ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΤΡΟΝΙΚΗΣ ΚΑΙ
ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Τεχνικές Βελτιστοποίησης για Αποθήκευση Πολυμεσικών Δεδομένων στο Cloud

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Λύδια Β. Ανυφαντάκη

Επιβλέπων : Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π

Αθήνα, Ιούλιος 2013

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΤΡΟΝΙΚΗΣ ΚΑΙ
ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Τεχνικές Βελτιστοποίησης για Αποθήκευση Πολυμεσικών Δεδομένων στο Cloud

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Λύδια Β. Ανυφαντάκη

Επιβλέπων : Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 17η Ιουλίου 2013.

.....	.....	.....
Θ. Βαρβαρίγου Καθηγήτρια Ε.Μ.Π.	Β. Λούμος Καθηγητής Ε.Μ.Π.	Σ. Παπαβασιλείου Αναπληρωτής Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2013

.....

Λύδια Β. Ανυφαντάκη

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π

Copyright © Λύδια Β. Ανυφαντάκη, 2013

Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

Ευχαριστίες

Η διπλωματική αυτή εκπονήθηκε στο Εργαστήριο Distributed Knowledge and Media Systems Group του Τομέα Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής της σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου, υπό την επίβλεψη της Καθηγήτριας Θεοδώρας Βαρβαρίγου.

Θα ήθελα να ευχαριστήσω θερμά την κ. Θεοδώρα Βαρβαρίγου για την υποστήριξη και καθοδήγηση που μου προσέφερε, καθώς και για την ευκαιρία που μου έδωσε να ασχοληθώ με την ερευνητική περιοχή του Cloud Computing.

Ιδιαίτερες ευχαριστίες απευθύνονται στους υποψήφιους διδάκτορες Βαφειάδης Γιώργος και Μουλός Βρεττός, οι οποίοι με την πολύτιμη βοήθεια τους και τις στοχευμένες συμβουλές τους συνέβαλλαν στην εκπόνηση αυτής της διπλωματικής εργασίας.

Τέλος θα ήθελα να ευχαριστήσω την οικογένειά μου για την διαρκή υποστήριξη που μου προσέφεραν καθ’ όλη τη διάρκεια των σχολικών και φοιτητικών μου χρόνων, καθώς και τους φίλους που βρίσκονταν δίπλα μου όλα αυτά τα χρόνια.

Περίληψη

Στην παρούσα διπλωματική εργασία μελετάται το πρόβλημα αναζήτησης δεδομένων με βάση το περιεχόμενό τους και εξετάζεται η αποθήκευση των metadata των αρχείων με τρόπους που ευνοούν την γρήγορη απόκριση της αναζήτησης.

Ο εντοπισμός δεδομένων με βάση το περιεχόμενό τους γίνεται με χρήση των metadata, αρχείων που περιέχουν την περιγραφή του περιεχομένου τους. Τα αρχεία στα οποία θα εκτελεστεί η αναζήτηση συνδυάζονται με metadata που υπακούν σε διάφορα πρότυπα περιγραφής περιεχομένου με γνωστότερο το MPEG-7. Τα αρχεία βρίσκονται αποθηκευμένα σε repositories, ενώ για τα metadata τους επιλέγεται κατάλληλος μετασχηματισμός και αποθήκευσή τους, ανάλογα με τον τύπο της υπηρεσίας που προσφέρει την εφαρμογή αναζήτησης και της διαθέσιμης υποδομής. Ευρέως χρησιμοποιούνται σκληροί δίσκοι και σχεσιακές βάσεις δεδομένων για την διατήρηση των metadata, πρακτικές που παρουσιάζουν διάφορα προβλήματα στην ταχύτητα εκτέλεσης της αναζήτησης, καθώς για την εξέταση των metadata απαιτούνται εργαλεία XML Parsing και εφαρμογή πολλών πολύπλοκων JOINS.

Η διπλωματική αυτή προσπαθεί να λύσει το παραπάνω πρόβλημα με εφαρμογή εναλλακτικών τρόπων αποθήκευσης, και συνεπώς εκτέλεσης της αναζήτησης, με χρήση των σύγχρονων τεχνολογιών cloud. Συγκεκριμένα μελετάται ο τρόπος αποθήκευσης και αναζήτησης περιεχομένου σε 3D αρχεία, τα metadata των οποίων υπακούν στα πρότυπα MPEG-7, MPEG-21 και X3D. Καθώς τα πρότυπα αυτά χαρακτηρίζονται από πολλές ελευθερίες στον τρόπο περιγραφής του περιεχομένου, η αναπαράστασή τους σε σχεσιακές βάσεις δεδομένων απαιτεί χρήση πολλών διαφορετικών tables και διατήρηση πολλών relations για την πλήρη αναπαράσταση των μεταξύ τους σχέσεων. Για το λόγο αυτό επιλέχθηκε για την αποθήκευση των metadata η NoSQL βάση δεδομένων MongoDB, η οποία επιτρέπει την ευέλικτη και χωρίς περιορισμούς αναπαράσταση των metadata σε αρχεία τύπου BSON, τα οποία διατηρούν τα δεδομένα τους σε key/value pairs. Ένας ακόμα λόγος που επιλέχθηκε η MongoDB είναι το άμεσο scalability που προσφέρει με προσθήκη κόμβων στο MongoDB cluster, κάτι που σε σχεσιακές βάσεις δεδομένων αποτελεί πολύπλοκη διαδικασία και απαιτεί μεγάλη μεταφορά δεδομένων στο δίκτυο. Η αναζήτηση εκτελείται με εφαρμογή MongoDB Queries, προς πλήρη εκμετάλλευση των βελτιστοποιημένων τεχνικών που έχουν αναπτυχθεί ειδικά για το σκοπό αυτό. Παράλληλα υλοποιήθηκε μια μέθοδος αναζήτησης ενός query που απαιτεί aggregation των δεδομένων με εφαρμογή του προγραμματιστικού μοντέλου MapReduce.

Στόχος είναι η συγκριτική μελέτη των σύγχρονων τεχνικών αναζήτησης περιεχομένου και της νέας προσέγγισης που προτείνεται από την διπλωματική αυτή, προς ανάδειξη των καταλληλότερων τεχνικών για κάθε είδους εφαρμογή που υλοποιεί αναζήτηση αρχείων με βάση το περιεχόμενό τους.

Λέξεις Κλειδιά

MPEG-21, MPEG-7, NoSQL βάσεις δεδομένων, X3D, XML, XSD, διάσχιση XML αρχείων, εικονοποίηση, ενδιάμεσο λογισμικό, κατανεμημένος προγραμματισμός, μεταδεδομένα, σχεσιακές βάσεις δεδομένων, υπολογιστικό νέφος

Abstract

The presented diploma thesis deals with the issues raised by content-based data search and examines different approaches on storing metadata in order to achieve fast response time.

Locating data according to their content is done by using their metadata, term that refers to the description of the content of the data, which obey in various standards of describing content. One of the most known standards for describing multimedia content is MPEG-7, introduced by the MPEG expert working group. Search regards files that are stored in repositories, while their metadata are appropriately modified and stored according to the needs and available infrastructure of the service provider. Most commonly metadata are stored in hard drives in a form of directory-hierarchy and alternatively within relational databases. Both practices have issues regarding the response time of the search, due to the XML Parsing tools used to access the metadata and to the complicated structure that is required to perform the search.

This diploma thesis attempts to solve these problems by applying alternative methods of storing the metadata and therefore by using different, most advanced tools to perform the search, with the use of modern cloud technologies. The search will be performed on 3D object files, whose metadata obey to the MPEG-7, MPEG-21 and X3D standards. These standards are characterized by many freedoms regarding the way of representing the description of the content of the data. Therefore the use of relational databases would require the maintenance of a large number of tables in conjunction with the complex in between relations to fully represent the structure of the metadata. To address this problem the NoSQL Database MongoDB is used for the storing of the metadata. MongoDB uses BSON files, which store their data in key/value pairs. This kind of format allows the representation of complex schemes in the simplest and most flexible ways. Another reason for choosing MongoDB is the easy and immediate scalability provided, simply by adding compute nodes to the MongoDB cluster. With relational databases such an addition would require large amount of data to be transferred through the network in order to perform even the simplest tasks. The search is performed with the use of MongoDB queries, in order to fully exploit the advantages of using the optimized internal techniques developed by MongoDB. Finally, in order to tackle the problems issued by applying aggregation functions to relational databases, a technique that conforms to the MapReduce programming model was developed.

Goal is the comparative study of current ways to perform content-based search with the new approach suggested by this diploma thesis, in order to highlight the advantages of each technique according to the type of provided service that performs content-based search.

Keywords

MPEG-21, MPEG-7, NoSQL databases, X3D, XML, XSD, XML Parsing, virtualization, middleware, distributed computing, metadata, relational databases, cloud computing

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Κατάλογος Ερωτημάτων

Κατάλογος Εξισώσεων

Κεφάλαιο 1
Εισαγωγή

Στην παρούσα διπλωματική θα παρουσιαστούν τεχνικές και μέθοδοι αναζήτησης πληροφορίας που αφορούν σε multimedia δεδομένα, χρησιμοποιώντας νέες μορφές διαχείρισης και αποθήκευσης δεδομένων, και συγκεκριμένα υποδομές Cloud. Για την αναζήτηση multimedia αρχείων με βάση το περιεχόμενό τους, χρησιμοποιήθηκαν τα metadata τους, που υπακούν στα πρότυπα περιγραφής περιεχομένου MPEG-7, MPEG-21 και X3D. Στόχος αποτελεί ο εντοπισμός της καταλληλότερης μεθόδου αποθήκευσης και αναζήτησης multimedia αρχείων, ώστε να προσφέρονται αποδοτικότερες υπηρεσίες ανάλογα με τον τύπο των εναλλακτικών εφαρμογών που χρησιμοποιούνται, των γνώσεων του ανθρώπινου δυναμικού που τις διαχειρίζεται, της διαθέσιμης υποδομής και της δυνατότητας μετάβασης σε μοντέλα που χρησιμοποιούν σύγχρονες και νεότερες τεχνολογίες.

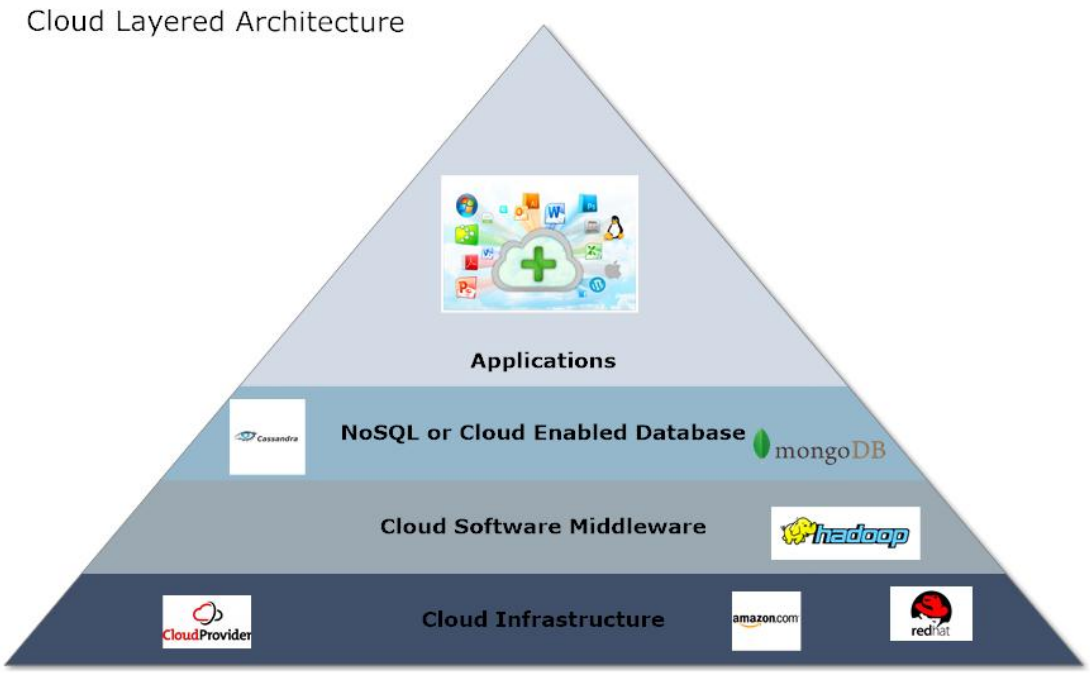
Οι μέθοδοι που αναπτύχθηκαν από αυτήν την διπλωματική χρησιμοποιούν τεχνολογίες που βασίζονται στην νοοτροπία που εισήγαγαν μοντέλα τύπου Cloud. Μπορούμε να διαχωρίσουμε την αρχιτεκτονική τέτοιων μοντέλων σε τέσσερα αφαιρετικά επίπεδα. Στο πρώτο επίπεδο βρίσκεται η υλικοτεχνική υποδομή, δηλαδή το hardware που θα χρησιμοποιηθεί για την ανάπτυξη και προσφορά των εφαρμογών αναζήτησης. Οι πόροι που ανήκουν σε αυτό το αφαιρετικό επίπεδο, αποτελούν μεταξύ άλλων ο αποθηκευτικός χώρος, οι επεξεργαστές και το δίκτυο. Βασικό χαρακτηριστικό της φιλοσοφίας που εισήγαγαν τα cloud μοντέλα παροχής υπηρεσιών αποτελεί η κατανομή της αποθήκευσης και επεξεργασίας των δεδομένων. Για το σκοπό αυτό αναπτύσσονται ολοένα και περισσότερο, clusters υπολογιστών που διαθέτουν μέχρι και εκατοντάδες χιλιάδες επεξεργαστές, που χρησιμοποιούνται για κατανομή της επεξεργαστικής ισχύς και παραλληλοποίησης των προβλημάτων. Εκτός από την προσφορά κατανεμημένης επεξεργαστικής ισχύς, προσφέρονται επίσης κατανεμημένα συστήματα αποθήκευσης δεδομένων, όπως στην περίπτωση της Amazon, η οποία μέσω του Amazon Elastic Compute Cloud παρέχει αποθηκευτικό χώρο, υποδομές δικτύου και επεξεργαστική ισχύ στους χρήστες που αναπτύσσουν εφαρμογές με ευέλικτο τρόπο, ώστε ανά πάσα στιγμή να χρησιμοποιούνται μόνο οι απαραίτητοι πόροι. Οι πόροι αυτοί παρέχονται μέσω virtual machines, εικονικών μηχανών, με τους χρήστες να έχουν απόλυτο έλεγχο στο λειτουργικό σύστημα, στο λογισμικό και σε όλα τα εργαλεία που θα εγκαταστήσουν σε αυτούς. Στην Εικόνα 1 στο χαμηλότερο επίπεδο της πυραμίδας παρουσιάζονται ορισμένα παραδείγματα παρόχων Cloud υποδομής.

Στο αμέσως υψηλότερο αφαιρετικό επίπεδο της αρχιτεκτονικής που ακολουθούν τα cloud μοντέλα, όπως φαίνεται και στην Εικόνα 1, βρίσκεται το Middleware. Το Middleware αποτελεί το ειδικό λογισμικό που εγκαθίσταται στην υποδομή που παρέχεται από το προηγούμενο αφαιρετικό επίπεδο και είναι υπεύθυνο για την διαχείρισή της. Εκμεταλλεύεται τα χαρακτηριστικά του hardware και προσφέρει επιπλέον λειτουργικότητες προς βελτίωση της απόδοσης των εφαρμογών που θα αναπτυχθούν. Μία από αυτές αποτελεί η διαχείριση των σκληρών δίσκων της υποδομής, όπως για παράδειγμα με εφαρμογή κάποιας τεχνικής αποθήκευσης RAID ανάλογα με την απαιτούμενη απόδοση και το επιθυμητό πλήθος αντιγράφων που διατηρούνται για τα δεδομένα. Μία άλλη παρεχόμενη λειτουργικότητα είναι η επιλογή της τοποθεσίας των δεδομένων, σε διαφορετικά racks ή datacenters, και το πλήθος

των αντιγράφων τους που θα διατηρούνται στο cluster, με σκοπό την προσφορά υψηλής διαθεσιμότητας στα δεδομένα. Ένα από τα πιο γνωστά Middleware αποτελεί το Apache Hadoop, λογισμικό που εγκαθίσταται σε commodity hardware και αναλαμβάνει την αποθήκευση των δεδομένων με χρήση του Hadoop Distributed File System, προσφέροντας υψηλό bandwidth σε όλο το cluster. Σε αυτό το κομμάτι περιλαμβάνεται και η χρήση του MapReduce Framework, του προγραμματιστικού μοντέλου που χρησιμοποιείται στην παράλληλη επεξεργασία μεγάλου όγκου ανεξάρτητων και κατανεμημένων δεδομένων, μέσω εφαρμογής τεχνικών διαίρει και βασίλευε.

Η νέα φιλοσοφία αρχιτεκτονικής των πληροφοριακών συστημάτων οδήγησε στην αναζήτηση νέων τρόπων διαχείρισης και δόμησης των βάσεων δεδομένων. Λόγω του καταμερισμού της επεξεργαστικής ισχύς και αποθήκευσης έπρεπε να δημιουργηθούν βάσεις ικανές να εκμεταλλευτούν στο μέγιστο βαθμό τις νέες τεχνολογίες με τα ιδιαίτερα χαρακτηριστικά τους και τον καινοτόμο τρόπο διαχείρισης των προβλημάτων. Την απάντηση έδωσαν οι NoSQL βάσεις δεδομένων, το κύριο χαρακτηριστικό των οποίων είναι η εφαρμογή τους σε κατανεμημένα συστήματα αποθήκευσης δεδομένων, η παροχή υψηλής διαθεσιμότητας στα δεδομένα και η ικανότητά τους να κλιμακώνονται οριζόντια. Χρησιμοποιούνται κυρίως για διαχείριση πολύ μεγάλου όγκου δεδομένων, σε web εφαρμογές και σε περιπτώσεις που δεν απαιτούνται πολλά JOINS, αλλά εφαρμογή απλών πράξεων, για παράδειγμα με χρήση του προγραμματιστικού μοντέλου MapReduce. Στις NoSQL βάσεις δεδομένων τα δεδομένα αποθηκεύονται συνήθως σε key/value pairs, μορφή ιδανική για την εφαρμογή απλών πράξεων, με άμεσο αποτέλεσμα την αύξηση της απόδοσης των εφαρμογών.

Στο τελευταίο αφαιρετικό επίπεδο της πυραμίδας παροχής υπηρεσιών cloud βρίσκονται οι τελικές εφαρμογές, δηλαδή το λογισμικό που αναπτύσσεται από τους χρήστες των NoSQL βάσεων δεδομένων. Καθώς τα εργαλεία που χρησιμοποιούνται είναι καινούργια ήταν απαραίτητη η αλλαγή στην νοοτροπία ανάπτυξης εφαρμογών και συγκεκριμένα στον τρόπο γραφής του κώδικα του παρεχόμενου λογισμικού. Αναπτύχθηκαν νέοι τρόποι διασύνδεσης και επικοινωνίας των εφαρμογών με τις βάσεις δεδομένων, νέα προγραμματιστικά περιβάλλοντα καθώς και νέα πακέτα λογισμικού για την μετάβαση από παλαιότερες τεχνολογίες σε νέες.



Εικόνα 1 Cloud Layered Architecture

1.1 Αντικείμενο της Διπλωματικής

Ένας από τους σημαντικότερους τομείς παροχής υπηρεσιών μέσω Διαδικτύου αφορά στην αναζήτηση πληροφορίας. Ο ολοένα αυξανόμενος όγκος δεδομένων που διακινείται έχει κάνει επιτακτική την ανάγκη παροχής εργαλείων αναζήτησης ικανών να εντοπίζουν τις ζητούμενες, έγκυρες πληροφορίες γρήγορα και αξιόπιστα. Τις μηχανές αναζήτησης διέπουν ορισμένοι βασικοί κανόνες λειτουργίες. Κάθε μία δέχεται ερωτήματα ορισμένης μορφής από τους χρήστες, πραγματοποιεί την αναζήτηση σε κάποια βάση δεδομένων και επιστρέφει τα αποτελέσματα που απαντούν στο υποβαλλόμενο ερώτημα πίσω στους χρήστες. Η σημασία ύπαρξης εξελιγμένων μηχανών αναζήτησης φαίνεται από την διευρυμένη βάση των χρηστών του Διαδικτύου, μικρό ποσοστό των οποίων γνωρίζει την ακριβή τοποθεσία των δεδομένων που αναζητεί. Ταυτόχρονα πολλοί είναι οι χρήστες που δεν γνωρίζουν ακριβώς πώς να εκφράσουν τις πληροφορίες που αναζητούν, με τις μηχανές αναζήτησης να πρέπει να μετασχηματίσουν τα ερωτήματα που υποβάλλονται και να αναζητήσουν δεδομένα με βάση το περιεχόμενό τους. Αυτός ο τύπος αναζήτησης γίνεται με χρήση των metadata των δεδομένων, δηλαδή της περιγραφής του περιεχομένου τους.

Η διπλωματική αυτή εξετάζει τους σύγχρονους τρόπους αναζήτησης με βάση το περιεχόμενο των αρχείων και προτείνει νέους που ακολουθούν τις νέες τεχνολογίες που παρουσιάστηκαν προηγουμένως. Συγκεκριμένα η αναζήτηση θα αφορά σε 3D αρχεία, τα metadata των οποίων υπακούν στα πρότυπα περιγραφής περιεχομένου MPEG-7, MPEG-21 και X3D και αποτελούν αρχεία τύπου XML. Μέχρι σήμερα οι τεχνικές που χρησιμοποιούνται για την αναζήτηση metadata χρησιμοποιούν repositories για την αποθήκευση των 3D αρχείων και directories και σχεσιακές βάσεις δεδομένων για την αποθήκευση των metadata τους. Κάθε φορά που υποβάλλεται ένα ερώτημα, πραγματοποιείται διάσχιση των XML αρχείων ή των tables της σχεσιακής βάσης, και επιστρέφονται τα ονόματα των 3D αρχείων που απαντούν στα χαρακτηριστικά του ερωτήματος που υποβλήθηκε. Ωστόσο έχει παρατηρηθεί πως αυτός ο τρόπος αναζήτησης σε πολλές περιπτώσεις δεν είναι αποδοτικός, καθώς απαιτούνται εργαλεία Parsing XML αρχείων και εφαρμογής πολλών JOINS προκειμένου να απαντηθούν τα ερωτήματα. Για το λόγο αυτό προτείνονται από την διπλωματική αυτή εναλλακτικοί τρόποι αποθήκευσης των metadata, με χρήση NoSQL βάσεων δεδομένων, και αναζήτηση σε αυτά με χρήση των εξειδικευμένων queries τέτοιων βάσεων και με εφαρμογή του προγραμματιστικού μοντέλου MapReduce. Εξετάζεται κατά πόσον αυτές οι τεχνικές οδηγούν σε ταχύτερες αποκρίσεις των εφαρμογών αναζήτησης και σε ποιες περιπτώσεις μπορούν να αντικαταστήσουν παλαιότερα μοντέλα αποθήκευσης και αναζήτησης πληροφορίας.

1.2 Οργάνωση Κειμένου

Το κείμενο της διπλωματικής μπορεί να χωριστεί σε δύο υποενότητες. Στην πρώτη, Κεφάλαια 2, 3, 4 και 5, γίνεται η ανάλυση των τεχνολογιών που συνθέτουν τα αφαιρετικά επίπεδα που αναλύθηκαν προηγουμένως και αφορούν σε όλα τα στάδια ανάπτυξης εφαρμογών, από το επίπεδο του hardware μέχρι το τελευταίο επίπεδο του software. Στην δεύτερη υποενότητα ανήκουν τα Κεφάλαιο 6, 7 και 8, όπου γίνεται η εκτενής ανάλυση και περιγραφή του προβλήματος που εξετάστηκε, της υλοποίησης των νέων τεχνικών αποθήκευσης δεδομένων και της παρουσίασης των συμπερασμάτων που προέκυψαν από την συγκριτική μελέτη των παλαιών και νέων μεθόδων αναζήτησης.

Πιο αναλυτικά στο Κεφάλαιο 2 γίνεται μια εισαγωγή στις έννοιες που εισήγαγε το Cloud Computing, παρουσιάζονται τα χαρακτηριστικά γνωρίσματά του, αναφέρονται οι διαφορετικές cloud υπηρεσίες που παρέχονται και αναλύονται ορισμένοι χαρακτηριστικοί πάροχοι τέτοιων υπηρεσιών, με σκοπό την ανάδειξη των πολλών δυνατοτήτων που προσφέρονται με χρήση της τεχνολογίας Cloud Computing. Στο Κεφάλαιο 3 παρουσιάζεται το αμέσως ανώτερο αφαιρετικό επίπεδο κατά την ανάπτυξη εφαρμογών, το Middleware.

Αναλύονται οι λόγοι που οδήγησαν στην ανάπτυξη των κατανεμημένων συστημάτων αποθήκευσης και επεξεργασίας δεδομένων και παρουσιάζεται εκτενώς το Apache Hadoop, καθώς αποτελεί χαρακτηριστικό παράδειγμα Middleware λογισμικού που χρησιμοποιείται για την βελτίωση του τρόπου αποθήκευσης των δεδομένων σε clusters υπολογιστών. Αναλύεται επίσης το προγραμματιστικό μοντέλο MapReduce και δίνεται, για την καλύτερη κατανόησή του, ένα παράδειγμα χρήσης του. Στο Κεφάλαιο 4 γίνεται η ανάλυση των NoSQL βάσεων δεδομένων, οι οποίες αποτελούν το προτελευταίο στάδιο ανάπτυξης υπηρεσιών, καθώς χρησιμοποιούνται από τις εφαρμογές για την αξιόπιστη αποθήκευση και ανάκτηση των δεδομένων τους. Παρουσιάζονται εισαγωγικά κομμάτια των βάσεων δεδομένων και αναλύονται οι λόγοι που οδήγησαν στην αλλαγή του τρόπου φιλοσοφίας δόμησής τους. Περιγράφονται επιγραμματικά τα διαφορετικά είδη NoSQL βάσεων δεδομένων που συναντώνται μέχρι σήμερα και αναλύονται εκτενώς δύο από αυτές, η CouchDB και η MongoDB. Σ το Κεφάλαιο 5 εισάγονται τα πρότυπα περιγραφής περιεχομένου MPEG-7 και MPEG-21 και δίνονται τα κυριότερα χαρακτηριστικά τους, με έμφαση σε εκείνα που βρίσκονται πιο κοντά στο αντικείμενο της διπλωματικής.

Ακολουθεί η συγκριτική μελέτη των παλαιότερων μεθόδων αναζήτησης και των νέων που προτείνονται από αυτή τη διπλωματική. Στο Κεφάλαιο 6 παρουσιάζεται το πρόβλημα που μας απασχόλησε και οι σημερινοί τρόποι επίλυσής του. Στη συνέχεια περιγράφεται εκτενώς η υλοποίηση των νέων τεχνικών, οι οποίες χωρίστηκαν σε δύο κατηγορίες. Στην πρώτη ανήκουν οι μέθοδοι αναζήτησης που πραγματοποιήθηκαν με χρήση των queries που δίνονται από την βάση δεδομένων. Στην δεύτερη ανήκουν οι μέθοδοι αναζήτησης που εφαρμόζουν το προγραμματιστικό μοντέλο MapReduce. Στο Κεφάλαιο 7 πραγματοποιείται η συγκριτική μελέτη όλων των τεχνικών, παλαιών και νεότερων, ανάλογα με την κατηγορία στην οποία ανήκουν. Ο λόγος που έγινε αυτός ο διαχωρισμός είναι η σύγκριση όμοιων φιλοσοφιών και η ανάδειξη της ανωτερότητας της κάθε μεθόδου στον τομέα που ανήκει. Τέλος παρουσιάζονται τα συμπεράσματα από την σύγκριση των μεθόδων στο Κεφάλαιο 8.

Κεφάλαιο 2
Cloud Computing

Με τον όρο "*Cloud Computing*" ή "*Υπολογιστικό Νέφος*" εννοούμε την παροχή υπολογιστικών πόρων ή/και υπηρεσιών μέσω ενός δικτύου. Οι υπηρεσίες Cloud Computing κάνουν δυνατή την ευέλικτη διάθεση τεχνολογικού κεφαλαίου των παρόχων στους τελικούς χρήστες, μέσω της δικτυακής παροχής πρόσβασης σε ένα σύνολο παραμετροποιήσιμων υπολογιστικών πόρων. Υπολογιστικούς πόρους ονομάζουμε κάθε φυσικό ή εικονικό component ενός συστήματος υπολογιστών, καθώς και κάθε συσκευή που είναι συνδεδεμένη σε αυτό. Στους υπολογιστικούς πόρους που διατίθενται μέσω cloud περιλαμβάνονται αποθηκευτικοί χώροι, χρόνος σε CPU, μνήμη, αρχεία, δίκτυα, servers, εφαρμογές και υπηρεσίες.

2.1 Εισαγωγή

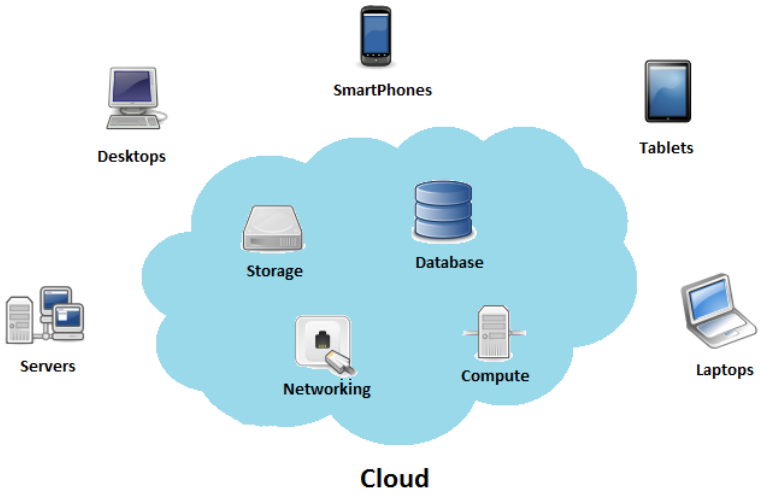
Πριν εμφανιστεί το cloud η ανάπτυξη εφαρμογών βασιζόταν αποκλειστικά σε ιδίους πόρους των χρηστών. Οι πάροχοι των υπηρεσιών ανέπτυσσαν τις εφαρμογές τους σε δικό τους εξοπλισμό και στη συνέχεια οι χρήστες των εφαρμογών εγκαθιστούσαν το λογισμικό στα δικά τους μηχανήματα. Παρότι με αυτόν τον τρόπο εξασφαλίζεται ανεξαρτησία και πλήρης έλεγχος των δεδομένων τόσο του παρόχου όσο και των τελικών χρηστών, δημιουργούνται ταυτόχρονα πολλά προβλήματα.

Για την παροχή υπηρεσιών απαιτείται κατ' αρχάς μεγάλο αρχικό κεφάλαιο το οποίο αφορά στο λογισμικό στο οποίο θα αναπτυχθεί η υπηρεσία, datacenters, servers, εγκατάσταση δικτύου, βάσεις δεδομένων καθώς και προσωπικό για την λειτουργία και ρύθμιση όλων των παραπάνω. Το κόστος συντήρησης είναι επίσης μεγάλο, καθώς είναι απαραίτητο να αναπτυχθούν μηχανισμοί ανάκτησης δεδομένων σε περίπτωση failover, να είναι εγγυημένη η ασφάλεια των δεδομένων των χρηστών και να εξασφαλίζεται διαρκής ενημέρωση και έλεγχος του συστήματος ώστε νέες εκδόσεις λογισμικού να μην θέτουν σε κίνδυνο την ομαλή λειτουργία της υπηρεσίας. Συχνά δεν χρησιμοποιείται ολόκληρος ο εξοπλισμός ή απαιτούνται περισσότερα μηχανήματα από τα διαθέσιμα προκειμένου να καλυφθούν οι ανάγκες των χρηστών. Στην πρώτη περίπτωση πόροι μένουν αναξιοποίητοι ενώ στη δεύτερη οι χρήστες δεν λαμβάνουν ομαλές και σίγουρες υπηρεσίες.

Για τους παραπάνω λόγους ήταν απαραίτητη μια αλλαγή στον τρόπο παροχής υπηρεσιών. Το cloud παρέχει λύσεις στους παραπάνω περιορισμούς ανάπτυξης εφαρμογών με ιδίους πόρους, εξασφαλίζοντας παράλληλα την ασφάλεια των δεδομένων των τελικών χρηστών και των παρόχων των υπηρεσιών. Το cloud, μία αναπαράσταση του οποίου δίνεται από την Εικόνα 2, προσφέρει τη δυνατότητα της απομακρυσμένης παροχής εφαρμογών και υπηρεσιών. Χρησιμοποιείται τόσο για την ανάπτυξη των υπηρεσιών, όσο και για την προσφορά τους στους τελικούς χρήστες. Όσον αφορά στην ανάπτυξη των εφαρμογών, το cloud χρησιμοποιείται για εξοικονόμηση πόρων. Οι πάροχοι των υπηρεσιών αναπτύσσουν τις εφαρμογές τους, όχι σε ιδίο εξοπλισμό, αλλά σε λογισμικό που τρέχει σε απομακρυσμένα μηχανήματα. Με αυτή τη τεχνική η εγκατάσταση και διαχείριση του απαραίτητου λογισμικού

πραγματοποιείται χωρίς τη συμμετοχή του παρόχου της αναπτυσσόμενη εφαρμογής, αλλά από την πλευρά του cloud, δηλαδή από τον πάροχο cloud computing. Ταυτόχρονα τα δεδομένα δεν διατηρούνται σε μηχανήματα του παρόχου, αλλά απομακρυσμένα, διαμοιραζόμενα datacenters. Σε αυτά έχουν αναπτυχθεί αυτόματοι μηχανισμοί failover και ασφάλειας των δεδομένων, καθώς και εργαλεία δυναμικής παραχώρησης μηχανημάτων ανάλογα με τις εκάστοτε απαιτήσεις των παρόχων σε υπολογιστικούς πόρους.

Το cloud χρησιμοποιείται επίσης για την διάθεση υπηρεσιών στους τελικούς χρήστες. Μέσω του cloud οι εφαρμογές τρέχουν απομακρυσμένα, χωρίς ανάγκη εγκατάστασης λογισμικού στον εξοπλισμό των τελικών χρηστών. Οι ενημερώσεις των εφαρμογών, η αναβάθμιση και συντήρησή τους δεν γίνεται στην πλευρά του χρήστη, αλλά από τον πάροχο. Οι χρήστες αρκεί να συνδεθούν στο δίκτυο, από οποιαδήποτε συσκευή, PC, tablet, Smartphone, και άμεσα μπορούν να χρησιμοποιήσουν τις εφαρμογές. Όλα τα παραπάνω χαρακτηριστικά του cloud, οι διαφορετικοί τύπου υπηρεσιών που παρέχονται καθώς και κάποια ζητήματα που εγείρονται κατά τη χρήση του, αναλύονται στις επόμενες ενότητες.



Εικόνα 2: Cloud Computing

2.2 Χαρακτηριστικά του Cloud Computing

Όπως αναφέρθηκε, ο κλάδος παροχής υπηρεσιών αντιμετωπίζει διάφορα προβλήματα. Από την πλευρά των παρόχων των υπηρεσιών απαιτείται μεγάλο αρχικό κεφάλαιο, ειδίκευση σε θέματα ασφαλείας των δεδομένων, ανάπτυξη μηχανισμών που εξασφαλίζουν την αξιοπιστία της υπηρεσίας, διασύνδεση των δεδομένων ώστε να τρέχει η υπηρεσία σε πολλές πλατφόρμες και διατήρηση εξοπλισμού και προσωπικού. Από την πλευρά των τελικών χρηστών είναι απαραίτητη η εγκατάσταση της υπηρεσίας σε κάθε μηχάνημά τους, η αγορά λογισμικού και η διαρκής ενημέρωση των εφαρμογών τους. Παρακάτω παρουσιάζονται τα χαρακτηριστικά του cloud τα οποία δίνουν λύσεις σε όλα τα παραπάνω προβλήματα.

Scalability – Επεκτασιμότητα

Στο cloud είναι δυνατή η χειροκίνητη ή δυναμική προσθήκη και αφαίρεση κόμβων εκτέλεσης εργασιών ανάλογα με την αυξομείωση των απαιτήσεων, χωρίς να αλλοιώνεται η υπηρεσία που παρέχεται στο χρήστη. Όσον αφορά στους πόρους, αυτοί εξοικονομούνται καθώς τα μηχανήματα χρησιμοποιούνται μόνο όταν χρειάζονται.

Virtualization - Εικονοποίηση

Με τη μέθοδο της εικονοποίησης οι λεπτομέρειες υλοποίησης και κατάστασης στην οποία βρίσκονται οι πόροι αποκρύπτονται από το χρήστη. Δίνεται η εντύπωση του ενιαίου κάτι που μπορεί να μην ισχύει, όπως στην περίπτωση των κατανεμημένων συστημάτων. Οι χρήστες δεν έχουν τρόπο να γνωρίζουν ακριβώς που αποθηκεύονται τα δεδομένα τους ή που τρέχουν οι εφαρμογές τους, αλλά έχουν πρόσβαση σε αυτά σαν να ήταν όλα στον προσωπικό τους υπολογιστή. Αυτό δίνει στον πάροχο της υπηρεσίας τη δυνατότητα να ελέγξει πως αποθηκεύονται τα δεδομένα, να αξιοποιήσει τυχόν ιδιαιτερότητές τους και να προσφέρει πιο γρήγορες και αποτελεσματικές υπηρεσίες.

Εικονικές μηχανές, virtual machines, VM, ονομάζουμε τις δομές που δίνουν την εντύπωση στο χρήστη ενός ολοκληρωμένου φυσικού μηχανήματος ενώ από φυσικής απόψεως είναι ένα σύνολο αρχείων και προγραμμάτων τα οποία τρέχουν πάνω σε ένα άλλο φυσικό μηχάνημα. Οι εικονικές μηχανές μπορούν να έχουν διαφορετικά χαρακτηριστικά και πόρους από το μηχάνημα του οποίου τους πόρους χρησιμοποιούν, όπως λειτουργικό σύστημα, αρκεί να μην ξεπερνούν τις δυνατότητες του μηχανήματος στο οποίο τρέχουν. Με αυτόν τον τρόπο είναι δυνατή η δημιουργία εικονικών μηχανών πλήρως ή μερικώς διαμορφωμένες από το χρήστη ανάλογα με τον πάροχο.

Reliability – Αξιοπιστία

Το cloud εγγυάται την αποθήκευση των δεδομένων των χρηστών και την αξιόπιστη μεταφορά τους. Τα δεδομένα αποθηκεύονται στους κόμβους από τους οποίους αποτελείται το cloud. Για την εξασφάλιση της αξιοπιστίας, τα δεδομένα δεν αποθηκεύονται σε ένα μόνο σημείο, Εικόνα 3. Δημιουργούνται αντίγραφα των δεδομένων είτε στο ίδιο κόμβο είτε σε άλλους κόμβους του cloud. Ο συνηθισμένος αριθμός αντιγράφων είναι τρία (3) (replication level three). Επίσης χρησιμοποιούνται μηχανισμοί ανάκτησης δεδομένων σε περίπτωση απώλειας ή αστοχίας ενός κόμβου αποθήκευσης.

Για την μέτρηση της αξιοπιστίας χρησιμοποιούνται δείκτες όπως ο *MTBF*, Mean Time Between Failures, δηλαδή ο μέσος χρόνος μεταξύ δύο διαδοχικών αποτυχιών του συστήματος, και ο *MTTF*, Mean Time to Failure, δηλαδή ο μέσος χρόνος μέχρι την αποτυχία του συστήματος. Ο MTBF χρησιμοποιείται για συστήματα στα οποία υπάρχει μηχανισμός ανάκτησης και ο MTTF χρησιμοποιείται για συστήματα στα οποία δεν υπάρχει μηχανισμός ανάκτησης, αλλά μηχανισμός αντικατάστασης.

$$MTBF = \frac{\sum(start\ of\ downtime - start\ of\ uptime\ )}{number\ of\ failures}$$

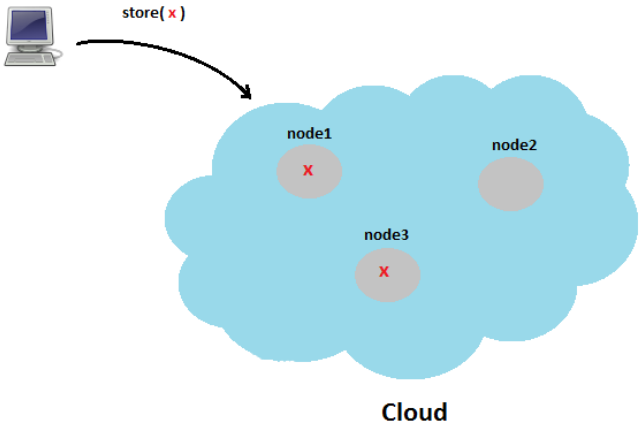
Εξίσωση 1 MTBF Equation

,όπου *downtime* είναι η χρονική στιγμή που το σύστημα απέτυχε και *uptime* η χρονική στιγμή που το σύστημα επανήλθε ακριβώς πριν το downtime.

$$MTTF \approx \frac{B_{10}}{0.1 n_{op}}$$

Εξίσωση 2 MTTF Equation

,όπου B_{10} είναι ο αριθμός των λειτουργιών που θα εκτελεστούν αν αποτύχει το 10% του συστήματος και n_{op} είναι το πλήθος των λειτουργιών που εκτελούνται ανά κύκλο λειτουργιών.



Εικόνα 3: Cloud Computing Reliability

Maintenance – Συντήρηση

Με τον όρο συντήρηση εννοούμε την διόρθωση των εφαρμογών από προγραμματιστικά λάθη και την εγκατάσταση νέων εκδόσεων της εφαρμογής. Με χρήση του cloud η συντήρηση των εφαρμογών είναι ευκολότερη. Οι χρήστες δεν ασχολούνται οι ίδιοι με την εγκατάσταση των ενημερώσεων σε κάθε συσκευή τους ξεχωριστά. Από όπου και αν τρέξει η εφαρμογή, προσωπικός υπολογιστής, tablet, Smartphone, είναι στη δικαιοδοσία του παρόχου να προσφέρει συμβατό λογισμικό, χωρίς καμία ενέργεια από την πλευρά του χρήστη. Ταυτόχρονα προγραμματιστικά λάθη μπορούν άμεσα να διορθωθούν σε όλους τους χρήστες.

Performance - Απόδοση

Η απόδοση αναφέρεται σε διάφορα κριτήρια όπως χαμηλό χρόνο απόκρισης, υψηλή διαμεταγωγή δεδομένων, χαμηλή χρήση υπολογιστικών πόρων, υψηλή διαθεσιμότητα του συστήματος, γρήγορη επεξεργασία δεδομένων και υψηλό εύρος ζώνης του δικτύου. Χρησιμοποιώντας το cloud ο τελικός χρήστης μπορεί να παρακολουθεί την απόδοση των εφαρμογών του. Οι πάροχοι των εφαρμογών έχουν στη διάθεσή τους μέσω του cloud καλύτερα μηχανήματα και περισσότερα εργαλεία βελτίωσης της απόδοσης, με αποτέλεσμα να παρέχουν πιο αποδοτικές υπηρεσίες. Οι τελικοί χρήστες μέσω των αναβαθμίσεων που γίνονται είτε σε επίπεδο υποδομής του cloud είτε σε επίπεδο βελτίωσης της εφαρμογής από τον πάροχο έχουν αυτόματα βελτιωμένη απόδοση.

Multitenancy

Κάθε χρήστης μιας υπηρεσίας cloud δεν χρειάζεται να διαθέτει δικό του μοναδικό αντίγραφο της εφαρμογής. Αρκεί ένα μοναδικό instance (στιγμιότυπο) της εφαρμογής το οποίο είναι ευέλικτο και μπορεί να προσαρμοστεί στην ανάγκες του κάθε χρήστη. Αυτό έχει ως αποτέλεσμα την εξοικονόμηση πόρων στο cloud, την ευκολότερη συντήρηση της εφαρμογής. Ταυτόχρονα διευκολύνεται η εξόρυξη δεδομένων (data mining). Οι πληροφορίες όλων των χρηστών είναι αποθηκευμένες σε ένα κεντρικό σημείο, χωρίς να υπολογιστούν οι μηχανισμοί αντιγραφής των δεδομένων για αξιοπιστία, το οποίο κάνει ευκολότερη την επεξεργασία των δεδομένων για εξαγωγή στατιστικών αποτελεσμάτων και πληροφοριών χρήσιμων στον πάροχο της υπηρεσίας.

Security – Ασφάλεια

Στην ασφάλεια συμπεριλαμβάνονται όλοι οι μηχανισμοί που προστατεύουν τα δεδομένα και τα μηχανήματα από ακούσια και μη εξουσιοδοτημένη πρόσβαση, αλλαγή ή καταστροφή, καθώς και από φυσικές καταστροφές. Στο cloud η ασφάλεια που παρέχεται είναι καλύτερη λόγω της κεντρικής αποθήκευσης των δεδομένων (centralization) και της προσφοράς περισσότερων εργαλείων που στοχεύουν στην ασφάλεια των δεδομένων. Σε κατανεμημένα συστήματα αποθήκευσης οι μηχανισμοί που εγγυούνται την ασφάλεια των δεδομένων είναι πιο πολύπλοκοι και δυσκολεύουν πολύ την παρακολούθηση του αρχείου καταγραφής ελέγχου (audit log). Μέσω του cloud οι χρήστες δεν χρειάζεται να ασχολούνται οι ίδιοι με την πολύπλοκη ασφάλεια των δεδομένων τους. Υπάρχουν και τα ιδιωτικά cloud στα οποία οι χρήστες έχουν έλεγχο πάνω στην υποδομή και σε ό,τι αφορά στην ασφάλεια της πληροφορίας, όπως ποιος έχει πρόσβαση στα δεδομένα, τι χρήση γίνεται. Ταυτόχρονα επιλέγονται και οι χειρισμοί για την ασφάλεια των προσωπικών και ευαίσθητων δεδομένων των χρηστών.

Cloud API (Application Programming Interface)

Διεπαφή Προγραμματισμού Εφαρμογών ονομάζουμε τη διεπαφή που χρησιμοποιούν οι εφαρμογές για να επικοινωνήσουν με το χρήστη ή άλλες εφαρμογές και μηχανήματα. Τα cloud APIs χρησιμοποιούνται για την επικοινωνία του λογισμικού με τις υπηρεσίες μέσω διεπαφών (interfaces). Τα cloud APIs συνήθως χρησιμοποιούν αρχιτεκτονικές τύπου REST. Απομακρύνονται από την λογική της αρχιτεκτονικής SOAP στην οποία κάθε λειτουργία πρέπει να ξαναγραφτεί και να σταλεί με XML αρχεία προς αποκωδικοποίηση. Η αρχιτεκτονική τύπου REST προσφέρει scalability και ανεξάρτητη ανάπτυξη των components του συστήματος.

Μειωμένα κόστη

Οι χρήστες και οι πάροχοι των υπηρεσιών – εφαρμογών δεν χρειάζεται να διαθέτουν και να συντηρούν δικό τους εξοπλισμό ή λογισμικό. Με το cloud είναι δυνατή η κοστολόγηση της χρήσης των υπολογιστικών πόρων κάτι που δεν είχε γίνει ποτέ στο παρελθόν. Το κόστος αφορούσε μόνο την αγορά του εξοπλισμού και όχι την χρήση του. Καθώς οι πάροχοι cloud χρησιμοποιούν πολιτική pay-as-you-go, δηλαδή οι χρήστες πληρώνουν ακριβώς όσο χρησιμοποιούν, τα κόστη τόσο για την ανάπτυξη μιας υπηρεσίας όσο και για τη χρήση της είναι εξαιρετικά μειωμένα.

On-Demand Self-Service

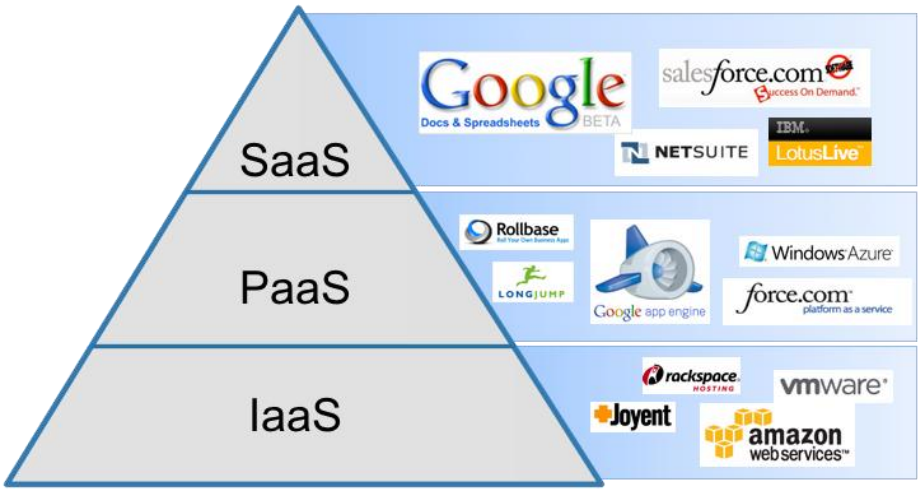
Ένα ακόμα χαρακτηριστικό του Νέφους είναι το “on-demand self-service”, το οποίο δίνει τη δυνατότητα στους χρήστες γρήγορα και εύκολα να δεσμεύουν και να παραμετροποιήσουν μηχανήματα και να αναπτύξουν εφαρμογές, χωρίς να είναι απαραίτητη η ανθρώπινη αλληλεπίδραση με υπευθύνους του cloud. Οι πάροχοι cloud προτρέπονται να δημιουργήσουν templates, πρότυπα, τα οποία, ύστερα από σύγκριση με άλλα, οδηγούν τους χρήστες να επιλέξουν γρήγορα τις υπηρεσίες που ταιριάζουν καλύτερα στις ανάγκες τους. Τα πρότυπα διαθέτουν προκαθορισμένα το λειτουργικό τους σύστημα, τη βάση δεδομένων, ρυθμίσεις ασφαλείας και υπηρεσίες web, καθώς και οδηγίες για εύκολη χρησιμοποίηση διαφορετικών cloud και παραμετροποίηση των εφαρμογών να ταιριάζουν σε αυτά. Διατίθενται επίσης εργαλεία για μετακίνηση εφαρμογών από το ένα περιβάλλον cloud στο άλλο.

2.3 Υπηρεσίες Cloud Computing

Σε αυτή την ενότητα παρουσιάζονται τα διαφορετικά είδη υπηρεσιών cloud computing που έχουν αναπτυχθεί μέχρι σήμερα. Ο διαχωρισμός των υπηρεσιών γίνεται με βάση το αφαιρετικό επίπεδο ελέγχου των χρηστών στους υπολογιστικούς πόρους που τους παρέχονται. Όπως φαίνεται στην Εικόνα 4, το υψηλότερο αφαιρετικό επίπεδο παροχής υπηρεσιών αποτελεί το SaaS. Σε αυτό παρέχονται στους τελικούς χρήστες εφαρμογές, οι οποίοι μπορούν μόνο να τις παραμετροποιήσουν ανάλογα με τις ανάγκες τους, χωρίς κανέναν έλεγχο στον τρόπο εκτέλεσής τους.

Αμέσως χαμηλότερα στην πυραμίδα βρίσκεται το PaaS, στο οποίο οι χρήστες αναπτύσσουν εφαρμογές. Σε αυτή την κατηγορία χρηστών παρέχονται έτοιμα προγραμματιστικά περιβάλλοντα, στα οποία αναπτύσσονται με απόλυτο έλεγχο οι εφαρμογές. Ωστόσο οι χρήστες δεν έχουν κανέναν έλεγχο στην υποδομή και το λειτουργικό σύστημα στο οποίο τρέχουν τα προγραμματιστικά περιβάλλοντα καθώς και στον τρόπο αποθήκευσης των δεδομένων τους.

Στο τελευταίο επίπεδο της πυραμίδας βρίσκεται η υπηρεσία IaaS. Σε αυτήν παρέχονται στους χρήστες μηχανήματα πλήρως ελέγξιμα, στα οποία επιλέγονται όλα τα χαρακτηριστικά τους, όπως λειτουργικό σύστημα και αποθηκευτικός χώρος. Ακολουθεί ανάλυση των τύπων cloud computing *IaaS*, *PaaS* και *SaaS*, καθώς και της πιο πρόσφατης υπηρεσίας *NaaS*, και δίνονται κάποια παραδείγματα παρόχων τους.



Εικόνα 4: Cloud Computing Services

Infrastructure as a Service (IaaS) – Υποδομή σαν Υπηρεσία

Σε αυτή την υπηρεσία παρέχεται στους χρήστες υποδομή, δηλαδή υπολογιστές, servers και υπολογιστικοί πόροι, είτε ως μηχανήματα ή πιο συχνά ως εικονικές μηχανές.

Ο πάροχος της υπηρεσίας είναι υπεύθυνος για την συντήρηση των μηχανημάτων στα οποία τρέχουν οι εικονικές μηχανές των χρηστών. Φροντίζει για την ασφάλεια των δεδομένων των χρηστών, την αξιοπιστία των μηχανημάτων ώστε σε περίπτωση αποτυχίας ενός κόμβου να μην αλλοιωθεί η απόδοση της υπηρεσίας και την επεκτασιμότητά τους ώστε δυναμικά να δίνονται στους χρήστες οι πόροι που χρειάζονται. Συνήθως εφαρμόζεται πολιτική πληρωμής ανάλογη με την ποσότητα των πόρων που δεσμεύονται και χρησιμοποιούνται.

Οι χρήστες έχουν τη δυνατότητα να ορίσουν μερικώς ή πλήρως τα χαρακτηριστικά των μηχανημάτων που χειρίζονται, CPU, RAM, σκληρός δίσκος, λειτουργικό σύστημα, δίκτυο

και άλλα. Έχουν τη δυνατότητα να χειριστούν τα μηχανήματα με απόλυτη ελευθερία, να εγκαταστήσουν λογισμικό, να στήσουν βάσεις δεδομένων και να υλοποιήσουν εφαρμογές ακριβώς όπως θα έκαναν σε δικά τους μηχανήματα. Το περιβάλλον πρέπει να είναι επίσης ικανό να χρησιμοποιηθεί για υλοποίηση υπηρεσιών PaaS και SaaS.

Ένα παράδειγμα αποτελεί το Rackspace Cloud, το οποίο ήταν ένας από τους πρώτους παρόχους IaaS. Διαθέτει τρεις κύριες υπηρεσίες, Cloud Files, Cloud Servers και Cloud Sites. Τα Cloud Files αποτελούν τον αποθηκευτικό χώρο των χρηστών. Η πρόσβαση σε αυτά γίνεται μέσω API ανοιχτού κώδικα. Τα δεδομένα κρατούνται σε τρία αντίγραφα. Με τους Cloud Servers οι χρήστες έχουν τη δυνατότητα να σηκώσουν servers μέσω ενός server API. Υποστηρίζεται η δυναμική διάθεση μηχανημάτων σε απότομες αλλαγές του φόρτου εργασίας. Με το Cloud Sites οι χρήστες διατηρούν απεριόριστο αριθμό ιστοσελίδων, e-mails και βάσεων δεδομένων.

Παράδειγμα: Amazon CloudFormation

Platform as a Service (PaaS) – Πλατφόρμα σαν Υπηρεσία

Το PaaS παρέχει στους χρήστες υπολογιστικές πλατφόρμες πάνω στις οποίες υλοποιούν εφαρμογές. Είναι το αμέσως παραπάνω λογικό επίπεδο παροχής υπηρεσιών από το IaaS.Το συνηθισμένο πακέτο περιλαμβάνει ήδη εγκατεστημένο λειτουργικό σύστημα, περιβάλλον προγραμματισμού, βιβλιοθήκες, βάση δεδομένων, δίκτυα και έναν web server. Προσφέρονται εργαλεία σχεδίασης των εφαρμογών και testing (ελέγχου) και υπηρεσίες σχεδιασμού βάσεων δεδομένων, υπηρεσιών web, application versioning, αποθήκευσης, ασφάλειας, αξιοπιστίας των δεδομένων, αυτόματου scaling ανάλογα με τις απαιτήσεις της εφαρμογής, διαχείρισης της κατάστασης των user interfaces και συνεργασίας απομακρυσμένων ομάδων ανάπτυξης μιας εφαρμογής.

Με το PaaS οι χρήστες αναπτύσσουν εφαρμογές που έχουν δημιουργήσει ή εφαρμογές που έχουν αποκτήσει, με τα εργαλεία και τη γλώσσα προγραμματισμού που υποστηρίζονται από τον πάροχο, αλλά δεν έχουν κανένα έλεγχο πάνω στην υποδομή. Δεν χρειάζεται να διατηρούν μηχανήματα και λογισμικό, κάτι που μειώνει σημαντικά το κόστος ανάπτυξης εφαρμογών. Προβλήματα συμβατότητας των εκδόσεων λογισμικού, ανάπτυξης μηχανισμών ασφαλείας για τα δεδομένα των χρηστών και χειροκίνητης δέσμευσης πόρων ανάλογα με τη χρήση της εφαρμογής εξαλείφονται. Τέλος η ποικιλία στους παρόχους βοηθάει στην επιλογή του κατάλληλου για την ανάπτυξη των επιθυμητών εφαρμογών.

Παράδειγμα: Windows Azure Compute

Software as a Service (SaaS) – Λογισμικό σαν Υπηρεσία

Το SaaS, γνωστό και ως "on-demand software", παρέχει στους χρήστες έτοιμες εφαρμογές, οι οποίες τρέχουν στο cloud και είναι προσβάσιμες μέσω ενός δικτύου. Είναι το αμέσως παραπάνω λογικό επίπεδο στην παροχή υπηρεσιών από το PaaS.Λογισμικό και δεδομένα βρίσκονται κεντρικά στο cloud και διατίθενται στους χρήστες μέσω ενός web browser. Ο πάροχος είναι υπεύθυνος για τη διάθεση εφαρμογών, τις οποίες προσφέρει στους χρήστες on-demand. Χρησιμοποιείται multi-tenant αρχιτεκτονική, με ένα μοναδικό version της εφαρμογής να εξυπηρετεί πολλούς χρήστες. Κάποιοι πάροχοι επιτρέπουν στους χρήστες την παραμετροποίηση των εφαρμογών. Οι ενημερώσεις και η συντήρηση των εφαρμογών γίνονται κεντρικά με αποτέλεσμα οι χρήστες να έχουν αυτόματα τις νέες εκδόσεις. Οι πάροχοι διαθέτουν μηχανισμούς scaling με αποτέλεσμα ο φόρτος εργασίας να κατανέμεται

ανάλογα με τις εκάστοτε απαιτήσεις. Καθώς το λογισμικό μπορεί να χρησιμοποιηθεί από διάφορες συσκευές, υπολογιστής, tablet, Smartphone, οι πάροχοι είναι υπεύθυνοι για την ανάπτυξη συμβατών εκδόσεων λογισμικού. Το SaaS προσφέρει αξιοπιστία και ασφάλεια των δεδομένων των χρηστών.

Οι χρήστες δεν έχουν τον έλεγχο της υποδομής ή τις πλατφόρμες πάνω στις οποίες τρέχουν οι εφαρμογές. Με το SaaS δεν απαιτείται πλέον η εγκατάσταση και διαρκής ενημέρωση του λογισμικού ούτε η δέσμευση αποθηκευτικού χώρου για αποθήκευση των δεδομένων σε κάθε συσκευή του χρήστη. Οι εφαρμογές αρκεί να νοικιαστούν μία φορά και μπορούν να χρησιμοποιηθούν από κάθε συσκευή του χρήστη. Με αυτόν τον τρόπο ζητήματα συμβατότητας μεταξύ των διάφορων συσκευών εξαλείφονται και δεν είναι απαραίτητη η μεταφορά δεδομένων σε περίπτωση που η εφαρμογή χρησιμοποιείται από διάφορα σημεία. Μέσω του scaling, διαδικασία διαφανή στον τελικό χρήστη, η εφαρμογή έχει κάθε στιγμή την ίδια απόδοση ανεξάρτητα από το πόσοι χρήστες τη χρησιμοποιούν ταυτόχρονα.

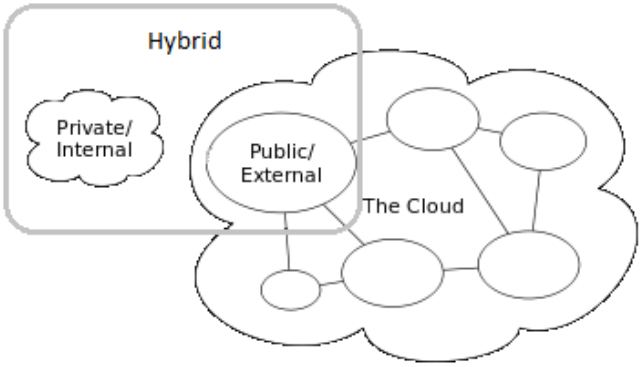
Παράδειγμα: Google Apps, όπως Gmail

Network as a Service (NaaS) – Δίκτυο σαν Υπηρεσία

Το NaaS αποτελεί μια νέα υπηρεσία παροχής υπηρεσιών μέσω cloud. Με το NaaS παρέχεται στους χρήστες υποδομή και υπηρεσίες δικτύων. Στόχος είναι η αποδοτικότερη χρήση της υποδομής του δικτύου ενός datacenter. Στην υπηρεσία περιλαμβάνονται η γνώση της τοπολογίας του δικτύου, ώστε χρήστες να μπορούν να επιλέξουν τους κόμβους που θα αποθηκεύουν τα δεδομένα τους, ο έλεγχος των προωθούμενων πακέτων στα switches του δικτύου, ορισμός firewalls από τους χρήστες και επεξεργασία των δεδομένων εσωτερικά του δικτύου, ώστε να περιορίζετε η κίνηση στο δίκτυο. Για υλοποίηση του NaaS είναι απαραίτητη μια multi-tenant αρχιτεκτονική και απομόνωση των πόρων που παρέχονται στους χρήστες. Παρέχεται επίσης API για την εύκολη πρόσβαση του χρήστη στις διάφορες υπηρεσίες του NaaS παρόχου. Το NaaS αποτελεί αρκετά νέα τεχνολογία.

2.4 Είδη Cloud Computing

Αντίστοιχα με την κατηγοριοποίηση των υπηρεσιών cloud ανάλογα με τον έλεγχο των χρηστών στους υπολογιστικούς πόρους, εντοπίζεται διαχωρισμός και όσον αφορά στην υποδομή που χρησιμοποιείται. Συγκεκριμένα εντοπίζονται τρία είδη Cloud Computing, το public cloud, το private cloud και ο συνδυασμός των δύο, hybrid cloud, ανάλογα με τον πάροχο της υποδομής που χρησιμοποιείται. Σχηματικά οι διαφορές τους φαίνονται στην Εικόνα 5. Ακολουθεί αναλυτική περιγραφή των τριών αυτών ειδών cloud.



Εικόνα 5: Είδη Cloud Computing

Public Cloud – Δημόσια Νέφη

Το public cloud αποτελείται από ένα σύνολο μηχανημάτων και υποδομής δικτύου, και χρησιμοποιείται για την παροχή υπολογιστικών πόρων και υπηρεσιών στο ευρύ κοινό αποκλειστικά μέσω δικτύου. Οι υπηρεσίες παρέχονται είτε χωρίς χρέωση είτε με εφαρμογή κάποιας τιμολογιακής πολιτικής. Ένα παράδειγμα δωρεάν παροχής υπηρεσιών είναι η περίπτωση των πανεπιστημιακών παρόχων.

Παραδείγματα: Amazon AWS, Microsoft, Google

Private Cloud – Ιδιωτικά Νέφη

Το private cloud δομείται σε υποδομή η οποία χρησιμοποιείται αποκλειστικά από έναν οργανισμού ή χρήστη. Η διαχείριση της υποδομής γίνεται είτε από τον ίδιο τον πάροχο είτε από εξουσιοδοτημένους τρίτους. Η υποδομή μπορεί να βρίσκεται οπουδήποτε. Το βάρος πέφτει στον πάροχο-οργανισμό ώστε κάθε βήμα του στησίματος των υπηρεσιών που παρέχονται από ιδιωτικά cloud να μην εγείρει ζητήματα ασφαλείας και αξιοπιστίας, όπως απώλεια δεδομένων. Η εικονοποίηση πρέπει να γίνει από τον πάροχο κάτι που απαιτεί ιδιαίτερα καλή τεχνογνωσία.

Η φιλοσοφία όμως του cloud είναι οι χρήστες να μην χρειάζεται να διαθέτουν εξοπλισμό και υποδομή και να εστιάζουν μόνο στην παροχή καλύτερων υπηρεσιών. Τελικά οι μικροί πάροχοι μπορεί να μην επωφελούνται από τα πλεονεκτήματα του cloud, εφόσον πρέπει να διαθέτουν, να διαχειρίζονται τον εξοπλισμό και να στήνουν εφαρμογές και υπηρεσίες.

Hybrid Cloud – Υβριδικά Νέφη

Τέλος διακρίνεται το hybrid cloud, το οποίο Αποτελεί συνδυασμό δύο ή περισσοτέρων ειδών cloud που παρότι στενά συνδεδεμένα ουσιαστικά είναι ξεχωριστά. Τα hybrid clouds εκμεταλλεύονται τα πλεονεκτήματα που δίνουν οι διαφορετικοί τύποι cloud, public και private. Οι χρήστες τέτοιου τύπου cloud συνδυάζουν αρχιτεκτονική με ανοχή σε σφάλματα (fault-tolerant) μαζί με άμεση διάθεση των δεδομένων τους χωρίς απαραίτητη σύνδεση με το δίκτυο. Για την υλοποίηση ενός Υβριδικού Νέφους απαιτείται τόσο ιδιωτική όσο και απομακρυσμένη server-based υποδομή. Κερδίζεται ευελιξία στις private εφαρμογές, αλλά

χάνεται η ασφάλεια και η αξιοπιστία που εγγυάται το cloud.

2.5 Ανοιχτά Ζητήματα – Θέματα

Παρά τα πολλά πλεονεκτήματα χρήσης του cloud, εγείρονται αρκετές επιφυλάξεις ως προς την ασφάλεια των δεδομένων των χρηστών, την ενδεχόμενη παραβίαση της ιδιωτικότητας και της πιθανής κατάχρησης των πληροφοριών που αποθηκεύονται στο cloud. Τα ζητήματα αυτά, καθώς και άλλα που αφορούν στην διαλειτουργικότητα των παρεχόμενων υπηρεσιών, παρουσιάζονται αναλυτικά σε αυτήν την ενότητα.

Ιδιωτικότητα

Οι πάροχοι υπηρεσιών cloud ελέγχουν και άρα έχουν τη δυνατότητα ανά πάσα στιγμή να έχουν πρόσβαση στα δεδομένα των χρηστών. Αυτό εγείρει ερωτήματα για το κατά πόσον τα δεδομένα που αποθηκεύουν οι χρήστες στο cloud παραμένουν ιδιωτικά. Λόγω της εκτεταμένης εικονοποίησης τα δεδομένα μπορεί να μην αποθηκεύονται στον ίδιο κόμβο ή στο ίδιο datacenter ή ακόμα και στο ίδιο cloud κάτι που εκτός του ότι πολυπλέκει την παροχή ιδιωτικότητας στα δεδομένα, θέτει νομικά ζητήματα για το ποιος τελικά έχει δικαιοδοσία σε αυτά. Κάποιο πάροχοι cloud, όπως η Amazon, έχουν δώσει τη δυνατότητα επιλογής της περιοχής στην οποία θα αποθηκεύονται τα δεδομένα του χρήστη.

Open Standards

Πολλοί cloud providers υλοποιούν APIs, τα οποία παρότι καλά τεκμηριωμένα δεν είναι διαλειτουργικά, δηλαδή δεν είναι συμβατά με άλλα clouds. Γίνονται προσπάθειες για ορισμό γενικών αρχών με βάση τις οποίες θα υλοποιούνται τα APIs για να περιοριστεί αυτό το πρόβλημα.

Security

Η ασφάλεια αφορά σε πολιτικές, τεχνολογίες και ελέγχους που χρησιμοποιούνται για την προστασία των εφαρμογών, της υποδομής και των δεδομένων των χρηστών. Ο φυσικός έλεγχος του εξοπλισμού και η άμεση εποπτεία των δεδομένων, μπορεί να εγγυηθεί την ασφάλεια των δεδομένων, κάτι που είναι πιο δύσκολο αν ο εξοπλισμός ελέγχεται από τρίτους. Στο cloud, η εγγύηση της ασφάλειας των δεδομένων αναλαμβάνεται από τους παρόχους των υπηρεσιών, οι οποίοι πρέπει να αναπτύξουν εξελιγμένους μηχανισμούς ασφαλείας για την διατήρηση των δεδομένων και την απόκρυψη της πληροφορίας από μη εξουσιοδοτημένους χρήστες. Αυτό που εγείρει αμφιβολίες όσον αφορά στα ευαίσθητα δεδομένα είναι η ακριβής χρήση, ο τρόπος αποθήκευσης και επεξεργασίας τους, η ιδιωτικότητα, ο χειρισμός των λαθών, η ανάκτηση, η προστασία από κακόβουλη χρήση και θέματα που αφορούν Multitenancy. Οι λύσεις που δίνουν οι πάροχοι ποικίλουν. Χρησιμοποιούνται μέθοδοι κρυπτογραφίας, χρήσης δημοσίου κλειδιού, public key infrastructure PKI, βελτίωσης της υποστήριξης των εικονικών μηχανών, χρήσης πολλών παρόχων, δημιουργίας προτύπων για τα cloud APIs και άλλα.

Sustainability – Διατηρησιμότητα

Σε όλα τα μεγάλα datacenters, όπου στεγάζονται πολλοί servers είναι απαραίτητα μεγάλα

συστήματα ψύξης για την λειτουργία των μηχανημάτων. Περιοχές με γενικά χαμηλές θερμοκρασίες όπως η Φιλανδία, η Σουηδία και η Ελβετία, αποτελούν ιδανικά περιβάλλοντα για την στέγαση των datacenters, αφού σε συνδυασμό με χρήση ανανεώσιμων πηγών ενέργειας είναι δυνατή η μεγάλη εξοικονόμηση ενέργειας και άρα η εξοικονόμηση χρημάτων και η προστασία του περιβάλλοντος.

Abuse

Υπάρχει ο κίνδυνος να χρησιμοποιηθούν οι υπηρεσίες cloud σε άνομες πράξεις με στόχο άλλους χρήστες του cloud. Είναι στην αρμοδιότητα των παρόχων να σταματούν τέτοιες ενέργειες και να προστατεύουν τα δεδομένα και τους ίδιους τους χρήστες.

2.6 Πλατφόρμες Cloud Computing

Στις προηγούμενες ενότητες παρουσιάστηκαν οι παροχές υπηρεσιών cloud προς την πλευρά των χρηστών, είτε αυτοί είναι πάροχοι εφαρμογών είτε τελικοί χρήστες τους. Σε αυτήν και την επόμενη ενότητα θα παρουσιαστεί η παροχή cloud computing υπηρεσιών από την πλευρά των παρόχων.

Μπορούμε να διακρίνουμε δύο κατηγορίες παρόχων cloud. Στην πρώτη εντάσσονται οι πάροχοι που προσφέρουν την πλατφόρμα στην οποία στήνονται cloud και στη δεύτερη οι πάροχοι που προσφέρουν υπηρεσίες cloud, όπως IaaS, PaaS και SaaS. Σε αυτή την ενότητα θα παρουσιάσουμε την πρώτη κατηγορία και στην επόμενη τη δεύτερη.

Οι πλατφόρμες cloud είναι το λογισμικό που χρησιμοποιείται για την δημιουργία και διαχείριση datacenters και cloud σε ιδιωτικά cluster. Εταιρείες, χρήστες και ερευνητικά εργαστήρια συχνά επιθυμούν να αναπτύξουν και να προσφέρουν δικές τους υπηρεσίες cloud σε δικές τους υποδομές. Έτσι κάθε πάροχος έχει απόλυτο έλεγχο πάνω στην υποδομή που θα στήσει το cloud, παραμετροποιεί το cluster ανάλογα με τις απαιτήσεις των υπηρεσιών που θα προσφέρει, επιλέγει τα εργαλεία και αναπτύσσει τεχνικές που θα χρησιμοποιηθούν για το storage. Παρακάτω παρουσιάζονται πάροχοι πλατφόρμων cloud.

2.6.1 OpenNebula

Ένας από τους πρώτους παρόχους cloud platform είναι η OpenNebula Community με το OpenNebula. Πρόκειται για ένα λογισμικό ανοιχτού κώδικα, το οποίο παρέχει μια γενική λύση για την ανάπτυξη και διαχείριση κατανεμημένων datacenters και IaaS clouds.

Με το OpenNebula χρησιμοποιείται υπάρχουσα υποδομή για το χτίσιμο όλων των υπηρεσιών cloud. Σε αντίθεση με άλλους παρόχους cloud platform που υλοποιούν υπηρεσίες ειδικού σκοπού σε προεπιλεγμένα περιβάλλοντα, το OpenNebula προσφέρει ένα ευέλικτο, ανοιχτό και εύκολα διαχειρίσιμο στρώμα υλοποίησης των υπηρεσιών. Ενσωματώνονται τεχνικές storage, εικονοποίησης, ασφάλειας, ελέγχου και δικτύωσης σε cluster υπολογιστών και είναι στην ευχέρεια του προγραμματιστή να επιλέξει τα εργαλεία που προσαρμόζονται καλύτερα στις ανάγκες της υπηρεσίας. Δίνεται έμφαση στην τυποποίηση, στη διαλειτουργικότητα και στο portability προσφέροντας στους χρήστες δυνατότητα επιλογής μεταξύ διάφορων διεπαφών cloud, όπως Amazon EC2 Query, και hypervisors, όπως Xen, KVM, VMware. Η διαλειτουργικότητα του OpenNebula κάνει δυνατή την χρήση του σε διάφορες αρχιτεκτονικές cloud και στην ανάπτυξη οποιασδήποτε υπηρεσίας.

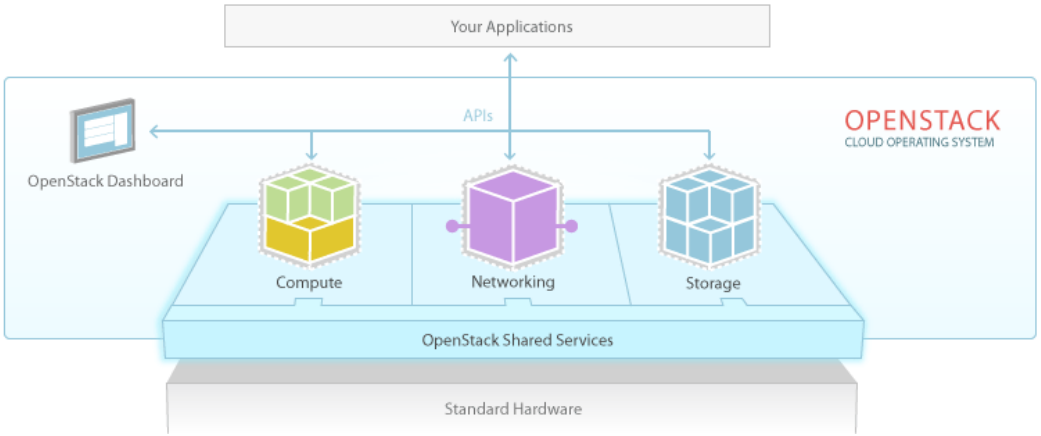
Μπορεί να εφαρμοστεί στη δημιουργία όλων των ειδών cloud που συναντήσαμε παραπάνω. Το OpenNebula μπορεί να χρησιμοποιηθεί για διαχείριση της υποδομής και

εφαρμογή τεχνικών εικονοποίησης στο cluster, δηλαδή στην εγκατάσταση ενός private cloud. Μπορεί ακόμα να συνδυάσει την τοπική υποδομή με υποδομή public cloud, πετυχαίνοντας καλύτερο scaling σε περιπτώσεις απότομης αύξησης της ζήτησης της υπηρεσίας, υλοποιώντας ένα hybrid cloud. Τέλος παρέχονται διεπαφές cloud για την διαχείριση των εικονικών μηχανών, της αποθήκευσης και της δικτύωσης, εργαλεία για στήσιμο ενός public cloud.

2.6.2 OpenStack

Μία ακόμα υπηρεσία παροχής cloud platform είναι το OpenStack, το οποίο διατίθεται με τους όρους χρήσης του Apache License. Όπως και το OpenNebula, το OpenStack είναι λογισμικό ανοιχτού κώδικα που χρησιμοποιείται για ανάπτυξη και προσφορά υπηρεσιών cloud πάνω σε ιδιωτικές υποδομές.

Το OpenStack αποτελεί ένα σύγχρονο λογισμικό για υλοποίηση public και private cloud. Στόχος του παρόχου είναι με την εύκολη εφαρμογή του λογισμικού, των πολλών χαρακτηριστικών που διαθέτει και του υψηλού scalability, να προσφέρει μια λύση για όλους τους τύπους των cloud που θέλουν να αναπτύξουν οι χρήστες. Το OpenStack χρησιμοποιείται από εταιρείες, ερευνητικά εργαστήρια, datacenters με δεδομένα μεγάλης κλίμακας που χρησιμοποιούν εργαλεία όπως το Hadoop και παρόχους private και public clouds με ανάγκες ευέλικτου scaling ανάλογα με τις εφαρμογές. Το λογισμικό είναι ανοιχτού κώδικα και ο πάροχος προτρέπει την υποβολή βελτιώσεων από τους χρήστες. Αυτό έχει σαν αποτέλεσμα την συνεισφορά πολλών για την ανάδειξη της υπηρεσίας και επίλυση προβλημάτων σε μεγαλύτερη κλίμακα. Διακρίνονται τέσσερις βασικές συνιστώσες του OpenStack οι οποίες παρουσιάζονται σχηματικά στην Εικόνα 6 και αναλύονται παρακάτω.



Εικόνα 6: OpenStack

OpenStack Compute

Η πρώτη συνιστώσα είναι το OpenStack Compute. Είναι το τμήμα της υπηρεσίας που επιβλέπει και διαχειρίζεται μεγάλα δίκτυα εικονικών μηχανών. Μέσω APIs οι πάροχοι των υπηρεσιών προσφέρουν on-demand επεξεργαστικούς πόρους με δυνατότητες scaling. Οι χρήστες των υπηρεσιών έχουν πρόσβαση στους πόρους αυτούς μέσω web interfaces. Η ευέλικτη αρχιτεκτονική του OpenStack κάνει δυνατή την ενσωμάτωση της υπηρεσίας με άλλα συστήματα. Οι πάροχοι χρησιμοποιούν κάποιον από τους υποστηριζόμενους hypervisors για τη διαχείριση του εικονικού περιβάλλοντος. Μια συχνή επιλογή που απευθύνεται στις περισσότερες περιπτώσεις χρήσης είναι οι KVM και XenServer.

OpenStack Storage

Η δεύτερη συνιστώσα είναι το OpenStack Storage. Για την αποθήκευση των δεδομένων εφαρμόζονται διάφορα μοντέλα ανάλογα με τις απαιτήσεις της υπηρεσίας προς υλοποίηση. Το OpenStack παρέχει δύο τρόπους αποθήκευσης των δεδομένων και πολλές επιλογές εφαρμογής τους για να καλύψει τις διαφορετικές ανάγκες των χρηστών, το Object Storage και το Block Storage.

Το Object Storage είναι ιδανικό για εφαρμογές που απαιτούν scaling και αποδοτική ως προς το κόστος αποθήκευση. Παρέχεται μια κατανεμημένη πλατφόρμα αποθήκευσης δεδομένων, διαχειρίσιμη μέσω API, η οποία μπορεί να ενσωματωθεί σε εφαρμογές και να χρησιμοποιηθεί για backup και αρχειοθέτηση. Δεν υπάρχει ένας κεντρικός μηχανισμός ελέγχου της αποθήκευσης. Τα αρχεία αποθηκεύονται σε πολλά σημεία στο cluster, με το OpenStack να φροντίζει για την εσωτερική αναπαραγωγή των δεδομένων. Το scaling επιτυγχάνεται με την προσθήκη servers. Το OpenStack Storage έχει ανοχή σε σφάλματα και σε περίπτωση αποτυχίας ενός τμήματος του cluster είναι υπεύθυνο για την ανάκτηση των πληροφοριών και την διατήρηση της αξιοπιστίας των δεδομένων.

Το Block Storage επιτρέπει τη χρήση block devices για καλύτερη απόδοση της υπηρεσίας και ενσωμάτωση της με πλατφόρμες αποθήκευσης που ικανοποιούν τις σύγχρονες ανάγκες μεγάλων εταιρειών. Παρέχεται persistent αποθήκευση των blocks. Το OpenStack Storage είναι υπεύθυνο για την δημιουργία και ανάθεση των block devices στους servers. Με εργαλεία διαχείρισης snapshots παρέχονται υψηλές δυνατότητες backup των δεδομένων. Τα snapshots μπορούν επίσης να χρησιμοποιηθούν για δημιουργία νέων block storage volumes.

OpenStack Networking

Η τρίτη συνιστώσα είναι το OpenStack Networking, η οποία υλοποιεί τη σύνδεση του cloud με τους χρήστες του μέσω δικτύου. Οι απαιτήσεις των datacenters σε servers, εξοπλισμό δικτύου, συσκευές αποθήκευσης καθώς και στη διαχείριση των IP addresses, της δρομολόγησης αιτημάτων και των μηχανισμών ασφαλείας που πρέπει να διέπουν το σύστημα, είναι τέτοιες που παραδοσιακά συστήματα διαχείρισης δικτύων δεν μπορούν να ικανοποιήσουν. Το OpenStack Networking παρέχει ευέλικτα μοντέλα δικτύωσης που ικανοποιούν τις ανάγκες του κάθε χρήστη. Αποτελεί ένα έτοιμο προς χρήση σύστημα διαχείρισης δικτύου και IP addresses, το οποίο ελέγχεται μέσω ενός API. Οι χρήστες μπορούν να δημιουργούν και να ελέγχουν δικά τους δίκτυα και να ορίσουν παραμέτρους του δικτύου, όπως firewalls, VPN και διαχείριση του φόρτου εργασιών. Είναι δυνατός ο ορισμός static και floating IP addresses. Με τα floating IP addresses οι αιτήσεις δρομολογούνται δυναμικά στα μηχανήματα που μπορούν να τις υποστηρίξουν βοηθώντας στο scaling και στη γρήγορη διαχείριση σφαλμάτων. Οι πάροχοι υπηρεσιών μπορούν να αξιοποιήσουν τεχνολογία software-defined networking, SDN, σε multitenant εφαρμογές για εξυπηρέτηση δεδομένων μεγάλης κλίμακας.

OpenStack Dashboard

Τέλος το OpenStack Dashboard παρέχει στους χρήστες και τους παρόχους των υπηρεσιών ένα γραφικό περιβάλλον για την πρόσβαση και αξιοποίηση των πόρων. Οι χρήστες μπορούν εύκολα να ελέγξουν τους επεξεργαστικούς, αποθηκευτικούς και δικτυακούς πόρους που χρησιμοποιούν. Από τη πλευρά τους οι πάροχοι μπορούν να ορίσουν ομάδες χρηστών και να αναθέσουν σε αυτούς συγκεκριμένους πόρους. Τέλος είναι εύκολη η προσθήκη άλλων υπηρεσιών που παρέχουν δυνατότητες εποπτείας, ελέγχου, χρέωσης καθώς και διαφόρων εργαλείων διαχείρισης της υπηρεσίας.

2.6.3 Synnefo

Το Synnefo είναι λογισμικό ανοιχτού κώδικα, που παρέχεται από το ΕΔΕΤ, για την δημιουργία public IaaS clouds υψηλής κλιμάκωσης. Για τον χειρισμό των εικονικών μηχανών χρησιμοποιείται το Google Ganeti, λογισμικό για τη διαχείριση εικονοποιημένων server δομημένων σε cluster.

Το Synnefo χρησιμοποιείται από το ~okeanos, υπηρεσία cloud computing που παρέχεται επίσης από το ΕΔΕΤ, που θα αναλυθεί στη συνέχεια. Είναι σχεδιασμένο να έχει το μεγαλύτερο δυνατό scalability και να είναι όσο πιο απλό γίνεται στη χρήση και στην εγκατάσταση. Απευθύνεται κυρίως σε ανθρώπους που αναπτύσσουν μεγάλες υπηρεσίες ώστε να φανούν οι μεγάλες δυνατότητές του σε επεκτασιμότητα, αλλά μπορεί να χρησιμοποιηθεί και για μικρότερες εφαρμογές. Οι υπηρεσίες που παρέχει το Synnefo μπορούν είτε να αναπτυχθούν ανεξάρτητα είτε να συνδυαστούν μεταξύ τους σε μεγάλες εφαρμογές. Παρακάτω παρουσιάζονται κάποιες από αυτές.

Identity Management (Astakos)

Η υπηρεσία Astakos χρησιμοποιείται από τον πάροχο για την πρώτη είσοδο των χρηστών στο σύστημα. Παρέχει τους μηχανισμούς πιστοποίησης της ταυτότητας των χρηστών, ορίζει πολιτικές πρόσβασης σε ομάδες χρηστών και δίνει στους χρήστες τα εργαλεία για τη διαχείριση των λογαριασμών τους. Προσφέρεται σαν λογισμικού ανοιχτού κώδικα με τους όρους χρήσης του BSD-2. Η πιστοποίηση των χρηστών μπορεί να γίνει είτε με ιδία μέσα είτε χρησιμοποιώντας το Shibboleth. Η υπηρεσία συντονίζει τις αιτήσεις για παροχή πόρων, όπως από τα συστήματα Pithos+ και Cyclades, και ενημερώνει την υπηρεσία Aquarium για είσοδο των χρηστών στο σύστημα και έναρξη της τιμολόγησης.

Object Storage System (Pithos+)

Η υπηρεσία Pithos+ είναι το σύστημα αποθήκευσης που χρησιμοποιεί το cloud. Είναι βασισμένο στο OpenStack Object Storage που παρουσιάστηκε παραπάνω και διαθέτει διάφορες επεκτάσεις. Χορηγείται σαν λογισμικό ανοιχτού κώδικα με τους όρους χρήσης του BSD-2. Με το Pithos+ οι χρήστες ανεβάζουν, κατεβάζουν, μοιράζονται και δημιουργούν αρχεία τα οποία αποθηκεύονται στο cloud. Τα δεδομένα αποθηκεύονται ως αντικείμενα και είναι προσβάσιμα από το λογαριασμό του χρήστη. Χρήστες του OpenStack μπορούν να χρησιμοποιούν το Pithos+, αλλά δεν έχουν πρόσβαση στα νέα χαρακτηριστικά που προστέθηκαν. Οι πιο σημαντικές επεκτάσεις αποτελούν η αποθήκευση των αντικειμένων σε blocks χωρίς περιορισμούς ως προς το μέγεθος τους και με μηχανισμούς deduplication, ο ορισμός permissions, η διατήρηση διαφορετικών versions και ο ορισμός policies για ενεργοποίηση του versioning και επιβολή quotas στα δεδομένα. Δόθηκε ακόμα η δυνατότητα ορισμού metadata στα δεδομένα και η εφαρμογή ερωτημάτων σε αυτά με βάση τα metadata. Στα δεδομένα ανατίθεται κατά τη δημιουργία τους ένας Universally Unique IDentifier (UUID) που χρησιμοποιείται για παραπομπές. Ακόμα οι χρήστες μπορούν να δημιουργούν αντικείμενα των οποίων τα δεδομένα μπορεί να προέρχονται από άλλα αντικείμενα. Με το Pithos+ API ο πάροχος έχει πρόσβαση στον τρόπο αποθήκευσης των δεδομένων σε blocks και μπορεί να χρησιμοποιήσει ιδιαίτερες τεχνικές για βελτιστοποίηση της διαχείρισης των δεδομένων.

Compute/Network Service (Cyclades)

Το Cyclades είναι το κομμάτι του Synnefo που ασχολείται με την διαχείριση των πόρων και τη δικτύωση του cloud. Η υλοποίηση είναι βασισμένη στο OpenStack Compute και διαθέτει διάφορες προεκτάσεις. Για την διαχείριση των εικονικών μηχανών χρησιμοποιείται το Google Ganeti και για την ανάπτυξη δικτυακών υπηρεσιών, που συχνά συνοδεύονται από πολύπλοκες βάσεις, χρησιμοποιείται το Django Web Framework, γραμμένο σε Python, που είναι προσβάσιμο μέσω ενός API. Οι χρήστες έχουν πρόσβαση σε εικονικές μηχανές μέσω του δικτύου ή μέσω του OpenStack Compute API και μπορούν να τις εκκινήσουν, να τις τερματίσουν, να τις δημιουργήσουν και να τις καταστρέψουν. Υπάρχει η δυνατότητα επιλογής κάποιων χαρακτηριστικών των εικονικών μηχανών, όπως CPU, RAM, δίσκος και λειτουργικό σύστημα. Το User Interface τρέχει αποκλειστικά στην πλευρά του χρήστη, μειώνοντας έτσι την κίνηση στο δίκτυο, αυξάνοντας την απόδοση των εφαρμογών. Οι λειτουργίες ικανοποιούνται με ασύγχρονη κλήση. Κάθε εικονική μηχανή έχει ανεξάρτητη σύνδεση με το διαδίκτυο και υπάρχει δυνατότητα ορισμού firewalls ή χρήσης ενός προεπιλεγμένου. Μπορούν να δημιουργηθούν δυναμικά, public και private εικονικά δίκτυα, που συνθέτουν αυθαίρετες τοπολογίες δικτύου. Τα public δίκτυα υλοποιούνται από τον πάροχο της υπηρεσίας και είναι προσβάσιμα από τους χρήστες της και τα private δίκτυα υλοποιούνται από τους χρήστες για απομονωμένη πρόσβαση στις εικονικές μηχανές τους.

Image Registry (Plankton)

Ένα στρώμα πάνω από το Pithos+ βρίσκεται το Plankton, η υπηρεσία του Synnefo για την διατήρηση των images των αρχείων. Κάθε image στο Plankton είναι ένα αρχείο στο Pithos+ και μπορεί να χρησιμοποιηθεί για δημιουργία νέων εικονικών μηχανών μέσα από το Cyclades.

2.6.4 Eucalyptus

Το τελευταίο cloud platform που θα αναλυθεί είναι το Eucalyptus, που παρέχεται από την Eucalyptus Systems. Πρόκειται για λογισμικό ανοιχτού κώδικα που χρησιμοποιείται προς ανάπτυξη συμβατών με το Amazon Web Service (AWS) private και hybrid clouds.

Το Eucalyptus δομείται πάνω σε ιδιωτική εικονοποιημένη υποδομή, προς δημιουργία πόρων cloud, οι οποίοι αξιοποιούνται στην επεξεργασία, την αποθήκευση και τη διασύνδεση του cloud με τους χρήστες. Προσφέρεται ένα ευέλικτο, ασφαλές και scalable περιβάλλον παροχής ελαστικών υπηρεσιών web, οι οποίες δυναμικά δεσμεύουν πόρους ανάλογα με το φόρτο εργασίας τους. Καθώς για την ανάπτυξη των web υπηρεσιών χρησιμοποιείται το AWS API, το Eucalyptus είναι ιδανικό για τους χρήστες που επιθυμούν να συνδυάσουν την υπηρεσία τους με τις υπηρεσίες της Amazon, όπως το EC2, S3 και EBS, κάποια από τα οποία θα αναλυθούν στην επόμενη ενότητα. Παρακάτω παρουσιάζονται κάποια από τα χαρακτηριστικά του Eucalyptus.

Συμβατότητα με το AWS

Η συμβατότητα του Eucalyptus με το AWS API προσφέρει στους χρήστες την δυνατότητα της μετακίνησης των instances τους από το private cloud του Eucalyptus στο public cloud της Amazon δημιουργώντας έτσι ένα hybrid cloud. Προσφέρονται επίσης εναλλακτικές μέθοδοι αποθήκευσης των δεδομένων, με τους χρήστες να επιλέγουν ανάμεσα στο bucket-based σύστημα αποθήκευσης, συμβατό με υπηρεσίες S3, και στο Elastic block-based σύστημα αποθήκευσης, συμβατό με υπηρεσίες EBS.

Ευέλικτη Διαχείριση – Εικονικές Μηχανές

Η διαχείριση των πόρων γίνεται ελαστικά με εύκολη τοποθέτηση, εισαγωγή και αφαίρεσή τους. Παρέχεται επίσης ευέλικτη διαχείριση της δικτύωσης, των ομάδων χρηστών στους οποίους ανατίθενται διαφορετικά δικαιώματα στις υπηρεσίες καθώς και της απομόνωσης της κίνησης στο δίκτυο ώστε οι χρήστες των υπηρεσιών να είναι απομονωμένοι.

Το λειτουργικό σύστημα που χρησιμοποιείται είναι το Linux, το οποίο μπορεί να υποστηρίξει εικονικές μηχανές σχεδόν για όλες τις εκδόσεις Linux και Windows. Παρέχεται έτσι μεγάλη ελευθερία στους χρήστες ως προς το εύρος των υπηρεσιών που μπορούν να υλοποιήσουν. Για την διαχείριση των εικονικών μηχανών υποστηρίζονται διάφοροι hypervisors, όπως Xen, KVM και VMware. Οι εικονικές μηχανές μπορούν να υποστηριχθούν από το Amazon EBS, το οποίο διαθέτει persistent storage volumes. Βελτιώνεται ο χρόνος εκκίνησης των εικονικών μηχανών και παρέχεται η δυνατότητα δημιουργίας fully-persistent images εικονικών μηχανών. Έτσι οι χρήστες διατηρούν τα δεδομένα τους και μετά την αποσύνδεσή τους από τις εικονικές μηχανές.

High Availability – Υψηλή Διαθεσιμότητα

Το Eucalyptus υποστηρίζει την ανάπτυξη υπηρεσιών υψηλής διαθεσιμότητας. Χρησιμοποιούνται μηχανισμοί αντιμετώπισης αποτυχιών και αποκατάστασης, χωρίς κανένα single point of failure, ώστε να μεγιστοποιηθεί η αξιοπιστία της υπηρεσίας.

Συσκευές SAN

Περιλαμβάνεται υποστήριξη για συσκευές SAN, Storage Area Networks, συσκευές που χρησιμοποιούνται στη δημιουργία συσκευών storage (disk arrays, tape libraries, και optical jukeboxes). Έτσι οι συσκευές storage είναι προσβάσιμες από τους servers και εμφανίζονται σαν μέρος του λειτουργικού συστήματος. Τα cloud που δημιουργούνται με το Eucalyptus μπορούν να εκμεταλλευτούν τις ιδιότητες των disk arrays για βελτίωση της απόδοσης, της αξιοπιστίας και της δυναμικής δέσμευσης και αποδέσμευσης πόρων της υπηρεσίας.

Eucalyptus Dashboard

Όπως και στην περίπτωση του OpenStack παρέχεται ένα γραφικό περιβάλλον παρακολούθησης και ελέγχου της κατανάλωσης των πόρων του cloud. Έτσι είναι δυνατή η ρύθμιση και ο έλεγχος της διάθεσης των εικονικών και φυσικών πόρων.

2.7 Πάροχοι Cloud Computing

Στην προηγούμενη ενότητα παρουσιάστηκαν οι πάροχοι Cloud Platform που δίνουν εργαλεία για ανάπτυξη clouds. Αυτή η μορφή Cloud Computing δεν απευθύνεται στο ευρύ κοινό, καθώς αποτελεί μια αρκετά απαιτητική διαδικασία. Οι περισσότεροι χρήστες έχουν ανάγκη από έτοιμα στημένα cloud και θέλουν να χρησιμοποιήσουν τις υπηρεσίες που προσφέρουν, IaaS, PaaS και SaaS, εύκολα και άμεσα. Σε αυτή την ενότητα θα παρουσιαστούν κάποιοι πάροχοι που προσφέρουν τέτοιες υπηρεσίες cloud.

2.7.1 Πάροχος IaaS – Amazon Elastic Compute Cloud, Amazon EC2

Η πρώτη υπηρεσία cloud computing που θα αναλυθεί είναι το Amazon Elastic Compute Cloud, Amazon EC2, που παρέχεται από την Amazon. Το Amazon EC2 ανήκει στο πακέτο των Amazon Web Services και είναι η υπηρεσία που παρέχει IaaS στους χρήστες.

Με το Amazon EC2 οι χρήστες νοικιάζουν εικονικούς υπολογιστές, δημιουργούν εικονικές μηχανές, instances και αναπτύσσουν δικές τους υπηρεσίες. Αρχικά οι χρήστες επιλέγουν μια Amazon Machine Image, AMI, ή μία δική τους που περιέχει βιβλιοθήκες, εφαρμογές και δεδομένα, η οποία δημιουργεί μία εικονική μηχανή στο Amazon EC2. Στη συνέχεια επιλέγονται οι ρυθμίσεις δικτύου, ασφαλείας και λειτουργίας του κάθε instance εικονικών μηχανών. Υπάρχει η δυνατότητα επιλογής του φυσικού χώρου που θα τρέξουν τα instances μεταξύ των Regions της Amazon. Παρακάτω αναλύονται τα πλεονεκτήματα χρήσης του Amazon EC2.

Ελαστικότητα

Οι χρήστες μπορούν άμεσα να έχουν διαθέσιμους όσους πόρους θέλουν χωρίς αναμονή. Υπάρχει επίσης η δυνατότητα του αυτόματου scaling στους πόρους που χρησιμοποιούν οι εφαρμογές, ώστε κάθε χρονική στιγμή να δεσμεύονται τόσοι υπολογιστικοί πόροι όσοι χρειάζονται, μειώνοντας το κόστος ανάπτυξης και παροχής υπηρεσιών.

Έλεγχος

Οι χρήστες έχουν απόλυτο έλεγχο πάνω στα instances, δηλαδή στις εικονικές μηχανές που χρησιμοποιούν. Μπορούν να τα εκκινήσουν κάθε χρονική στιγμή, να τα ελέγχουν μέσω root access όπως θα έκαναν για ιδία μηχανήματα, να τερματίσουν τη λειτουργία τους και μέσω web service APIs να τα επανεκκινήσουν.

Ευελιξία

Οι χρήστες μπορούν να επιλέξουν μέσα από μια μεγάλη ποικιλία τύπων instances, λειτουργικού συστήματος και πακέτων λογισμικού. Υπάρχει η δυνατότητα επιλογής των χαρακτηριστικών που θα έχουν οι εικονικές μηχανές όπως CPU, αποθηκευτικό χώρο, RAM και λειτουργικό σύστημα, που ταιριάζουν καλύτερα στις ανάγκες των υπηρεσιών που θα αναπτύξουν.

Συμβατότητα με AWS

Για να αναπτυχθούν εφαρμογές δεν απαιτείται μόνο το υπολογιστικό κομμάτι που παρέχει το Amazon EC2. Χρειάζονται χώροι αποθήκευσης των δεδομένων καθώς και βάσεις δεδομένων για την διαχείρισή τους. Οι απαιτήσεις σε αποθηκευτικό χώρο ικανοποιούνται από το Amazon Simple Storage Service, Amazon S3, ενώ όσον αφορά στις βάσεις δεδομένων παρέχονται η Amazon Relational Database Service, Amazon RDS, αν απαιτείται η χρήση σχεσιακής βάσης δεδομένων και το Amazon SimpleDB αν απαιτείται χρήση NoSQL βάσης δεδομένων. Αυτές οι υπηρεσίες περιλαμβάνονται στο πακέτο AWS και μαζί με άλλες συνιστούν ένα ολοκληρωμένο σύστημα ανάπτυξης web υπηρεσιών μέσω cloud.

Αξιοπιστία

Το Amazon EC2 αποτελεί ένα αξιόπιστο περιβάλλον ανάπτυξης εφαρμογών και εγγυάται την διαθεσιμότητα των υπηρεσιών του σε ποσοστό 99,95%. Η αξιοπιστία αφορά και στην μη απώλεια των δεδομένων, κάτι που εξασφαλίζεται με διατήρηση πολλαπλών αντιγράφων σε διάφορα σημεία της υποδομής.

Ασφάλεια

Το Amazon EC2 παρέχει διάφορους μηχανισμούς ασφαλείας για τους πόρους των χρηστών. Υπάρχει η δυνατότητα ρύθμισης των firewalls για έλεγχο της κίνησης ανάμεσα σε χρήστες και instances. Για παροχή περαιτέρω απομόνωσης δίνεται η επιλογή της δημιουργίας ενός Amazon Virtual Private Cloud, Amazon VPC, για διασύνδεση της ιδίας υποδομής των χρηστών με instances που ανήκουν στην Amazon και ορισμού του εύρους IP που θα χρησιμοποιούνται.

2.7.1.1 Τύποι Instances Amazon EC2

Μπορούμε να διακρίνουμε τρεις (3) διαφορετικούς τύπους instances τα οποία προσφέρουν διαφορετικά πλεονεκτήματα στους χρήστες. Ανάλογα με τη χρήση του καθενός εφαρμόζεται αντίστοιχη πολιτική χρέωσης με αποτέλεσμα κάθε εφαρμογή να υλοποιείται στο κατάλληλο τύπο instance με το ελάχιστο δυνατό κόστος. Στη συνέχεια αναφέρουμε αυτούς τους τύπους instance.

On-Demand Instances

Χρήστες που δεν γνωρίζουν εκ των προτέρων τη χρήση που θα κάνουν ή γνωρίζουν ότι οι εφαρμογές τους έχουν μεγάλες διαφοροποιήσεις στις απαιτήσεις τους, μπορούν να επιλέξουν τα on-demand instances, στα οποία η τιμολόγηση είναι ανάλογη της χρήσης. Δεν απαιτείται εκ των προτέρων δέσμευση πόρων και οι χρήστες πληρώνουν με την ώρα ό,τι χρησιμοποιούν.

Reserved Instances

Στα reserved instances οι χρήστες πληρώνουν εκ των προτέρων το ποσό για την ενοικίασή τους και έχουν μια σημαντική έκπτωση στην ωριαία χρέωση. Χρησιμοποιούνται από χρήστες που ξέρουν πόσους πόρους θα χρειαστούν σε ένα βάθος χρόνου, όπως στην περίπτωση εκπόνησης ενός project. Διατίθενται τρεις τύποι reserved instances ανάλογα με τις απαιτήσεις των χρηστών, Light, Medium και Heavy Utilization, με ανάλογη τιμολόγηση για το καθένα. Μέσα από το Reserved Instance Marketplace χρήστες που δεν χρειάζονται πλέον τα reserved instances τους έχουν τη δυνατότητα να τα πουλήσουν.

Spot Instances

Στα spot instances οι χρήστες ορίζουν μια δική τους τιμή για τους πόρους. Όταν αυτή η τιμή είναι μεγαλύτερη της εκάστοτε Spot Price το instance προσφέρεται στο χρήστη, διαφορετικά το instance αποδεσμεύεται. Η Spot Price αλλάζει ανάλογα με τη διαθεσιμότητα και τη ζήτηση. Οι τιμές αυτές κυμαίνονται σε αρκετά χαμηλότερα επίπεδα από αυτές των on-

demand instances. Συνεπώς τα spot instances είναι ιδανικά για εφαρμογές με ευελιξία ως προς το πότε θα τρέξουν. Ακόμα μια συνηθισμένη περίπτωση χρήσης spot instances είναι οι fault-tolerant υπηρεσίες, στις οποίες είναι εύκολη η διαχείριση κατάρρευσης του συστήματος με αποδέσμευση του instance στο οποίο τρέχει, όταν δηλαδή η τιμή που έχει ορίσει ο χρήστης είναι χαμηλότερη από την Spot Price.

2.7.1.2 Χαρακτηριστικά και Εργαλεία Amazon EC2

Το Amazon EC2 προσφέρει τα εργαλεία για την ανάπτυξη εφαρμογών με δυνατότητες μεγάλης κλιμάκωσης, οι οποίες παραμένουν ανθεκτικές σε αποτυχίες και χωρίς την ανάγκη διατήρησης ιδίου εξοπλισμού κάτι που μειώνει πολύ το κόστος παροχής υπηρεσιών. Στη συνέχεια παρουσιάζονται τα χαρακτηριστικά του Amazon EC2.

Amazon Elastic Block Store

Τα instances έχουν δύο τρόπους αποθήκευσης των δεδομένων. Ο πρώτος είναι το instance-storage, στο οποίο τα δεδομένα διατηρούνται όσο το instance λειτουργεί και σε περίπτωση reboot, επανεκκίνηση. Όταν όμως οι χρήστες αποσυνδεθούν από το instance αυτά τα δεδομένα χάνονται. Ο δεύτερος είναι τα volumes, τόμοι, που προσφέρονται από το Amazon EBS και το καθένα αποθηκεύει δεδομένα από 1GB έως 1TB. Η υπηρεσία Elastic Block Storage, EBS, προσφέρει στα volumes persistent αποθήκευση. Με την persistent αποθήκευσή τους, τα volumes διατηρούν τα δεδομένα τους ανεξάρτητα από την κατάσταση των instances, δηλαδή και μετά την αποσύνδεση των χρηστών από τις εικονικές μηχανές. Μπορούν να χρησιμοποιηθούν είτε για το boot, εκκίνηση, των instances είτε σαν standard block devices ενός instance. Όταν χρησιμοποιούνται για την εκκίνηση των instances οι χρήστες μπορούν να αποσυνδέονται και να επανασυνδέονται στο instance και να χρεώνονται μόνο για τους πόρους που χρησιμοποιούνται όσο το instance είναι ενεργό. Τα volumes μπορούν να μετακινηθούν από το ένα instance στο άλλο ενώ αυτά είναι ενεργά. Αναπαράγονται αυτόματα στην υποδομή της Amazon και χαρακτηρίζονται από υψηλή διαθεσιμότητα, αξιοπιστία και ασφάλεια. Υπάρχει η δυνατότητα δημιουργίας snapshots, τα οποία αποθηκεύονται στο Amazon Simple Storage Service, Amazon S3, υπηρεσία του AWS για την αποθήκευση των δεδομένων. Τα snapshots διατηρούνται σε πολλαπλά σημεία και μπορούν να χρησιμοποιηθούν για τη δημιουργία νέων volumes. Υπάρχουν δύο τύποι volumes, Standard και Provisioned IOPS. Σε εφαρμογές με μέτρια ή με απότομες εναλλαγές I/O requests επιλέγονται τα standard volumes, ώστε οι χρήστες να πληρώνουν ακριβώς τους πόρους που χρησιμοποιούν κάθε χρονική στιγμή. Σε εφαρμογές με έντονα I/O requests και με υψηλές απαιτήσεις στην απόδοση, όπως σε βάσεις δεδομένων, χρησιμοποιούνται τα provisioned IOPS volumes.

EBS-Optimized Instances

Κάποια επιλεγμένα instances μπορούν, με επιπλέον χρέωση, να εκκινηθούν σαν EBS-Optimized instances και να αξιοποιήσουν όλες τις I/O δυνατότητες των EBS volumes που παρουσιάστηκαν παραπάνω. Οι IOPS volumes που ανήκουν σε ένα EBS-Optimized instance είναι σχεδιασμένοι να λειτουργούν σε εύρος 10% των δυνατοτήτων τους στο 99% των περιπτώσεων.

Multiple Locations

Η υποδομή του Amazon συνιστάται από Regions, περιοχές, και Availability Zones, ζώνες διαθεσιμότητας. Τα Regions βρίσκονται σε διάφορα σημεία του πλανήτη. Κάθε Region έχει μία ή περισσότερες Availability Zones, οι οποίες έχουν ξεχωριστή υποδομή, και είναι σχεδιασμένες να μην επηρεάζονται από αποτυχίες σε άλλες Availability Zones. Μεταξύ των Availability Zones του ίδιου Region πραγματοποιείται γρήγορη δικτυακή μεταφορά δεδομένων. Οι χρήστες μπορούν να επιλέξουν το Region που θα αποθηκευτούν τα δεδομένα τους, πετυχαίνοντας βελτιστοποίηση του latency. Υπάρχει δυνατότητα αποθήκευσης των δεδομένων σε πολλά Availability Zones του ίδιου Region, προστατεύοντάς τα από αποτυχία σε μια συγκεκριμένη Availability Zone. Τέλος οι χρήστες μπορούν να αποθηκεύουν τα instances και τα δεδομένα τους σε διαφορετικά Regions, πετυχαίνοντας ακόμα μεγαλύτερη διαθεσιμότητα.

Elastic IP Addresses

Οι Elastic IP Addresses είναι σχεδιασμένες για dynamic cloud computing. Συσχετίζονται με accounts των χρηστών και όχι με συγκεκριμένες εικονικές μηχανές. Αυτό σημαίνει πως σε περίπτωση αποτυχίας ενός instance, προγραμματιστικά είναι δυνατή η ανάθεση της Elastic IP Address του σε ένα νέο instance άμεσα. Εξαλείφεται έτσι η ανάγκη μεσολάβησης τεχνικού ή DNS για ενημέρωση όλων των χρηστών της υπηρεσίας. Η Elastic IP Address ανήκει στο account μέχρι να επιλέξει ο χρήστης την αποδέσμευση της.

Amazon Virtual Private Cloud (VPC)

Η υπηρεσία Amazon VPC δίνει στους χρήστες τη δυνατότητα να συνδέουν την υποδομή που διαθέτουν με τους πόρους που διατηρούν στο AWS cloud. Αυτή η διασύνδεση γίνεται μέσω ενός Virtual Private Network, VPN, και δίνει στους χρήστες εκτεταμένες δυνατότητες διαχείρισης ρυθμίσεων, όπως παραμέτρους ασφαλείας και δικτύου, οι οποίες εφαρμόζονται στις υπηρεσίες τους.

Amazon CloudWatch

Το Amazon CloudWatch είναι η web υπηρεσία ελέγχου των εφαρμογών και των πόρων του cloud που χρησιμοποιούνται. Οι χρήστες επιλέγουν τα instances που θέλουν να παρακολουθήσουν και μέσω APIs και εντολών λαμβάνουν πληροφορίες για αυτά. Οι πληροφορίες αφορούν στην χρήση πόρων, στην επεξεργαστική απόδοση και στα patterns ζήτησης, όπως χρήση της CPU, reads και writes στο δίσκο, κίνηση δικτύου και άλλα. Ακόμα προσφέρονται στατιστικά αποτελέσματα, γραφήματα και η δυνατότητα ορισμού συνθήκων κάτω από τις οποίες στέλνονται ειδοποιήσεις στο χρήστη, alarms. Οι μετρήσεις που συλλέγονται χρησιμοποιούνται από την Auto Scaling λειτουργία, που θα παρουσιαστεί στη συνέχεια. Μπορούν επίσης να εφαρμοστούν ιδία μετρικά εργαλεία και μετρήσεις του χρήστη.

Auto Scaling

Μαζί με το Amazon CloudWatch, δίνεται η δυνατότητα ορισμού συνθηκών κάτω από τις οποίες προστίθενται ή αφαιρούνται πόροι ανάλογα με τις απαιτήσεις. Σε περιπτώσεις απότομης αύξησης της ζήτησης προστίθενται instances στην υπηρεσία για να διατηρείται η απόδοσή της, ενώ σε περιπτώσεις μειωμένης ζήτησης αφαιρούνται instances και μειώνεται το κόστος. Αυτή η υπηρεσία είναι ιδιαίτερα αποτελεσματική για εφαρμογές που δεν έχουν πάντοτε συγκεκριμένες απαιτήσεις σε πόρους και η χρήση εμφανίζει έντονη μεταβλητότητα.

Elastic Load Balancing

Με το Elastic Load Balancing τα requests από τις εφαρμογές αυτόματα κατανέμονται στα instances, δίνοντας στις υπηρεσίες μεγαλύτερη ανοχή σε σφάλματα, fault-tolerant. Τα instances που δεν είναι ικανά να ικανοποιήσουν τις αιτήσεις απομονώνονται και η κίνηση κατανέμεται στα υπόλοιπα instances μέχρι να αποκατασταθεί το σφάλμα. Το Elastic Load Balancing μπορεί να εφαρμοστεί είτε στην ίδια Availability Zone είτε μεταξύ πολλών, προσφέροντας έτσι καλύτερη απόδοση, καθώς πιθανό σφάλμα σε μια Availability Zone δεν θα προκαλέσει σφάλματα, ασυνέπειες και μη διαθεσιμότητα στις υπηρεσίες. Το Amazon CloudWatch προσφέρει μετρήσεις που αφορούν στην κατανομή των εργασιών στα instances, όπως πλήθος αιτήσεων και συνολικό latency.

High Performance Compute (HPC) Clusters

Εφαρμογές με ιδιαίτερα πολύπλοκες απαιτήσεις, όπως πολλές παράλληλες διεργασίες, χαμηλό latency και με απαιτήσεις για μεγάλη επεξεργαστική ισχύ δεν μπορούν να ικανοποιηθούν από τα συνηθισμένα instances. Τέτοιες εφαρμογές χρησιμοποιούν επιστήμονες και ερευνητικά εργαστήρια για επίλυση προβλημάτων στο χώρο των επιστημών και των επιχειρήσεων. Τέτοιες εφαρμογές χρησιμοποιούν το Cluster Compute και Cluster GPU instances που έχουν σχεδιαστεί για παροχή υπηρεσιών δικτύου υψηλής απόδοσης. Δίνεται η δυνατότητα οι εφαρμογές να εκτελούνται σε cluster, ώστε να παρέχεται το χαμηλό latency που απαιτείται για την επικοινωνία μεταξύ των κόμβων και των στενά συνδεδεμένων εκτελούμενων διεργασιών. Παρέχεται επίσης αυξημένο throughput ιδανικό για εφαρμογές με μεγάλες απαιτήσεις δικτύου.

High I/O Instances

Χρήστες με μεγάλες ανάγκες σε χαμηλό latency και πολλά τυχαία I/O στα δεδομένα τους χρησιμοποιούν τα High I/O Instances. Οι Solid-State Drives, SSD, είναι αποθηκευτικά μέσα που χρησιμοποιούν ολοκληρωμένα κυκλώματα σαν μνήμη για την persistent αποθήκευση δεδομένων. Σε αντίθεση με την παραδοσιακή τεχνολογία των μαγνητικών δίσκων, οι SSDs δεν χρησιμοποιούν ακίδα για τον εντοπισμό των δεδομένων στο δίσκο, αλλά ηλεκτρονικές διεπαφές συμβατές με σκληρούς δίσκους που αποθηκεύουν τα δεδομένα σε blocks, επιταχύνοντας πολύ την πρόσβαση στα δεδομένα. Αυτό είναι το κύριο πλεονέκτημα χρήσης των SSDs. Χρησιμοποιώντας την τεχνολογία SSD στα instances επιτυγχάνεται υψηλή απόδοση στην εξυπηρέτηση των I/O που κυμαίνεται σε πάνω από 100000 IOPS. Είναι ιδανικά για χρήστες που υλοποιούν και χρειάζονται μεγάλης απόδοσης NoSQL και σχεσιακές βάσεις δεδομένων.

VM Import/Export

Οι χρήστες μπορούν να εισάγουν images εικονικών μηχανών που έχουν σε δικές τους συσκευές και να τις χρησιμοποιήσουν σαν νέα instances στο Amazon EC2. Επίσης μπορούν να εξάγουν τις images εικονικών μηχανών. Με αυτό τον τρόπο παλαιότερες επενδύσεις σε τεχνολογίες για δημιουργία εικονικών μηχανών αξιοποιούνται και οι χρήστες μπορούν να ορίζουν τις ρυθμίσεις δικτύου, ασφαλείας και συμβατότητας που επιθυμούν.

AWS Marketplace

Χρησιμοποιείται για αγοραπωλησία πόρων και λογισμικού τόσο μεταξύ των χρηστών όσο και με την Amazon. Παραδείγματα αποτελούν Reserved Instances που δεν χρειάζονται πλέον στον αρχικό χρήστη και λογισμικό όπως η NoSQL βάση MongoDB. Οι χρεώσεις φαίνονται αμέσως στον χρήστη που αγοράζει τους πόρους.

2.7.2 Πάροχος IaaS – ~okeanos

Το ~okeanos είναι η IaaS υπηρεσία που παρέχεται από το ΕΔΕΤ και χρησιμοποιεί την πλατφόρμα cloud computing Synnefo που παρουσιάστηκε στην προηγούμενη ενότητα. Απευθύνεται στην ερευνητική και ακαδημαϊκή κοινότητα, διατίθεται δωρεάν και την περίοδο που συντάχθηκε αυτή η διπλωματική βρισκόταν στο στάδιο Alpha Testing. Προσφέρονται δύο υπηρεσίες το Cyclades, που αφορά σε διάθεση υπολογιστικών πόρων και πόρων δικτύου, και το Pithos+, που υλοποιεί το storage. Στη συνέχεια παρουσιάζονται αυτές οι δύο υπηρεσίες.

Cyclades

Η υπηρεσία Cyclades προσφέρει επεξεργαστικούς πόρους, με τη μορφή εικονικών μηχανών. Οι χρήστες δημιουργούν, καταστρέφουν, διαχειρίζονται τις εικονικές μηχανές τους και συνδέονται σε αυτές μέσω δικτύου. Καθώς το σύστημα είναι ακόμα σε φάση παραγωγής, οι χρήστες μπορούν να έχουν ταυτόχρονα μέχρι δύο (2) εικονικές μηχανές. Το λειτουργικό σύστημα επιλέγεται μεταξύ Windows, Ubuntu, Debian, CentOS και Fedora. Κάθε εικονική μηχανή μπορεί να έχει μέχρι 4 CPUs, 2GB μνήμης RAM και 40 GB αποθηκευτικού χώρου. Όσον αφορά στις ρυθμίσεις δικτύου οι χρήστες μπορούν να επιλέξουν μεταξύ τριών (3) firewalls. Η διαχείριση των εικονικών μηχανών και του δικτύου γίνεται μέσω ενός εύχρηστου RESTful API. Είναι επίσης δυνατή η δημιουργία VPNs για μεγαλύτερη απομόνωση των εικονικών μηχανών. Τέλος οι χρήστες έχουν την δυνατότητα να εισάγουν αρχεία στις εικονικές μηχανές τους και να δημιουργούν νέες από δικά τους images.

Pithos+

Μέσω της υπηρεσία Pithos+ οι χρήστες αποθηκεύουν με ασφάλεια αρχεία στην υποδομή του ~okeanos, και έχουν πρόσβαση σε αυτά μέσω δικτύου και μέσω των εικονικών τους μηχανών. Δίνεται επίσης η δυνατότητα διαμοιρασμού των αρχείων τους με άλλους χρήστες ή ομάδες χρηστών. Κάθε χρήστης έχει στη διάθεσή του 50GB αποθηκευτικού χώρου καθώς και ένα User Interface για εύκολη διαχείριση, αποθήκευση και οργάνωση των αρχείων του. Τέλος υπάρχει η δυνατότητα διατήρησης ενός τοπικού directory το οποίο είναι πάντοτε συγχρονισμένο με το Pithos+, δίνοντας μεγαλύτερες ελευθερίες στους χρήστες, καθώς μπορεί να χρησιμοποιηθεί και σαν εργαλείο backup.

2.7.3 Πάροχος PaaS – Windows Azure

Η PaaS υπηρεσία της Microsoft ονομάζεται Windows Azure. Δίνει επιλογές χρήσης πολλών γλωσσών προγραμματισμού, βάσεων δεδομένων και εργαλείων για την ανάπτυξη των εφαρμογών, χωρίς την ενασχόληση των χρηστών με την συντήρηση της υποδομής. Παράλληλα το scaling των εφαρμογών γίνεται αυτόματα, ενώ ταυτόχρονα παρέχονται

εργαλεία για την άμεση εποπτεία των χρησιμοποιούμενων πόρων. Υποστηρίζονται σχεσιακές αλλά και NoSQL βάσεις δεδομένων, καθώς και χρήση του Hadoop Framework για εφαρμογή τεχνικών data mining. Καθώς τα datacenters του Windows Azure, στα οποία τρέχουν οι εφαρμογές, βρίσκονται σε πολλές φυσικές τοποθεσίες, είναι δυνατή η ανάπτυξη και εκτέλεσή τους κοντά στους τελικούς χρήστες, μειώνοντας έτσι το φόρτο δικτύου και συμβάλλοντας στην παροχή καλύτερων υπηρεσιών. Ταυτόχρονα δίνεται η επιλογή χρήσης υβριδικών λύσεων στην παροχή υπηρεσιών, με την ταυτόχρονη χρήση πόρων που παρέχονται μέσω των datacenters του Windows Azure και της ιδιωτικής υποδομής των χρηστών που αναπτύσσουν νέες εφαρμογές. Παρέχονται επίσης δυνατότητες distributed caching, τεχνική που ενισχύει την απόδοση των εφαρμογών. Οι χρήστες επικοινωνούν με τις παρεχόμενες από το Windows Azure υπηρεσίες μέσω ανοιχτών πρωτοκόλλων τύπου REST.

2.7.4 Πάροχος PaaS – Openshift

Το Openshift αποτελεί την PaaS υπηρεσία που έχει αναπτυχθεί από την Red Hat. Προσφέρει δυνατότητες ανάπτυξης εφαρμογών σε περιβάλλον cloud, με έτοιμα προγραμματιστικά εργαλεία. Διακρίνονται τρεις εκδόσεις του ανάλογα με τις ανάγκες των χρηστών, Openshift Online, Openshift Enterprise και Openshift Origin. Αποτελεί την πλατφόρμα για την ανάπτυξη και το hosting των εφαρμογών, με την διαχείριση της υποδομής και το scaling των πόρων που χρησιμοποιούνται να γίνεται αυτόματα από το λογισμικό χωρίς την συμμετοχή των χρηστών. Δίνεται μια πληθώρα επιλογής γλωσσών προγραμματισμού, developer tools και βάσεων δεδομένων που μπορούν να χρησιμοποιηθούν για την ανάπτυξη νέων εφαρμογών. Κάποιες από αυτές είναι οι γλώσσες προγραμματισμού Java, Ruby, PHP, Node.js, Python και Perl και οι βάσεις δεδομένων MySQL και MongoDB. Το Openshift Origin διατίθεται ως λογισμικό ανοιχτού κώδικα διευκολύνοντας την διαλειτουργικότητα των αναπτυσσόμενων εφαρμογών.

2.7.5 Πάροχος SaaS – Google Apps

Ένας από τους SaaS παρόχους είναι η Google μέσω του πακέτου Google’s App Engine, στο οποίο περιλαμβάνονται και οι τρεις τύποι παροχής cloud computing υπηρεσιών, IaaS, PaaS και SaaS. Σε αυτή την ενότητα θα παρουσιαστεί το κομμάτι του Google’s App Engine που παρέχει έτοιμο λογισμικό προς χρήση. Οι εφαρμογές που διατίθενται μέσω του cloud λέγονται Google Apps και περιλαμβάνουν μεταξύ άλλων τα Gmail, Google Talk, Google Docs, Google Calendar και Google Maps. Προκειμένου να χρησιμοποιήσουν οι χρήστες αυτές τις υπηρεσίες απαιτείται σύνδεση στο διαδίκτυο και για κάποιες από αυτές Google Account, ενώ αποτελεί υποχρέωση της Google η διαχείριση, ενημέρωση και αναβάθμιση των υπηρεσιών. Υπάρχει δυνατότητα παραμετροποίησης των εφαρμογών ώστε να ικανοποιούν απόλυτα τις ανάγκες των χρηστών, και καθώς τα δεδομένα αποθηκεύονται σε datacenters της Google η πρόσβαση σε αυτές μπορεί να γίνει από κάθε συσκευή του χρήστη, ανεξάρτητα από το λειτουργικό της σύστημα και της υποδομής της. Ωστόσο η απομακρυσμένη αποθήκευση των δεδομένων εγείρει ερωτήματα σε θέματα ασφαλείας και ιδιωτικότητας των δεδομένων τους.

Κεφάλαιο 3
Middleware

Με τον όρο "*Middleware*" εννοούμε το λογισμικό που χρησιμοποιείται ως ενδιάμεσο επίπεδο επικοινωνίας μεταξύ της hardware υποδομής και των εφαρμογών που τη χρησιμοποιούν. Δεν αποτελεί λειτουργικό σύστημα, ή σύστημα διαχείρισης βάσεων δεδομένων, αλλά παρέχει συμπληρωματικές υπηρεσίες που δεν μπορούν να προσφέρει το λειτουργικό σύστημα. Συχνά χρησιμοποιείται για την περιγραφή του λογισμικού που επιτρέπει την επικοινωνία και διαχείριση δεδομένων σε κατανεμημένες εφαρμογές. Ένα παράδειγμα Middleware, που θα αναλυθεί διεξοδικά σε αυτό το κεφάλαιο, είναι το Apache Hadoop, σύστημα που χρησιμοποιείται για την διαχείριση κατανεμημένων δεδομένων που αποθηκεύονται στην υποδομή που προσφέρεται μέσω cloud.

3.1 Distributed Computing

Τη δεκαετία του '70 τα προβλήματα που οι υπολογιστές καλούνταν να επιλύσουν γίνονταν όλο και πιο πολύπλοκα με αποτέλεσμα η επεξεργαστική ισχύς ενός μόνο μηχανήματος να μην είναι αρκετή για την παραγωγή αποτελεσμάτων σε εύλογο χρονικό διάστημα. Τότε αναπτύχθηκε η ιδέα του distributed computing, των παράλληλων και κατανεμημένων συστημάτων, δηλαδή του χωρισμού των προβλημάτων σε υποπροβλήματα και της παράλληλης ανάθεσης και επίλυσης αυτών από περισσότερους επεξεργαστές. Με αυτόν τον τρόπο μπορούσε να διανεμηθεί ο υπολογιστικός φόρτος σε περισσότερα μηχανήματα και να μειωθεί σημαντικά ο χρόνος παραγωγής αποτελεσμάτων για πολύπλοκα προβλήματα. Τα πρώτα clusters υπολογιστών δημιουργήθηκαν σαν απάντηση στην παραπάνω ανάγκη και αναπτύχθηκαν παράλληλα με τα πρώτα δίκτυα υπολογιστών καθώς μέσω δικτύων επιτυγχάνεται η διασύνδεση υπολογιστικών πόρων, δεδομένων και περιφερειακών συσκευών. Το cluster αποτελείται από πολλά ίδια μηχανήματα, με ίδιο λειτουργικό σύστημα, filesystem και επεξεργαστή, τα οποία το καθένα επεξεργάζεται μέρος του παραλληλοποιημένου προβλήματος. Ένα παράδειγμα τέτοιου cluster είναι το Titan, ένας supercomputer της Cray.inc που διαθέτει 560000 επεξεργαστές.

Τα clusters που παρουσιάστηκαν παραπάνω εκμεταλλεύονται την παραλληλοποίηση των προβλημάτων και καταμερίζουν την επεξεργαστική ισχύ που απαιτείται, σε πολλά, απομακρυσμένα από τον τελικό χρήστη, ίδια μηχανήματα. Καθώς εξελίσσονταν οι απαιτήσεις των προβλημάτων και αναπτύσσονταν τα δίκτυα προέκυψε ακόμα η ανάγκη της γρήγορης αποθήκευσης και ανάκτησης πληροφοριών. Πολλοί πάροχοι υπηρεσιών έπρεπε να αποθηκεύουν και να επεξεργάζονται μεγάλο όγκο δεδομένων σε πολύ σύντομο χρονικό διάστημα. Αντίστοιχα με το distributed computing αναπτύχθηκαν συστήματα απομακρυσμένης αποθήκευσης δεδομένων, κλάδος που ονομάστηκε distributed data store. Αυτά τα συστήματα διαθέτουν μηχανισμούς κλωνοποίησης των δεδομένων και μηχανισμούς failover σε περίπτωση που ένας κόμβος αστοχήσει. Η αρχιτεκτονική δομή που βασίζεται στην φιλοσοφία του μοντέλου των ομοιόμορφων nodes (cluster) χρησιμοποιήθηκε σαν ιδέα

από την Yahoo! για να υλοποιήσει τη δική της πλατφόρμα παροχής distributed computing and data store, που την ονόμασε Apache Hadoop. Στο εξής με τον όρο cluster εννοείται η διασύνδεση πολλών, ενδεχομένως ανομοιόμορφων μηχανημάτων, στα οποία γίνεται αποθήκευση και επεξεργασία κατανεμημένων δεδομένων.

3.2 MapReduce

Προκειμένου να γίνει κατανοητή η ανάλυση ενός από τα βασικά πακέτα υπηρεσιών που προσφέρει το Apache Hadoop είναι απαραίτητη μία εισαγωγή στο MapReduce framework. Το MapReduce είναι ένα μοντέλο προγραμματισμού για διαχείριση και επεξεργασία μεγάλου όγκου κατανεμημένων δεδομένων σε clusters υπολογιστών. Το MapReduce framework χρησιμοποιείται για ανάπτυξη εφαρμογών που μπορούν να παραλληλοποιηθούν. Μέσα σε αυτό οι εφαρμογές αυτόματα κατανέμονται και συγχρονίζονται στους κόμβους του cluster. Το MapReduce αρχικά παρουσιάστηκε από την Google [1], ενώ πλέον υπάρχουν πολλές υλοποιήσεις με πιο γνωστή την ελεύθερη υλοποίηση του Apache Hadoop.

Η ανάπτυξη της τεχνικής MapReduce προέκυψε από την ανάγκη επιβολής απλών ερωτημάτων, όπως web request logs, εντοπισμός των πιο δημοφιλών web pages και άλλα, σε μεγάλο όγκο δεδομένων τα οποία ήταν κατανεμημένα σε κόμβους. Με συμβατικές μεθόδους η απάντηση σε αυτά τα ερωτήματα δίνεται μετά από μεγάλη μεταφορά δεδομένων στο δίκτυο, πολλαπλή επεξεργασία των ίδιων δεδομένων, μεγάλη χρονική καθυστέρηση και ανάπτυξη πολύπλοκων μηχανισμών παραλληλοποίησης του κάθε ερωτήματος. Ήταν απαραίτητη μια μέθοδος που θα επεξεργαζόταν τα δεδομένα με απλό τρόπο και θα έκρυβε τους μηχανισμούς παραλληλοποίησης, συγχρονισμού και εφαρμογής fault-tolerant μηχανισμών. Αυτή η μέθοδος είναι το MapReduce εμπνευσμένο από τις συναρτήσεις *map* και *reduce* των συναρτησιακών γλωσσών προγραμματισμού.

Επιγραμματικά για την υλοποίηση του MapReduce πρέπει να οριστούν από τον χρήστη οι συναρτήσεις map και reduce. Στο στάδιο του map παράγεται ένα ενδιάμεσο key/value pair για κάθε δεδομένο, η βιβλιοθήκη του MapReduce συγκεντρώνει όλα τα ενδιάμεσα δεδομένα με το ίδιο ενδιάμεσο κλειδί και το αποτέλεσμα περνάει στη φάση του reduce. Στο στάδιο του reduce συγχωνεύονται τα δεδομένα με το ίδιο key και εξάγεται ένα τελικό αποτέλεσμα.

Σχηματικά:

$$map(key, value) \rightarrow list(i\_key, i\_value)$$

$$reduce\big(i\_key, list(i\_value)\big) \rightarrow list(f\_value)$$

,όπου *(key,value)* είναι τα αρχικά δεδομένα, *(i_key,i_value)* είναι τα ενδιάμεσα αποτελέσματα μετά το map και *f_value* είναι το τελικό αποτέλεσμα που παράγεται από το reduce.

Όπως αναφέρθηκε το πλεονέκτημα χρήσης του MapReduce είναι η απλοποίηση της εφαρμογής κατανεμημένων αλγορίθμων χωρίς ο προγραμματιστής να χρειάζεται να γνωρίζει ακριβώς πως υλοποιείται ο καταμερισμός. Ωστόσο για μέγιστη παραλληλοποίηση είναι απαραίτητη η μελέτη του προβλήματος και η εφαρμογή map και reduce functions καθώς και ενδιάμεση επεξεργασία των δεδομένων με τέτοιο τρόπο που να αξιοποιούνται όλες οι ιδιαιτερότητες του.

3.2.1 Παράδειγμα

Παρουσιάζεται ένα παράδειγμα χρήσης του MapReduce για μέτρηση του αριθμού εμφάνισης κάθε λέξης σε ένα κείμενο. Η συνάρτηση map παράγει την κάθε λέξη μαζί με μία μέτρηση, το "1". Η συνάρτηση reduce αθροίζει όλες τις μετρήσεις για την κάθε λέξη και επιστρέφει για κάθε μία το πλήθος των εμφανίσεών της στο κείμενο. Ο ψευδοκώδικας δίνεται στους Πίνακας 1 και Πίνακας 2.

map function

```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");
```

Πίνακας 1 Map Function Example1

reduce function

```
reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

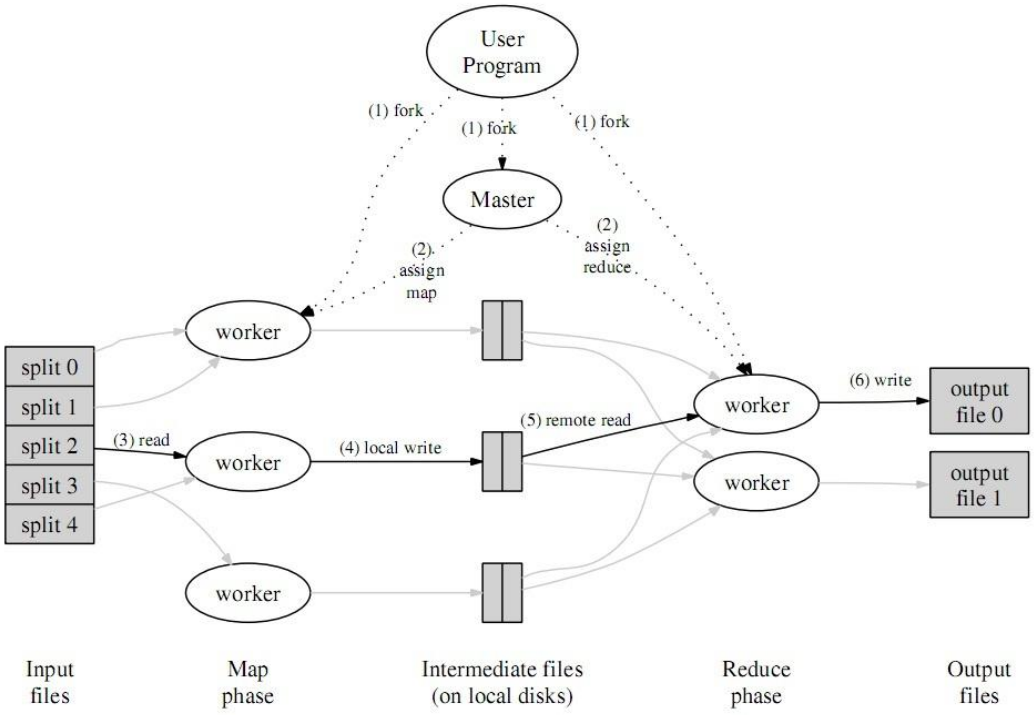
Πίνακας 2 Reduce Function Example1

3.2.2 Αρχιτεκτονική

Το MapReduce εκτελείται σε διάφορα στάδια μέσα στο cluster, το οποίο αποτελείται από κόμβους. Ένας από τους κόμβους χαρακτηρίζεται σαν master κόμβος και κάποιοι άλλοι σαν worker ή slave κόμβοι. Ο master κόμβος είναι υπεύθυνος για την ανάθεση εργασιών στους workers, από τους οποίους πλήθος M αναλαμβάνουν το κομμάτι του map και πλήθος R το κομμάτι του reduce. Ο master κόμβος είναι επίσης υπεύθυνος για τη διαχείριση των αστοχιών του συστήματος.

Στο πρώτο στάδιο ο master κόμβος παίρνει τα δεδομένα στα οποία θα εφαρμοστεί το MapReduce, τα χωρίζει σε M κομμάτια και τα κατανέμει στους map κόμβους για παράλληλη επεξεργασία. Στη συνέχεια τα δεδομένα περνάνε από έναν input reader, ο οποίος παράγει key/value ζευγάρια για κάθε δεδομένο. Οι map κόμβοι εφαρμόζουν στα key/value δεδομένα την map function και ταξινομούν τα αποτελέσματα με βάση το key. Στη συνέχεια ο master κόμβος χωρίζει τα ενδιάμεσα αποτελέσματα με βάση μια συνάρτηση κατακερματισμού ώστε κάθε reduce κόμβος να λάβει περίπου ίδιο όγκο δεδομένων για ισομοιρασμό της εργασίας. Μια τέτοια συνάρτηση είναι συνήθως η $hash(key)$ **mod** R, όπου R το πλήθος των reduce κόμβων. Η hash function και το R ορίζονται από το χρήστη. Στη συνέχεια ο master κόμβος

ειδοποιεί τους reduce κόμβους για την τοποθεσία των ενδιάμεσων key/value δεδομένων στα οποία θα εφαρμοστεί η reduce function. Στη συνέχεια οι reduce κόμβοι διαβάζουν τα key/value δεδομένα, εφαρμόζουν σε αυτά τη reduce function και παράγουν για κάθε key ένα αποτέλεσμα, το οποίο επιστρέφεται στο χρήστη. Μια σχηματική αναπαράσταση αυτής της διαδικασίας φαίνεται στην Εικόνα 7.



Εικόνα 7 MapReduce Execution Overview

3.3 Εισαγωγή στο Apache Hadoop

Το Apache Hadoop είναι ένα software framework ανοιχτού κώδικα για την ανάπτυξη κατανεμημένων εφαρμογών και παρέχεται με τους όρους χρήσης του Apache v2 license. Επιτρέπει την διαχείριση μεγάλου όγκου κατανεμημένων δεδομένων τα οποία είναι αποθηκευμένα σε clusters υπολογιστών. Είναι σχεδιασμένο να παρέχει υψηλή κλιμάκωση στις εφαρμογές, καθώς μπορεί να εφαρμοστεί σε clusters που αποτελούνται από μερικούς servers μέχρι χιλιάδες ανεξάρτητα και ανομοιόμορφα μηχανήματα. Προσφέρει επίσης μια εναλλακτική, ελεύθερη υλοποίηση του προγραμματιστικού μοντέλου MapReduce, που παρουσιάστηκε από την Google. Πλέον αποτελεί την πιο γνωστή υλοποίηση του MapReduce framework και είναι γραμμένο στην γλώσσα προγραμματισμού Java. Στη συνέχεια παρουσιάζεται επιγραμματικά το σύνολο των πακέτων που περιλαμβάνονται στο Apache Hadoop Framework, ενώ στις επόμενες ενότητες του κεφαλαίου θα αναλυθούν εκτενέστερα κάποια από αυτά.

- **Hadoop Common**: Παρέχει εργαλεία για την πρόσβαση στο filesystem και στα πακέτα που υποστηρίζονται από το Hadoop και για την εγκατάσταση ενός Hadoop Single Node ή Cluster.

- **HDFS**: Το Hadoop filesystem που προσφέρει υψηλό throughput στην πρόσβαση στα δεδομένα.

- **Hadoop Yarn**: Ένα framework για δρομολόγηση των εργασιών και διαχείριση των πόρων του cluster.

- **Hadoop MapReduce**: Είναι ένα σύστημα βασισμένο στο Hadoop Yarn για παράλληλη επεξεργασία μεγάλου όγκου δεδομένων.

Το Apache Hadoop χρησιμοποιείται για την ανάπτυξη και εκτέλεση κατανεμημένων εφαρμογών και την διαχείριση petabytes δεδομένων. Για την αποθήκευση των δεδομένων διατίθεται το Hadoop Distributed File System HDFS, το οποίο παρέχει πολύ υψηλό bandwidth σε όλο το cluster. Προσφέρει επεκτασιμότητα, αξιοπιστία και υψηλή διαθεσιμότητα στις εφαρμογές, καθώς η διαχείριση των σφαλμάτων, όπως στην περίπτωση αστοχίας ενός κόμβου, γίνεται στο επίπεδο του middleware, με την αυτόματη ανακατανομή των εργασιών σε διαθέσιμους κόμβους. Ακολουθούν τα κυριότερα χαρακτηριστικά του.

Προσβασιμότητα – Το Hadoop εφαρμόζεται σε μεγάλα clusters από commodity hardware και σε συστήματα υπηρεσιών cloud.

Αξιοπιστία – Η συχνή αστοχία κάποιων κόμβων θεωρείται δεδομένη σε συστήματα clusters. Το Hadoop διαθέτει μηχανισμούς διαχείρισης τέτοιων σφαλμάτων υλικού, μεταφέροντας την κίνηση σε άλλους κόμβους μέχρι να διορθωθεί το πρόβλημα.

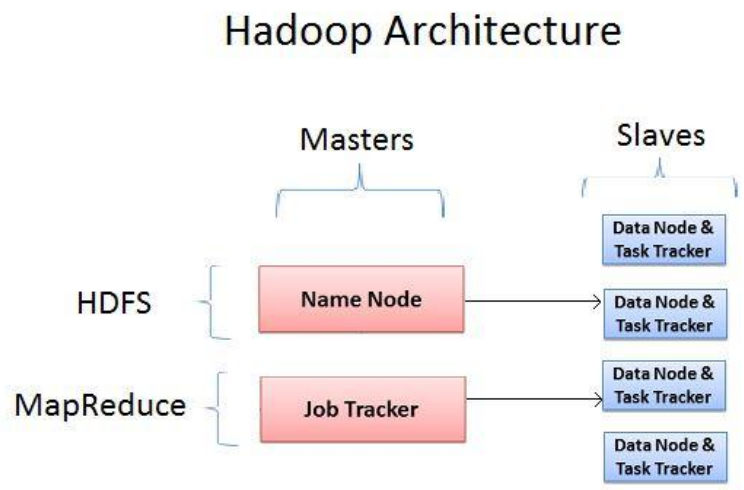
Επεκτασιμότητα – Με την προσθήκη νέων κόμβων στο cluster πραγματοποιείται αυτόματα γραμμικό scaling out. Καθώς ο NameNode, που θα αναλυθεί σε επόμενη ενότητα, είναι ο μόνος που διατηρεί indexes για το filesystem, το μέγεθός του αποτελεί τον μόνο περιορισμό στο scalability.

Ευκολία – Το Hadoop επιτρέπει την εύκολη ανάπτυξη παραλληλοποιημένων εφαρμογών, με χρήση του ενσωματωμένης υλοποίησης του MapReduce.

3.4 Αρχιτεκτονική Hadoop

Τα Hadoop clusters υλοποιούν αρχιτεκτονική τύπου master-slave. Το ρόλο του master κόμβου αναλαμβάνουν ο *NameNode* και ο *JobTracker*. Ο NameNode διατηρεί indexes για το filesystem και ο JobTracker είναι υπεύθυνος για την δρομολόγηση των MapReduce εργασιών στους slaves. Οι slaves διακρίνονται στους *DataNodes* και *TaskTrackers*. Στους DataNodes γίνεται η αποθήκευση των δεδομένων ενώ οι TaskTrackers αποδέχονται και εκτελούν τις αιτήσεις του JobTracker. Οι NameNodes και DataNodes διαθέτουν ενσωματωμένους web servers κάνοντας εύκολη την διαρκή ενημέρωση για την κατάσταση του cluster, διατηρώντας στατιστικά δεδομένα, πληροφορίες για τους DataNodes και εργαλεία για την περιήγηση στο filesystem. Σε cluster που δεν χρησιμοποιούν το HDFS οι NameNode, δευτερεύων NameNode και DataNodes αντικαθίστανται από τους ισοδύναμους που ορίζει το filesystem. Από τα παραπάνω είναι φανερό πως στο κομμάτι του HDFS ανήκουν οι NameNodes και DataNodes, ενώ την εκτέλεση του Hadoop MapReduce αναλαμβάνουν οι κόμβοι JobTracker

και TaskTrackers. Στην Εικόνα 8 δίνεται μια σχηματική αναπαράσταση της αρχιτεκτονικής του Hadoop.



Εικόνα 8 Apache Hadoop Architecture

3.5 Hadoop Common

Στο πακέτο Hadoop Common περιλαμβάνονται τα εργαλεία για εγκατάσταση και εκκίνηση του Hadoop σε συστήματα ενός ή μερικών χιλιάδων κόμβων. Στην περίπτωση συστημάτων ενός μόνο κόμβου παρέχονται μηχανισμοί για την εκτέλεση απλών εργασιών χρησιμοποιώντας το MapReduce και το HDFS. Σε clusters που απαρτίζονται από περισσότερα μηχανήματα προσφέρονται εργαλεία για την εκτέλεση απαιτητικών, κατανεμημένων εφαρμογών με χρήση των πακέτων MapReduce, Yarn και HDFS. Διακρίνονται τρεις τρόποι λειτουργίας του cluster.

Standalone Mode

Το Hadoop σε έναν κόμβο τρέχει σαν μία single Java process. Καθώς όμως δεν χρειάζεται να επικοινωνεί με άλλους κόμβους δεν χρησιμοποιεί το HDFS. Είναι χρήσιμο για ανάπτυξη και debugging απλών εφαρμογών που έχουν δημιουργηθεί στη λογική του MapReduce.

Pseudo-Distributed Mode

Σε αυτόν τον τρόπο λειτουργίας το cluster αποτελείται από έναν μόνο κόμβο, ο οποίος δουλεύει και ως server, στον οποίο τρέχει το Hadoop με τα πακέτα HDFS και Hadoop MapReduce. Οι κόμβοι αντιστοιχούν στα αρχεία που είναι αποθηκευμένα στον κόμβο και τα οποία δομούνται όπως ένα Hadoop cluster. Χρησιμοποιείται για ανάπτυξη και debugging πιο πολύπλοκων εφαρμογών που χρησιμοποιούν HDFS και Hadoop MapReduce.

Fully-Distributed Mode

Το fully-distributed mode αφορά clusters υπολογιστών που αποτελούνται από πολλούς κόμβους καθένας από τους οποίους έχει εγκατεστημένο το Hadoop. Είναι συστήματα πλήρως κατανεμημένα που αξιοποιούν όλες τις δυνατότητες που προσφέρει το Hadoop Project.

3.6 Hadoop Distributed File System, HDFS

Το HDFS αποτελεί το κύριο και καταλληλότερο εργαλείο αποθήκευσης δεδομένων σε ένα Hadoop cluster, ενώ είναι στην ευχέρεια του χρήστη να το αντικαταστήσει με άλλα συστήματα διαχείρισης δεδομένων. Το HDFS είναι κατάλληλο για κατανεμημένη αποθήκευση και επεξεργασία δεδομένων. Προσφέρει μηχανισμούς ανοχής σε σφάλματα, είναι scalable και διαθέτει ενσωματωμένη υλοποίηση του MapReduce framework. Για τις περισσότερες εφαρμογές δεν είναι απαραίτητες επιπλέον ρυθμίσεις από την πλευρά του προγραμματιστή, ενώ σε περιπτώσεις πολύ μεγάλων clusters παρέχονται εργαλεία βελτιστοποίησης της απόδοσης. Είναι γραμμένο στη γλώσσα προγραμματισμού Java και υποστηρίζεται από τα περισσότερα προγραμματιστικά περιβάλλοντα παρέχοντας εργαλεία για απευθείας πρόσβαση στο HDFS.

3.6.1 Αρχιτεκτονική του HDFS

Όπως αναφέρθηκε το Hadoop και κατ' επέκταση το HDFS διαθέτει αρχιτεκτονική τύπου master-slave. Εδώ θα παρουσιαστούν αναλυτικά τα χαρακτηριστικά των κόμβων που απαρτίζουν ένα HDFS cluster, το οποίο αναπαρίσταται σχηματικά στην Εικόνα 9.

NameNode

Ο master κόμβος του HDFS cluster αντιπροσωπεύεται από έναν μοναδικό NameNode, ο οποίος διαχειρίζεται το namespace του filesystem και εξυπηρετεί τις αιτήσεις των χρηστών για πρόσβαση στα δεδομένα, δρομολογώντας τους στους DataNodes. Κάθε αρχείο χωρίζεται σε blocks τα οποία αποθηκεύονται σε έναν ή περισσότερους DataNodes. Ο NameNode διατηρεί metadata για την τοποθεσία των δεδομένων και καθορίζει ποια blocks θα αποθηκευτούν στον κάθε DataNode. Είναι επίσης υπεύθυνος για όλες τις open, close και rename operations που αφορούν αρχεία ή directories. Ο NameNode λαμβάνει περιοδικά *Heartbeats* από τους DataNodes, σήματα επιβεβαίωσης της ορθής λειτουργίας τους, καθώς και λίστες με τα blocks που διατηρεί ο κάθε ένας, τα *Blockreports*. Σημειώνεται πως οι αιτήσεις για δημιουργία αρχείου δεν φτάνουν άμεσα στον NameNode. Το αρχείο δημιουργείται και αποθηκεύεται προσωρινά στο application layer και όταν φτάσει σε μέγεθος ένα block αποθήκευσης τότε αποστέλλεται η αίτηση στον NameNode και δρομολογείται το αρχείο στους DataNodes.

Στον NameNode αποθηκεύεται το namespace του HDFS σε μία εικόνα δίσκου, την *FsImage*. Κάθε αλλαγή στο filesystem που εκτελούν οι χρήστες αποθηκεύεται σε log file του NameNode, το *EditLog*. Παραδείγματα τέτοιων αλλαγών είναι η δημιουργία ενός νέου αρχείου και η αλλαγή του replication factor ενός αρχείου. Ο NameNode ενημερώνεται κατά την εκκίνησή του για την κατάσταση του HDFS από την FsImage, εφαρμόζει τις αλλαγές του EditLog σε αυτή, αποθηκεύει τη νέα εικόνα του HDFS και ξεκινάει τη λειτουργία του με άδειο EditLog. Καθώς το EditLog συγχωνεύεται με την FsImage, και άρα διαγράφεται το περιεχόμενό του, μόνο κατά την εκκίνηση του NameNode, σε clusters με πολλές κινήσεις δύναται να αυξηθεί πολύ το μέγεθος του log file. Για την αποφυγή μεγάλου EditLog και την ενίσχυση της αξιοπιστίας των δεδομένων, χρησιμοποιούνται οι επιπρόσθετοι κόμβοι

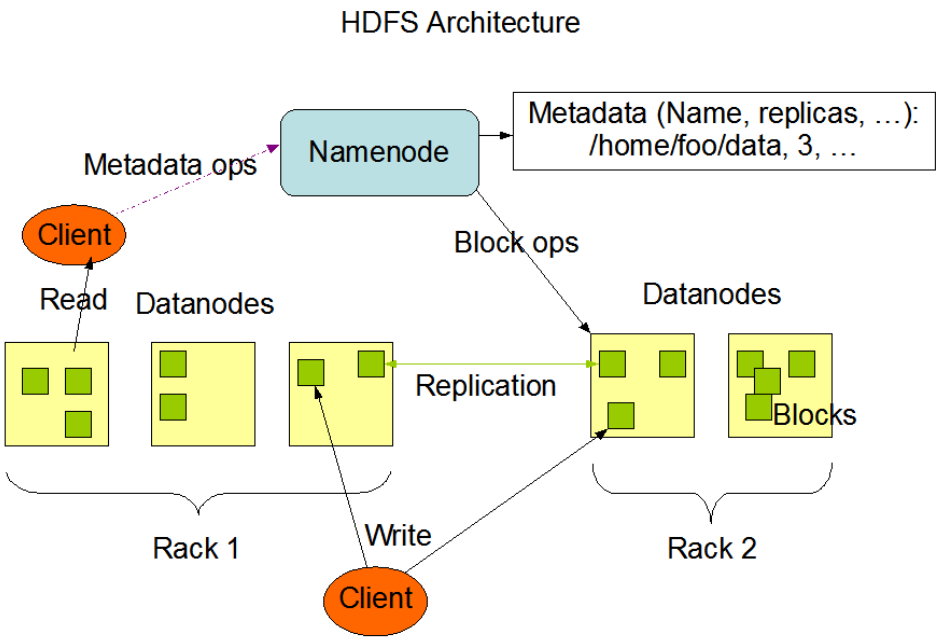
Secondary NameNode, *Checkpoint Node* και *Backup Node*, οι οποίοι παρουσιάζονται παρακάτω.

Για την εξυπηρέτηση των αιτήσεων πρόσβασης στα αρχεία, ο NameNode στέλνει στους clients μια λίστα με τους DataNodes που διατηρούν τα ζητούμενα δεδομένα. Αν ο NameNode αστοχήσει, οι clients δεν έχουν εναλλακτικό τρόπο να γνωρίζουν την τοποθεσία των δεδομένων και το filesystem καθίσταται μη προσβάσιμο. Είναι φανερό πως ο NameNode αποτελεί single point of failure για όλο το HDFS cluster. Σε περίπτωση που δεν είναι διαθέσιμος ο NameNode και δεν υπάρχει το EditLog καθώς και η εικόνα του HDFS, υπάρχει κίνδυνος απώλειας δεδομένων.

DataNodes

Οι DataNodes αποτελούν τους slave nodes ενός HDFS cluster. Είναι υπεύθυνοι για την διαχείριση και αποθήκευση των δεδομένων και αναλαμβάνουν την εκτέλεση των reads και writes που υποβάλουν οι χρήστες. Επικοινωνούν μεταξύ τους καθώς και με τον NameNode για την δημιουργία, την διαγραφή και το replication των blocks των αρχείων. Οι χρήστες για να έχουν πρόσβαση στο filesystem, χρησιμοποιούν την λίστα που τους έστειλε ο NameNode για την τοποθεσία των ζητούμενων δεδομένων και εκτελούν απευθείας I/O στους DataNodes.

Τα blocks δεδομένων αποθηκεύονται σε αρχεία, σε directories στο τοπικό filesystem των DataNodes. Χρησιμοποιούνται ευριστικοί αλγόριθμοι για τον αριθμό των αρχείων που θα ανήκουν στο κάθε directory και δημιουργούνται αντίστοιχα sub-directories. Κατά την εκκίνησή τους, οι DataNodes στέλνουν στον NameNode όλα τα blocks δεδομένων που διαθέτουν με το *Blockreport*.



Εικόνα 9 HDFS Architecure

Secondary NameNode

Έχει ξεπεραστεί πλέον και αντικατασταθεί από τους Checkpoint και Backup Nodes. Περιοδικά συγχωνεύει την εικόνα του HDFS με το EditLog περιορίζοντας το μέγεθος του σε ένα συγκεκριμένο όριο.

Checkpoint Node

Περιοδικά συγχωνεύει τοπικά την εικόνα του HDFS και το EditLog του NameNode δημιουργώντας *checkpoints*. Τα checkpoints αποθηκεύονται στον NameNode σαν νέα εικόνα του HDFS και δημιουργείται νέο, άδειο EditLog. Αν δεν υπάρχουν Backup Nodes στο cluster, επιτρέπεται η διατήρηση πολλών Checkpoint Nodes.

Backup Node

Έχει την ίδια λειτουργία με τον Checkpoint Node και ταυτόχρονα διατηρεί δικό του αντίγραφο του namespace του NameNode. Βρίσκεται σε διαρκή συγχρονισμό με τον NameNode, ο οποίος στέλνει όλες τις αλλαγές του HDFS namespace. Οι αλλαγές αυτές εφαρμόζονται στο namespace που διατηρεί ο Backup Node δημιουργώντας έτσι ένα backup του. Καθώς διατηρεί μια συνεπή εικόνα του filesystem, αρκεί μόνο το EditLog του NameNode για να δημιουργηθεί νέα HDFS εικόνα, κάνοντάς τον πιο αποδοτικό από τον Checkpoint Node. Μπορεί να υπάρχει μόνο ένας Backup Node στο cluster, ενώ στο μέλλον θα υποστηρίζονται περισσότεροι. Επίσης αν λειτουργεί ένας Backup Node δεν μπορούν να υπάρχουν Checkpoint Nodes.

3.6.2 High Availability HDFS Cluster

Στην τελευταία έκδοση του Hadoop έχουν συμπληρωθεί εργαλεία για την εξυπηρέτηση εφαρμογών που απαιτούν High Availability. Σε ένα High Availability (HA) HDFS cluster υπάρχουν δύο κόμβοι τύπου NameNode, ένας *Active* και ένας *Standby*. Ο Active κόμβος είναι υπεύθυνος για όλες τις ενέργειες των χρηστών στο cluster και ο Standby διατηρεί όσες πληροφορίες είναι απαραίτητες για γρήγορο failover σε περίπτωση αστοχίας του Active. Κάθε namespace modification καταγράφεται από τον Active σε ένα log, το οποίο συμβουλεύεται ανά τακτά χρονικά διαστήματα ο Standby κόμβος και εφαρμόζει τις αλλαγές στο δικό του namespace. Αν αποτύχει ο Active, ο Standby επιβεβαιώνει ότι έχει πραγματοποιήσει όλες τις ενέργειες του log και αναλαμβάνει σαν Active. Στην περίπτωση αυτή για γρήγορο failover είναι απαραίτητο ο Standby NameNode να διατηρεί πληροφορίες για το που βρίσκονται τα δεδομένα στο cluster. Για αυτό το λόγο οι DataNodes στέλνουν και στους δύο NameNodes πληροφορίες για τα δεδομένα που διατηρούν. Κάθε χρονική στιγμή μόνο ένας από τους δύο NameNodes επιτρέπεται να είναι Active, διαφορετικά τα namespaces τους μπορεί να αποκλίνουν και να οδηγήσουν σε απώλεια δεδομένων.

3.6.3 Χαρακτηριστικά και Εργαλεία του HDFS

Τα Hadoop clusters συχνά αποτελούνται από εκατοντάδες μηχανήματα, καθένα από τα οποία μπορεί ανά πάσα στιγμή να αποτύχει. Για την ομαλή εκτέλεση των εφαρμογών είναι απαραίτητοι μηχανισμοί γρήγορου εντοπισμού των μη διαθέσιμων κόμβων και επαναδρομολόγηση των εργασιών σε διαθέσιμους. Αυτοί οι μηχανισμοί αποτελούν κύριο χαρακτηριστικό του HDFS. Παρακάτω παρουσιάζονται τα κύρια χαρακτηριστικά και εργαλεία του HDFS, που προσφέρουν αξιοπιστία, διαθεσιμότητα και γρήγορη απόκριση στις εφαρμογές.

Εκτέλεση κοντά στα Δεδομένα

Για την καλύτερη απόδοση των εφαρμογών, το HDFS παρέχει εργαλεία για την μεταφορά της εκτέλεσης των εφαρμογών όσο γίνεται πιο κοντά στα σημεία αποθήκευσης των δεδομένων. Με αυτόν τον τρόπο η επεξεργασία των δεδομένων γίνεται τοπικά χωρίς να χρειάζεται μεταφορά μεγάλου όγκου πληροφορίας, ελαχιστοποιώντας την κίνηση στο δίκτυο και αυξάνοντας το ολικό throughput.

Απλό Coherency Model

Οι εφαρμογές που τρέχουν στο HDFS πρέπει να ακολουθούν το μοντέλο πρόσβασης στα αρχεία write-one-read-many. Αυτό σημαίνει πως τα αρχεία που δημιουργούνται και αποθηκεύονται στους κόμβους δεν θα πρέπει να αλλάξουν στο μέλλον. Με αυτό το μοντέλο, που ταιριάζει με την φιλοσοφία του MapReduce, διευκολύνεται η επίτευξη coherency στα δεδομένα και μεγιστοποιείται το throughput στην πρόσβαση σε αυτά. Σε επόμενες εκδόσεις του Hadoop θα υποστηρίζεται το update σε αρχεία που έχουν δημιουργηθεί.

Replication

Ένα από τα βασικά χαρακτηριστικά του HDFS είναι η εγγυημένη και αξιόπιστη αποθήκευση μεγάλων αρχείων σε όλο το cluster. Όπως έχει αναφερθεί τα αρχεία χωρίζονται σε blocks ίσου μεγέθους, τα οποία αποθηκεύονται και γίνονται replicate στους DataNodes που ορίζει ο NameNode. Το μέγεθος των blocks και ο αριθμός των αντιγράφων που θα αποθηκευτεί μπορεί να οριστεί από το χρήστη για κάθε αρχείο. Υπενθυμίζεται ότι το HDFS δεν υποστηρίζει αλλαγές στα αρχεία, δεν γίνονται updates, και θέτει σαν περιορισμό μόνο ένας χρήστης να γράφει σε ένα αρχείο κάθε χρονική στιγμή. Κατά το replication χρησιμοποιούνται οι τεχνικές *Rack Awareness, Rebalancer* και *Safemode* που αναλύονται παρακάτω.

Όταν χρησιμοποιείται replication level 3, δηλαδή τα δεδομένα διατηρούνται σε τρία αντίγραφα, συνηθίζεται ένα αντίγραφο να βρίσκεται σε έναν κόμβο, το δεύτερο σε έναν διαφορετικό κόμβο του ίδιου rack και το τρίτο σε διαφορετικό rack. Καθώς η αστοχία ενός rack είναι αρκετά σπάνια, αυτή η τοποθέτηση δεδομένων μειώνει την κίνηση στο, βελτιώνει την απόδοση των writes και δεν επηρεάζει την αξιοπιστία και διαθεσιμότητα των δεδομένων. Ωστόσο αν τα replicas ήταν σε διαφορετικά racks τα read operations θα εκμεταλλεύονταν καλύτερα το bandwidth των racks και θα εκτελούνταν γρηγορότερα. Είναι στην ευχέρεια του προγραμματιστή να επιλέξει τον τρόπο αποθήκευση των replicas ανάλογα με τις ανάγκες της εφαρμογής του.

Rebalancer

Εργαλείο για την ανακατανομή των δεδομένων σε περίπτωση που αυτά δεν είναι ισοκατανεμημένα στους DataNodes. Συνήθως αυτό παρατηρείται μετά από εισαγωγή νέων DataNodes στο cluster. Ο NameNode είναι υπεύθυνος για την δρομολόγηση των writes στους κόμβους σύμφωνα με διάφορα κριτήρια. Για την ελαχιστοποίηση της κίνησης στο δίκτυο replicated δεδομένα αποθηκεύονται στον ίδιο κόμβο. Επίσης ένα από τα replicas συχνά επιλέγεται να αποθηκευτεί σε κόμβο που πραγματοποιεί αλλαγές στα δεδομένα. Ταυτόχρονα είναι συνετό τα αντίγραφα των δεδομένων να αποθηκεύονται σε πολλούς κόμβους, ώστε αστοχία σε έναν από αυτούς να μην οδηγήσει σε απώλεια δεδομένων. Είναι πιθανό κατά τη διάρκεια εξυπηρέτησης όλων των παραπάνω ένας κόμβος να διατηρεί μεγάλο όγκο

δεδομένων, ενώ ένας καινούργιος DataNode να έχει πολύ λιγότερα. Ο Rebalancer είναι το εργαλείο των administrators για ισοκατανομή των δεδομένων, όταν παρατηρηθούν μεγάλες αποκλίσεις.

Rack Awareness

Τα large scale Hadoop clusters συνήθως αποτελούνται από racks με πολλούς κόμβους το καθένα, οι οποίοι επικοινωνούν δικτυακά μεταξύ τους. Καθώς η κίνηση στο δίκτυο επιβαρύνει την απόδοση των εφαρμογών, είναι επιθυμητό οι κόμβοι με συχνή μεταξύ τους επικοινωνία να τοποθετούνται στο ίδιο rack. Επίσης σε περίπτωση αστοχίας ενός κόμβου είναι αποδοτικότερο η δρομολόγηση των εργασιών να γίνεται σε κόμβους του ίδιου rack, ώστε να διατηρείται σταθερή η κίνηση στο δίκτυο. Τέλος οι μηχανισμοί fault-tolerant απαιτούν τα replicas των δεδομένων να αποθηκεύονται σε διαφορετικά racks, ώστε αν ένας κόμβος αστοχήσει, να μην υπάρξει κίνδυνος απώλειας δεδομένων. Με τη διασπορά των replicas σε διαφορετικά racks ευνοούνται τα reads καθώς οι αιτήσεις δρομολογούνται στους κόμβους με το υψηλότερο bandwidth, αλλά οι εγγραφές γίνονται πιο αργά, καθώς πρέπει να ενημερωθούν όλοι οι κόμβοι πριν επιστρέψει το write. Για τους παραπάνω λόγους για την ανάθεση εργασιών και την αποθήκευση replicated δεδομένων λαμβάνεται υπόψιν, από τον NameNode, η φυσική τοποθεσία των κόμβων. Μέσω Rack Awareness ελαχιστοποιείται η κίνηση στο δίκτυο και οι απώλειες δεδομένων.

Safemode

Κατά την εκκίνησή του ο NameNode λαμβάνει μια εικόνα του HDFS και το EditLog με τις αλλαγές που πρέπει να δρομολογήσει. Αν αυτή η διαδικασία ξεκινήσει πριν οι DataNodes ενημερώσουν τον NameNode για τα blocks δεδομένων που διαθέτουν ενδέχεται να δημιουργηθούν παραπάνω replicas από όσα χρειάζονται. Για το λόγο αυτό όταν ο NameNode εκκινηθεί, μπαίνει στην κατάσταση λειτουργίας Safemode, όπου επιτρέπονται μόνο read-only operations στο filesystem, αποτρέποντας έτσι το παραπάνω πρόβλημα. Ο NameNode βγαίνει από το Safemode αφότου ενημερωθεί από τους DataNodes για το ποια δεδομένα διατηρεί ο καθένας. Η κατάσταση λειτουργίας Safemode μπορεί να ενεργοποιηθεί ανά πάσα στιγμή προγραμματιστικά.

3.7 Hadoop MapReduce

Το Hadoop MapReduce αποτελεί το framework του Hadoop Project για την ανάπτυξη εφαρμογών που χειρίζονται μεγάλο όγκο κατανεμημένων δεδομένων παράλληλα. Εφαρμόζεται σε μεγάλα clusters υπολογιστών, προσφέρει μηχανισμούς fault-tolerant και αξιοπιστία στα δεδομένα.

Το MapReduce χρησιμοποιεί την τεχνική διαίρει και βασίλευε για την επεξεργασία μεγάλου όγκου δεδομένων. Το πρόβλημα που καλείται να λύσει η εφαρμογή σπάει σε πολλά ανεξάρτητα μικρότερα, τα οποία δρομολογούνται σε πολλούς κόμβους παράλληλα. Κάθε κόμβος επιστρέφει τα αποτέλεσμα των υποπροβλημάτων και στη συνέχεια αυτά συγχωνεύονται σε μία απάντηση του γενικού προβλήματος. Ο βασικός λόγος ανάπτυξης του Hadoop είναι η παράλληλη επεξεργασία μεγάλου όγκου δεδομένων με απλούς μηχανισμούς, αξιόπιστα και γρήγορα. Το MapReduce προσφέρει αυτές τις μεθόδους επίλυσης προβλημάτων και για αυτό αποτελεί βασικό κομμάτι του Hadoop Project.

Όπως παρουσιάστηκε στην προηγούμενη ενότητα, τα Hadoop clusters που χρησιμοποιούν το HDFS, διαθέτουν έναν NameNode και πολλούς DataNodes για την

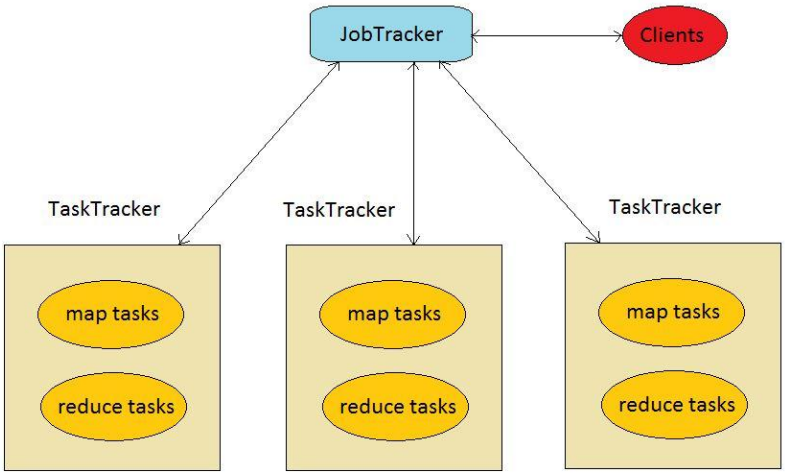
αποθήκευση και ανάκτηση των δεδομένων. Αντίστοιχα για την εκτέλεση του MapReduce είναι απαραίτητοι δύο άλλοι τύποι κόμβων, ο JobTracker και οι TaskTrackers. Στην Εικόνα 10 δίνεται η σχηματική αναπαράσταση της αρχιτεκτονικής του Hadoop MapReduce.

JobTracker

Σε κάθε Hadoop cluster υπάρχει ένας μοναδικός JobTracker. Συμπεριφέρεται σαν master και επικοινωνεί με τους clients που στέλνουν τις MapReduce εφαρμογές προς εκτέλεση. Αναλαμβάνει την τμηματοποίηση των δεδομένων εισόδου και είναι υπεύθυνος για την δρομολόγηση των map-reduce tasks στους κατάλληλους TaskTrackers. Ελέγχει τι επιστρέφει ο κάθε ένας και ξαναστέλνει τα tasks που δεν εκτελέστηκαν.

TaskTracker

Κάθε κόμβος του Hadoop cluster διαθέτει έναν TaskTracker. Οι TaskTrackers, που αποτελούν τους slaves της αρχιτεκτονικής του MapReduce, εκτελούν τα map και reduce tasks που τους αναθέτει ο JobTracker. Ανά τακτά χρονικά διαστήματα επικοινωνούν με τον JobTracker επιβεβαιώνοντας την ορθή λειτουργία τους. Αν το JobTracker δεν λάβει τα αναμενόμενα μηνύματα από τους TaskTrackers θεωρεί ότι ο κόμβος έχει αστοχήσει και στέλνει τα tasks που του είχε αναθέσει σε άλλους TaskTrackers του cluster.



Εικόνα 10 Hadoop MapReduce Architecture

Το Hadoop MapReduce framework δέχεται σαν είσοδο, επεξεργάζεται και παράγει σαν έξοδο αποκλειστικά δεδομένα της μορφής key/value pairs. Τα δεδομένα εισόδου και εξόδου αποθηκεύονται στο filesystem. Καθώς η επεξεργασία και αποθήκευση των δεδομένων γίνεται συνήθως στους ίδιους κόμβους, είναι πιο αποδοτικό η εκτέλεση των maps και reduces να πραγματοποιείται όσο πιο κοντά στα δεδομένα είναι δυνατόν. Το framework είναι υπεύθυνο για την δρομολόγηση των tasks στους κόμβους που απαιτούν την ελάχιστη μεταφορά δεδομένων εντός του cluster και για την επανεκτέλεση όσων αποτυγχάνουν.

3.7.1 Ανάλυση του Hadoop MapReduce Framework

Μία Hadoop MapReduce εφαρμογή αποτελείται από δύο στάδια. Το στάδιο του *map* και το στάδιο του *reduce*. Τα δεδομένα εισόδου σπάνε σε ανεξάρτητα κομμάτια, στα οποία εφαρμόζεται παράλληλα η map function. Στα ενδιάμεσα αποτελέσματα προαιρετικά εφαρμόζεται για ομαδοποίησή μια *combine* function. Στη συνέχεια αυτά ταξινομούνται και στέλνονται στους κόμβους που θα εκτελέσουν το reduce, οι οποίοι θα παράξουν τα τελικά αποτελέσματα. Παρακάτω παρουσιάζονται αναλυτικά τα επιμέρους κομμάτια του Hadoop MapReduce, καθώς και η Εικόνα 11 για καλύτερη κατανόηση των επιμέρους σταδίων.

Map function – Mapper interface

Ο *Mapper* κάνει map τα input key/value pairs σε intermediate key/value pairs. Ορίζει τον τύπο δεδομένων των keys και των values που δέχεται και επιστρέφει η map function, όπως φαίνεται παρακάτω.

```
public interface Mapper<K1,V1,K2,V2>
```

Στο `K1` και `V1` ορίζονται οι τύποι δεδομένων των input records και στα `K2` και `V2` οι τύποι δεδομένων των intermediate records που επιστρέφονται. Όπως φαίνεται μπορούν να οριστούν διαφορετικοί τύποι δεδομένων για την είσοδο και την έξοδο της map function.

Ο χρήστης χωρίζει την είσοδο σε τμήματα. Το κάθε ένα από αυτά διαθέτει δεδομένα στη μορφή key/value pairs. Ο Mapper γεννά ίσο αριθμό map tasks με το πλήθος των τμημάτων της εισόδου. Το κάθε map task δέχεται σαν είσοδο τα δεδομένα ενός τμήματος εισόδου. Σε αυτά εφαρμόζει την ορισμένη από το χρήστη map function και παράγει ενδιάμεσα αποτελέσματα πάντα στη μορφή key/value pairs. Κάθε key/value της εισόδου μπορεί να παράξει από κανένα ή πολλά key/value εξόδου. Στη συνέχεια τα αποτελέσματα ομαδοποιούνται, ταξινομούνται, χωρίζονται σε τμήματα με βάση το key και περνούν σαν είσοδο στον *Reducer*, ο οποίος θα δρομολογήσει τα reduce tasks. Η map function περιγράφεται από τον παρακάτω τύπο.

$$map(key, value) \rightarrow list(i\_key, i\_value)$$

Πριν τα αποτελέσματα ταξινομηθούν και περάσουν στο στάδιο του reduce, οι χρήστες μπορούν να ορίσουν μια combine function η οποία ομαδοποιεί τοπικά τα ενδιάμεσα αποτελέσματα του κάθε map tasks. Με αυτόν τον τρόπο επιτυγχάνεται λιγότερη μεταφορά δεδομένων από τον Mapper στον Reducer. Οι χρήστες μπορούν να ελέγξουν την ομαδοποίηση του Mapper με χρήση ενός *Comparator*. Προαιρετικά υλοποιείται και ένας *Partitioner*, ο οποίος ορίζει ποια keys, δηλαδή ποια δεδομένα, θα προωθηθούν στο κάθε reduce task.

Combine function

Η combine function ορίζεται από τους χρήστες για την συνένωση των output των map tasks πριν την αποστολή τους στα reduce tasks. Όπως είδαμε παραπάνω τα αποτελέσματα του map πριν σταλθούν στον Reducer ταξινομούνται με βάση το key. Όταν οριστεί combine function η ταξινόμηση δεν γίνεται στα αποτελέσματα των map tasks, αλλά πραγματοποιείται αμέσως μετά την επιστροφή του combine. Όταν ολοκληρωθεί η εκτέλεση της combine, τα αποτελέσματα ταξινομούνται με βάση το key, ομαδοποιούνται και περνούν σαν είσοδο στον Reducer. Η combine function περιγράφεται από τον παρακάτω τύπο.

$$combine(i\_key, i\_value) \rightarrow list(i\_key, list(merged\_value))$$

Η combine function χρησιμοποιείται για την μείωση του όγκου δεδομένων που μεταφέρονται μεταξύ του Mapper και του Reducer βελτιώνοντας την απόδοση των εφαρμογών. Μπορεί να εκτελεστεί μία ή περισσότερες φορές μετά από maps και reduces, στους map nodes και reduce nodes αντίστοιχα. Στις combine functions δεν επιτρέπονται τα *side effects*, δηλαδή εκτός των αποτελεσμάτων που επιστρέφουν δεν επιτρέπεται να επηρεάζουν οποιοδήποτε άλλο τμήμα της εφαρμογής. Ακόμα οι τύποι δεδομένων των input key και values πρέπει να είναι ίδιοι με αυτούς των output key και values αντίστοιχα. Πρακτικά η combine function έχει την ίδια λειτουργία με την reduce function, με τη διαφορά ότι κάθε κόμβος που έχει εκτελέσει map tasks, ή reduce tasks, καλεί μια φορά την combine function, η οποία συγχωνεύει τοπικά τα map output, ή reduce output, του κόμβου. Με αυτόν τον τρόπο δεν απαιτείται μεταφορά δεδομένων μεταξύ των map και reduce nodes.

Partitioner Interface

Όπως αναφέρθηκε παραπάνω τα ενδιάμεσα αποτελέσματα που παράγει η map function, ή η combine function αν έχει χρησιμοποιηθεί, ταξινομούνται και χωρίζονται με βάση το key για να περάσουν σαν είσοδο στους reducers. Την τμηματοποίηση των ενδιάμεσων αποτελεσμάτων του map αναλαμβάνει ο *Partitioner*, ο οποίος χωρίζει το key space σε *partitions*. Στη συνέχεια τα key/value pairs του κάθε partition προωθούνται στους αντίστοιχους reducers. Παρακάτω παρουσιάζεται το interface των Partitioners.

```
public interface Partitioner<K, V>
```

Το interface προσδιορίζει σε ποιο partition ανήκει το κάθε key τύπου `K` με values τύπου `V`. Ο Partitioner καλείται μία φορά σε κάθε κόμβο με είσοδο τα ομαδοποιημένα αποτελέσματα όλων των map tasks που έχουν εκτελεστεί σε αυτόν. Είναι πιθανό keys που έχουν παραχθεί σε διαφορετικούς κόμβους να ανήκουν στο ίδιο partition, δηλαδή να πρέπει να καταλήξουν στον ίδιο reduce node. Για λόγους απόδοσης πρέπει η εκτέλεση του Partitioner στον κάθε κόμβο να είναι ανεξάρτητη και να μην χρειάζεται ανταλλαγή πληροφοριών μεταξύ των map nodes για να την επιλογή του partition στο οποίο ανήκει το κάθε key.

Ένας καλός Partitioner χωρίζει τα δεδομένα έτσι ώστε όλα τα partition να έχουν περίπου το ίδιο μέγεθος. Σε διαφορετική περίπτωση δεν επιτυγχάνεται μέγιστη παραλληλία, καθώς κάποιοι κόμβοι θα πρέπει να επεξεργαστούν δυσανάλογα μεγάλο όγκο δεδομένων, ενώ άλλοι να μην έχουν δεδομένα προς επεξεργασία. Ο default Partitioner είναι ο *HashPartitioner*, ο οποίος χρησιμοποιεί μια hush function ανάλογα με το πλήθος των reduce nodes για τον χωρισμό των keys. Ο χρήστης μπορεί να ορίσει δικούς του Partitioners καθώς και το πλήθος των partitions, δηλαδή το πλήθος των reduce nodes που θέλει να χρησιμοποιηθούν από την εφαρμογή. Πρέπει να λαμβάνονται υπόψιν οι ανάγκες των εφαρμογών πριν την υλοποίηση Partitioner από τον προγραμματιστή, ώστε να παράγονται partitions με περίπου ίδιο μέγεθος.

Reduce function – Reducer Interface

Η reduce function αποτελεί το τελευταίο στάδιο του Hadoop MapReduce. Ο *Reducer* εφαρμόζει την ορισμένη από τους χρήστες reduce function στα intermediate key/value pairs

που έχουν προκύψει μετά το στάδιο του map και παράγει τα τελικά αποτελέσματα στη μορφή key/value pairs. Παρακάτω φαίνεται το interface του Reducer.

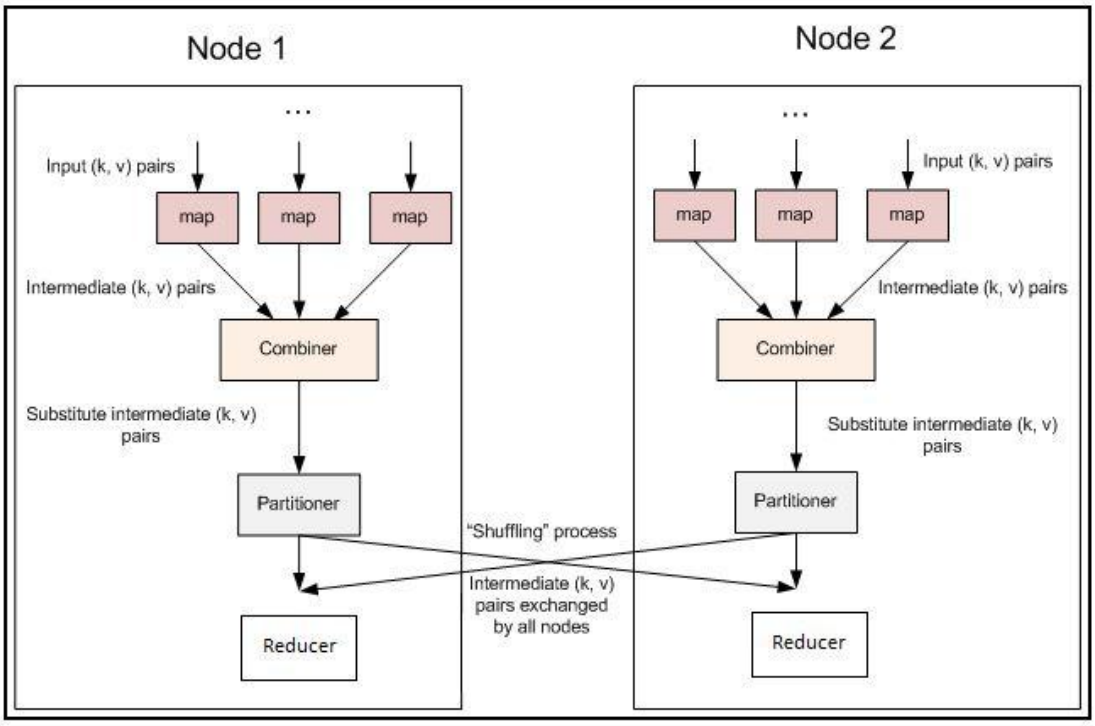
```
public interface Reducer<K2,V2,K3,V3>
```

Το `K2` και `V2` ορίζουν τον τύπο δεδομένων των intermediate keys και values που επιστρέφει ο Mapper και αποτελούν την είσοδο του Reducer. Το `K3` και `V3` αποτελούν το output του Reducer και είναι ίδιου τύπου δεδομένων με τα `K2` και V2 αντίστοιχα. Ενώ ο Reducer επιτρέπεται να κάνει αλλαγές στα values της εισόδου του, δεν μπορεί να μεταβάλλει τα keys.

Όπως έχει αναλυθεί παραπάνω μετά το map προαιρετικά εφαρμόζεται μία combine function, η οποία ομαδοποιεί τα δεδομένα κάθε map κόμβου παράγοντας keys/list_of_values pairs. Στη συνέχεια αυτά χωρίζονται τοπικά σύμφωνα με έναν Partitioner και ανατίθενται σε κάποιο reduce task. Το πλήθος των reduce tasks είναι ίσο με τον αριθμό των partition των intermediate key/value pairs που προκύπτουν μετά την map ή την combine. Το framework αναλαμβάνει την αποστολή όλων των partitions στους αντίστοιχους reduce nodes, την εκτέλεση της reduce function στα key/value pairs και την επαναδρομολόγησή της σε περίπτωση αποτυχίας. Στην περίπτωση που δεν θέλει ο προγραμματιστής να χρησιμοποιήσει reduce function, τα αποτελέσματα των map tasks αποθηκεύονται στο filesystem, χωρίς να υποστούν κάποια ταξινόμηση. Η reduce function περιγράφεται από τον παρακάτω τύπο.

$$reduce(i\_key, list(merged\_value)) \rightarrow list(f\_key.f\_value)$$

Πιο αναλυτικά ο Reducer αποτελείται από τρία επιμέρους στάδια. Τα στάδια *Shuffle*, *Sort* και *Reduce*. Στο στάδιο του Shuffle το framework στέλνει τα partition από όλους τους map nodes στα reduce tasks που έχουν ανατεθεί. Καθώς το partition γίνεται με βάση το key, δεδομένα με ίδιο key θα σταλούν στο ίδιο reduce task. Το κάθε map task ωστόσο εκτελείται ανεξάρτητα και είναι πιθανό διαφορετικοί map nodes να παράξουν το ίδιο key. Για λόγους απόδοσης η reduce function πρέπει να εφαρμοστεί σε key/value pairs, με το κάθε key να εμφανίζεται μόνο μία φορά. Αυτό επιτυγχάνεται στο στάδιο του Sort, όπου λίστες values με κοινό κλειδί ομαδοποιούνται σε μία. Τέλος στο στάδιο του Reduce η reduce function καλείται για όλα τα key/list_of_values και παράγει τα τελικά key/value pairs. Το αποτέλεσμα γράφεται στο filesystem και αν προκύψουν περισσότερα του ενός key/value pairs αυτά δεν ταξινομούνται από το MapReduce.



Εικόνα 11 Hadoop MapReduce Execution Progress

3.7.2 Παράδειγμα WordCount με το Hadoop MapReduce

Σε αυτή την ενότητα θα παρουσιαστεί ένα παράδειγμα εφαρμογής του Hadoop MapReduce. Η εφαρμογή *WordCount* δέχεται σαν είσοδο αρχεία και επιστρέφει το πλήθος εμφάνισης όλων των λέξεων τους. Στο παράδειγμα η Java class *WordCount.java* ορίζει δύο classes, την *Map* και τη *Reduce*, και την μέθοδο *main* η οποία τις καλεί για επίλυση του προβλήματος. Ακολουθεί ο κώδικας και η ανάλυση της λειτουργίας των μεθόδων.

WordCount.java

```
   public class WordCount {

      public static class Map extends MapReduceBase
               implements Mapper<LongWritable, Text, Text,
IntWritable> {

        private final static IntWritable one = new
IntWritable(1);
        private Text word = new Text();

        public void map(LongWritable key, Text value,
                        OutputCollector<Text, IntWritable>
output,
                        Reporter reporter) throws IOException {
          String line = value.toString();
          StringTokenizer tokenizer = new
StringTokenizer(line);
          while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
```

```
            output.collect(word, one);
          }
        }
      }

      public static class Reduce extends MapReduceBase
               implements Reducer<Text, IntWritable, Text,
IntWritable> {

        public void reduce(Text key, Iterator<IntWritable>
values,
                           OutputCollector<Text, IntWritable>
output,
                           Reporter reporter) throws
IOException {
          int sum = 0;
          while (values.hasNext()) {
            sum += values.next().get();
          }
          output.collect(key, new IntWritable(sum));
        }
      }

      public static void main(String[] args) throws Exception {
        JobConf conf = new JobConf(WordCount.class);
        conf.setJobName("wordcount");
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);
        conf.setMapperClass(Map.class);
        conf.setCombinerClass(Reduce.class);
        conf.setReducerClass(Reduce.class);
        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);
        FileInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new
Path(args[1]));
        JobClient.runJob(conf);
      }
   }
```

Η μέθοδος main ορίζει τις κλάσεις που θα χρησιμοποιηθούν για το map, το combine και το reduce και τους τύπους δεδομένων των key/values της εισόδου και της εξόδου του MapReduce. Στο παράδειγμα η Map class θα χρησιμοποιηθεί για το map και η Reduce class για το combine και το reduce. Τα input key/value pairs του map δημιουργούνται με την εντολή `conf.setInputFormat(TextInputFormat.class)`, με την οποία τα αρχεία χωρίζονται σε γραμμές. Στο key αποθηκεύεται η θέση στο αρχείο και στο value το περιεχόμενο της γραμμής. Το input key/values του map είναι τύπου `<LongWritable/Text>` και τα output key/values του reduce είναι τύπου `<Text/IntWritable>`.

Η Map class υλοποιεί έναν Mapper που σαν είσοδο παίρνει key/value pairs της μορφής `<LongWritable/Text> Text` και επιστρέφει intermediate key/value pairs στη μορφή `<Text/IntWritable>`. Ορίζει δύο μεταβλητές, την `one`, έναν `IntWritable` με τιμή ίση με 1, και την `word` που είναι τύπου `Text`. Υλοποιείται η μέθοδος map, η οποία

μετατρέπει το value της εισόδου, που είναι μία γραμμή του αρχείου, σε `String` και από αυτό εξάγει όλες τις λέξεις `word`. Κάθε `word` γίνεται emit ως `(word,one)` key/value pair.

Η Reduce class υλοποιεί έναν Reducer, ο οποίος σαν είσοδο παίρνει key/value pairs της μορφής `<Text/IntWritable>` και αφού εφαρμόσει σε αυτή τη reduce μέθοδο επιστρέφει key/value pairs στη μορφή `<Text/IntWritable>`. Η μέθοδος reduce αθροίζει για κάθε key όλα τα values που έχει και κάνει emit ένα key/value pair, όπου στο key βρίσκεται η λέξη και στο value το πλήθος των εμφανίσεών της.

Ακολουθεί ένα παράδειγμα εκτέλεσης της WordCount για τα αρχεία *file1* και *file2*.

file1	file2
run forest run	i run in the forest

Η map method παράγει τα ακόλουθα key/value pairs.

file1	file2
(run,1) (forest,1) (run,1)	(i,1) (run,1) (in,1) (the,1) (forest,1)

Στο παράδειγμα για το combine χρησιμοποιείται η Reduce class. Αυτό σημαίνει πως πριν τα key/value pairs σταλθούν στους reduce nodes, ομαδοποιούνται τα keys στο κάθε αρχείο. Δηλαδή η μέθοδος reduce εφαρμόζεται τοπικά στα key/value pairs του κάθε αρχείου εισόδου ξεχωριστά και ομαδοποιεί τις εμφανίσεις των ίδιων keys. Η combine επιστρέφει τα ακόλουθα key/value pairs.

file1	file2
(run,2) (forest,1)	(i,1) (run,1) (in,1) (the,1) (forest,1)

Η reduce method δέχεται σαν είσοδο τα παραπάνω και παράγει τα τελικά key/value pairs.

output
(i,1) (run,3) (in,1) (the,1) (forest,2)

3.7.3 Χαρακτηριστικά και Εργαλεία του Hadoop MapReduce

Το Hadoop MapReduce Framework χρησιμοποιείται για την επεξεργασία δεδομένων που είναι αποθηκευμένα σε cluster υπολογιστών. Ο κατανεμημένος τρόπος χειρισμού των δεδομένων αποτελεί το μεγάλο πλεονέκτημα του Hadoop αλλά καθιστά την δημιουργία και

συντήρηση των εφαρμογών που το χρησιμοποιούν διαφορετικές από άλλα μοντέλα προγραμματισμού. Για αυτό το λόγο προσφέρονται εργαλεία και μηχανισμοί για την ανάπτυξη, τον χειρισμό, τον έλεγχο, το testing και το debugging των προγραμμάτων που χρησιμοποιούν το Hadoop MapReduce. Παρακάτω παρουσιάζονται ορισμένα από αυτά.

Distributed Cache

Αποτελεί το μηχανισμό του Hadoop για την αποδοτική κατανομή μεγάλων read-only αρχείων σε όλους του κόμβους του cluster. Τα ζητούμενα δεδομένα αποθηκεύονται στην cache των κόμβων πριν από την εκτέλεση των tasks που θα τα χρειαστούν. Τα αρχεία δεν πρέπει να υποστούν επεξεργασία από τις εφαρμογές ή εξωτερικά όσο χρόνο το task που τα χρησιμοποιεί εκτελείται. Ένα παράδειγμα χρήσης είναι η κατανομή δεδομένων που είναι απαραίτητα σε όλους τους Mappers.

JobTracker Web UI

Για την παρακολούθηση της προόδου εκτέλεσης των διεργασιών ο JobTracker παρέχει ένα web interface. Καθώς οι εφαρμογές τρέχουν κατανεμημένα είναι πολύ δύσκολος ο έλεγχος των tasks που εκτελούνται. Το JobTracker Web UI, μέσω του *job ID* που ανατίθεται σε κάθε διεργασία στην εκκίνησή της, εντοπίζει το κάθε task και παρέχει πληροφορίες για την εξέλιξη του map, του reduce, το μέγεθος των I/O που πραγματοποιούνται καθώς και άλλες χρήσιμες πληροφορίες.

Counters

Οι Counters χρησιμοποιούνται για την παρουσίαση της συνολικής απόδοσης των εφαρμογών. Οι προγραμματιστές ορίζουν counters, οι οποίοι αυξάνουν τις τιμές τους μετά από συγκεκριμένες ενέργειες του συστήματος. Τα αποτελέσματα παρουσιάζονται στο JobTracker Web UI μαζί με τους counters που έχει ενσωματωμένους το Hadoop. Ένα παράδειγμα χρήσης είναι η μέτρηση των εγγραφών που απορρίπτονται κατά την εφαρμογή του MapReduce. Παρότι δεν έχει αντίκτυπο στην ίδια την εφαρμογή μια τέτοια μέτρηση δίνει καλύτερη εικόνα της συμπεριφοράς της, βοηθώντας τους προγραμματιστές να βελτιώσουν την απόδοσή της.

Skipping Bad Records

Καθώς σε πολλές περιπτώσεις οι εφαρμογές χειρίζονται μεγάλο όγκο δεδομένων είναι πιθανό κάποια από αυτά να έχουν λάθη, δηλαδή να μην έχουν την δομή ή τις πληροφορίες που αναμένονται. Οι εφαρμογές θα πρέπει να εντοπίζουν τέτοια αρχεία και να μπορούν να τα χειριστούν χωρίς να επηρεάζεται η εκτέλεσή τους. Το Hadoop παρέχει έναν μηχανισμό για το skipping εγγραφών που μπορεί να προκαλέσουν την αστοχία των εφαρμογών. Κατά την επαναλαμβανόμενη εκτέλεση ενός task, αυτό θα οδηγηθεί στον TaskTracker ο οποίος θα προσδιορίσει τις εγγραφές που είναι υπεύθυνες για την αστοχία, θα τις διαγράψει και θα το επαναδρομολογήσει. Αυτός ο μηχανισμός θα πρέπει να ενεργοποιηθεί από τους διαχειριστές και μπορεί να εφαρμοστεί στο στάδιο του map και στο στάδιο του reduce.

Isolation Runner

Για το debugging των MapReduce jobs μπορούν να χρησιμοποιηθούν τα log files που αποθηκεύονται στον NameNode, τα οποία όπως έχει παρουσιαστεί διατηρούν όλες τις αλλαγές που έχουν υποβληθεί στα αρχεία. Ωστόσο καθώς κάθε MapReduce job αποτελείται από πολλά tasks, τα οποία τρέχουν παράλληλα σε πολλούς κόμβους, σε περιπτώσεις αποτυχίας το debugging είναι πολύ δύσκολο. Πρέπει να εντοπιστεί ο κόμβος που έγινε το σφάλμα, το input του συγκεκριμένου task και ο ακριβής προσδιορισμός του τρόπου εκτέλεσής του. Το Hadoop MapReduce προσφέρει τον Isolation Runner, ένα εργαλείο για τον εντοπισμό του task που προκάλεσε την αποτυχία και την παρακολούθηση των βημάτων που οδήγησαν σε αυτήν. Όταν μια job αποτύχει, χρησιμοποιείται το JobTracker Web UI για τον εντοπισμό του task που προκάλεσε την αστοχία και ο κόμβος στον οποίο έτρεχε. Με τον Isolation Runner απομονώνεται το task και επανεκτελείται στον κόμβο με το ίδιο input. Παρακολουθώντας την εκτέλεση του με έναν debugger, προσδιορίζεται το ακριβές σημείο που προκάλεσε την αστοχία. Στην παρούσα έκδοση του Hadoop υποστηρίζεται η επανεκτέλεση μόνο των map tasks.

Compression

Μετά την ολοκλήρωση των map tasks τα δεδομένα διανέμονται μέσω δικτύου στους κόμβους όπου θα εκτελεστεί το στάδιο του reduce. Μέχρι τώρα ο μόνος τρόπος που έχει αναφερθεί για την μείωση της συνολικής μεταφερόμενης πληροφορίας είναι η χρήση της combine function αμέσως μετά το map. Στην επόμενη ενότητα θα παρουσιαστούν και άλλες τεχνικές για την μείωση του συνολικού όγκου των δεδομένων. Το Hadoop MapReduce Framework προσφέρει για περαιτέρω βελτίωση της απόδοσης των εφαρμογών εργαλεία για την συμπίεση των δεδομένων τόσο μετά το στάδιο του map, για την μείωση της κίνησης στο δίκτυο, όσο και μετά το reduce, για μείωση του συνολικού output των MapReduce jobs. Περιλαμβάνονται επίσης εργαλεία για την αποσυμπίεση των δεδομένων. Είναι στην ευχέρεια των προγραμματιστών να επιλέξουν τα κατάλληλα εργαλεία για την συμπίεση και αποσυμπίεση των αρχείων. Από το Hadoop προσφέρεται το εξειδικευμένο *sequence file format* για την συμπίεση αρχείων με δεδομένα της μορφής key/value pairs. Μπορεί επίσης να χρησιμοποιηθεί για αποθήκευση binary αρχείων, όπως εικόνες. Η παραλληλοποίηση στο Hadoop επιτυγχάνεται με τμηματοποίηση του input και διαμοιρασμό του στους κόμβους για ταυτόχρονη επεξεργασία. Γι' αυτό το λόγο το sequence file format είναι σχεδιασμένο να συμπιέζει τα αρχεία και ταυτόχρονα να τα καθιστά τμηματοποιήσιμα, κάνοντας δυνατή την παράλληλη αποσυμπίεσή τους στα reduce tasks.

3.7.4 Advanced MapReduce

Όσο εξελίσσονται οι ανάγκες των εφαρμογών, απαιτείται η υποβολή ολοένα πιο σύνθετων ερωτημάτων. Οι απλές ενέργειες που εφαρμόζονται στα δεδομένα από ένα MapReduce job συχνά δεν είναι αρκετές. Επίσης σε πολλές περιπτώσεις απαιτείται η επεξεργασία δεδομένων που ανήκουν σε διαφορετικά datasets, σε διαφορετικές collections. Τότε είναι απαραίτητη η υλοποίηση κάποιου είδους join, τα οποία όμως στην μορφή που συναντώνται στις σχεσιακές βάσεις δεδομένων δεν υποστηρίζονται από το Hadoop MapReduce. Για αυτούς τους λόγους το Hadoop framework περιλαμβάνει εργαλεία και τεχνικές που ικανοποιούν τέτοιες πολύπλοκες απαιτήσεις.

3.7.4.1 Διαδοχικά MapReduce tasks

Για την εξαγωγή απαντήσεων σε πολύπλοκα ερωτήματα πολλές φορές δεν αρκεί μια MapReduce Job, αλλά περισσότερες, οι οποίες εκτελούνται είτε σε διαφορετικά δεδομένα ανεξάρτητα είτε η έξοδος της μιας είναι η είσοδος της επόμενης. Στις τεχνικές με διαδοχική εκτέλεση maps και reduces, αποτελεί γενικό κανόνα τα output key/value pairs μιας συνάρτησης να ταιριάζουν με τα input key/value pairs της επόμενης. Για λόγους απόδοσης το Hadoop MapReduce διαθέτει εργαλεία για την αυτοματοποιημένη εκτέλεση των MapReduce Jobs που υλοποιούν οι χρήστες.

Σε περιπτώσεις που απαιτείται διαδοχική εκτέλεση δύο ή περισσοτέρων MapReduce Jobs, αυτά θα μπορούσαν να εκτελεστούν σε σειρά χειροκίνητα από τους προγραμματιστές. Δηλαδή να εκτελεστεί το πρώτο MapReduce, να αποθηκευτούν τα αποτελέσματα στο filesystem και στη συνέχεια να ξεκινήσει η εκτέλεση του επόμενου MapReduce σε αυτά. Με το Hadoop MapReduce οι χρήστες μπορούν να αυτοματοποιήσουν την εκτέλεση διαδοχικών MapReduce Jobs σε μία, με το output μιας ενδιάμεσης MapReduce Job να γίνεται input της επόμενης.

Μέσω των *Job* και *JobControl* classes προσφέρονται μηχανισμοί για εκτέλεση πιο πολύπλοκων συνδυασμών MapReduce Jobs. Παρέχονται εργαλεία παρακολούθησης και διαχείρισης των tasks και ορίζονται κανόνες, όπως εξάρτησης ενός MapReduce από ένα άλλο μέσω της *addDependingJob* class. Όταν ένα task εξαρτάται από ένα άλλο, δεν μπορεί να ξεκινήσει την εκτέλεσή του μέχρι να επιστρέψει το ορισμένο προηγούμενο. Αντίθετα ανεξάρτητα tasks μπορούν να εκτελεστούν παράλληλα αυξάνοντας την απόδοση της εφαρμογής.

Σε εφαρμογές που απαιτούν πολλά στάδια pre-processing και post-processing η συνήθης τεχνική είναι αυτά να υλοποιούνται σαν ανεξάρτητα MapReduce Jobs. Ωστόσο όταν κάθε στάδιο απαιτεί πολλά I/O και μεγάλη αποθήκευση πληροφορίας, είναι αποδοτικότερο να ομαδοποιούνται τα pre-processing και post-processing στάδια καθώς έτσι μειώνεται η πολλαπλή μεταφορά και αποθήκευση των ίδιων δεδομένων. Μέσω των *ChainMapper* και *ChainReducer* classes είναι δυνατή η ομαδική εκτέλεση των pre-processing και post-processing τμημάτων της εφαρμογής αντίστοιχα. Ένα MapReduce Job που χρησιμοποιεί αυτήν την τεχνική αποτελείται από έναν ChainedMapper, ένα MapReduce task και έναν ChainedReducer. Οι ChainedMapper και ChainedReducer υλοποιούνται μέσω map functions. Οι συναρτήσεις map και reduce εκτελούνται με τη σειρά, με την έξοδο της μίας να είναι η είσοδος της επόμενης. Ακολουθεί ένα τέτοιο παράδειγμα.

$$Map1|Map2|Map3|Reduce|Map4|Map5$$

Στο παράδειγμα οι συναρτήσεις $Map1$ και $Map2$ αποτελούν το pre-processing κομμάτι της εφαρμογής, οι $Map3$ και $Reduce$ αποτελούν το κομμάτι του MapReduce και οι $Map4$ και $Map5$ εκτελούν το post-processing τμήμα. Το πλήθος των pre και post-processing συναρτήσεων ποικίλει ανάλογα με τις ανάγκες των χρηστών.

3.7.4.2 Τεχνικές Join στο Hadoop MapReduce

Το Hadoop MapReduce εφαρμόζεται σε clusters υπολογιστών και συγκεκριμένα σε αρχεία. Καθώς αποτελεί ένα γενικό εργαλείο εξαγωγής πληροφοριών με συναρτήσεις που ορίζει ο χρήστης, δεν απαιτείται ένα ορισμένο schema στη δομή των αρχείων. Ανάλογα με τις ανάγκες τους οι προγραμματιστές μπορούν να διαλέξουν τον τρόπο δόμησης των δεδομένων τους και να υλοποιήσουν συναρτήσεις που αξιοποιούν όλες τις ιδιαιτερότητές τους.

Όσο οι ανάγκες των εφαρμογών γίνονται πολυπλοκότερες, συχνά απαιτείται ο συνδυασμός πληροφορίας από αρχεία που ανήκουν σε διαφορετικές πηγές. Στην περίπτωση των σχεσιακών βάσεων δεδομένων και μέσω της σχεσιακής άλγεβρας αυτή η συνένωση

δεδομένων υλοποιείται με τα JOINS. Με χρήση JOINSεγγραφές που ανήκουν σε διαφορετικά tables δημιουργούν απαντήσεις που περιέχουν πληροφορίες από τα επιθυμητά tables. Στο παράδειγμα που ακολουθεί θέλουμε το inner join δύο tables, People Πίνακας 4 και Movies Πίνακας 3, δηλαδή όλες τις στήλες οι οποίες θα περιέχουν τις εγγραφές των People που είναι Directors σε τουλάχιστον 1 Movie. Το αποτέλεσμα παρουσιάζεται στον Πίνακας 5.

M_id	Title	P_id
Ax01	foo	2
Ax02	bar	3

Πίνακας 3 Table Movies

P_id	Name	Surname	Address	Phone
1	A	B	Street	No_1
2	C	D	Road	No_2

Πίνακας 4 Table People

P_id	Name	Surname	Address	Phone	M_id	Title
2	C	D	Road	No_2	Ax01	foo

Πίνακας 5 Inner Join of Tables Movies and People

Στις σχεσιακές βάσεις δεδομένων με την υποβολή queries τα JOINS αναλαμβάνονται αυτόματα από τη βάση, καθώς αυτή είναι δομημένη με συγκεκριμένο schema. Στο Hadoop MapReduce όμως τα δεδομένα δομούνται με schema-free τρόπο και ανάλογα με τις εκάστοτε ανάγκες των προγραμματιστών. Ταυτόχρονα οι συναρτήσεις map και reduce που ορίζονται από τους χρήστες, μπορούν να εφαρμοστούν μόνο σε δεδομένα που ανήκουν στην ίδια πηγή, όπως στην ίδια collection των NoSQL βάσεων δεδομένων. Για αυτούς τους λόγους το Hadoop MapReduce δεν υποστηρίζει την εφαρμογή JOINSαυτόματα στα δεδομένα όπως στις σχεσιακές βάσεις. Σε αυτήν την ενότητα θα παρουσιαστούν οι εναλλακτικές τεχνικές που χρησιμοποιεί το Hadoop MapReduce για την υλοποίησή τους.

Reduce-side join

Μια τεχνική που χρησιμοποιεί το Hadoop MapReduce για την υλοποίηση JOINSείναι το reduce-side join, μέσω του *datajoin package*. Παρότι δεν είναι η πιο αποδοτική, αποτελεί την πιο γενική μέθοδο που χρησιμοποιείται ως βάση για πιο εξειδικευμένες τεχνικές. Το reduce-side join εισάγει τις έννοιες *data source*, *tag* και *group key*. Η data source, το ανάλογο των tables στις σχεσιακές βάσεις δεδομένων, είναι οι διαφορετικές πηγές από τις οποίες θα αντληθούν πληροφορίες για την υλοποίηση του join. Κάθε data source αποτελείται από ένα ή περισσότερα αρχεία, τα οποία υποχρεωτικά έχουν το ίδιο schema. Το tag χρησιμοποιείται για την διατήρηση metadata στα δεδομένα ακόμα και μετά την επεξεργασία τους. Στο reduce-side join τα records, το αντίστοιχο των εγγραφών στις σχεσιακές βάσεις δεδομένων, γίνονται tagged με τις data sources τους, ώστε κάθε στιγμή να είναι γνωστή η προέλευσή τους. Το group key είναι το ανάλογο του join key των σχεσιακών βάσεων δεδομένων, αλλά με πολύ πιο ευέλικτη χρήση, καθώς μπορεί να είναι οποιαδήποτε συνάρτηση ορίσουν οι χρήστες.

Στο reduce-side join οι χρήστες ορίζουν τέτοιες map functions, ώστε τα δεδομένα να έχουν κατάλληλη μορφή για το join που θα πραγματοποιηθεί στο στάδιο του reduce. Για τον ορισμό των map και reduce functions στο reduce-side join χρησιμοποιούνται οι classes *DataJoinMapperBase* και *DataJoinReducerBase* αντίστοιχα. Οι map functions δέχονται σαν είσοδο και επιστρέφουν σαν έξοδο key/value pairs. Ορίζεται ως output key το group key με

βάση το οποίο θα γίνει το join και ως output value το record. Κάθε record γίνεται tagged με το data source του, δηλαδή με το αρχείο από το οποίο προήλθε, μέσω της class *TaggedMapOutput*. Πριν επιστρέψουν οι map functions ομαδοποιούν τα key/value pairs με βάση το key. Καθώς το key έχει οριστεί ως το group key, όλες οι εγγραφές με κοινό group key που πρέπει να γίνουν join καταλήγουν στο list_of_values ενός key/list_of_values pair. Ακολουθεί σχηματικά το προηγούμενο παράδειγμα, όπου το group key ορίζεται το P_id.

Movies	People
Ax01,foo,2	1,A,B,Street,No_1
Ax02,bar,3	2,C,D,Road,No_2

Map Function Before Return
(2, [Ax01, foo, 2 || *Movies*])
(3, [Ax02, bar, 3 || *Movies*])
(1, [1, A, B, Street, No_1 || *People*])
(2, [2, C, D, Road, No_2 || *People*])

Map Function Output
(2, list{[Ax01, foo, 2 || *Movies*], [2, C, D, Road, No_2 || *People*]})
(3, [Ax02, bar, 3 || *Movies*])
(1, [1, A, B, Street, No_1 || *People*])

Στα παραπάνω key/value pairs το key είναι το **P_id** και στο value οι εγγραφές των δύο data sources, *Movies* και *People*. Στην έξοδό της η map έχει ομαδοποιήσει τις εγγραφές με ίδιο **P_id**.

Στη συνέχεια η reduce function δέχεται σαν είσοδο τα key/list_of_values pairs που επιστρέφει η map function και επιστρέφει για το κάθε ένα όλους τους συνδυασμούς των list_of_values που δεν έχουν ίδιο data source. Αυτός ο έλεγχος γίνεται διότι το join ορίζεται για εγγραφές διαφορετικών data sources. Στο παράδειγμα αν μια εγγραφή από το People είχε σκηνοθετήσει δύο ή περισσότερες Movies, στο join δεν ανήκει η ένωση των δύο Movies, αλλά η ένωση του People με την κάθε Movie. Ακολουθεί σχηματικά το παράδειγμα κατά την εφαρμογή της reduce function.

Reduce Function Input
(2, list{[Ax01, foo, 2 || *Movies*], [2, C, D, Road, No_2 || *People*]})
(3, [Ax02, bar, 3 || *Movies*])
(1, [1, A, B, Street, No_1 || *People*])

Reduce Function Output
2,Ax01,foo,*Movies*,C,D,Road,No_2

Είναι φανερό ότι αυτή η μέθοδος join παρότι απλή δεν είναι ιδιαίτερα αποδοτική, καθώς το join δεν πραγματοποιείται παρά μόνο στο στάδιο του reduce. Έτσι απαιτείται μεταφορά μεγάλου όγκου πληροφορίας, ενώ πολλά από τα δεδομένα τελικά απορρίπτονται.

Replicated Joins με χρήση Distributed Cache

Ένας πιο αποδοτικός τρόπος εφαρμογής join είναι αυτό να γίνει κατά το στάδιο του map. Ωστόσο συχνά ο Mapper δεν έχει πρόσβαση σε όλα τα data sources από τα οποία θα γίνει

άντληση πληροφοριών. Μια λύση είναι να γνωρίζει εκ των προτέρων ο Mapper την ακριβή τοποθεσία των data sources, το group key που θα χρησιμοποιηθεί και τη δομή των επιμέρους αρχείων. Στις περισσότερες περιπτώσεις αυτή η πρακτική απαιτεί την εφαρμογή άλλων MapReduces για να έρθουν τα δεδομένα στην επιθυμητή μορφή, κάνοντάς τη μη αποδοτική.

Μια άλλη λύση είναι η αντιγραφή μιας data source σε όλους τους Mappers και η εφαρμογή join τοπικά σε καθέναν από αυτούς. Η πράξη έχει δείξει ότι σε πολλές εφαρμογές η μια data source είναι τάξεις μεγέθους μεγαλύτερη από την άλλη. Σε αυτές τις περιπτώσεις και αν η μικρή data source χωράει στη μνήμη του κάθε Mapper, είναι αποδοτικό στο στάδιο του map να τμηματοποιείται η μεγάλη data source και να στέλνεται σε όλα τα map tasks η μικρότερη. Με αυτόν τον τρόπο από το στάδιο του map γίνεται το join και απορρίπτονται τα δεδομένα που δεν ανήκουν σε αυτό. Αυτή η τεχνική ονομάζεται replicated join, καθώς ένα κομμάτι των δεδομένων αντιγράφεται σε όλους τους κόμβους που συμμετέχουν στο MapReduce. Για την μεταφορά των μικρότερων data sources σε όλους τους κόμβους χρησιμοποιείται ο μηχανισμός της Distributed Cache, που έχει παρουσιαστεί παραπάνω.

Κατά την εκτέλεση της map function το τμήμα της μεγάλης data source στο κάθε map task και η μικρότερη data source δημιουργούν key/value pairs, με key το group key και value την εγγραφή. Κάθε φορά που ένα key/value pair της μεγάλης data source βρίσκει ένα key/value pair της μικρής με ίδιο key, τα values ομαδοποιούνται. Τα key/value pairs με key που δεν υπάρχει στα key/value pairs της μικρής data source απορρίπτονται και διαγράφονται. Με αυτόν τον τρόπο το join πραγματοποιείται στο στάδιο του map, χωρίς αχρείαστη μεταφορά πληροφορίας εγγραφών που στη συνέχεια απορρίπτονται.

Semi-join: reduce-side join με map-side filtering

Σε περιπτώσεις που οι παραπάνω μέθοδοι δεν μπορούν να εφαρμοστούν είτε για λόγους απόδοσης είτε επειδή δεν είναι αρκετά μικρή η μια data source ώστε να χωράει στην τοπική μνήμη των Mappers, δίνεται η εναλλακτική του semi-join. Το semi-join αποτελεί μια πιο αποδοτική έκδοση του reduce-side join που παρουσιάστηκε παραπάνω. Κατά το reduce-side join ο Mapper κάνει tag στις εγγραφές την data source από την οποία προήλθαν, ομαδοποιεί σύμφωνα με το group key και στέλνει όλες τις εγγραφές στο reduce στάδιο. Αυτό έχει ως αποτέλεσμα να μεταφέρεται μεγάλος όγκος δεδομένων που δεν ανήκουν στο join και τα οποία τελικά απορρίπτονται. Εναλλακτικά με χρήση semi-join η απόρριψη των δεδομένων που δεν ανήκουν στο join γίνεται κατά το στάδιο του map, αφαιρώντας την πλεονάζουσα μεταφορά δεδομένων στο δίκτυο. Αυτό επιτυγχάνεται με την εφαρμογή κατάλληλων φίλτρων στο στάδιο του map σύμφωνα με τα δομικά χαρακτηριστικά των δεδομένων και τις ανάγκες των εφαρμογών.

Κεφάλαιο 4
NoSQL Βάσεις Δεδομένων

Σε αυτό το κεφάλαιο θα παρουσιαστεί το κομμάτι του λογισμικού που εγκαθίσταται πάνω από το επίπεδο του middleware και αποτελεί συνδετικό κρίκο της επικοινωνίας του με τις τελικές εφαρμογές που αναπτύσσουν οι χρήστες. Το λογισμικό αυτό αφορά στις βάσεις δεδομένων και συγκεκριμένα σε αυτήν την διπλωματική στις NoSQL βάσεις δεδομένων. Στις επόμενες ενότητες θα αναλυθούν οι λόγοι που οδήγησαν στις ανάπτυξη των NoSQL βάσεων δεδομένων, θα δοθούν τα χαρακτηριστικά και οι κανόνες που πρέπει να διακρίνουν κάθε βάση και θα παρουσιαστούν οι πιο χαρακτηριστικές NoSQL βάσεις που εμφανίζονται μέχρι σήμερα, με έμφαση σε εκείνη που επιλέχθηκε από αυτήν την διπλωματική.

4.1 Βάσεις Δεδομένων

Μια βάση δεδομένων είναι μια συλλογή από δεδομένα, που οργανώνονται σε δομές, τέτοιες ώστε να αναπαριστούν την πραγματικότητα με τον καλύτερο δυνατό τρόπο. Οι διαφορετικές δομές ορίζουν και τα διαφορετικά μοντέλα σύμφωνα με τα οποία οργανώνεται η βάση. Η βάση δεδομένων έρχεται σε συνδυασμό με ένα σύστημα διαχείρισής της, το *Σύστημα Διαχείρισης Βάσεων Δεδομένων, ΣΔΒΔ*, ή *Database Management System – DBMS*. Υπάρχουν διάφορα μοντέλα στα οποία μπορεί να βασίζεται ένα DBMS καθώς και η γλώσσα προγραμματισμού για το χειρισμό του και την υποβολή ερωτημάτων. Οι παραδοσιακές βάσεις δεδομένων χρησιμοποιούν το σχεσιακό μοντέλο, το σχεσιακό DBMS, *Relational DBMS – RDBMS*, και τη γλώσσα προγραμματισμού SQL. Ένα νέο μοντέλο δόμησης και διαχείρισης δεδομένων αποτελούν οι NoSQL Databases. Διακρίνονται πολλά είδη NoSQL βάσεων ανάλογα με τον τύπο των δεδομένων προς αποθήκευση, κάθε ένα από τα οποία ορίζεται από τη δική του γλώσσα υποβολής ερωτημάτων.

4.2 Ανάγκη Ανάπτυξης NoSQL Βάσεων Δεδομένων

Το σχεσιακό μοντέλο είναι γενικά ευέλικτο, διαθέτει πολλά εργαλεία διαχείρισης των δεδομένων, ενημέρωσης της βάσης, *update*, εκτέλεσης συναλλαγών με αυτήν, *transactions*, και παροχής πληροφοριών στο χρήστη μέσω υποβολής ερωτημάτων. Αναπαριστά τα δεδομένα με δισδιάστατους πίνακες και τις σχέσεις μεταξύ τους μέσω κλειδιών. Είναι μια εύκολη στην κατανόηση δομή και καλύπτει τις ανάγκες των περισσότερων εφαρμογών.

Ωστόσο όλο και περισσότερο έχει παρατηρηθεί η ανάγκη για ταυτόχρονη αποθήκευση και διαχείριση μεγάλου όγκου κατανεμημένων δεδομένων, τα οποία εμφανίζουν πολυμορφία. Παρουσιάζεται συχνά η ανάγκη μεταφοράς των δεδομένων πάνω από ένα δίκτυο και η άντληση γνώσης από συστήματα *Peer to Peer, P2P*. Τέλος απαιτείται η χρήση των υπηρεσιών πάνω από διαφορετικές πλατφόρμες, σε ποικίλες εφαρμογές, το λογισμικό και τα εργαλεία των οποίων μπορεί να διαφέρουν. Για αυτού του είδους τις εφαρμογές οι βάσεις

δεδομένων που υλοποιούν το σχεσιακό μοντέλο δεν είναι ιδανικές, καθώς τα ερωτήματα που τίθενται στη βάση και η εκτέλεση των JOINS προς απάντησή τους, απαιτούν την επεξεργασία μεγάλου όγκου δεδομένων και τη μεταφορά τους πάνω από το δίκτυο, προκαλώντας μεγάλο latency και αυξημένο χρόνο απόκρισης των υπηρεσιών.

Για την υλοποίηση σύνθετων, κατανεμημένων υπηρεσιών δεν αρκούν παλαιότερα μοντέλα ανάπτυξης, όπως το μοντέλο του καταρράκτη, αλλά συντομότερα, διαδραστικότερα μοντέλα, όπου η ανάπτυξη και σχεδίαση είναι πιο κοντά στην παραγωγή των τελικών προϊόντων. Ακόμα οι διακυμάνσεις στις απαιτήσεις των χρηστών έκανε επιτακτική την ανάγκη ανάπτυξης μηχανισμών γρήγορου scale up και scale down στους πόρους που χρησιμοποιεί η εφαρμογή ανάλογα με τις ανάγκες κάθε χρονική στιγμή. Οι σχεσιακές βάσεις δεν έχουν δημιουργηθεί για τόσο ευέλικτες περιπτώσεις. Αναπτύσσονται σε γνωστούς εκ των προτέρων πόρους, με γνωστό schema και δεν επιτρέπουν πολυμορφία.

Για τους παραπάνω λόγους και για εφαρμογές που απαιτούν διαχείριση ολοένα αυξανόμενου όγκου δεδομένων, σε πραγματικό χρόνο ή σε στατιστικές μελέτες, ήταν απαραίτητη η δημιουργία ενός άλλου τρόπου αναπαράστασης και διαχείρισης των δεδομένων. Αναπτύχθηκαν οι NoSQL βάσεις δεδομένων, όπου τα δεδομένα δεν είναι απαραίτητα δομημένα σε πίνακες, όπως στο σχεσιακό μοντέλο, αλλά με διάφορους τρόπους όπως θα δούμε παρακάτω. Αυτό δίνει μεγάλη ελευθερία στους χρήστες που θέλουν να αναπτύξουν υπηρεσίες καθώς μπορούν να επιλέξουν το μοντέλο που ταιριάζει καλύτερα στις ανάγκες της εφαρμογής τους. Δεν υπάρχει συγκεκριμένη γλώσσα προγραμματισμού για το χειρισμό τους. Κάθε NoSQL βάση παρέχει εργαλεία για την διαχείριση των δεδομένων καθώς και δική της γλώσσα υποβολής ερωτημάτων, *query language*.

Οι NoSQL βάσεις παρέχουν επεκτασιμότητα, πολυμορφία και αύξηση στις επιδόσεις των εφαρμογών. Εφαρμόζονται σε clusters, υποστηρίζουν διαχείριση κατανεμημένων δεδομένων και παρέχουν μηχανισμούς ανάκτησης των δεδομένων μετά από καταστροφή, failure tolerant. Για την παρουσίαση των NoSQL βάσεων είναι απαραίτητη η αναφορά των χαρακτηριστικών που πρέπει να τις διακρίνουν και των κανόνων που πρέπει να ακολουθούν προκειμένου να αποτελούν αξιόπιστα εργαλεία αποθήκευσης και διαχείρισης δεδομένων. Στις επόμενες ενότητες δίνεται μια προεπισκόπηση των ιδιοτήτων που πρέπει να διακρίνουν τις βάσεις δεδομένων, καθώς και οι μηχανισμοί που χρησιμοποιούνται για την εκτέλεση των transactions.

4.3 Ιδιότητες ACID

Transaction, συναλλαγή, ονομάζεται κάθε σειρά ενεργειών, όπου κάθε ενέργεια διαβάζει ή γράφει αντικείμενα σε μια βάση δεδομένων. Η εφαρμογή των ιδιοτήτων ACID σε κάθε transaction εγγυάται την αξιοπιστία των σχεσιακών βάσεων δεδομένων. Ακολουθεί η ανάλυση των ιδιοτήτων ACID.

Ατομικότητα, Atomicity – Εξασφαλίζεται ότι είτε θα πραγματοποιηθούν όλες οι πράξεις ενός transaction είτε ότι θα αποτύχουν όλες, αφήνοντας τη βάση ανεπηρέαστη. Σε περίπτωση κατάρρευσης του συστήματος κατά τη διάρκεια ενός transaction θα αναιρεθούν ό,τι αλλαγές έχουν γίνει στη βάση και αυτή θα έρθει στη μορφή που ήταν πριν ξεκινήσει η εκτέλεση του.

Συνέπεια, Consistency – Εξασφαλίζεται ότι πριν και μετά την εκτέλεση ενός transaction, οι κανόνες και περιορισμοί που διέπουν τη βάση δεδομένων πληρούνται.

Απομόνωση, Isolation – Κάθε transaction θεωρεί πως είναι το μοναδικό που τρέχει στο σύστημα. Τα ενδιάμεσα αποτελέσματα ενός transaction δεν επηρεάζουν άλλα transaction που ενδεχομένως έχουν πρόσβαση στα ίδια δεδομένα. Το τελικό αποτέλεσμα της βάσης είναι το

ίδιο, αν όλα τα transactions έτρεχαν σειριακά.

Μονιμότητα, Durability – Οι αλλαγές που προκαλεί στη βάση ένα transaction που επιτυγχάνει θα παραμείνουν ακόμα και μετά από κατάρρευση του συστήματος.

4.4 Μηχανισμός Ελέγχου Συντονισμού

Για την αποφυγή ασυνεπειών στη βάση τα transactions πρέπει να φέρνουν τα ίδια αποτελέσματα σαν να εκτελούνταν σειριακά. Η σειριακή εκτέλεση δεν μπορεί να πραγματοποιηθεί για λόγους απόδοσης. Σε ένα περιβάλλον όπου τα παράλληλα transactions εκτελούνται ανεξέλεγκτα είναι εύκολο να δημιουργηθούν ασυνέπειες στη βάση. Είναι απαραίτητος ένας μηχανισμός ελέγχου συντονισμού, concurrency control mechanism, των transactions ώστε να αποφεύγονται τέτοια φαινόμενα. Ένα παράδειγμα ασυνέπειας είναι το "*lost update problem*", όπου ένα transaction Β γράφει στο δεδομένο που έχει ήδη γράψει ένα transaction Α και κάθε transaction που τρέχει ταυτόχρονα διαβάζει αυτό που έχει γράψει το transaction Β, ενώ θα έπρεπε να διαβάσει αυτό που έγραψε αρχικά το transaction Α. Ένα άλλο πρόβλημα είναι το "*dirty-read problem*", όπου ένα transaction Β διαβάζει δεδομένα που έχουν γραφτεί από ένα transaction Α που απέτυχε και καταλήγει με λάθος αποτελέσματα.

Για τον συντονισμό των transactions χρησιμοποιούνται διάφοροι αλγόριθμοι οι οποίοι μπορούν να διαχωριστούν είτε με βάση τη θεώρηση του αλγορίθμου για το μέλλον του transaction, δηλαδή αν μετά την ολοκλήρωση θα τερματίσει επιτυχώς ή ανεπιτυχώς, είτε με βάση τα μέσα που χρησιμοποιούνται για το συντονισμό. Στην πρώτη κατηγορία εντάσσονται οι αισιόδοξοι, *optimistic*, και απαισιόδοξοι, *pessimistic*, αλγόριθμοι και στη δεύτερη ο αλγόριθμος που βασίζεται σε κλειδώματα, *locks*, και ο αλγόριθμος που βασίζεται σε χρονικές σημάνσεις, *timestamps*. Από τους αλγορίθμους που χρησιμοποιούν timestamps θα αναλύσουμε τον *Multiversioning Concurrency Protocol* [2]. Παρακάτω παρουσιάζονται αυτοί οι αλγόριθμοι.

Αισιόδοξοι Αλγόριθμοι

Οι αισιόδοξοι αλγόριθμοι συντονισμού transactions κάνουν την υπόθεση πως το transaction θα τερματίσει επιτυχώς χωρίς συγκρούσεις. Αυτοί οι αλγόριθμοι δεν μπλοκάρουν τις αναγνώσεις και τις εγγραφές και στο τέλος, όταν τα transactions είναι έτοιμα να τερματίσουν επιτυχώς, αποφασίζουν αν υπήρξαν συγκρούσεις και αν παραβιάστηκαν οι κανόνες της βάσης, οπότε και το transaction αποτυγχάνει.

Απαισιόδοξοι Αλγόριθμοι

Οι απαισιόδοξοι αλγόριθμων ελέγχουν αν υπάρχει κίνδυνος σύγκρουσης στα transactions πριν την αρχή εκτέλεσής τους. Αν υπάρχει μπλοκάρουν το transaction μέχρι αυτός ο κίνδυνος να εξαφανιστεί. Γενικά χρήση τέτοιων αλγορίθμων επηρεάζει αρνητικά την απόδοση των συστημάτων.

Κλειδώματα

Σε αυτούς τους αλγορίθμους τα transactions εξασφαλίζουν κλειδώματα για δεδομένα που επεξεργάζονται και δεν επιτρέπουν σε άλλα transactions την πρόσβαση σε αυτά. Στις μη

κατανεμημένες βάσεις δεδομένων ο πιο διαδεδομένος αλγόριθμος με βάση τα κλειδώματα είναι ο αλγόριθμος κλειδώματος σε δύο φάσεις, *2 Phase Locking – 2PL*. Στα κατανεμημένα συστήματα διαχείρισης βάσεων δεδομένων τα κλειδώματα ελέγχονται από έναν διαχειριστή κλειδωμάτων, *lock manager*. Τα transactions ενημερώνουν τον διαχειριστή κλειδωμάτων ποια κομμάτια της βάσης επεξεργάζονται και ενημερώνονται από εκείνον ποια κομμάτια είναι διαθέσιμα προς επεξεργασία. Γίνεται μια αρκετά εκτενής μεταφορά πληροφορίας πριν ξεκινήσει η εκτέλεση του κάθε transaction. Αυτή η τεχνική δημιουργεί προβλήματα στις κατανεμημένες βάσεις δεδομένων, καθώς ο διαχειριστής κλειδωμάτων αποτελεί bottleneck για όλο το σύστημα.

Multiversioning Concurrency Protocol (MCC ή MVCC)

Είναι ο αλγόριθμος συντονισμού των transactions που χρησιμοποιείται σε πολλές σχεσιακές βάσεις δεδομένων καθώς και σε πολλές κατανεμημένες βάσεις δεδομένων. Κάθε φορά που ένα transaction κάνει ένα update στη βάση, δεν αντικαθιστούνται τα παλιά δεδομένα, αλλά δημιουργείται μία νέα έκδοση της βάσης. Παρότι υπάρχουν πολλές εκδόσεις στη βάση μόνο μία είναι η νεότερη. Αυτό υλοποιείται με χρήση timestamps ή *transaction IDs*. Με χρήση του MVCC κανένα transaction δεν χρειάζεται να περιμένει μέχρι να γίνει διαθέσιμο ένα αντικείμενο της βάσης όπως στην περίπτωση των κλειδωμάτων. Όλα τα αντικείμενα είναι διαθέσιμα ανά πάσα στιγμή. Παρακάτω παρουσιάζεται μια γενική περιγραφή του αλγορίθμου.

Κάθε transaction χαρακτηρίζεται μοναδικά από ένα transaction ID, το οποίο αυξάνεται όσο προστίθεται transactions. Το transactions ID δείχνει με ποια σειρά θα έπρεπε να εκτελεστούν τα transactions. Θεωρούμε transaction T_i και timestamp αυτού $ts(T_i)$. Σε κάθε έκδοση ενός αντικείμενου της βάσης ανατίθεται ένα write timestamp και ένα read timestamp σε μορφή διανύσματος $[wts, rts]$. Το wts είναι ίσο με το timestamp του transaction που έκανε το write, $ts(T_i)$, και το rts ισούται με το timestamp του πιο πρόσφατου transaction που έκανε read. Αν δεν έχει ακόμα γίνει read τίθεται $wts = rts$. Θεωρούμε αντικείμενο x και έκδοση του αντικειμένου x_k. Κάθε φορά που ένα transaction T_i θέλει να κάνει read, διαβάζει την έκδοση x_k με το μεγαλύτερο wts το οποίο είναι μικρότερο από $ts(T_i)$. Αν $rts < ts(T_i)$ τότε $rts = ts(T_i)$, δηλαδή αν το transaction T_i είναι το νεότερο που έχει κάνει read, το read timestamp της έκδοσης του αντικειμένου x_k γίνεται ίσο με το timestamp του transaction, $ts(T_i)$. Όταν ένα transaction T_i θέλει να κάνει write σε δεδομένο x γίνεται ο ακόλουθος έλεγχος. Εντοπίζεται η έκδοση x_k με το μεγαλύτερο wts το οποίο είναι μικρότερο από $ts(T_i)$. Αν $rts > ts(T_i)$, το write απορρίπτεται και το transaction ξεκινάει από την αρχή. Αυτό γίνεται διότι ένα νεότερο transaction έχει κάνει read μια παλαιότερη έκδοση x_k και έχει βασίσει την εκτέλεσή του σε αυτή την τιμή. Για να μην υπάρχουν ασυνέπειες πρέπει το write να απορριφθεί. Αν $rts \leq ts(T_i)$ το write πραγματοποιείται, δημιουργείται καινούργια έκδοση x_m και τίθενται $wts = rts = ts(T_i)$. Όταν δεν υπάρχει άλλος χώρος για αποθήκευση των versions ή των $[wts, rts]$ διαγράφονται οι παλαιότερες εκδόσεις των αντικειμένων της βάσης.

Καθώς τα reads ποτέ δεν μπλοκάρονται αυτός ο αλγόριθμος συντονισμού είναι ιδανικός για βάσεις δεδομένων που έχουν κυρίως read operations. Σε μία document-oriented βάση δεδομένων, όπως η CouchDB που θα δούμε παρακάτω, βελτιστοποιείται η αποθήκευση αρχείων καθώς τα αρχεία αποθηκεύονται συνεχόμενα στο δίσκο. Όταν γίνεται update ολόκληρα τα αρχεία γράφονται στη βάση συνεχόμενα και δεν υπάρχει ανάγκη διατήρησης συνδέσμων, *links*, που ενώνουν τα κομμάτια του αρχείου, όπως στην περίπτωση των βάσεων που δεν δομούνται σε συνεχόμενες θέσεις στο δίσκο.

4.5 Consistency Models

Σε κατανεμημένα συστήματα, όπως σε συστήματα κατανεμημένων βάσεων δεδομένων, χρησιμοποιούνται *Consistency Models*, τα οποία ορίζουν κανόνες για την σειρά εκτέλεσης των updates στα δεδομένα. Τα κατανεμημένα συστήματα αποθήκευσης δεδομένων δομούνται από κόμβους. Τα δεδομένα αποθηκεύονται σε έναν ή περισσότερους κόμβους και η πρόσβαση σε αυτά γίνεται με reads και writes. Παρακάτω αναλύονται δύο από τα πιο συνηθισμένα μοντέλα συντονισμού των updates, το *strong consistency* και το *eventual consistency*. Για να γίνει πιο κατανοητή η διαφορά των μοντέλων και ο τρόπος που υλοποιούνται χρησιμοποιείται η παρακάτω σημειογραφία.

N – Ο αριθμός των αντιγράφων των δεδομένων που θα διατηρεί η βάση

R – Ο αριθμός των αντιγράφων που θα πρέπει να διαβαστούν για να είναι έγκυρο ένα read

W – Ο αριθμός των αντιγράφων που θα πρέπει να ενημερωθούν πριν ολοκληρωθεί ένα write

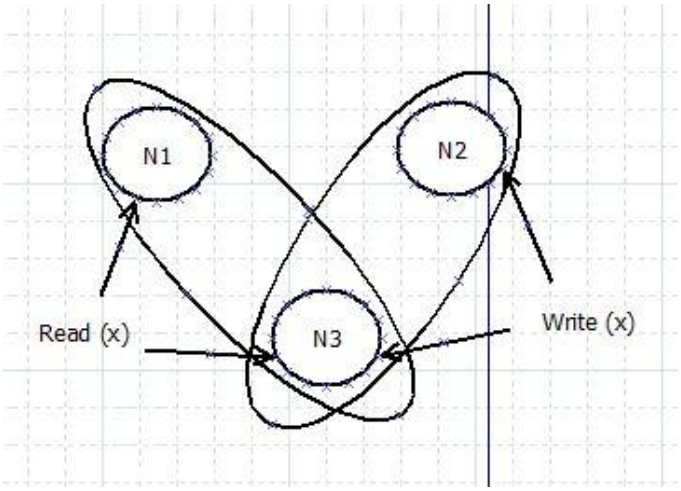
Σημειώνεται πως οι παραπάνω συμβολισμοί δείχνουν πόσα αντίγραφα πρέπει να ενημερωθούν ή διαβαστούν για να θεωρηθεί έγκυρο ένα write και ένα read. Τελικά όλοι οι κόμβοι που έχουν τα δεδομένα θα αποκτήσουν την τελευταία έκδοσή τους. Αυτό επιτυγχάνεται με έναν από τους μηχανισμούς ελέγχου συντονισμού των διεργασιών που περιεγράφηκαν στην προηγούμενη ενότητα.

Η κάθε βάση επιλέγει τέτοια *N, R, W* που να ευνοούν τις πιο κοινές χρήσεις της. Βάσεις που θέλουν υψηλή διαθεσιμότητα στα δεδομένα τους επιλέγουν μεγάλο N και $R = 1$, ώστε ένα μοναδικό read να επιστρέψει αποτέλεσμα. Βάσεις με περισσότερα reads επιλέγουν μικρό R και μεγάλο W. Έτσι εξυπηρετούνται γρήγορα τα reads και τα λιγότερα writes γράφουν σε πολλά αντίγραφα, ώστε τα επόμενα reads να διαβάσουν σίγουρα τη σωστή έκδοση των δεδομένων. Αντίστοιχα βάσεις με περισσότερα writes επιλέγουν μικρό W και μεγάλο R, ώστε να εξυπηρετούνται γρήγορα τα writes και τα λιγότερα reads να διαβάζουν από πολλά αντίγραφα σε κάποιο ή κάποια από τα οποία υπάρχει η τελευταία έκδοση των δεδομένων.

Strong Consistency

Το μοντέλο αυτό διασφαλίζει ότι κάθε χρονική στιγμή όλες οι διεργασίες του κατανεμημένου συστήματος βλέπουν την τελευταία έκδοση των δεδομένων. Οι ενημερώσεις που γίνονται στη βάση είναι άμεσα διαθέσιμες σε κάθε ανάγνωση του δεδομένου, μέχρι να προκύψει καινούργια ενημέρωση. Σε περίπτωση σφάλματος στην επικοινωνία κόμβων η βάση παύει να είναι διαθέσιμη μέχρι την αποκατάσταση του προβλήματος.

Για να εξασφαλιστεί strong consistency πρέπει $W + R > N$. Αυτό σημαίνει πως ο αριθμός των αντιγράφων που γράφονται σε ένα write, *write set*, και ο αριθμός των αντιγράφων που διαβάζονται σε ένα read, *read set*, είναι αρκετά μεγάλος ώστε τα δύο σύνολα να επικαλύπτονται, το οποίο εγγυάται ότι θα διαβαστεί τουλάχιστον ένα αντίγραφο με την τελευταία έκδοση του δεδομένου. Στην Εικόνα 12 παρουσιάζεται ένα παράδειγμα εφαρμογής του strong consistency model με $N = 3$ και $W = R = 2$.



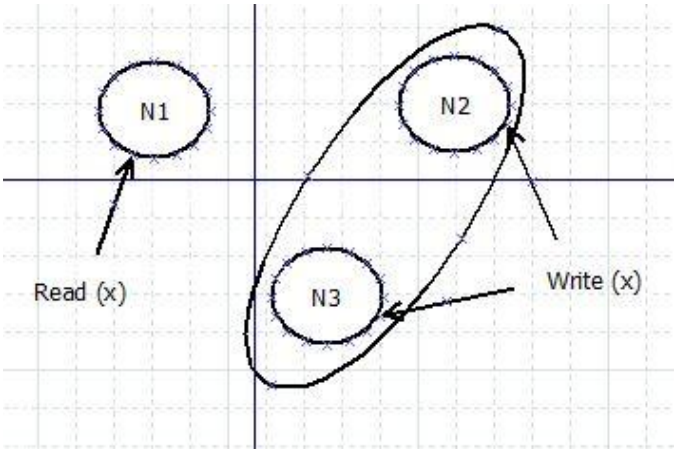
Εικόνα 12 Strong Conistency Model Example

Θεωρούμε το δεδομένο *x* το οποίο διατηρείται σε τρεις κόμβους. Κάθε φορά που γίνεται write η νέα έκδοση του *x* αντιγράφεται σε δύο κόμβους. Κάθε read διαβάζει από δύο κόμβους με τουλάχιστον τον έναν από αυτούς να έχει την τελευταία έκδοση του *x*. Σε περίπτωση σφάλματος σε σύνδεση τα writes και reads δεν επιτυγχάνονται και η βάση γίνεται προσωρινά μη διαθέσιμη.

Eventual Consistency

Σε αυτό το μοντέλο εξασφαλίζεται ότι μετά το πέρας ενός χρονικού διαστήματος στο οποίο δεν γίνονται άλλες ενημερώσεις, θα έχουν γίνει με τη σωστή σειρά όλες οι ενημερώσεις και όλα τα αντίγραφα στη βάση θα είναι συνεπή. Αυτό σημαίνει πως υπάρχει ένα παράθυρο ασυνέπειας, ένα χρονικό διάστημα όπου μπορεί να διαβαστούν παλαιότερες εκδόσεις του αντικειμένου. Αυτό συμβαίνει όταν ο κόμβος από τον οποίο διαβάζονται τα δεδομένα δεν έχει ενημερωθεί για ένα update. Σε περίπτωση αδυναμίας πρόσβασης σε κόμβο είτε λόγω σφαλμάτων στη σύνδεση είτε λόγω αστοχίας hardware η βάση συνεχίζει να είναι διαθέσιμη, με τον κίνδυνο να διαβαστούν παλαιότερες εκδόσεις των δεδομένων.

Σε συστήματα με eventual consistency ισχύει $R + W \leq N$. Αυτό σημαίνει πως ο αριθμός των αντιγράφων που γράφονται σε ένα write, *write set*, και ο αριθμός των αντιγράφων που διαβάζονται σε ένα read, *read set*, δεν εξασφαλίζει την επικάλυψη των δύο συνόλων με αποτέλεσμα να υπάρχει πιθανότητα ένα read να μην διαβάσει την τελευταία έκδοση του δεδομένου. Στην Εικόνα 13 παρουσιάζεται ένα παράδειγμα εφαρμογής του μοντέλου eventual consistency με $N = 3$, $R = 1$ και $W = 2$.



Εικόνα 13 Eventual Conistency Model Example

Θεωρούμε το δεδομένο *x* το οποίο διατηρείται σε τρεις κόμβους. Κάθε φορά που γίνεται write η νέα έκδοση του *x* αντιγράφεται σε δύο κόμβους. Κάθε read διαβάζει από έναν κόμβο. Είναι φανερό από το παραπάνω σχήμα το read να μην γίνει από κόμβο με την νεότερη έκδοση του δεδομένου, το οποίο οδηγεί σε ασυνέπεια στη βάση. Τελικά όλοι οι κόμβοι θα ενημερωθούν αλλά μέχρι τότε υπάρχει ένα παράθυρο ασυνέπειας.

4.6 CAP Theorem

Το 2000 ο Eric Brewer εισήγαγε κάποιες υποθέσεις για την ανάδειξη των trade-offs που πρέπει να γίνουν στην ανάπτυξη ενός κατανεμημένου συστήματος αποθήκευσης δεδομένων. Αργότερα ήρθε η απόδειξη των ισχυρισμών του, οι οποίοι μετονομάστηκαν στο γνωστό CAP Theorem [3] ή αλλιώς Brewer's Theorem. Σύμφωνα με αυτό σε ένα κατανεμημένο σύστημα αποθήκευσης δεδομένων μόνο δύο ιδιότητες από τις *Consistency*, *Availability* και *Partition Tolerance* μπορούν να ικανοποιούνται ταυτόχρονα.

Με το όρο *Consistency* – Συνέπεια εννοούμε την υλοποίηση του μοντέλου strong consistency, όπου όλοι οι κόμβοι του κατανεμημένου συστήματος βλέπουν κάθε χρονική στιγμή τα ίδια δεδομένα. Με τον όρο *Availability – Διαθεσιμότητα* εννοούμε πως οι χρήστες μπορούν κάθε χρονική στιγμή να διαβάσουν και να γράψουν δεδομένα. Με τον όρο *Partition Tolerance* εννοούμε πως το σύστημα έχει ανοχή σε προσωρινά προβλήματα σύνδεσης και επιτρέπει το χωρισμό του κατανεμημένου συστήματα σε τμήματα, τα οποία συνεχίζουν να δουλεύουν κανονικά.

Στις κατανεμημένες βάσεις δεδομένων το Partition Tolerance νομοτελειακά πρέπει να ικανοποιείται. Οι σχεδιαστές των βάσεων καλούνται να επιλέξουν ανάμεσα στο consistency και στο availability. Αν επιλεγεί strong consistency σε περίπτωση προβλήματος στην επικοινωνία των κόμβων, τα δεδομένα δεν είναι διαθέσιμα μέχρι να επιδιορθωθεί το πρόβλημα και η βάση να είναι σε συνεπή μορφή. Αν επιλεγεί availability τα δεδομένα είναι κάθε στιγμή διαθέσιμα, αλλά υπάρχει ο κίνδυνος να μην διαβαστούν οι τελευταίες εκδόσεις των δεδομένων. Τότε υλοποιείται το eventual consistent μοντέλο που αναλύθηκε παραπάνω.

4.7 Είδη NoSQL Βάσεων Δεδομένων

Ανάλογα με τον τρόπο που αναπαριστούν τα δεδομένα μπορούμε να χωρίσουμε τις NoSQL βάσεις σε διάφορες κατηγορίες. Παρακάτω παρουσιάζονται μερικές από αυτές, ενώ στις επόμενες ενότητες θα αναλυθούν διεξοδικά οι NoSQL βάσεις δεδομένων *CouchDB* και *MongoDB*, καθώς αυτές διαθέτουν τα απαιτούμενα από αυτήν την διπλωματική

χαρακτηριστικά.

Document store

Στις document-oriented βάσεις τα δεδομένα αποθηκεύονται σε αρχεία με κάποιο συγκεκριμένο μοτίβο, και όχι σε συσχετιζόμενους πίνακες όπως στις σχεσιακές βάσεις. Τα αρχεία μπορεί να είναι XML, JSON, BSON, YAML, PDF, αρχεία από το Microsoft Office και άλλα. Οι διάφορες υλοποιήσεις δίνουν διαφορετικούς τρόπους οργάνωσης και διαχείρισης των δεδομένων. Τα δεδομένα μπορεί να είναι οργανωμένα σε συλλογές, *collections*, σε ιεραρχίες καταλόγου, *directory hierarchies, non-visible metadata* ή *tags*. Τα non-visible metadata είναι metadata τα οποία δεν συμμετέχουν σε ερωτήματα στη βάση, αλλά είναι χρήσιμα σε τεχνικές data mining.

Οι συλλογές μπορούν να θεωρηθούν όμοιες με τους πίνακες στις σχεσιακές βάσεις και τα αρχεία, *documents*, ως οι εγγραφές. Με αυτόν τον τρόπο επιτυγχάνεται schema-less αναπαράσταση των δεδομένων. Ανάλογα με τις εκάστοτε ανάγκες, documents του ίδιου collection μπορούν να διατηρούν διαφορετικά πεδία. Η προσθήκη ενός νέου πεδίου σε ένα document δεν επηρεάζει τα υπόλοιπα documents της βάσης. Τα αρχεία αντιπροσωπεύονται μέσω ενός μοναδικού κλειδιού, *Universally Unique Identifier – UUID*, κάτι που κάνει σχεδόν αδύνατο δύο documents να έχουν το ίδιο UUID. Αντίθετα στις σχεσιακές βάσεις δεδομένων για τον μονοσήμαντο προσδιορισμό της κάθε εγγραφής σε ένα πίνακα χρησιμοποιούνται τα primary keys μαζί με κάποιο μηχανισμό παραγωγής τους, όπως auto-increment. Ωστόσο σε περιπτώσεις κατανεμημένων σχεσιακών βάσεων, όπου τα δεδομένα δεν αποθηκεύονται σε ένα μοναδικό σημείο, η προσθήκη νέων εγγραφών σε δύο σημεία της βάσης μπορεί να οδηγήσει σε διαφορετικές εγγραφές με το ίδιο primary key. Αυτό εξαλείφεται με τη χρήση του UUID στις document-store βάσεις δεδομένων. Τέλος η βάση μπορεί να παρέχει APIs για την πρόσβαση στα δεδομένα μέσω του περιεχομένου τους.

Παραδείγματα: Apache CouchDB, MongoDB, SimpleDB, Apache Cassandra

Graph Database

AOι Graph Databases σχεδιάστηκαν για δεδομένα που αναπαρίστανται καλύτερα με τη μορφή γράφου, όπως αναπαράσταση κοινωνικών σχέσεων, μεταφορές, χάρτες, τοπολογίες δικτύου και άλλα. Οποιοδήποτε σύστημα αποθήκευση που παρέχει διασύνδεση αντικειμένων μέσω δεικτών μπορεί να χρησιμοποιηθεί σαν graph database. Μια graph database αποτελείται από κόμβους, ακμές και τις ιδιότητες των κόμβων. Οι κόμβοι αναπαριστούν οντότητες – αντικείμενα, οι ιδιότητες αποτελούν τα χαρακτηριστικά και τις πληροφορίες που αφορούν στις οντότητες και οι ακμές τις σχέσεις που τις συνδέουν. Οι πληροφορίες αποθηκεύονται στις ακμές.

Συγκριτικά με τις σχεσιακές βάσεις δεδομένων οι graph databases είναι γρηγορότερες για δεδομένα που συσχετίζονται μεταξύ τους και ταιριάζουν καλύτερα στις αντικειμενοστραφείς φιλοσοφίες. Συνήθως δεν απαιτούν πολύπλοκες συνενώσεις, JOINS, είναι καλύτερες στο scaling και καθώς δεν απαιτούν ένα αυστηρό schema είναι πιο αποδοτικές για δεδομένα που αλλάζει συνεχώς το schema τους.

Παραδείγματα: Neo4J, Infinite Graph

Key–value store

Το key-value store είναι ο τρόπος αποθήκευσης των δεδομένων σε μορφή ζευγαριών key-value. Στο value αποθηκεύονται τα δεδομένα, είτε σε μορφή αντικειμένων είτε σε οποιοδήποτε άλλο τύπο δεδομένων, και το key είναι το κλειδί που ανατίθεται στο value. Η αναζήτηση των δεδομένων γίνεται σε πολλά συστήματα με αυτό το κλειδί. Οι key-value store βάσεις επιτρέπουν στις εφαρμογές να αποθηκεύουν τα δεδομένα με έναν schema-less τρόπο.

Παραδείγματα: Apache Cassandra, BigTable, MongoDB, Redis

Object Database

Σε αυτόν τον τύπο βάσης δεδομένων τα δεδομένα αποθηκεύονται σε μορφή αντικειμένων, όπως αυτά ορίζονται στον αντικειμενοστραφή προγραμματισμό. Οι περισσότερες object databases παρέχουν και μία γλώσσα υποβολής ερωτημάτων, *Object Query Language – OQL* . Με αυτόν τον τρόπο η ανάκτηση των αντικειμένων γίνεται με ένα πιο προστακτικό τρόπο.

Παραδείγματα: db4o, Versant Object Database

Column-oriented Database

Στις column-oriented βάσεις δεδομένων τα δεδομένα στους πίνακες αποθηκεύονται κατά στήλες, σε αντίθεση με τις σχεσιακές βάσεις δεδομένων όπου τα δεδομένα αποθηκεύονται κατά γραμμές. Αυτό είναι αποδοτικό σε συστήματα online analytical processing (OLAP), όπου απαιτούνται συνενώσεις μεγάλου όγκου παρόμοιων δεδομένων, όπως σε αποθήκες δεδομένων (data warehouses), συστήματα διαχείρισης πελατειακών σχέσεων (Customer relationship management (CRM)) και παρόμοια συστήματα, όπου απαιτούνται λιγότερα πολύπλοκα ερωτήματα.

4.8 CouchDB

Η CouchDB είναι μια ανοιχτού κώδικα NoSQL, document-oriented βάση δεδομένων και αποτελεί Project του Apache Foundation. Είναι γραμμένη στη συναρτησιακή γλώσσα προγραμματισμού Erlang, η οποία επιλέχθηκε επειδή διαθέτει μηχανισμούς συντονισμού, κατανομής των δεδομένων και διαχείρισης των σφαλμάτων. Με χρήση της Erlang ενισχύεται η αξιοπιστία και το scalability των εφαρμογών.

4.8.1 Αρχιτεκτονική

Η CouchDB χρησιμοποιεί μηχανισμό optimistic replication [4] στα αντίγραφα των δεδομένων που διατηρούνται στη βάση και εγγυάται eventual consistency σε αυτά. Η επικοινωνία με τη CouchDB και η πρόσβαση στα δεδομένα γίνεται μέσω ενός RESTful JSON API. Η αποθήκευση στην βάση γίνεται σε B-trees, μία δεντρική δομή που επιτρέπει την προσθήκη, αφαίρεση, αλλαγή και αναζήτηση δεδομένων σε λογαριθμικό χρόνο. Αυτός ο μηχανισμός αποθήκευσης χρησιμοποιείται για τα *documents* και τα *views*. Στις σχεσιακές βάσεις δεδομένων χρησιμοποιούνται συχνά μηχανισμοί locking για την εξασφάλιση της συνέπειας της βάσης. Στην CouchDB για τον συντονισμό των transactions δεν

χρησιμοποιούνται locks, αλλά το Multiversioning Concurrency Protocol (MVCC), που παρουσιάστηκε σε προηγούμενη ενότητα. Κάθε χρήστης διαθέτει ένα snapshot της τελευταίας έκδοσης της βάσης και οι αλλαγές που κάνει δεν είναι προσβάσιμες από τους άλλους χρήστες μέχρι να γίνουν commit. Σε περίπτωση conflict η εφαρμογή επιλέγει ποια έκδοση της βάσης είναι η σωστή. Ένα ακόμα χαρακτηριστικό της αρχιτεκτονικής της CouchDB αποτελεί η εφαρμογή μιας τεχνικής συμπίεσης, *compaction*, για την ανάκτηση αναξιοποίητου χώρου. Σε περιοδικό χρόνο ή όταν εντοπιστεί ορισμένος αναξιοποίητος χώρος, τα active data αντιγράφονται σε ένα νέο αρχείο, μεταφέρονται οι χρήστες που τα χρησιμοποιούσαν σε αυτό και το παλιό αρχείο διαγράφεται. Όσο διαρκεί αυτή η διαδικασία δεν επιτρέπονται reads και updates στο αρχείο.

4.8.2 Documents

Στην CouchDB τα δεδομένα αποθηκεύονται σε documents τύπου JSON με schema-free τρόπο. Όπως έχει αναφερθεί τα documents διαθέτουν πεδία, οι τιμές των οποίων μπορεί να είναι strings, αριθμοί, ημερομηνίες, λίστες και πίνακες. Στις σχεσιακές βάσεις δεδομένων κάθε εγγραφή δεσμεύει χώρο υποχρεωτικά για όλα τα πεδία του πίνακα ακόμα και για εκείνα που δεν έχει τιμή. Στην CouchDB κάθε document μπορεί να περιέχει διαφορετικά πεδία ανάλογα με τις ανάγκες. Αυτό κάνει ευέλικτη την αποθήκευση δεδομένων, καθώς δεν αποθηκεύονται κενά πεδία. Ο μόνος περιορισμός είναι πως δεν επιτρέπονται στο ίδιο document δύο ή περισσότερα πεδία με το ίδιο όνομα. Τα documents περιέχουν και metadata, δηλαδή πληροφορίες για το document και τα δεδομένα που περιέχει. Σε αυτά περιλαμβάνονται το *_id*, που είναι το UUID του document, και το *_rev*, που είναι το id του revision του. Τα documents αποθηκεύονται στα B-trees με βάση το *_id* και το *_rev*. Ένα παράδειγμα document παρουσιάζεται στην Εικόνα 14.

```
{
    "_id" : "document_id",
    "_rev" : "revision_id",
    "name" : "Jack",
    "languages" : [ "English", "German", "French" ]
}
```

Εικόνα 14 CouchDB JSON Document

Κάθε φορά που οι χρήστες τροποποιούν ένα document, οι αλλαγές δεν εφαρμόζονται στο αρχικό, αλλά δημιουργείται ένα revision. Στη βάση παραμένουν όλα τα revisions ενός document. Ποιο revision ενός document επεξεργάζεται ο χρήστης φαίνεται από το *_rev*. Για να τροποποιηθεί ένα document οι χρήστες το φορτώνουν, εφαρμόζουν τις αλλαγές και το σώζουν πίσω στη βάση. Αν στο ενδιάμεσο χρονικό διάστημα ένας άλλος χρήστης έχει υποβάλλει αλλαγές στο ίδιο document, εμφανίζεται ένα μήνυμα conflict. Ο πρώτος χρήστης μπορεί να επαναφορτώσει το document, να ξανακάνει τις αλλαγές και να προσπαθήσει να το ξανακαταχωρήσει στη βάση. Η συνέπεια της βάσης διατηρείται με την εξασφάλιση ότι τα updates σε ένα document είτε αποθηκεύονται όλα είτε κανένα. Δεν μένουν στη βάση documents μερικώς updated.

4.8.3 Views

Καθώς η βάση είναι δομημένη με schema-free τρόπο είναι απαραίτητη η ύπαρξη ενός μηχανισμού δομημένης αναπαράστασης των δεδομένων για το συμπερασμό των σχέσεων

μεταξύ των documents, την ομαδοποίησή τους και την παρουσίαση τους μετά την υποβολή ερωτημάτων. Για αυτό το λόγο η CouchDB χρησιμοποιεί τα views, τον μηχανισμό ενοποίησης και παρουσίασης των documents με τρόπο κατανοητό στους χρήστες.

Τα views παράγονται δυναμικά και ανάλογα με τις ανάγκες μπορούν να δημιουργηθούν διαφορετικά views των ίδιων δεδομένων. Τα views αποτελούν μια αναπαράσταση των δεδομένων του instance της βάσης που βλέπει ο χρήστης και δεν επηρεάζουν άμεσα τα documents που είναι αποθηκευμένα στη βάση. Τα views ορίζονται σε ειδικά αρχεία, τα *design documents*, που αντιγράφονται στα instances των βάσεων όπως και τα υπόλοιπα documents. Έτσι εφαρμογές που αποτελούνται από documents και design documents μπορούν επίσης να γίνονται replicated στην CouchDB.

Τα αποτελέσματα των views υπολογίζονται με JavaScript συναρτήσεις που υλοποιούν το MapReduce και συγκεκριμένα το map κομμάτι του. Το view εφαρμόζει τη συνάρτηση map στα documents και επιστρέφει τις πληροφορίες εκείνες που τελικά θα συμπεριληφθούν στο view και θα εμφανιστούν στο χρήστη. Τα key/value ζευγάρια που παράγονται κατά την δημιουργία των views ταξινομούνται σε ένα B-tree με βάση το key τους. Με αυτόν τον τρόπο τα δεδομένα μπορούν να κατανεμηθούν σε πολλούς κόμβους χωρίς αλλοίωση των αποτελεσμάτων των queries και μπορούν να αξιοποιηθούν οι μηχανισμοί των B-trees για γρήγορη αναζήτηση με βάση το key. Για τη γρήγορη εμφάνιση των views η CouchDB χρησιμοποιεί indexes, τα οποία ανανεώνονται με κάθε προσθήκη, αφαίρεση ή τροποποίηση των documents.

4.8.4 Replication

Η CouchDB είναι μια peer-based κατανεμημένη βάση. Παρέχει μηχανισμούς optimistic replication, δηλαδή τα αντίγραφα ενός δεδομένου εγγυημένα έχουν όλα την ίδια τιμή μετά το πέρας κάποιου χρόνου. Αυτός ο μηχανισμός εφαρμόζεται σε συνδυασμό με το eventual consistency, που ενισχύει την διαθεσιμότητα των δεδομένων, τον συντονισμό και την παραλληλοποίηση των υπολογισμών. Το optimistic replication εφαρμόζεται στους servers και στους χρήστες, δίνοντάς τους τη δυνατότητα να κάνουν update στα ίδια μοιραζόμενα δεδομένα ενώ είναι αποσυνδεδεμένοι και αφού συνδεθούν αμφίδρομα να ενημερώσουν για τις αλλαγές, μηχανισμός που ονομάζεται *bi-directional replication*. Καθώς το replication αφορά σε δεδομένα και εφαρμογές, το bi-directional replication είναι μεγάλης σημασίας χαρακτηριστικό για χρήστες που επιθυμούν να δουλέψουν ενώ δεν είναι συνδεδεμένοι στη βάση ή σε περιπτώσεις που οι servers στεγάζονται σε χώρους με αμφίβολης ποιότητας σύνδεση στο διαδίκτυο. Η διαδικασία παραγωγής αντιγράφων είναι incremental. Κάθε φορά ελέγχονται μόνο τα updated documents και αυτά γίνονται replicate στη βάση. Τέλος αναφέρουμε τα χαρακτηριστικά που παρέχει η CouchDB μέσα από το replication framework που προσφέρει.

- master-master replication
- master-slave replication
- filtered replication
- incremental and bi-directional replication
- conflict management

4.9 MongoDB

Η MongoDB ανήκει στην οικογένεια των document-oriented NoSQL βάσεων δεδομένων

και αναπτύχθηκε από την εταιρεία 10gen. Πρόκειται για μία βάση ανοιχτού κώδικα, διαθέσιμη με τους όρους χρήσης του GNU Affero General Public License. Είναι σχεδιασμένη να προσφέρει υψηλή απόδοση στις εφαρμογές, επεκτασιμότητα, υψηλή διαθεσιμότητα και δυνατότητα υποβολής σύνθετων ερωτημάτων. Για τα δεδομένα της εγγυάται eventual consistency. Αποτελεί το πιο δημοφιλές σύστημα διαχείρισης NoSQL βάσεων δεδομένων και είναι γραμμένη στην γλώσσα προγραμματισμού C++.

4.9.1 Αρχιτεκτονική

Με το σύστημα διαχείρισης δεδομένων MongoDB είναι δυνατή η ανάπτυξη κατανεμημένων εφαρμογών που τρέχουν σε clusters υπολογιστών, όπου με την προσθήκη νέων κόμβων πραγματοποιείται αυτόματη κατανομή του φόρτου εργασίας. Στο cluster διακρίνονται τρεις τύποι κόμβων, οι *shards*, οι *configuration servers* και οι *routers*. Αυτή η δομή των clusters είναι δυνατή λόγω του *sharding* που εφαρμόζεται στα δεδομένα και το οποίο θα περιγραφεί σε επόμενη ενότητα. Για την κατανόηση της αρχιτεκτονικής που ακολουθεί η MongoDB δίνεται σχηματικά στην Εικόνα 15 ένα MongoDB cluster.

Shard Nodes

Οι shard κόμβοι είναι υπεύθυνοι για την αποθήκευση των δεδομένων. Κάθε shard αποτελείται από ένα *replica set*, έναν ή περισσότερους κόμβους που διατηρούν αντίγραφα των ίδιων δεδομένων. Από τους κόμβους που δομούν το shard, ένας θεωρείται *primary*, πρωτεύων, και οι υπόλοιποι *secondary*, δευτερεύοντες. Σε περίπτωση αποτυχίας του primary ένας από τους secondary αναλαμβάνει ως primary. Τα writes και τα consistent reads δρομολογούνται στον primary κόμβο, ενώ τα eventual consistent reads κατανέμονται στους secondary. Η MongoDB προκειμένου να είναι γρήγορη και ευέλικτη χρησιμοποιεί απλούς μηχανισμούς locking για την εξυπηρέτηση των ταυτόχρονων reads και writes από πολλούς clients και όχι πολύπλοκους μηχανισμούς όπως το MVCC. Το λογισμικό που χρησιμοποιούν οι shards ονομάζεται *mongod*.

Configuration servers

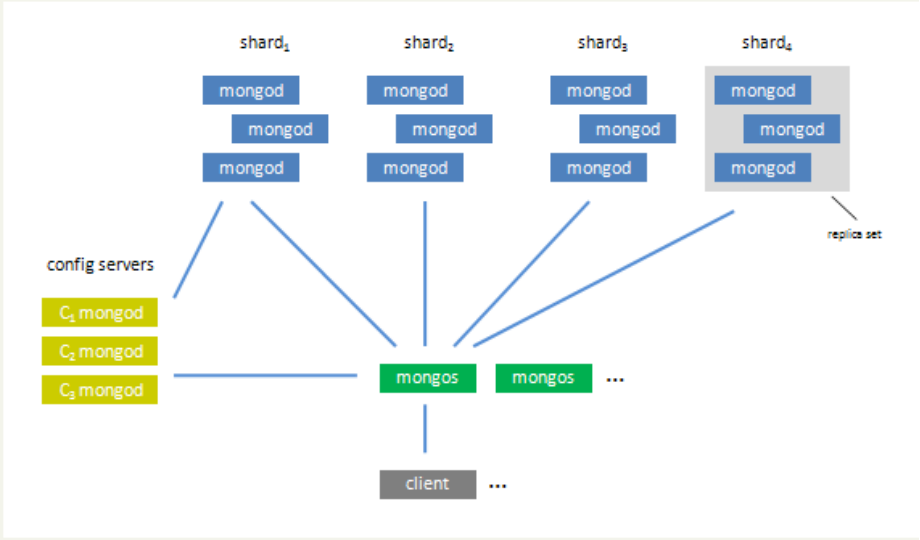
Οι configuration servers αποθηκεύουν metadata και πληροφορίες δρομολόγησης που υποδεικνύουν ποια δεδομένα διατηρούνται στο κάθε shard. Η πρόσβαση σε αυτούς γίνεται μέσω των shards, οι οποίοι ενημερώνουν ποια δεδομένα διαθέτουν, και μέσω των routers, οι οποίοι ζητούν πληροφορίες δρομολόγησης των requests που υποβάλλουν οι χρήστες. Το λογισμικό που χρησιμοποιούν οι configuration servers ονομάζεται *mongod*.

Routers

Οι routers χρησιμοποιούνται στην επικοινωνία ενός ή περισσότερων clients με τη βάση. Δέχονται requests για reads και writes από τους χρήστες με τη μορφή queries και updates, συμβουλεύονται τους configuration servers για το ποιο shard έχει τα ζητούμενα δεδομένα και δρομολογούν τις εργασίες στα αντίστοιχα shards. Αφού ολοκληρωθούν τα requests, οι routers ομαδοποιούν τα αποτελέσματα και τα στέλνουν πίσω στο χρήστη. Το λογισμικό που χρησιμοποιούν οι routers ονομάζεται *mongos*.

Clients

Με τον όρο clients εννοείται ολόκληρη η εφαρμογή ή κομμάτι της. Η επικοινωνία τους με τη βάση γίνεται μέσω του κατάλληλου *mongo driver*, ο οποίος επιλέγεται ανάλογα με τη γλώσσα που είναι γραμμένη η εφαρμογή.



Εικόνα 15 MongoDB Cluster

4.9.2 Storage

Σε αυτήν την ενότητα θα παρουσιαστούν τα εργαλεία που χρησιμοποιεί η MongoDB για την αποθήκευση και ανάκτηση των δεδομένων. Γνωρίζοντας τον τρόπο αποθήκευσης των δεδομένων γίνονται σαφέστερες οι ανάγκες των εφαρμογών σε αποθηκευτικά μέσα, RAM και δίσκο. Επίσης είναι απαραίτητη η κατανόηση της αξιοπιστίας και των μηχανισμών ασφάλειας που εφαρμόζονται στα δεδομένα, όπως πότε έχει επιστρέψει ένα write, για την προσαρμογή τους στις εκάστοτε ανάγκες των χρηστών. Τέλος καθώς η πρόσβαση στο δίσκο, δηλαδή τα I/O, αποτελούν bottleneck για τις περισσότερες εφαρμογές, η αξιοποίηση των παρακάτω μηχανισμών μπορεί να οδηγήσει σε αποδοτικότερες υπηρεσίες. Θα αναλυθούν οι διαδικασίες αποθήκευσης των δεδομένων, ο τρόπος αποθήκευσής τους και τεχνικές για τον γρήγορο εντοπισμό και την ανάκτησή τους.

Preallocate files

Σε πολλές περιπτώσεις ένας από τους λόγους χαμηλής επίδοσης των υπηρεσιών είναι τα I/O, η επικοινωνία της εφαρμογής με το δίσκο. Για την αντιμετώπιση αυτού του προβλήματος η MongoDB χρησιμοποιεί *pre-allocation*. Κατά το pre-allocation όταν ένα αρχείο φτάσει ένα ορισμένο μέγεθος, δεσμεύεται αυτόματα χώρος για το επόμενο, πριν η εφαρμογή ζητήσει την δημιουργία του. Το αρχείο που δημιουργείται δεν περιέχει δεδομένα.

Χωρίς pre-allocation τα αρχεία θα δέσμευαν δυναμικά το χώρο που χρειάζονταν, με τον κίνδυνο να αυξήσουν το μέγεθός τους ανεξέλεγκτα. Κατά την δυναμική δέσμευση μνήμης υπάρχει περίπτωση το αρχείο να μην αποθηκευτεί ολόκληρο σε διαδοχικές θέσεις μνήμης, αλλά σε κομμάτια, *fragments*, προκαλώντας *filesystem fragmentation*. Γενικά αυτό δεν είναι επιθυμητό, καθώς σε μια τέτοια περίπτωση θα πρέπει να διατηρούνται πληροφορίες για τις θέσεις που είναι αποθηκευμένες το αρχείο, με αποτέλεσμα η πρόσβαση σε αυτό να είναι πιο αργή και να μειώνεται η συνολική επίδοση των εφαρμογών.

Τα αρχεία που δημιουργούνται με pre-allocation έχουν εκ των προτέρων γνωστό και σταθερό μέγεθος. Έτσι αποφεύγεται η ανεξέλεγκτη αύξηση του μεγέθους των αρχείων και το filesystem fragmentation που προκαλεί. Τα αρχεία αποθηκεύονται σε διαδοχικές θέσεις μνήμης, κάνοντας πιο εύκολη την πρόσβαση σε αυτά και αυξάνοντας την απόδοση των εφαρμογών. Όταν μια εφαρμογή χρειαστεί να κάνει write δεδομένα σε αρχείο που δεν έχει χώρο να τα αποθηκεύσει, πρέπει να ζητηθεί η δημιουργία νέου αρχείου και στη συνέχεια να γίνει write σε αυτό. Αυτή η διαδικασία είναι χρονοβόρα και δημιουργεί activity και latency spikes, καθώς απαιτείται την ίδια χρονική στιγμή η δημιουργία του καθώς και η αποθήκευση των αρχείων. Με pre-allocation πριν χρειαστεί να γίνει write, το αρχείο έχει ήδη δημιουργηθεί, μειώνοντας έτσι την μεγάλη διακύμανση επεξεργαστικής ισχύς. Αυτό έχει ως αποτέλεσμα τα writes να γίνονται άμεσα, χωρίς αναμονή για την δημιουργία του αρχείου.

Για κάθε βάση δεδομένων η MongoDB ξεκινά το pre-allocation με ένα αρχείο μεγέθους 64MB. Κάθε φορά που απαιτείται νέο pre-allocation το μέγεθος του αρχείου διπλασιάζεται μέχρι να φτάσει τα 2GB. Είναι φανερό πως κάθε βάση στη MongoDB μπορεί να έχει αχρησιμοποίητο χώρο μέχρι 2GB. Η πράξη έχει δείξει ότι στις περισσότερες εφαρμογές, παρά το pre-allocation αρχείου 2GB, ο χώρος που μένει αναξιοποίητος είναι ελάχιστος.

Padding

Η MongoDB διαθέτει ένα μηχανισμό εφαρμογής *atomic upserts* στα documents. Τα upserts αποτελούν ένα flag των update operations, που μεταβάλλουν την συμπεριφορά τους από απλή ενημέρωση των documents σε εισαγωγή νέων δεδομένων. Πιο συγκεκριμένα οι χρήστες ορίζουν το query με βάση το οποίο θα αναζητηθούν documents στα οποία θα εκτελεστεί το ορισμένο update. Αν βρεθεί κάποιο document που ταιριάζει με τα κριτήρια που έχουν οριστεί στο query, το update εκτελείται, ενώ αν δεν βρεθεί εκτελείται insert με δεδομένα αυτά που έχουν οριστεί στο query.

Αυτή η λειτουργία, παρότι χρήσιμη για πολλές εφαρμογές, μπορεί να οδηγήσει σε προβλήματα στην απόδοση και το storage. Αν ένα upsert οδηγήσει σε αύξηση του μεγέθους του αρχείου, μπορεί να χρειαστεί η μεταφορά του σε σημείο του δίσκου που να μπορεί να αποθηκευτεί σε συνεχόμενες θέσεις στη μνήμη, ενώ παράλληλα θα πρέπει να ενημερωθούν τα indexes και τα shards, διαδικασία ιδιαίτερα χρονοβόρα. Για αυτό το λόγο η MongoDB εκτελεί ευριστικούς αλγορίθμους για τον εντοπισμό των documents που μπορεί να αυξηθούν σε μέγεθος πιο συχνά. Σε αυτά τα documents εφαρμόζεται *padding*, δηλαδή προσθήκη χώρου στο τέλος του document, δίνοντάς του τη δυνατότητα να μεγαλώσει λίγο σε μέγεθος χωρίς να χρειαστεί η μεταφορά του. Πόσος χώρος δίνεται ελέγχεται από τον *paddingFactor*, ο οποίος αυξάνεται όταν εμπειρικά εντοπιστούν από τη MongoDB collections, τα documents των οποίων τείνουν να αυξάνουν σε μέγεθος.

Memory mapped files

Για την διαχείριση των δεδομένων η MongoDB χρησιμοποιεί memory-mapped files. Κατά το memory mapping η MongoDB ζητά από το λειτουργικό, και συγκεκριμένα από τον *Virtual Memory Manager* του λειτουργικού συστήματος, να κάνει map όλα τα αρχεία από το δίσκο στη RAM του μηχανήματος που τα διατηρεί. Τα αρχεία αντιστοιχίζονται σε blocks εικονικής μνήμης και είναι δυνατός ο χειρισμός τους σαν να ήταν αποθηκευμένα σε φυσική μνήμη. Ο Virtual Memory Manager αποφασίζει ποια δεδομένα θα διατηρούνται στην RAM και ποια στο δίσκο.

Η πρόσβαση στα δεδομένα, τα reads και τα writes γίνονται, μέσω buffers της RAM, στις εικόνες των δεδομένων που έχουν προκύψει από το memory mapping. Το λειτουργικό σύστημα είναι υπεύθυνο για την μεταφορά των αρχείων που βρίσκονται στη RAM στην cache του filesystem και την εκτέλεση των reads και writes απευθείας από εκεί. Η MongoDB

περιοδικά συγχρονίζεται με το δίσκο, κάνοντας flush τα writes που βρίσκονται στους buffers των RAM. Παρότι με χρήση των memory-mapped files η MongoDB δεν μπορεί να ελέγξει πότε τα δεδομένα αποθηκεύονται στο δίσκο, αξιοποιείται στο μεγαλύτερο δυνατό βαθμό η χρήση μνήμης και ενισχύεται η επίδοση των εφαρμογών.

Με χρήση memory-mapped files δεν απαιτείται ανάπτυξη πολύπλοκου κώδικα από την MongoDB για την μεταφορά των δεδομένων από τη RAM στο δίσκο, καθώς το λειτουργικό σύστημα έχει ήδη υλοποιήσει το memory mapping αποδοτικά για όλους τους τύπους δεδομένων. Ωστόσο παρατηρούνται και μειονεκτήματα χρήσης του, όπως λόγω του RAM *read-ahead*. Κατά το read-ahead όταν ζητείται η μεταφορά δεδομένων από το δίσκο στην RAM, δεν αντιγράφονται μόνο οι ζητούμενες pages του filesystem, αλλά και κάποιες επόμενες γειτονικές. Αυτό πραγματοποιείται για λόγους απόδοσης, καθώς η RAM θεωρεί πως τα επιπλέον δεδομένα που αποθηκεύει θα ζητηθούν αργότερα και για γρηγορότερη απόκριση του συστήματος τα μεταφέρει νωρίτερα. Το RAM read-ahead έχει σαν αποτέλεσμα το γέμισμα της RAM με δεδομένα που δεν χρειάζονται. Αυτό οδηγεί στην εμφάνιση page faults όταν αναζητηθούν στην RAM δεδομένα που δεν υπάρχουν, επειδή απομακρύνθηκαν λόγω read-ahead. Έτσι απαιτείται νέα μεταφορά δεδομένων, κάτι που προκαλεί προβλήματα στην επίδοση των εφαρμογών. Ένα ακόμα μειονέκτημα χρήσης memory-mapped αρχείων είναι το RAM fragmentation που οφείλεται στο ακριβές mapping του fragmented δίσκου στη RAM. Όπως έχει παρουσιαστεί παραπάνω το fragmentations γενικά δεν είναι επιθυμητό, καθώς οδηγεί σε μειωμένες επιδόσεις στις εφαρμογές λόγω της μη αποδοτικής χρήσης του αποθηκευτικού μέσου.

Journal

Για την εξασφάλιση συνέπειας στη βάση χρησιμοποιείται *write ahead logging* σε ένα journal. Τα write operations γράφονται πρώτα στο journal και στη συνέχεια εφαρμόζονται περιοδικά στα δεδομένα. Αν κάθε φορά που γινόταν ένα write αυτό γραφόταν αμέσως από το journal στο δίσκο η επίδοση της βάσης θα μειωνόταν σημαντικά. Για αυτό το λόγο το journal κάνει group τα updates και τα προωθεί μαζικά και περιοδικά στο δίσκο. Οι χρήστες έχουν την δυνατότητα να ορίσουν κάθε πότε τα write operations γράφονται στο δίσκο. Από την MongoDB διατίθεται επίσης εργαλεία για το monitor του journal. Όταν τα write operations εφαρμοστούν στα δεδομένα, τα journal files διαγράφονται. Διαγράφονται επίσης όταν πραγματοποιηθεί shutdown της MongoDB, καθώς αυτό επιτυγχάνει μόνο όταν εφαρμοστούν όλα τα updates των journals στη βάση.

Σε περίπτωση αστοχίας υλικού, η βάση συμβουλεύεται το journal, το οποίο δεν καταστρέφεται, και επανεκτελεί τα write operations στα δεδομένα. Με αυτόν τον τρόπο εξασφαλίζεται η συνέπεια της βάσης. Παρότι είναι δυνατή η απενεργοποίηση του journaling, αυτό αποθαρρύνεται, καθώς σε περίπτωση αστοχίας οι εφαρμογές δεν έχουν τρόπο να επιβεβαιώσουν την συνέπεια της βάσης και πρέπει να εφαρμόσουν άλλους μηχανισμούς επαναφοράς της σε συνεπή μορφή.

Όταν ξεκινάει το journaling πραγματοποιείται pre-allocation ενός journal file, το οποίο δεσμεύει 1GB αποθηκευτικού χώρου. Κατά τη διάρκεια του pre-allocation δεν είναι δυνατή η επικοινωνία της εφαρμογής με τη βάση. Όταν εξαντληθεί ο διαθέσιμος χώρος ενός journal file, δημιουργείται νέο κάνοντας πάλι pre-allocation 1GB. Οι περισσότερες εφαρμογές διαθέτουν δύο ή τρία journal files, άρα για τη διατήρησή τους χρησιμοποιούνται 2 με 3GB αποθηκευτικού χώρου. Στην περίπτωση που συναντώνται πολλές databases με μικρό όγκο δεδομένων η κάθε μία, δίνεται η δυνατότητα της χρήσης journal files με μέγεθος 128MB, που ονομάζονται *smallfiles*. Ωστόσο τα μικρά σε μέγεθος journal files μπορεί να οδηγήσουν στην δημιουργία πάρα πολλών μικρών αρχείων, που μπορεί να επηρεάσει αρνητικά την επίδοση των εφαρμογών και για αυτό πρέπει να χρησιμοποιούνται με προσοχή.

4.9.3 Data Model

Στην MongoDB τα δεδομένα αποθηκεύονται με schema-free τρόπο σε αρχεία τύπου BSON (Binary JSON), τα οποία ονομάζονται documents. Κάθε document αντιπροσωπεύει μια εγγραφή στη ορολογία των σχεσιακών βάσεων δεδομένων και περιέχει πεδία σε μορφή key/value pairs. Στο key αποθηκεύεται σε ένα string το όνομα του πεδίου, και στο value μπορούν να αποθηκευτούν τύποι δεδομένων που υποστηρίζονται από JSON αρχεία καθώς και άλλοι, όπως Date. Τα documents έχουν μέγιστο μέγεθος 16MB, ενώ για αποθήκευση μεγαλύτερων σε μέγεθος δεδομένων, όπως εικόνες, χρησιμοποιείται το GridFS. Δίνεται ακόμα η δυνατότητα της embedded, ενσωματωμένης, αποθήκευσης Objects και Arrays στο εσωτερικό των documents κάτι που μειώνει την ανάγκη χρήσης joins στην πλευρά του χρήστη. Ένα σύνολο από documents ορίζει μία collection, που είναι το ανάλογο των πινάκων στις σχεσιακές βάσεις δεδομένων. Ένα σύνολο από collections συνιστά μία βάση δεδομένων. Η MongoDB ως σύστημα διαχείρισης διατηρεί ένα σύνολο από βάσεις δεδομένων.

Κάθε document υποχρεωτικά πρέπει να διαθέτει ένα πεδίο όπου αποθηκεύεται το μοναδικό id του document και χρησιμοποιείται σαν primary key για τον εντοπισμό του. Το key αυτού του πεδίου ονομάζεται *_id* και το value είναι τύπου *ObjectID*, ή οποιοσδήποτε άλλου τύπου εκτός από array, και έχει μοναδική τιμή για κάθε document. Σε όλες τις βάσεις δημιουργείται αυτόματα index με βάσει το *_id*. Ένα παράδειγμα ενός BSON αρχείου με ένα embedded document με πεδία *type* και *number* παρουσιάζεται στην Εικόνα 16.

```
{
    "_id" : ObjectID("document_id"),
    "name" : "Jack",
    "birth" : new Date("Apr 25, 1995"),
    "phone" : {
                "type" : "mobile",
                "number" : "some_number"
              },
    "languages" : ["English", "German", "French"]
}
```

Εικόνα 16 MongoDB BSON Document

Η MongoDB προσφέρει μεγάλες ελευθερίες στον τρόπο αποθήκευσης των δεδομένων και δίνει στους προγραμματιστές τη δυνατότητα να εκμεταλλευτούν τις ιδιαιτερότητες της εφαρμογής τους, να δημιουργήσουν ανάλογα μοντέλα αναπαράστασης των δεδομένων και να παρέχουν αποδοτικότερες υπηρεσίες. Καθώς η βάση δομείται με schema-free τρόπο, τα documents μιας collection δεν χρειάζεται να έχουν τα ίδια πεδία και δομή και δεν απαιτείται να διαθέτουν τους ίδιους τύπους δεδομένων, μπορεί δηλαδή το κάθε ένα να διαθέτει το δικό του schema. Η ευελιξία στο schema επιτρέπει την μοντελοποίηση των documents ώστε να αναπαριστούν καλύτερα τα objects της εφαρμογής και τις σχέσεις μεταξύ τους.

Για την αποτελεσματικότερη εκτέλεση των εφαρμογών, και καθώς δεν υποστηρίζονται joins στην MongoDB, για την επιλογή του κατάλληλου schema από τους σχεδιαστές πρέπει να λαμβάνονται υπόψιν τα queries που θα υποβάλλονται, η συχνότητα των updates, των reads και των writes, οι MapReduce operations και ο ρυθμός αύξησης του μεγέθους των documents. Είναι φανερό ότι η μοντελοποίηση των δεδομένων πρέπει να επιλέγεται με βάση τον τρόπο χρήσης τους, και όχι με βάση τον τρόπο αποθήκευσής τους όπως γίνεται στις σχεσιακές βάσεις δεδομένων, όπου η εκτέλεση των queries είναι κυρίως βασισμένη στα joins.

Για την εξαγωγή αποτελεσμάτων μέσω queries είναι απαραίτητη η αναπαράσταση των σχέσεων μεταξύ των documents. Η MongoDB δίνει δύο τρόπους συσχέτισης των δεδομένων, τη χρήση *Embedded Objects* και *Arrays* και τη χρήση *References*.

Embedded documents

Η τεχνική της embedded αποθήκευσης Object και Arrays εντός των documents χρησιμοποιείται όταν τα ενσωματωμένα δεδομένα εξετάζονται κυρίως σε συνδυασμό με τα documents που τα περιέχουν. Καθώς η επεξεργασία πληροφοριών του ίδιου document απαιτεί λιγότερο φόρτο εργασίας στους κόμβους συγκριτικά με τη συλλογή δεδομένων από διαφορετικά documents, στις περιπτώσεις που η αποθήκευση συσχετιζόμενων πληροφοριών στο ίδιο document είναι δυνατή, προτιμάται η embedded αποθήκευση documents. Τα embedded objects υλοποιούν το ανάλογο των joins για τις σχεσιακές βάσεις δεδομένων και λειτουργούν σαν pre-joined δεδομένα κάνοντας τις document operations πιο εύκολες επεξεργαστικά. Μειονέκτημα αυτής της μεθόδου αποτελεί το γεγονός ότι εύκολα αυξάνεται ανεξέλεγκτα το μέγεθος των documents που περιέχουν ενσωματωμένα άλλα objects.

References

Στις περιπτώσεις που το παραπάνω μοντέλο δεν βολεύει τους σχεδιαστές, η MongoDB δίνει την εναλλακτική των references, δηλαδή της διατήρησης των *_id* κάποιων documents σε άλλα για αναπαράσταση των μεταξύ τους σχέσεως. Τα documents που θα συσχετιστούν μπορεί να ανήκουν σε διαφορετικές collections, ακόμα και σε διαφορετικές βάσεις. Τα references δουλεύουν σαν τα foreign keys των σχεσιακών βάσεων και στόχο έχουν τον ορισμό των σχέσεων μεταξύ των documents. Με αυτή την τεχνική δίνεται ευελιξία στον ορισμό των σχέσεων των documents και μπορεί να ελεγχθεί καλύτερα το μέγεθός τους σε αντίθεση με τη χρήση embedded documents. Ωστόσο χρειάζεται προσοχή, καθώς ο επεξεργαστικός φόρτος στους κόμβους γίνεται μεγαλύτερος, αφού απαιτείται μεγαλύτερη μεταφορά δεδομένων στο δίκτυο. Η μέθοδος αυτή παρουσιάζει καλύτερα αποτελέσματα όταν η embedded αποθήκευση θα είχε σαν αποτέλεσμα την αποθήκευση των ίδιων δεδομένων πολλές φορές, όταν είναι απαραίτητη η δημιουργία πολλών σχέσεων μεταξύ των documents και όταν είναι δυνατή η αναπαράσταση των δεδομένων σε μορφή δένδρου.

4.9.4 Query Model

Για την υποβολή των queries χρησιμοποιούνται JSON-like αρχεία, τα οποία στέλνονται στη βάση σαν BSON objects μέσω του driver που χρησιμοποιεί η εφαρμογή. Στα αρχεία αυτά εμπεριέχονται τα πεδία, οι τιμές και οι συνθήκες που πρέπει να ικανοποιούν τα documents για να συμπεριληφθούν στο αποτέλεσμα που θα επιστραφεί. Για πιο γρήγορη εκτέλεση των queries είναι δυνατή η δημιουργία indexes, ωστόσο καθώς αυτά καταναλώνουν χώρο και επιβραδύνουν την εκτέλεση των updates πρέπει να χρησιμοποιούνται με προσοχή.

Τα queries υποβάλλονται σε όλα τα documents μίας μόνο collection κάθε φορά και συμπεριλαμβάνονται όλα τα embedded Objects και Arrays που περιέχουν. Για την υποβολή ερωτημάτων σε documents περισσότερων collection θα πρέπει το query να εφαρμοστεί σε όλες τις collections που διαθέτουν τα ζητούμενα documents. Παρακάτω παρουσιάζεται ένα παράδειγμα query που επιστρέφει τα documents που έχουν σαν name την τιμή Jack, σαν phone το αντικείμενο με type την τιμή mobile και σαν language την τιμή German, MongoDB-Query 1.

```
{“name”: “Jack”, “phone.type”: “mobile”, “languages”: “German”}
```

MongoDB-Query 1 MongoDB Example Query

Το Query Model της MongoDB υποστηρίζει, μεταξύ άλλων, τα ακόλουθα χαρακτηριστικά:

- υποβολή queries σε documents και τα embedded subdocuments
- υποβολή geospatial queries
- χρήση τελεστών σύγκρισης $(>, \geq, <, \leq, =)$
- χρήση λογικών τελεστών (and, nor, not, or)
- χρήση τελεστών ικανοποίησης συνθηκών $(equals, exists, in, mod, type, \ldots)$
- χρήση aggregation functions (συναρτήσεις ομαδοποίησης) $(count, sum, \ldots)$

Εκτός από τα queries, για την υλοποίηση πολύπλοκων aggregation functions, χρησιμοποιείται μια παραλλαγή του MapReduce που έχει υλοποιηθεί από τη MongoDB. Στην έκδοση αυτή του MapReduce, η map function επεξεργάζεται την είσοδο και παράγει key/value pairs. Τα key/value pairs ομαδοποιούνται με βάση το key και για κάθε ένα προκύπτει ένας πίνακας από values. Τα αποτελέσματα προωθούνται στην reduce function, η οποία εφαρμόζεται σε όλα τα στοιχεία του values πίνακα και επιστρέφει μία μοναδική τιμή για το κάθε key. Το αποτέλεσμα που επιστρέφεται από την reduce απαιτείται να είναι ίδιου τύπου δεδομένων με των values της map function. Η reduce function δεν πρέπει να έχει πρόσβαση στη βάση. Η σειρά των στοιχείων του values πίνακα δεν πρέπει να επηρεάζει το αποτέλεσμα που θα επιστραφεί και πρέπει η reduce function να είναι idemponent, αντιμεταθετική, δηλαδή να ικανοποιείται ο παρακάτω τύπος.

Ο χρήστης μπορεί προαιρετικά να ορίσει ακόμα μία συνάρτηση, τη finalize function. Αυτή η συνάρτηση εφαρμόζεται στα αποτελέσματα της reduce function και δίνει ένα μοναδικό αποτέλεσμα. Χρησιμοποιείται για την ομαδοποίηση των αποτελεσμάτων της reduce function, όπως για τον υπολογισμό ενός μέσου όρου. Η ανάγκη δημιουργία της finalize function στην έκδοση αυτή του MapReduce προκύπτει από τον περιορισμό της reduce function να είναι αντιμεταθετική και προσεταιριστική.

Τα αποτελέσματα του MapReduce αποθηκεύονται σε μία collection, είτε δημιουργώντας την αν δεν υπάρχει, είτε αντικαθιστώντας τα δεδομένα της με τα καινούργια. Υπάρχει ακόμα η δυνατότητα υποβολής ερωτημάτων στα δεδομένα εισόδου της map function, κάτι που μπορεί να μειώσει σημαντικά τις εργασίες της. Τέλος οι χρήστες μπορούν να εφαρμόζουν ταξινόμηση στα δεδομένα εισόδου, κάτι που είναι χρήσιμο για βελτιστοποιήσεις, καθώς σε ορισμένες εφαρμογές ταξινομώντας τα δεδομένα σύμφωνα με το key γίνονται λιγότερες πράξεις στην reduce function.

4.9.5 Indexing

Στην MongoDB για γρηγορότερη και αποδοτικότερη εκτέλεση των ερωτημάτων χρησιμοποιούνται indexes, δομές δεδομένων που εντοπίζουν γρήγορα documents με βάση ένα ή περισσότερα συγκεκριμένα πεδία. Τα indexes αποθηκεύονται σε μορφή B-trees και είναι όμοια με αυτά που συναντώνται σε πολλές άλλες βάσεις δεδομένων. Εφαρμόζονται στα keys των fields και embedded Objects και Arrays ενός document μίας μόνο collection. Όταν ένα document μεγαλώσει πολύ σε μέγεθος, λόγω του sharding που θα αναλυθεί στη συνέχεια, πρέπει να μετακινηθεί. Τότε πραγματοποιείται update σε όλα τα index keys που αφορούν στο document. Σε διαφορετική περίπτωση update γίνεται μόνο όταν αλλάξουν οι τιμές των πεδίων με βάση τα οποία έχει γίνει το indexing.

Όπως αναφέρθηκε η MongoDB δημιουργεί αυτόματα index με βάση το *_id* του κάθε document. Indexes μπορούν να οριστούν για όλα τα fields ενός document. Το indexing στα

πεδία με Arrays, που μπορεί να περιέχουν tags, γίνεται με χρήση των *multikeys*, όπου δημιουργείται ένα ξεχωριστό index για κάθε στοιχείο του πίνακα. Ένα τέτοιο index μπορεί να χρησιμοποιηθεί για την αναζήτηση αρχείων με βάση τα tags. Τέλος διακρίνονται indexes με βάση περισσότερα του ενός fields, τα *compound indexes*.

Οι σχεδιαστές ανάλογα με τα queries, τη συχνότητα που εκτελούνται, την αναλογία των reads και writes και την διαθέσιμη μνήμη επιλέγουν την δημιουργία των κατάλληλων indexes. Όταν τα index keys διατηρούν όλη την πληροφορία που αφορά ένα query, ελαχιστοποιούνται τα documents που πρέπει να αποθηκευτούν ολόκληρα στη μνήμη και μεγιστοποιείται η απόδοση και το throughput της βάσης. Καθώς όμως η διατήρηση indexes προκαλεί καθυστέρηση στα updates πρέπει να δημιουργούνται προσεκτικά και για τα πιο συχνά χρησιμοποιούμενα queries. Μια πρακτική για γρήγορη εκτέλεση των queries είναι τα indexes να διατηρούνται στην RAM. Αν παρατηρείται αργή εκτέλεση των queries μπορεί να υποδεικνύει κακή επιλογή indexes είτε ότι αυτά δεν βρίσκονται στη RAM.

Σε κάθε query χρησιμοποιείται μόνο ένα index. Η επιλογή γίνεται εμπειρικά από τον *query optimizer*, ο οποίος μπορεί να παρακαμφθεί. Ο query optimizer εκτελεί queries περιστασιακά με χρήση διαφορετικών indexes και επιλέγει αυτό που δίνει την καλύτερη απόδοση. Ένα ακόμα πλεονέκτημα της χρήσης indexes είναι η βελτίωση της απόδοσης των queries στις περιπτώσεις που εφαρμόζεται MapReduce.

4.9.6 Replication

Οι βάσεις δεδομένων υλοποιούν κάποιο μηχανισμό replication για να εξασφαλίσουν συνέπεια και υψηλή διαθεσιμότητα στα δεδομένα, αξιοπιστία και ανοχή σε σφάλματα. Η MongoDB χρησιμοποιεί για replication τα replica sets. Κάθε replica set, που συνήθως είναι ένας shard node, αποτελείται από έναν primary κόμβο και έναν ή περισσότερους secondary κόμβους. Όλοι έχουν εγκατεστημένο το λογισμικό mongod.

Ο primary κόμβος είναι υπεύθυνος για όλα τα writes και consistent reads και διατηρεί log με εγγραφές για όλες τις εργασίες που έχουν γίνει στα δεδομένα του. Οι secondary κόμβοι ενημερώνονται ασύγχρονα από το log του primary κόμβου, εφαρμόζουν στα δεδομένα τους τις αλλαγές που τους αφορούν και διατηρούν eventually consistent δεδομένα. Για να επιστρέψει ένα write αρκεί να γραφτούν τα δεδομένα στον primary κόμβο. Τότε σε περίπτωση αποτυχίας του primary και αν δεν έχουν προλάβει οι secondary κόμβοι να ενημερωθούν από το log, υπάρχει κίνδυνος να χαθούν δεδομένα. Σε περιπτώσεις εφαρμογών που απαιτείται αξιοπιστία των δεδομένων κάθε χρονική στιγμή οι σχεδιαστές μπορούν να επιλέξουν να επιστρέφει ένα write αφότου έχουν γραφτεί τα δεδομένα στον primary και σε όλους τους secondary κόμβους που τα διατηρούν.

Με χρήση replica sets διασφαλίζεται το αυτόματο failover σε περίπτωση σφάλματος. Οι κόμβοι ενός replica set χρησιμοποιούν heartbeats για τον εντοπισμό των κόμβων που έχουν αστοχήσει. Αν ο primary κόμβος αποτύχει, εκλέγεται αυτόματα ένας νέος από τους secondary κόμβους του replica set. Όταν οι κόμβοι που έχουν αστοχήσει επανέλθουν, συγχρονίζονται με τον primary του replica set και συνεχίζουν την λειτουργία τους ως secondary. Ο περιορισμός των replica sets είναι ότι κάθε ένα δεν μπορεί να διαθέτει παραπάνω από 12 κόμβους, έναν primary και 11 secondary. Σε περιπτώσεις που απαιτούνται περισσότεροι secondary κόμβοι η MongoDB δίνει την επιλογή χρήσης master-slave replication, όπου όμως δεν υποστηρίζονται μηχανισμοί αυτόματου failover.

4.9.7 Auto-Sharding

Με τον όρο *sharding* εννοούμε τον καταμερισμό μιας βάσης δεδομένων σε ένα cluster. Τα cluster στη MongoDB αποτελούνται από shard nodes, στους οποίους αποθηκεύονται τα

δεδομένα, configuration servers και routers. Η MongoDB υποστηρίζει το αυτόματο sharding, ισοκατανέμοντας τον όγκο των δεδομένων και το φόρτο εργασίας σε όλους τους κόμβους του cluster. Με την προσθήκη νέων κόμβων τα δεδομένα καταμερίζονται αυτόματα, παρέχοντας επεκτασιμότητα στη βάση.

Οι σχεδιαστές επιλέγουν σε ποιες βάσεις και σε ποιες collection τους θα ενεργοποιήσουν το sharding. Κάθε τέτοια collection διαμερίζει τα documents της σύμφωνα με ένα ορισμένο από το χρήστη *shard key*, έτσι ώστε κάθε shard να πάρει documents με ένα συγκεκριμένο εύρος τιμών του shard key. Το shard key, που είναι αρκετά όμοιο με ένα index, είναι ένα field το οποίο υποχρεωτικά υπάρχει σε κάθε document της collection. Μπορούν να δημιουργηθούν shard keys με περισσότερα του ενός πεδία.

Κάθε shard διατηρεί ταξινομημένα σύμφωνα με το shard key τα document που του έχουν ανατεθεί. Τα documents στα shards χωρίζονται σε *chunks*, που το καθένα διαθέτει ταξινομημένα documents από ένα start shard key μέχρι ένα end shard key. Αν ένα chunk μεγαλώσει πολύ σε μέγεθος χωρίζεται. Τα chunks χρησιμοποιούνται για την ισοκατανομή των δεδομένων στο cluster με την αυτόματη μετακίνηση τους, όπως στην περίπτωση προσθήκης ή αφαίρεσης κόμβων. Όταν ένα shard ξεπεράσει ένα ορισμένο όγκο δεδομένων μετακινεί chunks του σε μικρότερα shards.

Οι configuration servers διατηρούν πληροφορίες για την θέση των δεδομένων μέσα στο cluster. Πιο συγκεκριμένα διατηρούν εγγραφές για το κάθε chunk, στις οποίες αποθηκεύονται το start shard key, το end shard key και το shard στο οποίο βρίσκεται. Με αυτόν τον τρόπο μπορούν οι routers, αφού συμβουλευτούν τους configuration servers, να δρομολογούν τα requests στα shards που περιέχουν τα ζητούμενα δεδομένα.

Κεφάλαιο 5
MPEG-7, MPEG-21

Στη διπλωματική αυτή εξετάζεται η αποθήκευση των metadata πολυμεσικών αρχείων με στόχο την εύρεση καλύτερων τρόπων αποθήκευσης, διαχείρισης και ανάκτησής multimedia αρχείων. Για το σκοπό αυτό είναι απαραίτητη η κατανόηση του τρόπου λειτουργίας των προτύπων MPEG-7 και MPEG-21, καθώς και της τελικής δομής των metadata που δημιουργούνται σύμφωνα με αυτά. Στο κεφάλαιο αυτό θα γίνει η παρουσίαση της ομάδας MPEG και των προτύπων που εισήγαγαν για την κωδικοποίηση multimedia αρχείων, θα αναλυθούν εκτενέστερα τα πρότυπα MPEG-7 και MPEG-21 και θα γίνει μια σύντομη αναφορά στην γλώσσα XML.

5.1 Εισαγωγή

Η Moving Pictures Experts Group, MPEG, είναι η ομάδα που δημιουργήθηκε από την ISO/IEC για την ορισμό προτύπων στην επεξεργασία, συμπίεση και μετάδοση αρχείων ήχου, audio, και εικόνας, video. Μέχρι σήμερα έχουν δημιουργηθεί πέντε πρότυπα που καλύπτουν διαφορετικές πλευρές του ίδιου προβλήματος, ανάλογα με τις ανάγκες του χρήστη, τα εργαλεία που διαθέτει και το υλικό προς επεξεργασία.

Όλα τα MPEG πρότυπα έχουν δημιουργηθεί ακολουθώντας την αρχή του να ορίζουν όσο το δυνατόν λιγότερα και να εγγυόνται διαλειτουργικότητα. Αυτό δίνει στους σχεδιαστές τη δυνατότητα της διαρκούς βελτιστοποίησης των εφαρμογών με χρήση ολοένα και καλύτερων εργαλείων, ενώ ταυτόχρονα επιτρέπει τον υγιή ανταγωνισμό μεταξύ των υπηρεσιών παρέχοντας αμοιβαία διαλειτουργικότητα. Με αυτόν τον τρόπο η διάρκεια ζωής των προτύπων αυξάνεται και παρέχονται διαρκώς καλύτερες υπηρεσίες. Παρακάτω παρουσιάζονται επιγραμματικά αυτά τα πρότυπα, ενώ θα αναλυθούν ειδικότερα κάποια από αυτά στις Ενότητες 5.2 MPEG-7 και 5.3 MPEG-21.

MPEG-1: "*Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s* (ISO/IEC 11172)".

Είναι το πρώτο πρότυπο για συμπίεση αρχείων audio και video και δημιουργήθηκε σαν εναλλακτική των αναλογικών βιντεοκασετών, VHS. Στόχος του προτύπου είναι να προσφέρει έναν τρόπο ψηφιακής κωδικοποίησης audio-visual δεδομένων για αποθήκευση σε ψηφιακά μέσα, όπως CD, DAT και optical drives. Για να ανταγωνιστεί τις βιντεοκασέτες, το πρότυπο έπρεπε να είναι ικανό να παρέχει όλες τις λειτουργικότητες τους, όπως fast forward, random access και fast reverse. Ταυτόχρονα έπρεπε η ποιότητα εικόνας και ήχου να είναι ανάλογη των VHS. Είναι φανερό πως ήταν απαραίτητη τόσο η βελτιστοποίηση του τρόπου συμπίεσης και κωδικοποίησης των αρχείων, αλλά και η παροχή επιπρόσθετης λειτουργικότητας ανάλογα με τις απαιτήσεις των χρηστών.

Καθώς το κύριο μέσο αποθήκευσης επρόκειτο να είναι το CD, το πρότυπο βελτιστοποιήθηκε για ρυθμό μετάδοσης, bitrate range, 1.5 Mbps. Ωστόσο το πρότυπο δουλεύει τόσο για χαμηλότερο όσο και για υψηλότερο ρυθμό μετάδοσης. Το MPEG-1 έκανε δυνατή την μεταφορά πληροφορίας audio και video σε CD και χρησιμοποιήθηκε στην μετάδοση video μέσω δικτύων χαμηλού bandwidth, όπως σε υπηρεσίες επίγειας και δορυφορικής ψηφιακής τηλεόρασης. Πιο γνωστό κομμάτι του MPEG-1 είναι το MP3 format για αρχεία ήχου.

MPEG-2: "*Generic coding of moving pictures and associated audio information* (ISO/IEC 13818)".

Πρόκειται για την επέκταση του MPEG-1. Προστίθενται νέα χαρακτηριστικά, όπως υποστήριξη video υψηλής ανάλυσης, τεχνικές interlacing, και ορίζονται formats για αποθήκευση videos, ταινιών και προγραμμάτων σε DVDs. Με το MPEG-2 είναι δυνατή η αποθήκευση και μετάδοση ταινιών υψηλής ανάλυσης με διαθέσιμα μέσα αποθήκευσης και bandwidth. Χρησιμοποιήθηκε σαν πρότυπο στην μετάδοση σήματος υψηλής ποιότητας για την ψηφιακή τηλεόραση, High Definition TV, HDTV. Στο MPEG-2 ενσωματώθηκε το MPEG-3, πρότυπο για χειρισμό High Definition TV σημάτων με ρυθμό μετάδοσης από 20 μέχρι 40 Mbps.

MPEG-4: "*Coding of audio-visual objects* (ISO/IEC 14496)".

Τα προηγούμενα πρότυπα, MPEG-1 και MPEG-2, ακολουθούσαν κυρίως ένα μοντέλο αναπαράστασης εικόνων, γνωστό ως *frame-based model*. Παρότι αυτός ο τρόπος αναπαράστασης των δεδομένων ήταν ικανός να ικανοποιήσει τις ανάγκες πολλών υπηρεσιών, δεν μπορούσε να προσφέρει τις όλο και πολυπλοκότερες δυνατότητες που απαιτούσαν οι σύγχρονες εφαρμογές. Το MPEG-4 αποτελεί πρότυπο για την υλοποίηση multimedia εφαρμογών και προσφέρει έναν εναλλακτικό τρόπο αναπαράστασης δεδομένων, το *object-based model*. Στο object-based model το audio-visual περιεχόμενο δεν θεωρείται απλή αναπαράσταση εικόνων, αλλά δομείται με χρήση αντικειμένων, objects. Οι εικόνες των audio-visual αρχείων μπορούν να δημιουργηθούν με την σύνθεση ενός ή περισσοτέρων objects, τα οποία μπορούν να υποστούν επεξεργασία, να κωδικοποιηθούν και να συμπιεστούν ανεξάρτητα το ένα από το άλλο. Αυτός ο τρόπος δόμησης των δεδομένων δίνει δυνατότητες αλληλεπίδρασης με το περιεχόμενο και επιλογής των αντικειμένων που θα μεταδοθούν εξοικονομώντας υπολογιστικούς πόρους και bandwidth.

Ένα από τα βασικότερα πλεονεκτήματα του MPEG-4 σε σύγκριση με τα προηγούμενα πρότυπα, παρότι διατηρεί πολλά από τα χαρακτηριστικά τους, είναι η δυνατότητα αναπαράστασης των στοιχείων που συνθέτουν το video stream, playground, audio και video, με objects διαφορετικής προέλευσης, όπως text, speech, 3D models, 2D meshes, synthetic audio και VRML objects. Το *VRML, Virtual Reality Modeling Language*, είναι ένας τύπος text αρχείων για την αναπαράσταση τρισδιάστατων διαδραστικών vector graphics. Ένα ακόμα πλεονέκτημα του MPEG-4 αποτελεί το γεγονός ότι δεν στοχεύει σε συγκεκριμένες εφαρμογές, αλλά μπορεί να εφαρμοστεί σε όλα τα μέσα, mobiles, PCs, και από χαμηλό bitrate μέχρι αποδοτική μετάδοση υψηλής ποιότητας multimedia αρχείων. Το MPEG-4 προσφέρει επίσης το MPEG-J, ένα Java interface που επιτρέπει τη χρήση Java classes στο περιεχόμενο των multimedia αρχείων. Με αυτό το εργαλείο μπορούν να οριστούν μηχανισμοί διαδραστικότητας των δεδομένων, ώστε οι τελικοί χρήστες να μπορούν να αλληλεπιδρούν με τα αντικείμενα των εφαρμογών. Υποστηρίζεται τέλος η χρήση εργαλείων για την διαχείριση και προστασία του περιεχομένου των αρχείων, όπως *Digital Rights Management*.

MPEG-7: "*Multimedia content description interface* (ISO/IEC 15938)".

Πρόκειται για ένα πρότυπο περιγραφής του περιεχομένου των multimedia αρχείων. Χρησιμοποιείται σε συνδυασμό με κάποιο από τα πρότυπα MPEG-1, MPEG-2 και MPEG-4, αλλά διαφοροποιείται στο ότι δεν αφορά στην κωδικοποίηση των δεδομένων, αλλά στην περιγραφή του περιεχομένου των αρχείων που κωδικοποιούνται σύμφωνα με κάποιο από αυτά. Επιτρέπει την γρήγορη και αποτελεσματική αναζήτηση υλικού σύμφωνα με τις απαιτήσεις του χρήστη. Το MPEG-7 περιγράφει εκτός από πληροφορίες περιεχομένου και πληροφορίες για το ίδιο το αρχείο, όπως την ημερομηνία δημιουργίας του, την κωδικοποίηση και την συμπίεση που επιλέγεται. Χρησιμοποιεί αρχεία τύπου XML για την αποθήκευση των metadata και μπορεί να συνδυαστεί με timecodes για τον σημείωση συγκεκριμένων γεγονότων. Το MPEG-7 θα παρουσιαστεί αναλυτικότερα σε επόμενη ενότητα.

MPEG-21: "*Multimedia framework* (ISO/IEC 21000)".

Το MPEG-21 ορίζει ένα framework για ανάπτυξη multimedia εφαρμογές και παρέχει μηχανισμούς διαχείρισης και προστασίας της πνευματικής ιδιοκτησίας. Εισάγονται δύο θεμελιώδεις έννοιες, ο User και το Digital Item. Κύριος στόχος του αποτελεί ο προσδιορισμός των απαραίτητων τεχνολογικών πόρων, ώστε οι χρήστες να έχουν δυνατότητες πρόσβασης, και διαχείρισης των Digital Items με τον πλέον αποδοτικό τρόπο. Πιο αναλυτικά το πρότυπο MPEG-21 θα παρουσιαστεί σε επόμενη ενότητα.

5.2 MPEG-7

Με την διαθεσιμότητα των προτύπων κωδικοποίησης MPEG-1, MPEG-2 και MPEG-4 έχει γίνει ευκολότερη η δημιουργία, η απόκτηση και ο διαμοιρασμός αρχείων με audio-visual περιεχόμενο. Οι απαιτήσεις των χρηστών, για γρήγορο και αποτελεσματικό εντοπισμό των δεδομένων που τους ενδιαφέρουν, ολοένα και αυξάνονται. Παράλληλα γίνεται διαθέσιμος όλο και μεγαλύτερος όγκος δεδομένων. Με συμβατικές μεθόδους η αναζήτηση των αρχείων που περιέχουν τις ζητούμενες από τους χρήστες πληροφορίες θα μείωνε πολύ την απόκριση των εφαρμογών.

Για τους παραπάνω λόγους ήταν απαραίτητη η ανάπτυξη και η προτυποποίηση ενός μηχανισμού περιγραφής του περιεχομένου των αρχείων, ώστε οι αναζητήσεις των χρηστών να κωδικοποιούνται με καθολικό τρόπο. Ακολουθώντας ένα τέτοιο πρότυπο η ανάκτηση αρχείων μπορεί να γίνει στοχευμένα, με βάση τις εκάστοτε απαιτήσεις, γρήγορα και αξιόπιστα.

Το MPEG-7 αποτελεί το πρότυπο για την περιγραφή του περιεχομένου αρχείων. Είναι σχεδιασμένο να παρέχει συμπληρωματικές λειτουργίες στα πρότυπα MPEG-1, MPEG-2 και MPEG-4, παρουσιάζοντας πληροφορίες για το περιεχόμενο των αρχείων. Το MPEG-7 απαρτίζεται από 11 κομμάτια, τα οποία θα αναλυθούν στην συνέχεια. Οι περιγραφές του MPEG-7 παρέχουν πληροφορίες για ένα μεγάλο εύρος εφαρμογών και εφαρμόζονται σε πολλά αφαιρετικά επίπεδα, από περιγραφή low-level και στατιστικών χαρακτηριστικών των αρχείων μέχρι high-level αναπαράσταση του σημασιολογικού περιεχομένου τους. Εφαρμόζονται στα δεδομένα ανεξάρτητα από το μέσο στο οποίο μεταδίδονται και τον τρόπο που έχουν κωδικοποιηθεί, η αναπαράσταση τους γίνεται με αντικειμενοστραφή τρόπο και μπορούν να επεκταθούν ανάλογα με τις ανάγκες των χρηστών.

5.2.1 MPEG-7 Description Tools

Όπως έχει αναφερθεί, τα MPEG πρότυπα έχουν σχεδιαστεί με τέτοιο τρόπο ώστε να

δίνουν στους σχεδιαστές ευελιξία στην παραγωγή εφαρμογών, οι οποίες απαιτείται να ακολουθούν ορισμένους μόνο πολύ γενικούς κανόνες. Στην ίδια λογική το MPEG-7 ορίζει μόνο τον τρόπο περιγραφής των metadata και το decoding και αφήνει στους σχεδιαστές την δημιουργία των περιγραφών. Ορίζονται δύο εργαλεία, *description tools*, για την περιγραφή του περιεχομένου των αρχείων, οι *Descriptors* και τα *Description Schemas*.

Descriptors (D) – Ένας Descriptor, περιγραφέας, ορίζει την σύνταξη και σημασιολογία της αναπαράστασης ενός στοιχειώδους χαρακτηριστικού. Ως χαρακτηριστικό ορίζεται ένα διακριτό γνώρισμα των δεδομένων, όπως το χρώμα. Ένας Descriptor μπορεί να περιγράψει low-level χαρακτηριστικά του αρχείου, όπως texture, shape και motion, μέχρι high-level σημασιολογικά δεδομένα, όπως το όνομα του συγγραφέα και τον τίτλο του αρχείου. Η δομή και ο τρόπος σύνταξης των Descriptors ορίζεται από την *Description Definition Language, DDL*. Η DDL προσφέρει εκτός από τους κανόνες που πρέπει να ακολουθούν οι Descriptors και εργαλεία για την δημιουργία νέων κανόνων και κατ' επέκταση την παραγωγή νέων Descriptors που ικανοποιούν τις ανάγκες των χρηστών. Κάποια παραδείγματα Descriptors αποτελούν οι time-codes για αναπαράσταση χρονικής διάρκειας, string χαρακτήρων για αναπαράσταση ενός ονόματος και ιστογράμματα για αναπαράσταση εικόνας.

Description Schemas (DS) – Ένα Description Schema, σχήμα περιγραφής, προσδιορίζει την σύνταξη και τη σημασιολογία των σχέσεων μεταξύ components, όπου components μπορούν να είναι Descriptors ή Description Schemes. Τα Description Schemes δημιουργούνται από την διασύνδεση των Descriptors και τον ορισμό των μεταξύ τους σχέσεων. Η δομή και η σύνταξη των Description Schemes ορίζονται από την DDL, η οποία σε αρκετές περιπτώσεις δίνει τη δυνατότητα δημιουργίας και νέων DSs. Ένα παράδειγμα Description Schema αποτελεί το *AudioVisualSegment DS* όπου ορίζεται ένας συνδυασμός από πληροφορίες audio και video για περιγραφή video με ενσωματωμένο ήχο.

5.2.2 XML

Λόγω της πληθώρας των μέσων στις οποίες καλείται να εφαρμοστεί το MPEG-7, έπρεπε να επιλεγεί ένας τρόπος αναπαράστασης των δεδομένων που να μπορεί να εφαρμοστεί σε κάθε εφαρμογή και σε όλες τις πλατφόρμες προγραμματιστικού περιβάλλοντος. Ταυτόχρονα τα αρχεία έπρεπε να είναι δομημένα με τρόπο εύκολα κατανοητό τόσο από τους χρήστες όσο και από τους υπολογιστές. Τέλος θα έπρεπε να είναι δυνατή η ενσωμάτωση πολύπλοκων πληροφοριών σε απλά schemes, κατανοητά και διαχειρίσιμα από τους χρήστες, ώστε να μπορούν να τα προσαρμόσουν εύκολα στις ανάγκες τους. Αυτοί οι λόγοι οδήγησαν στην επιλογή της XML για γλώσσα δόμησης των αρχείων που ακολουθούν το πρότυπο MPEG-7. Συνεπώς οι Descriptors και τα Description Schemas που χρησιμοποιούνται για την αποθήκευση των metadata των αρχείων αποτελούν αρχεία τύπου XML.

Η χρήση της XML κάνει το πρότυπο MPEG-7 γενικό, απλό και εύκολα κατανοητό. Η δύναμη του προτύπου έγκειται στην χρήση απλών schemes για αναπαράσταση σύνθετων δεδομένων. Ακόμα λόγω της μορφής των XML αρχείων γίνεται πολύ αποδοτική η επεξεργασία και αναζήτηση δεδομένων. Καθώς τα αρχεία XML αποτελούν γνωστή τεχνολογία και χρησιμοποιούνται σε πληθώρα εφαρμογών, επιτυγχάνεται άμεση εξοικείωση των χρηστών με το MPEG-7 και την διαχείρισή του.

Τα XML αρχεία αποτελούνται από ένα string χαρακτήρων. Διακρίνονται δύο βασικά δομικά χαρακτηριστικά τους, τα *elements* και τα *attributes*. Τα elements αποτελούν τα components ενός αρχείου, περικλείονται από tags που σηματοδοτούν την αρχή και το τέλος του, μπορούν να έχουν ό,τι όνομα επιθυμούν οι χρήστες και περιέχουν τις απαραίτητες πληροφορίες για την εκτέλεση των απαιτούμενων και ορισμένων από το χρήστη ενεργειών. Η πληροφορία που περιέχουν μπορεί να είναι είτε το κενό, είτε ένα text, είτε άλλα elements.

Κάθε element μπορεί να έχει ιδιότητες, attributes, οι οποίες αποτελούνται από το όνομα του attribute, που είναι ένα text string και το value του. Ανάλογα με τις απαιτήσεις οι χρήστες ορίζουν την ακριβή σημασία των attributes και τις ενέργειες που πραγματοποιούνται με βάση αυτά. Ένα παράδειγμα αρχείου XML παρουσιάζεται στην Εικόνα 17.

```
<messages>
      <note id="501">
            <to>Tove</to>
            <from>Jani</from>
            <heading>Reminder</heading>
            <body>Don't forget me this weekend!</body>
      </note>
      <note id="502">
            <to>Jani</to>
            <from>Tove</from>
            <heading>Re: Reminder</heading>
            <body>I will not</body>
      </note>
</messages>
```

Εικόνα 17 XML Example Document

Στην Εικόνα 17 διακρίνεται το root element *messages* που περιέχει δύο child elements που ονομάζονται *note*. Κάθε *note* child element έχει ένα attribute *id* και περιέχει τέσσερα child elements, τα *to*, *from*, *heading* και *body*, καθένα από τα οποία περιέχει ένα text element, τα *Tove*, *Jani*, *Reminder* και *Don't forget me this weekend* αντίστοιχα για το note με *id = "501"* και *Jani, Tove, Re: Reminder* και *I will not* αντίστοιχα για το note με *id = "502"*.

5.2.3 MPEG-7 Parts

Τα MPEG πρότυπα περιλαμβάνουν το κάθε ένα το σύνολο των απαραίτητων προδιαγραφών για τον τρόπο διαχείρισης του περιεχομένου που τα αφορά. Ωστόσο αυτές οι προδιαγραφές θα έπρεπε να μπορούν να χρησιμοποιηθούν και ξεχωριστά, συνδυάζοντας τεχνολογίες, χρησιμοποιώντας μόνο ότι είναι απαραίτητο για την κάθε εφαρμογή και ανάλογα με τους περιορισμούς που αντιμετωπίζει ο κάθε χρήστης. Για αυτό το λόγο όλα τα MPEG πρότυπα οργανώνονται σε Parts καθένα από τα οποία περιλαμβάνει τις προδιαγραφές για επίλυση διαφορετικών πτυχών του προβλήματος. Παρακάτω παρουσιάζονται τα επιμέρους Parts του προτύπου MPEG-7, ενώ σε επόμενη ενότητα θα αναλυθεί εκτενέστερα το Part 2 που περιγράφει το Description Definition Language (DDL).

Part 1: Systems: Ορίζονται τα εργαλεία για τα παρακάτω:

i. Μεταφορά και αποθήκευση των MPEG-7 περιγραφών με αποδοτικό τρόπο. Μπορεί να θεωρηθεί και ένα εργαλείο για συμπίεση XML αρχείων.
ii. Συγχρονισμό των MPEG-7 περιγραφών με το περιεχόμενο που περιγράφουν. Οι MPEG-7 περιγραφές μπορεί να μεταφέρονται ανεξάρτητα ή μαζί με το περιεχόμενο που περιγράφουν.
iii. Διαχείριση και προστασία της πνευματικής ιδιοκτησίας του περιεχομένου που σχετίζεται με τις MPEG-7 περιγραφές.

Part 2: Description Definition Language (DDL)

Προσδιορίζεται η γλώσσα για την δημιουργία νέων Description Schemes και για την επέκταση και τροποποίηση των ήδη ενσωματωμένων στο MPEG-7. Η DDL που έχει επιλεγεί είναι βασισμένη στην XML Schema Language της W3C. Για να ικανοποιηθούν οι ανάγκες περιγραφής αρχείων με multimedia περιεχόμενο στην DDL περιλαμβάνονται εκτός από την XML Schema και κάποιες προεκτάσεις.

Part 3: Visual

Ορίζονται τα εργαλεία για την περιγραφή visual περιεχομένου. Περιλαμβάνονται Descriptors και Description Schemes για την περιγραφή των visual χαρακτηριστικών των αρχείων και για τον εντοπισμό των αντικειμένων που περιγράφονται μέσα στην εικόνα ή το video. Πέντε από τα βασικά χαρακτηριστικά μιας εικόνας περιγράφονται με Descriptors που περιέχονται σε αυτό το κομμάτι του MPEG-7. Αυτά τα βασικά χαρακτηριστικά είναι color, texture, shape, motion και localization. Περιλαμβάνονται επίσης Descriptors για την περιγραφή προσώπων.

Part 4: Audio

Περιγράφονται τα audio χαρακτηριστικά ενός αρχείου. Τα εργαλεία που περιέχονται χωρίζονται σε τομείς περιγραφής ήχου, όπως timbre, melody, silence, sound effects και spoken content.

Part 5: Multimedia Description Schemes (MDS)

Σε αυτό το κομμάτι του MPEG-7 ορίζονται τα εργαλεία για την περιγραφή περιεχομένου που δεν είναι audio ή visual, δηλαδή περιεχομένου generic και multimedia. Είναι φανερό πως εδώ περιλαμβάνεται ο μεγαλύτερος αριθμός description tools και μπορούν να βρεθούν από απλά εργαλεία για τον ορισμό της δομής των περιγραφών, μέχρι τρόποι περιγραφής των user preferences και των τεχνικών για προσθήκη στις περιγραφές audio και visual description tools. Ανάλογα με τη λειτουργικότητά τους διακρίνονται οι παρακάτω έξι κατηγορίες MDS description tools.

i. Basic Elements: αφορούν στις generic οντότητες που χρησιμοποιούνται από όλα τα description tools για το χτίσιμο των περιγραφών τους. Σε αυτά περιλαμβάνονται βασικά datatypes (αριθμοί, πίνακες, διανύσματα), εργαλεία διασύνδεσης και προσδιορισμού τοποθεσίας (time locators, referencing tools) και βασικοί description tools για την περιγραφή τοποθεσιών, ατόμων και άλλων.
ii. Schema Tools: περιλαμβάνουν τα εργαλεία για χρήση των description tools από εφαρμογές και την ομαδοποίηση συγγενών description tools σε φακέλους.
iii. Content Description Tools: ορίζονται τα εργαλεία για την αναπαράσταση των πληροφοριών συμπεριλαμβανομένων των δομικών και σημασιολογικών πτυχών του περιεχομένου. Το Structure Description Tool χρησιμοποιείται για την περιγραφή των δεδομένων όσον αφορά στα δομικά χαρακτηριστικά τους. Τα δομικά κομμάτια της περιγραφής αναφέρονται στο Content Management Description Tools που θα παρουσιαστούν παρακάτω. Το σημασιολογικό περιεχόμενο των αρχείων περιγράφετε με χρήση των Semantic Description Tools, δηλαδή με χρήση αντικειμένων (objects),

γεγονότων (events), εννοιών και σχέσεων. Τα Semantic και Structural Tools μπορούν να συσχετιστούν μεταξύ τους για παραγωγή πιο ολοκληρωμένων πληροφοριών.

iv. Content Management Description Tools: αποτελούν τα εργαλεία για την περιγραφή των χαρακτηριστικών των αρχείων και την δημιουργία και χρήση του περιεχομένου τους. Διακρίνονται τρείς κατηγορίες εργαλείων. Με τα Media Description Tools περιγράφεται ο τρόπος αποθήκευσης των δεδομένων, η κωδικοποίηση που χρησιμοποιείται, η ποιότητα τους και άλλα. Τα Creation Description Tools επιτρέπουν την περιγραφή της διαδικασίας δημιουργίας των αρχείων. Τέλος τα Usage Description Tools δίνουν τις απαραίτητες πληροφορίες για τον επιτρεπόμενο τρόπο χρήσης των δεδομένων, σύμφωνα με τη διαθεσιμότητά τους και των rights που διαθέτουν οι χρήστες, καθώς και ιστορικά στοιχεία για την χρήση των αρχείων.

v. Navigations and Access Description Tools: χρησιμοποιούνται για τον προσδιορισμό του τρόπου πλοήγησης των αρχείων με σκοπό τη διευκόλυνση της αναζήτησης και ανάκτησης δεδομένων με βάση το περιεχόμενό τους. Με χρήση των Summaries Description Tools είναι δυνατή η πλοήγηση με διάφορους τρόπους στο αρχείο, προσφέροντας αποδοτική πρόσβαση και προεπισκόπηση του multimedia περιεχομένου. Τα Partitions and Decomposition Tools που περιλαμβάνονται παρέχουν τα εργαλεία για προοδευτική πρόσβαση στα αρχεία.

vi. User Interaction Description Tools: επιτρέπουν την περιγραφή user preferences καθώς και ιστορικών πληροφοριών χρήσης αρχείων με multimedia περιεχόμενο.

Part 6: Reference Software

Περιλαμβάνει το απαραίτητο λογισμικό για την υλοποίηση των εργαλείων των Parts 1-5 που παρουσιάστηκαν παραπάνω.

Part 7: Conformance Testing

Περιλαμβάνονται οι διαδικασίες ελέγχου της εγκυρότητας των περιγραφών που έχουν υποβάλλει οι χρήστες. Οι περιγραφές αυτές πρέπει να είναι σύμφωνες με τις ορισμένες προδιαγραφές των Parts 1-5. Τα εργαλεία που χρησιμοποιούνται για τον έλεγχο της συμμόρφωσης με τις προδιαγραφές γίνεται σύμφωνα με το profile και το level των περιγραφών, όπως αυτά ορίζονται στο Part 9.

Part 8: Extraction and Use of MPEG-7 Descriptors

Παρέχει πληροφορίες για την εξαγωγή και την χρήση των MPEG-7 Descriptors.

Part 9: Profiles and Levels

Ορίζει το profile και το level των περιγραφών. Κάθε profile περιλαμβάνει ένα υποσύνολο των description tools του MPEG-7, τα οποία υποστηρίζουν συγκεκριμένες λειτουργίες και ικανοποιούν τις ανάγκες ενός συγκεκριμένου συνόλου εφαρμογών. Ανάλογα με την πολυπλοκότητα και τους περαιτέρω περιορισμούς που θέλουν να αποδώσουν οι χρήστες στις εφαρμογές ορίζονται τα levels του κάθε profile. Για κάθε περιγραφή MPEG-7 ορίζεται το profile και το level της και σύμφωνα με αυτά εφαρμόζονται έλεγχοι της συμμόρφωσής τους με το πρότυπο.

Part 10: Schema Definition

Ορίζει το schema definition που περιλαμβάνει όλα τα Parts του MPEG-7. Ταυτόχρονα συλλέγει τα description tools του προτύπου, προσδιορίζει τα namespaces και με χρήση της DDL ορίζει ένα μοναδικό schema για την περιγραφή της εφαρμογής.

Part 11: Profile Schemas

Περιλαμβάνει τα Schemas για τα profiles του προτύπου.

5.2.4 Description Definition Language, DDL

Το MPEG-7 διαθέτει στους χρήστες κάποια βασικά Ds και DSs για την περιγραφή του περιεχομένου των αρχείων τους. Πολλές φορές όμως, λόγω της αυξανόμενης πολυπλοκότητας του περιεχομένου των εφαρμογών, τα υπάρχοντα Ds και DSs δεν είναι αρκετά για την ακριβή περιγραφή των δεδομένων. Για την ικανοποίηση τέτοιων αναγκών ενσωματώθηκε στο MPEG-7 η Description Definition Language, DDL.

Η DDL, που αποτελεί βασικό κομμάτι του προτύπου MPEG-7, *Part 2: Description Definition Language (DDL)*, προσφέρει τα απαραίτητα εργαλεία για την ανάπτυξη και προσαρμογή των Ds και DSs σε εξειδικευμένες ανάγκες. Μέσω της DDL δίνονται στους χρήστες δυνατότητες επέκτασης, συνδυασμού και βελτίωσης των κανόνων που πρέπει να ικανοποιούν τα διαθέσιμα DSs καθώς και ορισμού νέων κανόνων για δημιουργία νέων Ds και DSs που να προσαρμόζονται πλήρως στις απαιτήσεις των εφαρμογών. Η ορθότητα των Ds και DSs, των τύπων δεδομένων που χρησιμοποιούν, του schema τους και ο έλεγχος της υπακοής τους στους ορισμένους από το MPEG-7 και τους χρήστες κανόνες, ελέγχεται από τον *DDL Parser*.

Λόγω των πληροφοριών που περιγράφει το MPEG-7, η DDL πρέπει να είναι ικανή να αναπαριστά τόσο πρωτογενείς τύπους δεδομένων, όπως strings και integers, αλλά και πολύπλοκους, όπως ιστογράμματα. Ακόμα είναι απαραίτητο να μπορεί να εκφράζει δομικές, κληρονομικές, χωρικές και νοηματικές σχέσεις μεταξύ των Ds και DSs, καθώς και κάθε είδους των μεταξύ τους διασυνδέσεων. Όπως έχει αναφερθεί τα MPEG-7 αρχεία αποτελούν αρχεία τύπου XML, για την διαχείριση των οποίων χρησιμοποιούνται XML Schema Languages. Η γλώσσα που έχει επιλεγεί σαν DDL του MPEG-7 είναι η *XML Schema Language* της *W3C, XSD*. Παράλληλα χρησιμοποιούνται και διάφορες επεκτάσεις για την προσαρμογή της XML Schema στην περιγραφή αρχείων με audio-visual περιεχόμενο. Πιο συγκεκριμένα η DDL αποτελείται από τρία βασικά τμήματα:

1 Δομικά στοιχεία της XML Schema
2 Τύπους δεδομένων της XML Schema
3 Επεκτάσεις του MPEG-7

5.2.4.1 Δομικά στοιχεία της XML Schema

Για την δημιουργία των κανόνων που πρέπει να ικανοποιούν οι περιγραφές του περιεχομένου των αρχείων χρησιμοποιούνται τα δομικά στοιχεία της XML Schema. Με αυτά οι χρήστες ορίζουν την δομή, τους περιορισμούς και τις ιδιότητες των Ds και DSs. Μέσω της XML Schema δίνονται στους χρήστες τα εργαλεία για την ευέλικτη παραγωγή schemas που να ικανοποιούν απόλυτα τις εκάστοτε ανάγκες τους. Τα schemas αποτελούν αφηρημένες συλλογές από metadata, που περιέχουν ένα σύνολο από schema components.

Τα schemas οργανώνονται με βάση το namespace. Το namespace αποτελεί μια συλλογή από μοναδικώς ορισμένους identifiers. Ο ίδιος identifier μπορεί να οριστεί σε περισσότερα του ενός namespaces, σε καθένα από τα οποία μπορεί να έχει διαφορετικό τρόπο λειτουργίας. Για να χρησιμοποιηθεί ένας identifier πρέπει να δοθεί τόσο το όνομά του, όσο και το namespace στο οποίο ανήκει. Με αυτόν τον τρόπο σε ένα XML αρχείο μπορούν να κληθούν identifiers που ανήκουν είτε στο ίδιο είτε σε διαφορετικά namespaces. Στην περίπτωση του MPEG-7 οι identifiers ορίζουν τα schema components. Σύμφωνα με το namespace μπορούν να κληθούν schema components που ανήκουν στο ίδιο ή σε διαφορετικό namespace από το schema που καλεί.

Στα κύρια components ενός schema περιλαμβάνονται μεταξύ άλλων *element* και *attribute declarations*, περιγραφές δηλαδή της δομής και του περιεχομένου των elements και attributes. Κατά τον ορισμό τους πρέπει επίσης να προσδιοριστεί το *type definition*, το οποίο αποτελεί ένα ακόμα κύριο component των schemas. Το type definition προσδιορίζει τους τύπους δεδομένων του περιεχομένου των elements και attributes. Δίνονται δύο τρόποι ορισμού του type, το *simple type definition* και το *complex type definition*, οι οποίοι θα αναλυθούν στη συνέχεια. Διακρίνονται επίσης τα *model* και *attribute group*. Μέσω των model και attribute groups οι χρήστες μπορούν να δημιουργήσουν macros, δηλαδή να ομαδοποιήσουν elements και attributes για χρήση τους σε πολλούς type definitions. Επιπλέον components αποτελούν το *attribute use*, που προσδιορίζει κατά πόσο είναι υποχρεωτική η χρήση ενός attribute, και το *element particle*, που προσδιορίζει τον ελάχιστο ή/και μέγιστο αριθμό εμφανίσεων ενός element.

Element Declarations

Τα element declarations χρησιμοποιούνται για τον ορισμό των ιδιοτήτων των elements του XML αρχείου. Στις ιδιότητες περιλαμβάνονται το όνομα του element, το namespace στο οποίο ανήκει, το type του, που προσδιορίζει τα attributes και τα children που μπορεί να περιέχει, καθώς και κανόνες κληρονομικότητας. Ακολουθεί ένα παράδειγμα ορισμού ενός element με όνομα `Age` και δεδομένα τύπου `integer` και δύο παραδείγματα χρήσης του.

```
<element name = "Age" type = "integer">
```

Example	Evaluation
<Age>20</Age>	✓
<Age>foo</Age>	✘

Attribute Declarations

Τα attribute declarations προσδιορίζουν τις ιδιότητες των attributes. Για κάθε attribute πρέπει να οριστούν το όνομά του, το namespace στο οποίο ανήκει και το type των δεδομένων που μπορεί να περιέχει. Προαιρετικά οι χρήστες μπορούν να επιλέξουν μια default ή μια fixed τιμή για το attribute. Ακολουθεί ένα παράδειγμα ορισμού ενός attribute με όνομα `id` και δεδομένα τύπου `ID`.

```
<attribute name = "id" type = "ID">
```

Simple Type Definitions

Με τους simple types προσδιορίζονται οι textual values ενός element ή attribute. Η XML Schema διαθέτει θεμελιώδεις τύπους δεδομένων, boolean, date, duration, float και άλλα, καθώς και ορισμένους παραγόμενους από αυτούς, token, integer, byte, date και άλλα, οι οποίοι μπορούν να χρησιμοποιηθούν σαν simple types σε ένα element ή attribute. Οι simple types δεν επιτρέπεται να περιέχουν άλλα elements ή attributes. Οι χρήστες μπορούν να ορίσουν δικούς τους simple types οι οποίοι θα πρέπει να υπακούσουν στους περιορισμούς που ορίζει η XML Schema. Ακολουθεί ένα παράδειγμα ορισμού ενός simple datatype με όνομα `exists` και δεδομένα τύπου `boolean` και τρία παραδείγματα χρήσης του.

```
<simpleType name = "exists">
     <restriction base = "boolean"/>
</simpleType>
```

Example	Evaluation
`<Ball type = "exists">true</Ball>`	✓
`<Ball type = "exists">false</Ball>`	✓
`<Ball type = "exists">foo</Ball>`	✗

Complex Type Definitions

Οι complex types περιγράφουν το επιτρεπόμενο περιεχόμενο των elements, προσδιορίζοντας τα attributes τους και τα child elements και texts. Κάθε child element μπορεί επίσης να περιέχει attributes και άλλα child elements. Οι χρήστες μπορούν να παράξουν νέα complex types από άλλα, είτε με περιορισμό κάποιων ιδιοτήτων τους, *restriction*, είτε επεκτείνοντάς τα, *extension*. Για τον ορισμό των element και attributes χρησιμοποιούνται αντίστοιχα τα element και attribute declarations που παρουσιάστηκαν προηγουμένως. Υπάρχει επίσης η δυνατότητα ορισμού κανόνων όσον αφορά στο πόσα, ποια και με τι περιεχόμενο elements επιτρέπονται στον κάθε complex type. Ανάλογα με το αν επιτρέπονται ή όχι child elements και texts διακρίνονται τέσσερεις κατηγορίες complex types. Οι empty complex types επιτρέπουν τη χρήση μόνο attributes, οι mixed complex types επιτρέπουν τη χρήση text μεταξύ των elements και των child elements, οι complexContent complex types επιτρέπουν μόνο τη χρήση elements και attributes και τέλος οι simpleContent complex types περιέχουν την ύπαρξη μόνο text. Ακολουθεί ένα παράδειγμα ορισμού ενός complex datatype με όνομα `ItemFeatures` και δεδομένα ένα attribute με όνομα `ItemPrice` και δεδομένα τύπου `price`.

```
<complexType name = "ItemFeatures">
     <attribute name = "ItemPrice" type = "price"/>
</complexType>
```

5.2.4.2 Τύποι Δεδομένων της XML Schema

Όπως παρουσιάστηκε παραπάνω οι χρήστες μπορούν να δημιουργήσουν νέους MPEG-7 Descriptors και σε κάποιες περιπτώσεις και Description Schemes. Τα schemes που δημιουργούν οι χρήστες αποτελούν XML αρχεία, τα οποία διαθέτουν elements, attributes και τύπους δεδομένων. Στην XML Schema Language της W3C περιλαμβάνονται κάποιοι *primitive*, θεμελιώδεις, τύποι δεδομένων, ένα σύνολο από *derived*, παραγόμενους, τύπους δεδομένων που παράγονται από τους primitive καθώς και μηχανισμοί που επιτρέπουν στους χρήστες τον ορισμό νέων τύπων δεδομένων ανάλογα με τις ανάγκες τους. Οι νέοι τύποι δεδομένων δημιουργούνται με επέκταση των ενσωματωμένων primitive και derived

datatypes και με εφαρμογή σε αυτούς *facets, list* και *unions*, εργαλεία που αναλύονται στη συνέχεια.

Facets

Οι χρήστες μπορούν να ορίσουν νέους τύπους δεδομένων εφαρμόζοντας περιορισμούς στους ήδη υπάρχοντες, αυτούς που είναι ενσωματωμένοι στην XML Schema και σε derived που έχουν ήδη δημιουργήσει. Αυτοί οι περιορισμοί ονομάζονται constraining facets και παρουσιάζονται στον Πίνακας 6 και στη συνέχεια δίνονται δύο παραδείγματα.

Name	Definition
length	Ορίζει τον αριθμό των μονάδων μήκους του value, όπως ορίζονται για το κάθε datatype (για string οι char)
minLength	Ορίζει τον ελάχιστο αριθμό των μονάδων μήκους του value
maxLength	Ορίζει τον μέγιστο αριθμό των μονάδων μήκους του value
pattern	Ορίζει το pattern που πρέπει να ακολουθεί το value
enumeration	Περιορίζει το value σε κάποιες συγκεκριμένες τιμές
whiteSpace	Μορφοποιεί τα white spaces του string εισόδου
maxInclusive	Ορίζει την inclusive max τιμή του value, value <= max
minInclusive	Ορίζει την inclusive min τιμή του value, value >= min
maxExclusive	Ορίζει την exclusive max τιμή του value, value < max
minExclusive	Ορίζει την exclusive min τιμή του value, value > min
totalDigits	Ορίζει το μέγιστο αριθμό ψηφίων για value τύπου decimal
fractionDigits	Ορίζει το μέγιστο αριθμό δεκαδικών ψηφίων για value τύπου decimal

Πίνακας 6 Facets of XSD

Στο παράδειγμα που ακολουθεί ορίζεται ο τύπος δεδομένων με όνομα `price`. Τα δεδομένα με datatype `price` είναι δεκαδικοί αριθμοί αυστηρά μεγαλύτεροι του μηδενός με το πολύ δύο δεκαδικά ψηφία.

```
<simpleType name = "price">
     <restriction base = "decimal">
          <minExclusive value = "0"/>
          <fractionDigits value = "2"/>
     </restriction>
</simpleType>
```

Ορίζεται το element `Item` που παίρνει δεδομένα τύπου `ItemFeatures` και δίνονται τα ακόλουθα παραδείγματα χρήσης του.

```
<element name = "Item" type = "ItemFeatures"/>
```

Example	Evaluation
`<Item ItemPrice = "10.21">foo</Item>`	✓
`<Item ItemPrice = "10"/>`	✓
`<Item ItemPrice = "-10.21"/>`	✗
`<Item ItemPrice = "10.213">foo</Item>`	✗
`<Item something = "else">foo</Item>`	✗

Στο παράδειγμα που ακολουθεί ορίζεται ο τύπος δεδομένων με όνομα `WeekDays`. Τα δεδομένα τύπου `WeekDays` μπορούν να πάρουν μία από τις ακόλουθες τιμές: `Monday`, `Tuesday`, `Wednesday`, `Thursday` ή `Friday`.

```
<simpleType name = "WeekDays">
     <restriction base = "string">
          <enumeration value = "Monday"/>
          <enumeration value = "Tuesday"/>
          <enumeration value = "Wednesday"/>
          <enumeration value = "Thursday"/>
          <enumeration value = "Friday"/>
     </restriction>
</simpleType>
```

List Datatype

Η XML Schema περιλαμβάνει ακόμα τον τύπο δεδομένων list. Δεδομένα τύπου list έχουν ως value μια ακολουθία πεπερασμένου μήκους με στοιχεία atomic values. Τα στοιχεία της list χωρίζονται με ένα white space. Οι χρήστες δεν μπορούν να παράγουν lists με επέκταση άλλων lists. Μπορούν ωστόσο να δημιουργήσουν νέα derived list datatypes χρησιμοποιώντας atomic datatypes και εφαρμόζοντας facets. Τα facets που μπορούν να χρησιμοποιηθούν παρουσιάζονται στον Πίνακας 7.

Name	Definition
length	Ορίζει το πλήθος στοιχείων που μπορεί να περιέχει η λίστα
minLength	Ορίζει το ελάχιστο πλήθος στοιχείων που μπορεί να περιέχει η λίστα
maxLength	Ορίζει το μέγιστο πλήθος στοιχείων που μπορεί να περιέχει η λίστα
pattern	Ορίζει το pattern που πρέπει να ακολουθεί το value
enumeration	Περιορίζει το value σε κάποιες συγκεκριμένες τιμές
whiteSpace	Μορφοποιεί τα white spaces του string εισόδου

Πίνακας 7 Facets used in List Datatype of XSD

Παρακάτω παρουσιάζεται ένα παράδειγμα ορισμού ενός list datatype με όνομα `stringList` και ενός δεύτερου list datatype με όνομα `FullName`. Δεδομένα τύπου `stringList` έχουν σαν value μια λίστα από `string`. Δεδομένα τύπου `FullName` έχουν σαν value μια `stringList`, η οποία έχει ακριβώς δύο στοιχεία.

```
<simpleType name = "stringList">
     <list itemType = "string"/>
</simpleType>
```

```
<simpleType name = "FullName">
     <restriction base = "stringList">
          <length value = "2"/>
     </restriction>
</simpleType>
```

Ακολουθούν μερικά παραδείγματα χρήσης του *FullName* list datatype.

Example	Evaluation
<customerName type="FullName">foo bar</customerName>	✓
<customerName type="FullName">foo 1</customerName>	✓
<customerName type="FullName">foo bar baz</customerName>	✗
<customerName type="FullName">foo</customerName>	✗
<customerName type="FullName">foo 1 2</customerName>	✗

Union Datatype

Με χρήση του union datatype, ομαδοποιούνται περισσότερα του ενός atomic και list datatypes σε ένα. Με αυτόν τον τρόπο ένα element ή attribute μπορεί να διατηρεί δεδομένα όχι ενός συγκεκριμένου datatype, αλλά όλων όσων περιέχονται στο union datatype. Οι datatypes που συμμετέχουν στον ορισμό ενός union datatype ονομάζονται memberTypes. Ακολουθεί ένα παράδειγμα ορισμού union datatype με memberTypes `integer` και `FullName`, της `stringList` που παρουσιάστηκε στην προηγούμενη ενότητα, και δύο παραδείγματα ορθής χρήσης του.

```
<element name = "CustomerID">
<simpleType>
          <union memberTypes = "integer FullName"/>
     </simpleType>
</element>
```

```
<CustomerID>123456789</CustomerID>
<CustomerID>foo bar</CustomerID>
```

Η σειρά με την οποία ορίζονται τα επιμέρους datatypes έχει σημασία, καθώς κατά την αναζήτηση του datatype των δεδομένων του value ελέγχονται με τη σειρά τα memberTypes και κλειδώνεται ο πρώτος datatype που ταιριάζει. Για να παρακαμφθεί αυτός ο μηχανισμός μπορεί να οριστεί ο τύπος δεδομένων του value κατά τον ορισμό του type των elements και attributes. Ακολουθεί ένα παράδειγμα ορισμού union datatype και τρία παραδείγματα κλήσης του και ο τύπος δεδομένων που ανατίθεται στο κάθε element.

```
<element name = "StringOrInt">
<simpleType>
          <union>
               <simpleType>
                    <restriction base = "string"/>
               </simpleType>
               <simpleType>
                    <restriction base = "integer"/>
               </simpleType>
          </union>
</simpleType>
</element>
```

Example	Datatype
<StringOrInt>foo</StringOrInt>	string
<StringOrInt>1</StringOrInt>	string
<StringOrInt type="integer">1</StringOrInt>	integer

5.2.4.3 Επεκτάσεις του MPEG-7

Εκτός από τα παραπάνω χαρακτηριστικά στην XML Schema Language που χρησιμοποιείται σαν DDL για το πρότυπο MPEG-7 προστέθηκαν και διάφορες προεκτάσεις. Με τα επιπλέον χαρακτηριστικά ικανοποιούνται οι ανάγκες περιγραφής αρχείων με multimedia περιεχόμενο. Σε αυτά περιλαμβάνονται array και matrix datatypes, typed references και κάποιοι derived datatypes για αναπαράσταση χρόνου και enumerated datatypes.

5.3 MPEG-21

Παρότι τα πρότυπα MPEG-1, MPEG-2 MPEG-4 και MPEG-7 προσέφεραν μεγάλες δυνατότητες στην ανάπτυξη εφαρμογών, δεν είχε επιτευχθεί πλήρης διαλειτουργικότητα στην μετάδοση multimedia περιεχομένου. Με τα προηγούμενα πρότυπα ήταν δυνατή η ανάπτυξη ξεχωριστών στοιχείων, τα οποία μπορούσαν να χρησιμοποιηθούν στην δημιουργία υποδομής για την μετάδοση και διαχείριση του multimedia περιεχομένου. Ωστόσο δεν υπήρχε τρόπος προσδιορισμού των σχέσεων αυτών των στοιχείων και του τρόπου διασύνδεσής τους για την παραγωγή πλήρως διαλειτουργικών εφαρμογών. Ήταν φανερό πως ήταν απαραίτητη η ανάπτυξη ενός προτύπου που θα όριζε τον τρόπο διασύνδεσης των διαθέσιμων στοιχείων, objects, metadata, και της ενοποίησης των διαφορετικών προτύπων.

Όπως αναφέρθηκε νωρίτερα, το MPEG-21 χρησιμοποιείται στην ανάπτυξη multimedia εφαρμογών, ενώ ταυτόχρονα παρέχει εργαλεία διαχείρισης και προστασίας της πνευματικής ιδιοκτησίας. Η πρώτη θεμελιώδης έννοια που ορίζει, ο User, αφορά σε κάθε οντότητα που αλληλεπιδρά με το περιβάλλον του MPEG-21 και κάνει χρήση των Digital Items. Ως Digital Item, που αποτελεί το δεύτερο θεμελιώδες στοιχείο του MPEG-21, ορίζεται κάθε αντικείμενο που συμμορφώνεται με το Digital Item Schema. Τα Digital Items είναι δομημένα, ψηφιακά αντικείμενα μέσα στο MPEG-21 framework με συγκεκριμένη αναπαράσταση, ταυτότητα και metadata. Αποτελούν τις θεμελιώδεις μονάδες μετάδοσης περιεχομένου εντός του framework.

Στο πρότυπο MPEG-21 ορίζεται το schema, δηλαδή ο τρόπος που πρέπει να δομούνται, τα αντικείμενα για να αποτελούν Digital Items καθώς και το framework για την περιγραφή των οντοτήτων. Προσφέρονται επίσης interfaces και πρωτόκολλα για την δημιουργία, διαχείριση, αναζήτηση, πρόσβαση, αποθήκευση και μεταφορά του multimedia περιεχομένου. Όσον αφορά στην πνευματική ιδιοκτησία παρέχονται εργαλεία για τον ορισμό δικαιωμάτων χρήσης των Digital Items σε ένα μεγάλο εύρος δικτύων και συσκευών. Το πρότυπο κάνει επίσης ικανή την παροχή διαλειτουργικότητας στις εφαρμογές και της διαφανούς πρόσβασης σε αυτές μέσω όλων των δικτύων και από όλες τις εναλλακτικές συσκευές. Τέλος μέσω interfaces είναι δυνατή η διαχείριση του περιεχομένου των εφαρμογών και των events που παρατηρούνται από τους χρήστες.

Κεφάλαιο 6
Αντικείμενο της Διπλωματικής

Στη διπλωματική αυτή μελετάται το πρόβλημα αποθήκευσης και αναζήτησης multimedia αρχείων. Η αναζήτηση αυτή πραγματοποιείται με βάση τα metadata, των δεδομένων που περιγράφουν το περιεχόμενο των αρχείων, και τα οποία υπακούν στο πρότυπο περιγραφής δεδομένων MPEG-7. Η ιδέα προέκυψε από το Project iPromotion, που ασχολείται με αποθήκευση και αναζήτηση αρχείων με 3D περιεχόμενο. Σε αυτό το κεφάλαιο παρουσιάζονται οι τεχνικές που χρησιμοποιούνται μέχρι σήμερα για την αναζήτηση δεδομένων με βάση τις MPEG-7 περιγραφές τους και προτείνονται καινούργιες τεχνικές που χρησιμοποιούν τις νέες τεχνολογίες που αναλύθηκαν στα προηγούμενα κεφάλαια.

6.1 Αναλυτική Περιγραφή Προβλήματος

Σε αυτό την ενότητα θα περιγραφεί η αναγκαιότητα της ανάπτυξης εργαλείων αναζήτησης περιεχομένου, θα παρουσιαστούν οι τεχνικές που εφαρμόζονται μέχρι σήμερα για την αναζήτηση multimedia αρχείων και θα αναφερθούν οι νέες προσεγγίσεις που προτείνονται από αυτή τη διπλωματική.

6.1.1 Αναζήτηση Περιεχομένου

Η ανάπτυξη του διαδικτύου συνέβαλλε αποφασιστικά στην διακίνηση ολοένα αυξανόμενου όγκου πληροφοριών. Καθώς η πρόσβαση σε αυτό είναι σε μεγάλο ποσοστό ελεύθερη, οι χρήστες του έχουν στη διάθεσή τους μια σχεδόν απεριόριστη πηγή πληροφοριών που αφορά σε πληθώρα διαφορετικών θεμάτων. Για τον εντοπισμό των ζητούμενων πληροφοριών θα ήταν απαραίτητη η γνώση τόσο του τι είναι αυτό το οποίο ενδιαφέρει τους χρήστες, όσο και της ακριβής τοποθεσίας των δεδομένων. Συχνά αυτό δεν είναι εφικτό, καθώς οι χρήστες δεν γνωρίζουν αφενός ακριβώς τι ψάχνουν και αφετέρου που είναι αποθηκευμένες οι πληροφορίες που τους ενδιαφέρουν.

Τη λύση στο παραπάνω πρόβλημα δίνει η τεχνική της αναζήτησης περιεχομένου. Αποτελεί μια όλο και πιο απαραίτητη υπηρεσία παροχής υπηρεσιών, καθώς δίνει στους χρήστες τα απαραίτητα εργαλεία για να εντοπίσουν εύκολα και γρήγορα τις πληροφορίες που αναζητούν. Ένα παράδειγμα υπηρεσίας αναζήτησης περιεχομένου είναι οι μηχανές αναζήτησης. Σε αυτές οι χρήστες υποβάλλουν το επιθυμητό περιεχόμενο και περιμένουν να τους επιστραφούν πληροφορίες για αυτό. Οι γενικές αρχές που ακολουθούν οι εφαρμογές που λειτουργούν ως μηχανές αναζήτησης είναι η αποδοχή ερωτημάτων από τους χρήστες, η επεξεργασία των ερωτημάτων, η υποβολή τους στη βάση δεδομένων που διατηρεί ο πάροχος και η παρουσίαση των αποτελεσμάτων πίσω στο χρήστη.

Διακρίνονται αρκετοί τρόποι για την υποβολή των ερωτημάτων, ανάλογα με τις ανάγκες των χρηστών και ανάλογα με τις δυνατότητες των παρόχων των μηχανών αναζήτησης. Ένας

από αυτούς αποτελεί η συμπλήρωση ενός text με βάση το οποίο θα επιστραφούν αποτελέσματα, ένας άλλος είναι η επιλογή συγκεκριμένων κριτηρίων και φίλτρων που έχουν οριστεί από τον πάροχο. Χρησιμοποιείται συχνά συνδυασμός των δύο προηγούμενων τρόπων καθώς και άλλοι για πιο εξειδικευμένες αναζητήσεις, όπως η υποβολή μιας εικόνας ή video για αναζήτηση αρχείων με παρόμοιο περιεχόμενο.

Από την πλευρά τους οι πάροχοι είναι υπεύθυνοι για την επεξεργασία των ερωτημάτων, την υποβολή τους στη βάση δεδομένων που είναι αποθηκευμένα τα δεδομένα και την παραγωγή έγκυρων αποτελεσμάτων, τα οποία οφείλεται να παρουσιάζονται στους χρήστες με απλό και κατανοητό τρόπο. Όσον αφορά στον τρόπο επεξεργασίας των ερωτημάτων που θέτουν οι χρήστες και του τρόπου παρουσίασης των απαντήσεων, οι πάροχοι υιοθετούν ένα από τα δύο ευρέως χρησιμοποιούμενα μοντέλα αναζήτησης και τα οποία περιγράφονται στη συνέχεια.

Πρώτο Μοντέλο

Στο πρώτο μοντέλο αναζήτησης, οι πάροχοι κάνουν την υπόθεση πως οι χρήστες δεν γνωρίζουν τον ακριβή τρόπο δόμησης της βάσης δεδομένων στην οποία υποβάλλονται τα queries. Παραδείγματα υπηρεσιών που εφαρμόζουν αυτήν την υπόθεση αποτελούν το Google Search και το eBay.

Οι βάσεις δεδομένων μπορούν να επεξεργαστούν queries όταν αυτά είναι γραμμένα στην κατάλληλη query language. Εφόσον οι χρήστες θεωρείται πως δεν έχουν τόσο εξειδικευμένες γνώσεις υποβολής ερωτημάτων, είναι απαραίτητη η επεξεργασία των queries που υποβάλλουν ώστε να είναι συμβατά με τη βάση δεδομένων. Αυτή η επεξεργασία ονομάζεται query expansion, εφαρμόζεται στο αρχικό query, το οποίο συνήθως είναι ένα text κείμενο, και με τροποποιήσεις δημιουργεί το τελικό query που τελικά θα υποβληθεί στη βάση.

Οι τροποποιήσεις που γίνονται στα ερωτήματα κατά το query expansion περιλαμβάνουν μεταξύ άλλων τις παρακάτω τεχνικές. Μέσω Term reordering, αναδιάταξη των όρων του query, ελαχιστοποιείται ο φόρτος εργασίας της βάσης. Για αύξηση της απόδοσης των queries εφαρμόζεται Stemming, με το οποίο εντοπίζεται η προέλευση των λέξεων του αρχικού query, και μπορούν να χρησιμοποιηθούν μικρότερες ισοδύναμες νοηματικά λέξεις οι οποίες βελτιστοποιούν τον χρόνο απόκρισης και την αποτελεσματικότητα των απαντήσεων. Παράλληλα διορθώνονται ως ένα βαθμό τυχόν ορθογραφικά λάθη των χρηστών. Τέλος τα queries εμπλουτίζονται με εξατομικευμένες πληροφορίες για τον κάθε χρήστη, με βάση του προφίλ του, παρέχοντας έτσι πιο στοχευμένες απαντήσεις σύμφωνα με τα προσωποποιημένα χαρακτηριστικά του καθενός. Πιο συγκεκριμένα με γνώση του ιστορικού των αναζητήσεων των χρηστών, παλαιότερων επιλογών τους και πληροφοριών για τον ίδιο, όπως τοποθεσία, φύλο, ηλικία, είναι δυνατή η επιλογή των αποτελεσμάτων που θα παρουσιαστούν στους χρήστες πιθανοτικά με βάση τι θεωρεί η βάση δεδομένων ότι τους ενδιαφέρει περισσότερο.

Είναι φανερό από τα παραπάνω ότι υπάρχει μια ασάφεια στον ακριβή τρόπο με τον οποίο υποβάλλονται τα queries στη βάση και έγκειται στην ευχέρεια του παρόχου το πώς θα τα χειριστεί και ποιες πληροφορίες τελικά θα παρουσιάσει στους χρήστες. Τα αποτελέσματα επιλέγονται με μία πιθανότητα εγκυρότητας, δηλαδή παρουσιάζονται αυτά τα οποία απαντούν στο query που υποβλήθηκε με φθίνουσα πιθανότητα να είναι έγκυρα, ορθά. Αυτό σημαίνει πως δεν ταιριάζουν όλα τα αποτελέσματα στο ακριβές query που υποβλήθηκε, αλλά συχνά σε μόνο μερικές από τις παραμέτρους του ή στο νόημα που θεωρείται ότι αναζητούσαν στην πραγματικότητα οι χρήστες.

Δεύτερο Μοντέλο

Στο δεύτερο μοντέλο απάντησης σε ερωτήματα, οι πάροχοι θεωρούν πως οι χρήστες ξέρουν στο μεγαλύτερο ποσοστό τον τρόπο ιεράρχησης και αποθήκευσης των δεδομένων στη βάση δεδομένων που υποβάλλονται τα queries. Εφόσον η δομή της βάσης είναι γνωστή οι πάροχοι τέτοιων εφαρμογών εστιάζουν στην ταχύτητα απόκρισης των queries, στην ακρίβεια των απαντήσεων που επιστρέφονται και στην εξασφάλιση της ορθότητας όλων των αποτελεσμάτων, δηλαδή όλα να είναι true. Αυτό αποτελεί και την κύρια διαφορά των δύο μοντέλων. Σε αντίθεση με το προηγούμενο μοντέλο, στο query που υποβάλλεται δεν προστίθενται πληροφορίες, ούτε αλλάζουν οι παράμετροί του. Όλα τα αποτελέσματα που επιστρέφονται απαντούν στο αρχικό query των χρηστών, είναι έγκυρα και δεν επιστρέφονται δεδομένα που δεν είναι true.

6.1.2 Σημερινές Τεχνικές

Μέχρι σήμερα για την αποθήκευση και αναζήτηση των MPEG-7 περιγραφών multimedia αρχείων χρησιμοποιούνται κατά κύριο λόγο δύο τεχνικές. Στην πρώτη η αναζήτηση περιεχομένου πραγματοποιείται απευθείας στα XML αρχεία με εφαρμογή τεχνικών XML Parsing. Η δεύτερη αφορά σε αποθήκευση των metadata σε RDBMS και αναζήτηση του περιεχομένου τους με εφαρμογή indexing και εξειδικευμένων queries στη βάση δεδομένων.

6.1.2.1 Αναζήτηση σε XML αρχεία

Όσον αφορά στην αναζήτηση πληροφορίας σε XML αρχεία διακρίνονται δύο κύριες προσεγγίσεις για τον τρόπο υποβολής των queries, η featured-based και η semantic-based. Με την featured-based προσέγγιση η αναζήτηση πραγματοποιείται στα δομικά, low-level χαρακτηριστικά των αρχείων, ενώ με την semantic-based τα queries που υποβάλλονται αφορούν στο σημασιολογικό περιεχόμενο των αρχείων που είναι πιο κοντά στην αντίληψη των τελικών χρηστών. Ωστόσο και οι δύο προσεγγίσεις επικεντρώνονται σε ένα πολύ μικρό ποσοστό της δομής και των περιγραφών περιεχομένου που περιλαμβάνονται στις MPEG-7 περιγραφές των multimedia αρχείων.

Έχουν προταθεί για την αναζήτηση περιεχομένου διάφορες querying languages που βασίζονται στην χρήση του XQuery [6] [7] [8]. Το XQuery είναι μια query language που έχει δημιουργηθεί από την W3C για την υποβολή queries σε συλλογές από XML αρχεία. Κομμάτι του XQuery αποτελεί το XPath, μιας γλώσσας υποβολής queries για τον εντοπισμό κόμβων σε ένα XML αρχείο σύμφωνα με το μονοπάτι στο οποίο βρίσκονται, όπως αυτό ορίζεται από την δενδρική μορφή του XML. Το μειονέκτημα αυτής της μεθόδου είναι ότι λαμβάνονται υπόψιν τα δομικά χαρακτηριστικά των metadata μόνο σαν XML αρχεία και όχι ως MPEG-7 περιγραφές. Χρήστες που εφαρμόζουν queries με χρήση τέτοιων εργαλείων θα ήταν απαραίτητο να έχουν εξειδικευμένη γνώση της δομής και του περιεχομένου των Ds και DSs του προτύπου MPEG-7. Ταυτόχρονα δεν είναι δυνατή η προσαρμογή των queries, ώστε να συμμορφώνονται με τις προσωποποιημένες απαιτήσεις των χρηστών στο περιεχόμενο που αναζητούν.

6.1.2.2 Αναζήτηση σε RDBMS

Κατά τη δεύτερη προσέγγιση αποθήκευσης και αναζήτησης multimedia αρχείων χρησιμοποιούνται για την αποθήκευση των metadata σχεσιακές βάσεις δεδομένων. Η εκτεταμένη χρήση των RDBMS σε πολλά είδη εφαρμογών, η αποδεδειγμένη μαθηματική θεωρία που τα υποστηρίζει, οι ενσωματωμένες τεχνικές βελτιστοποίησης των queries και οι προηγμένοι μηχανισμοί επεξεργασίας των δεδομένων είναι μερικά από τα πλεονεκτήματα

που κάνουν τις σχεσιακές βάσεις δεδομένων μια πολύ καλή λύση για την αποθήκευση MPEG-7 δεδομένων.

Κατά την δημιουργία μιας σχεσιακής βάσης δεδομένων, είναι απαραίτητη η δημιουργία ενός κατάλληλου schema με βάση το οποίο θα αποθηκεύονται τα δεδομένα. Όταν πρόκειται να αποθηκευτούν XML αρχεία και ιδιαίτερα MPEG-7 περιγραφές, για την δημιουργία του schema απαιτείται να ληφθούν υπόψιν τα ιδιαίτερα χαρακτηριστικά των MPEG-7 αρχείων. Σε αυτά περιλαμβάνονται μεταξύ άλλων το γεγονός ότι τα metadata, παρότι δομούνται σύμφωνα με τους κανόνες που ορίζει η XML Schema Language, χαρακτηρίζονται από πολλές ελευθερίες στον τρόπο αναπαράστασης της πληροφορίας. Παράλληλα λόγω των δυνατοτήτων της DDL είναι δυνατός ο ορισμός νέων τύπων δεδομένων και συνεπώς προκύπτει η ανάγκη αποθήκευσης πολύπλοκων παραγόμενων datatypes, όπως μια λίστα από integers, καθώς και η υποστήριξη εργαλείων για διαφορετικό χειρισμό των datatypes που χρησιμοποιούνται στο πρότυπο MPEG-7. Είναι φανερό πως είναι απαραίτητος ένας μηχανισμός mapping των MPEG-7 περιγραφών στο schema της σχεσιακής βάσης δεδομένων. Έχουν προταθεί πολλοί τρόποι υλοποίησης τέτοιου mapping. Όπως υποστηρίζεται στο [9] το schema με βάση το οποίο θα γίνει το mapping των XML αρχείων στο schema των σχεσιακών βάσεων δεδομένων χωρίζεται σε δύο κατηγορίες, το structure-mapping και το model-mapping.

Κατά το structure-mapping δίνεται βάση στην κατανόηση και χρήση του XML Schema σαν βοηθητικό εργαλείο για την αποθήκευση των δεδομένων. Για κάθε element type ορίζεται μια relation και χρησιμοποιούνται primary και foreign keys για την περιγραφή των ιεραρχικών σχέσεων που ορίζονται από την δενδρική μορφή του XML αρχείου. Το πλεονέκτημα αυτής της μεθόδου είναι η γρήγορη απόκριση κατά την υποβολή των queries, καθώς επιτυγχάνεται εύκολη εφαρμογή indexes με χρήση των ενσωματωμένων indexes της βάσης. Ωστόσο, δεν διατηρούνται πληροφορίες για την αρχική δομή του XML αρχείου. Αυτό κάνει την εφαρμογή πολύπλοκων queries που βασίζονται στο XPath αδύνατη καθώς τα δεδομένα του XML είναι τμηματοποιημένα, χωρίς δυνατότητα ανακατασκευής του αρχικού αρχείου. Η διατήρηση ολόκληρου του XML αρχείου είναι απαραίτητη σε multimedia αρχεία καθώς κατά πρωταρχικό λόγο έχουν δημιουργηθεί για ανταλλαγή πληροφοριών μεταξύ ετερογενών συστημάτων με καθολικά όμοιο τρόπο.

Όσον αφορά στο model-mapping επιλέγεται ένα σταθερό schema για την βάση δεδομένων με βάση το οποίο αποθηκεύονται τα XML αρχεία, χωρίς βοηθητική χρήση του XML Schema όπως στην περίπτωση του structure-mapping. Σε αυτήν την τεχνική ολόκληρη η δομή του XML αποθηκεύεται στη βάση, κάνοντας δυνατή τη χρήση του XPath για εφαρμογή πολύπλοκων queries και την ανακατασκευή του αρχικού αρχείου. Καθώς δεν χρησιμοποιείται βοηθητικό schema, όλα τα values του XML αποθηκεύονται σε μία column της σχεσιακής βάσης ως string. Αυτός ο τρόπος αποθήκευσης των metadata κάνει αδύνατη τόσο την ακριβή απεικόνιση όλων των διαφορετικών, πολύπλοκων datatypes που εμφανίζονται στο πρότυπο MPEG-7, όσο και την εφαρμογή indexes για γρήγορη απόκριση των queries.

Το σύστημα SM3, combined **S**tructure-**m**apping and **M**odel-**m**apping, [10] παρουσιάζει ακόμα μια μέθοδο αποθήκευσης, mapping, indexing και αναζήτησης MPEG-7 δεδομένων σε σχεσιακές βάσεις δεδομένων που συνδυάζει τις δύο παραπάνω τεχνικές. Αυτή η μέθοδος στόχο έχει την διατήρηση των πλεονεκτημάτων των μεθόδων structure-mapping και model-mapping και την αποφυγή των μεγαλύτερων μειονεκτημάτων τους. Τα XML αρχεία διαθέτουν δενδρική μορφή, η οποία ορίζει το XPath του κάθε leaf node. Οι εσωτερικοί κόμβοι, δηλαδή οι κόμβοι που έχουν child elements, απεικονίζουν την δομή του XML αρχείου και είναι χρήσιμοι για την διάσχιση του XML δένδρου. Αντίθετα οι leaf nodes διατηρούν τα δεδομένα του XML αρχείου. Είναι φανερό πως δεν μπορούν να χρησιμοποιηθούν για την διάσχιση του αρχείου, καθώς αποτελούν το τελευταίο σημείο του XPath. Για τους παραπάνω λόγους προτείνεται η αποθήκευση των εσωτερικών κόμβων, δηλαδή της εσωτερικής δομής του XML, με χρήση της model-mapping προσέγγισης και η αποθήκευση των leaf nodes με χρήση της structure-mapping τεχνικής. Με αυτόν τον τρόπο

είναι δυνατή τόσο η υποβολή πολύπλοκων XPath queries και ο ορισμός indexes, όσο και η αποθήκευση πολύπλοκων MPEG-7 datatypes. Καθώς όμως οι τιμές των leaf nodes είναι αποθηκευμένες σε διαφορετικά tables, παρατηρείται χαμηλή απόδοση στην περίπτωση queries που απαιτούν JOINS ή την επανακατασκευή του αρχικού XML αρχείου. Ένα ακόμα εμφανές μειονέκτημα αυτής της τεχνικής είναι η ανάγκη δημιουργίας εξειδικευμένου mapping schema για όλους τους MPEG-7 Ds και DSs.

6.1.2.3 Προσέγγιση της Διπλωματικής

Παρότι οι τεχνικές που αναφέρθηκαν στην προηγούμενη ενότητα παρέχουν λύσεις για πολλά είδη εφαρμογών, παρουσιάζουν ταυτόχρονα μειονεκτήματα όσον αφορά στο χρόνο απόκρισης των queries, στην αδυναμία καταμερισμού του φόρτου εργασίας και στην παροχή υπηρεσιών συμβατών με κάθε συσκευή και προγραμματιστικό περιβάλλον.

Για τους παραπάνω λόγους εξετάζεται από την διπλωματική αυτή η περίπτωση χρήσης τεχνικών cloud computing για την απομακρυσμένη παροχή υπηρεσιών αναζήτησης multimedia αρχείων. Η αποθήκευση των δεδομένων επιλέγεται να γίνει σε NoSQL βάσεις δεδομένων, καθώς σε αυτές τα αρχεία δεν απαιτείται να διαθέτουν όμοιο schema και ταυτόχρονα λόγω του clustering επιτυγχάνεται εύκολα καταμερισμός της επεξεργαστικής ισχύς που απαιτείται για την εκτέλεση των queries. Τέλος ελέγχεται ο τρόπος ανταπόκρισης της αναζήτησης στην περίπτωση εφαρμογής MapReduce.

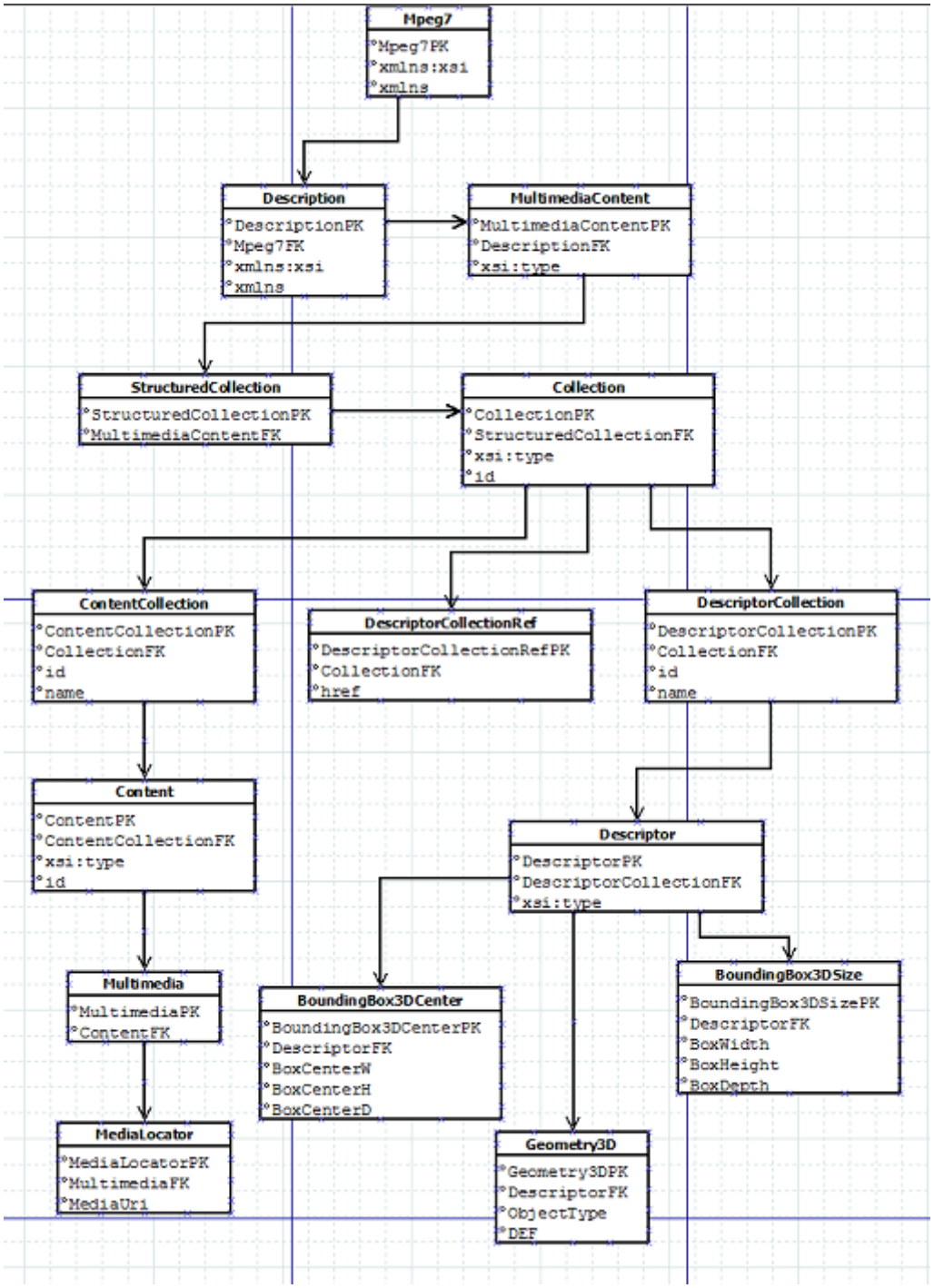
Όπως παρουσιάστηκε, μέχρι σήμερα κυριαρχούσαν δύο τρόποι αποθήκευσης των metadata για την αναζήτηση του περιεχομένου multimedia αρχείων. Ο πρώτος είναι η αποθήκευση των MPEG-7 περιγραφών, δηλαδή των XML αρχείων, σε directories και η εφαρμογή τεχνικών αναζήτησης των δομικών στοιχείων των XML δένδρων για τον εντοπισμό της ζητούμενης πληροφορίας. Ο δεύτερος είναι η μετατροπή των XML αρχείων και η αποθήκευσή τους σε σχεσιακές βάσεις δεδομένων, με την αναζήτηση να πραγματοποιείται με υποβολή εξειδικευμένων queries στη βάση.

Η διπλωματική αυτή προτείνει μια τρίτη προσέγγιση στην αποθήκευση και αναζήτηση multimedia αρχείων. Για την αποθήκευση των metadata των multimedia αρχείων επιλέχθηκαν οι NoSQL βάσεις δεδομένων και συγκεκριμένα η MongoDB, η οποία όπως είναι γνωστό, για την αποθήκευση των δεδομένων χρησιμοποιεί αρχεία τύπου JSON. Όσον αφορά στην αναζήτηση περιεχομένου, αυτή πραγματοποιείται στα key/value pairs των JSON αρχείων. Συνεπώς για την αποθήκευση και αναζήτηση πληροφορίας, είναι απαραίτητη η μετατροπή των XML αρχείων σε valid JSON documents. Για την καλύτερη σύγκριση των διαφορετικών τρόπων αποθήκευσης και ανάκτησης των metadata παρουσιάζεται ένα παράδειγμα XML αρχείου, Εικόνα 18, των tables που προκύπτουν από την αποθήκευσή του με structure-mapping σε μια σχεσιακή βάση δεδομένων, Εικόνα 19, και το JSON αρχείο που προκύπτει από το αρχικό XML και το οποίο θα αποθηκευτεί στην MongoDB, Εικόνα 20.

mpeg7Description.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Mpeg7 xmlns="urn:mpeg:mpeg7:schema:2001">
    <Description xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                                          xsi:type="ContentEntityType">
    <MultimediaContent xsi:type="MultimediaCollectionType">
        <StructuredCollection>
            <Collection xsi:type="ContentCollectionType" id="Textures">
                <ContentCollection id="Textures_N1002B"
                                            name="Textures_dad_Box1">
                    <Content xsi:type="MultimediaType"
                                            id="MovieTexture_N1003F">
                        <Multimedia>
                            <MediaLocator>
                                <MediaUri>"autopoieticbf.avi"</MediaUri>
                            </MediaLocator>
                        </Multimedia>
                    </Content>
                </ContentCollection>
            </Collection>
            <Collection xsi:type="DescriptorCollectionType"
                                                    id="Geometries">
                <DescriptorCollection id="Geometry_N10025"
                                                 name="Geometry_GROUND">
                    <Descriptor xsi:type="BoundingBox3DType">
                        <BoundingBox3DSize BoxWidth="-1" BoxHeight="-1"
                                                  BoxDepth="-1"/>
                        <BoundingBox3DCenter BoxCenterW="0" BoxCenterH="0"
                                                 BoxCenterD="0"/>
                    </Descriptor>
                    <Descriptor xsi:type="Geometry3DType">
                        <Geometry3D ObjectType="Box" DEF="GeoBox1"/>
                    </Descriptor>
                </DescriptorCollection>
                <DescriptorCollectionRef href="#GROUND"/>
            </Collection>
        </StructuredCollection>
    </MultimediaContent>
</Description>
</Mpeg7>
```

Εικόνα 18 MPEG-7 Description as XML Document



Εικόνα 19 MPEG-7 Description in RDBMS

Mpeg7Description.json


```
{
  "Mpeg7": {
    "-xmlns": "urn:mpeg:mpeg7:schema:2001",
    "Description": {
      "-xsi:type": "ContentEntityType",
      "-xmlns:xsi": "http://www.w3.org/2001/XMLSchema-instance",
      "MultimediaContent": {
        "-xsi:type": "MultimediaCollectionType",
        "StructuredCollection": {
          "Collection": [
            {
              "-xsi:type": "ContentCollectionType",
              "-id": "Textures",
              "ContentCollection": {
                "-id": "Textures_N1002B",
                "-name": "Textures_dad_Box1",
                "Content": {
                  "-xsi:type": "MultimediaType",
                  "-id": "MovieTexture_N1003F",
                  "Multimedia": {
                    "MediaLocator": { "MediaUri": "\"autopoieticbf.avi\"" }
                  }
                }
              }
            },
            {
              "-xsi:type": "DescriptorCollectionType",
              "-id": "Geometries",
              "DescriptorCollection": {
                "-id": "Geometry_N10025",
                "-name": "Geometry_GROUND",
                "Descriptor": [
                  {
                    "-xsi:type": "BoundingBox3DType",
                    "BoundingBox3DSize": {
                      "-BoxWidth": "-1",
                      "-BoxHeight": "-1",
                      "-BoxDepth": "-1"
                    },
                    "BoundingBox3DCenter": {
                      "-BoxCenterW": "0",
                      "-BoxCenterH": "0",
                      "-BoxCenterD": "0"
                    }
                  },
                  {
                    "-xsi:type": "Geometry3DType",
                    "Geometry3D": {
                      "-ObjectType": "Box",
                      "-DEF": "GeoBox1"
                    }
                  }
                ]
              },
              "DescriptorCollectionRef": { "-href": "#GROUND" }
            }
          ]
        }
      }
    }
  }
}
```


Εικόνα 20 MPEG-7 Description as JSON Document

Όπως φαίνεται από τα παραπάνω σχήματα, όταν χρησιμοποιούνται σχεσιακές βάσεις δεδομένων για την αποθήκευση των metadata, τα elements του XML αποθηκεύονται ως tables στην σχεσιακή βάση δεδομένων και τα attributes αποθηκεύονται σε columns του κάθε

table. Η δενδρική μορφή του XML αναπαρίσταται με την διασύνδεση των tables της βάσης με χρήση primary και foreign keys. Η αναζήτηση πραγματοποιείται με υποβολή queries στη βάση και επιστροφή των ονομάτων των multimedia αρχείων που το ικανοποιούν.

Όσον αφορά στην αποθήκευση των MPEG-7 περιγραφών στην MongoDB, κάθε XML αρχείο μετατρέπεται στο αντίστοιχο JSON Document, το οποίο διατηρεί key/value pairs που περιέχουν τις MPEG-7 περιγραφές. Για κάθε κόμβο του XML δημιουργείται ένα key/value pair. Στο key διατηρείται το όνομα του κόμβου, δηλαδή το element name, και στο value δημιουργείται ένα νέο JSON Object στο οποίο αποθηκεύονται τα attributes και τα πιθανά child nodes του κόμβου. Το κάθε attribute μετατρέπεται σε ένα key/value pair, όπου στο key αποθηκεύεται το όνομα του attribute και στο value η τιμή του. Αν ο κόμβος περιέχει child nodes, το κάθε ένα μετατρέπεται σε ένα key/value pair με το child node name να γίνεται το key και το value ένα νέο JSON Objects στο οποίο αποθηκεύονται τα δεδομένα του child node με την ίδια διαδικασία. Όμοια elements, δηλαδή elements με ίδιο όνομα, αποθηκεύονται σε JSON Arrays. Η αναζήτηση στα JSON Documents γίνεται με υποβολή ενός MongoDB query και επιστροφή των ονομάτων των αρχείων που το ικανοποιούν.

6.2 Αναλυτική Περιγραφή Υλοποίησης

Σε αυτή την ενότητα θα παρουσιαστεί αναλυτικά το πρόβλημα που εξετάστηκε καθώς και οι λύσεις που υλοποιήθηκαν. Για την συγκριτική μελέτη θα χρησιμοποιηθεί η αποθήκευση των metadata σε directories και η αναζήτηση σε αυτά με χρήση ενός XML Parser. Ακολουθεί αναλυτική περιγραφή των δεδομένων που χρησιμοποιήθηκαν, των διαφορετικών τρόπων αποθήκευσής τους και του ακριβούς τρόπου αναζήτησης πληροφοριών σε αυτά.

6.2.1 Δεδομένα

Όπως αναφέρθηκε στην αρχή του κεφαλαίου, τη διπλωματική αυτή ενέπνευσε το iPromotion, project που ασχολείται με την αναζήτηση αρχείων με 3D περιεχόμενο και που αποτελεί μια πλατφόρμα υψηλής κλιμάκωσης παροχής διαδραστικών διαφημίσεων εικονικής πραγματικότητας μέσω του Web. Αντικείμενο της διπλωματικής αποτελεί η αναζήτηση αρχείων 3D με βάση το περιεχόμενό τους.

Για την ολοκληρωμένη περιγραφή του περιεχομένου των 3D αρχείων είναι απαραίτητη η διατήρηση πληροφοριών τόσο για την περιγραφή του περιεχομένου τους, όσο και για την αναπαράσταση και περιγραφή των 3D γραφικών που χρησιμοποιούνται. Για την περιγραφή του περιεχομένου χρησιμοποιείται μια ανεπίσημη επέκταση του προτύπου MPEG-7, ενώ για την περιγραφή του 3D περιεχομένου το ISO πρότυπο X3D, στο οποίο τα δεδομένα αποθηκεύονται σε αρχεία τύπου XML. Είναι φανερό πως κάθε multimedia αρχείο με 3D περιεχόμενο περιγράφεται από δύο αρχεία τύπου XML, ένα για την X3D και ένα για την MPEG-7 περιγραφή του, που υπακούει στην ανεπίσημη επέκτασή του προτύπου. Συνεπώς η αναζήτηση περιεχομένου θα αφορά και στις δύο περιγραφές. Παρότι η επέκταση του MPEG-7 που χρησιμοποιείται δεν έχει προτοτυποποιηθεί κατά ISO και δεν έχει προστεθεί στην τωρινή έκδοση του MPEG-7, για λόγους ευκολίας η περιγραφή πριεχομένου που αφορά σε αυτήν θα αναφέρεται ως MPEG-7 περιγραφή.

Στη MongoDB θα αποθηκευτούν τα metadata των 3D αρχείων, δηλαδή οι X3D και MPEG-7 περιγραφές τους. Για κάθε 3D αρχείο, δημιουργείται ένα JSON document το οποίο περιέχει key/value pairs. Σε ένα key/value pair αποθηκεύεται το μοναδικό *_id* που δίνεται από την MongoDB και σε ένα δεύτερο το *name* του αρχείου. Χρησιμοποιούνται επίσης key/value pairs για την αποθήκευση των metadata. Θέμα της διπλωματικής είναι ποια συγκεκριμένα key/value pairs θα χρησιμοποιούνται για την παρουσίαση των X3D και MPEG-7 περιγραφών

του 3D αντικειμένου που περιγράφεται από τα αρχεία. Ένα προσχέδιο ενός JSON document παρουσιάζεται στην Εικόνα 21.

```
{
   "_id" : ObjectID("document_id"),
   "name" : "File1",
   "x3d" : "x3d_description",
   "mpeg7" : "mpeg7_description"
}
```

Εικόνα 21 Draft JSON Document

Για πλήρη εκμετάλλευση των δυνατοτήτων scalability και υψηλής απόδοσης που προσφέρει η MongoDB θεωρήθηκε πως όλα τα τμήματα των XML αρχείων έπρεπε να μετατραπούν σε key/value pairs προς σύνθεση ενός valid JSON document. Ωστόσο πολλές είναι οι εφαρμογές που χρησιμοποιούν συστήματα αναζήτησης περιεχομένου με απευθείας αναζήτηση σε directories στο δίσκο. Άλλες έχουν αναπτύξει ή χρησιμοποιούν εξειδικευμένους μηχανισμούς αναζήτησης σε XML για παράδειγμα μέσω XPath, μιας query language για αναζήτηση κόμβων σε XML αρχεία. Για τέτοιες εφαρμογές, που ήδη έχουν επενδύσει σε μηχανισμούς αναζήτησης και διαχείρισης δεδομένων, θα παρουσιαστούν δύο ξεχωριστές λύσεις που υλοποιήθηκαν και οι οποίες σκοπό έχουν να συνδυάσουν την τεχνολογία των NoSQL βάσεων δεδομένων με τις τεχνολογίες και τα εργαλεία που ήδη χρησιμοποιούνται. Έτσι αξιοποιείται όλος ο διαθέσιμος εξοπλισμός, ενώ παράλληλα δεν απαιτείται υψηλή κατανόηση και εξοικείωση με νέα εργαλεία από την πλευρά των διαχειριστών των εφαρμογών.

6.2.2 Λύσεις που Εξετάστηκαν

Εξετάστηκαν τέσσερεις διαφορετικοί τρόποι αποθήκευσης των metadata των 3D αρχείων. Ο πρώτος, που ονομάστηκε *Legacy*, αφορά στην αποθήκευση των XML αρχείων σε directories στο δίσκο και στην αναζήτηση απευθείας σε αυτά με χρήση ενός XML Parser. Αυτός ο τρόπος χρησιμοποιείται σαν αφετηρία για την συγκριτική μελέτη των παραδοσιακών τεχνικών με τις νέες που προτείνονται στη συνέχεια.

Στις νέες τεχνικές χρησιμοποιείται η MongoDB για την αποθήκευση των δεδομένων, με κάθε μία να διατηρεί διαφορετικά key/value pairs. Συγκεκριμένα τα δεδομένα όλων των τεχνικών αποθηκεύονται στην ορισμένη database σε διαφορετικές collections, ανάλογα με τη δομή των JSON Documents. Η πρώτη εναλλακτική τεχνική αποθήκευσης δεδομένων ονομάζεται *PathXML*, στην οποία τα δεδομένα αποθηκεύονται στην collection "*PathXML*". Στα key/value pairs διατηρείται το *directoryPath* στο οποίο βρίσκονται αποθηκευμένα τα XML αρχεία με τα metadata. Στην δεύτερη, που ονομάζεται *EmbeddedXML*, χρησιμοποιείται η collection "*EmbeddedXML*" και ολόκληρο το XML αρχείο των MPEG-7 και X3D περιγραφών αποθηκεύεται ως String στα *mpeg7* και *x3d* key/value pairs αντίστοιχα. Στην τελευταία τεχνική αποθήκευσης, η οποία ονομάστηκε *KeyValue*, ολόκληρο το XML μετατρέπεται σε JSON Document με την μεθοδολογία που παρουσιάστηκε για την Εικόνα 20. Για αυτή τη μέθοδο δημιουργείται η collection "*KeyValue*".

Όσον αφορά στην αναζήτηση υλοποιήθηκαν επτά διαφορετικοί τρόποι εκτέλεσής της. Ο πρώτος αφορά σε δεδομένα που είναι αποθηκευμένα με την τεχνική *Legacy* και ονομάζεται *SearchLegacy*. Αντίστοιχα για τις τεχνικές αποθήκευσης *PathXML* και *EmbeddedXML* υλοποιούνται οι μέθοδοι αναζήτησης *SearchPathXML* και *SearchEmbeddedXML*. Για τα δεδομένα που έχουν αποθηκευτεί με την τεχνική *KeyValue* υλοποιούνται δύο τρόποι αναζήτησης, οι *SearchKeyValueQueue* και *SearchKeyValuePath*. Υλοποιείται επίσης το προγραμματιστικό μοντέλο αναζήτησης MapReduce, σε δεδομένα που έχουν αποθηκευτεί

σύμφωνα με την *KeyValue* τεχνική. Αυτός ο τρόπος αναζήτησης, που στη συνέχεια θα ονομάζεται *MapReduce*, αναπτύχθηκε με στόχο την μελέτη της συμπεριφοράς του συστήματος σε queries που θα απαιτούσαν πολλά JOINS σε σχεσιακές βάσεις δεδομένων. Για την αξιολόγησή της ωστόσο ήταν απαραίτητη η μελέτη της απόκρισης του συστήματος στην περίπτωση που δεν χρησιμοποιούνται τα εργαλεία του προγραμματιστικού μοντέλου MapReduce για την εκτέλεση ενός όμοιου query. Για το σκοπό αυτό υλοποιείται ακόμα μια μέθοδος αναζήτησης, η *NoMapReduce*, με την οποία στα ίδια δεδομένα εκτελείται το ίδιο query με αυτό του *MapReduce*, αλλά χωρίς χρήση των μηχανισμών του MapReduce.

Ανάλογα με τον τύπο των queries που υποβάλλονται, διακρίνονται δύο κατηγορίες αναζήτησης, η *Search* και η *MapReduce*, κάθε μία από τις οποίες διαθέτει διαφορετικές τεχνικές για την εκτέλεση της αναζήτησης, ανάλογα με την μορφή των queries που εξετάζονται. Όσον αφορά στην κατηγορία *Search*, υποβάλλονται queries όμοια με αυτά των σχεσιακών βάσεων δεδομένων, που μπορούν δηλαδή να απαντηθούν άμεσα από την εξέταση των δεδομένων εισόδου. Αντίθετα στον τομέα μελέτης *MapReduce* εκτελούνται queries που απαιτούν κάποιου είδους aggregation των δεδομένων, δηλαδή χρησιμοποιούνται για την εξαγωγή αποτελεσμάτων που αφορούν τα δεδομένα συνολικά και μετά από εφαρμογή συναρτήσεων συγχώνευσής τους. Στις επόμενες ενότητες αναλύονται οι τεχνικές αποθήκευσης και αναζήτησης δεδομένων σύμφωνα με τον τομέα μελέτης που ανήκει η κάθε μία.

6.2.3 Τομέας Μελέτης *Search*

Όλες οι τεχνικές αναζήτησης που ανήκουν στον τομέα μελέτης *Search*, οι *Legacy*, *PathXML*, *EmbeddedXML*, *KeyValueQueue* και *KeyValuePath*, υλοποιήθηκαν με χρήση της γλώσσας προγραμματισμού Java. Για τις ανάγκες της διπλωματικής θεωρήσαμε πως η αναζήτηση επιστρέφει τα ονόματα των 3D αρχείων, τα οποία σε κάποια από τις MPEG-7 ή X3D περιγραφές τους, περιέχουν τουλάχιστον ένα element με όνομα *elementName*, το οποίο έχει ένα attribute με όνομα *attributeName* και τιμή *attributeValue*. Τα *elementName*, *attributeName* και *attributeValue* εισάγονται από τους χρήστες. Για την σύγκριση των τεχνικών *Legacy*, *PathXML*, *EmbeddedXML*, *KeyValueQueue* και *KeyValuePath* υποβλήθηκαν 6 διαφορετικά Queries για έλεγχο απόκρισης του συστήματος σε διαφορετικές αναζητήσεις. Τα Queries παρουσιάζονται αναλυτικά στον Πίνακας 8.

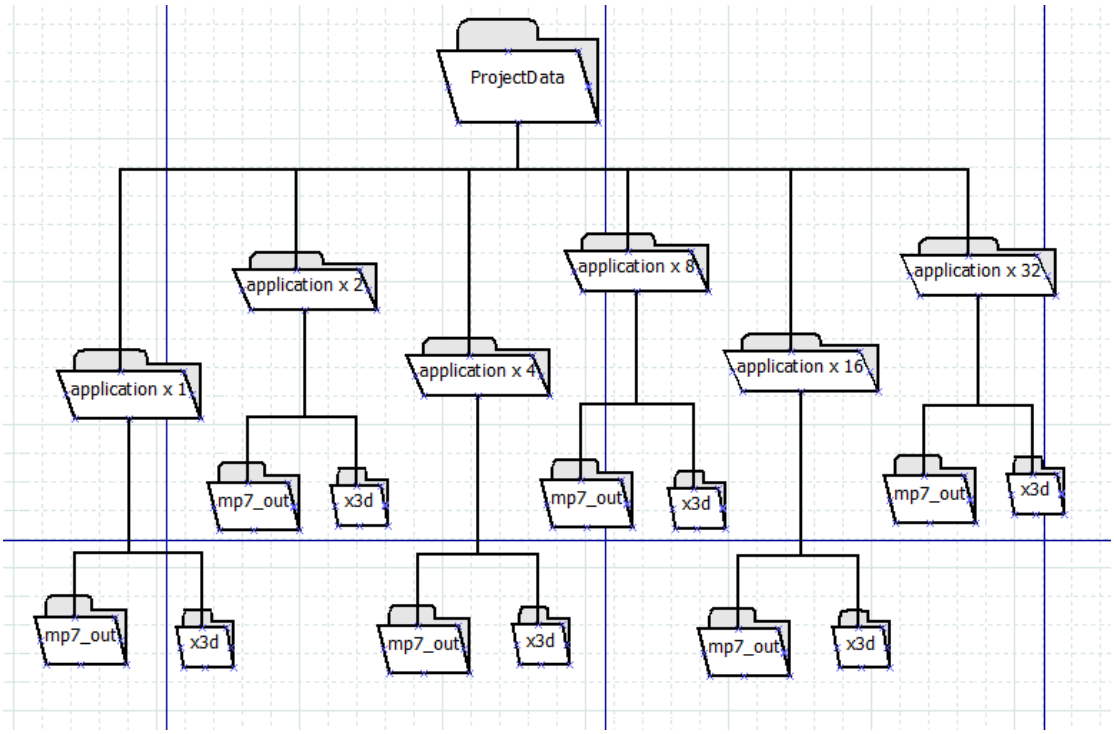
Query	elementName	attributeName	attributeValue
Q1	Collection	id	Transformations
Q2	Cylinder	radius	0.5
Q3	Geometry3D	ObjectType	Box
Q4	meta	name	title
Q5	Transform	DEF	Trans
Q6	X3D	profile	Immersive

Πίνακας 8 Examined Queries

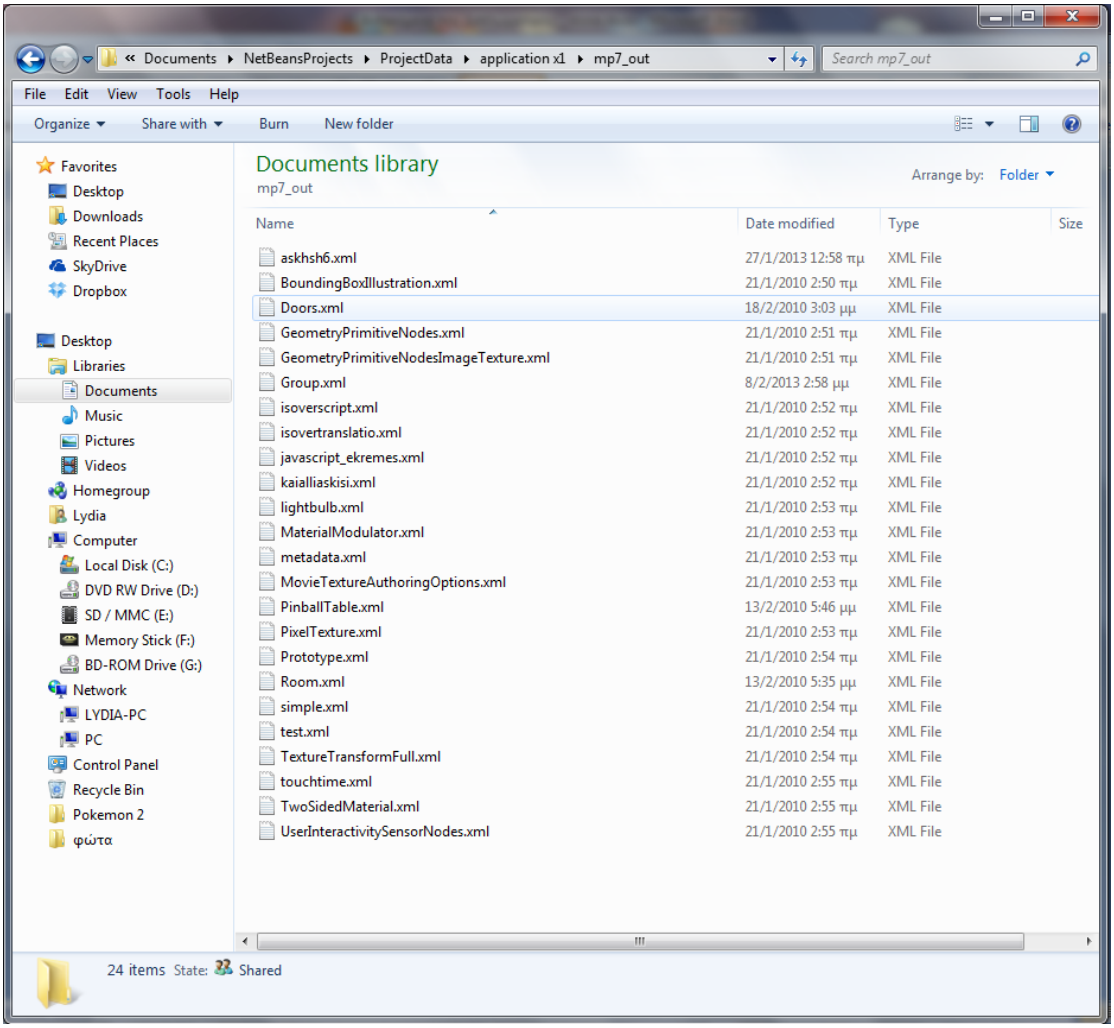
Τα δεδομένα στα οποία θα πραγματοποιηθεί η αναζήτηση είναι αποθηκευμένα σε ένα φάκελο που ονομάστηκε *ProjectData*. Για την σύγκριση του scalability των μεθόδων η αναζήτηση έπρεπε να πραγματοποιηθεί σε ολοένα και μεγαλύτερο όγκο δεδομένων. Επιλέχθηκε σε κάθε αναζήτηση να διπλασιάζεται το πλήθος των αρχείων 3D προς εξέταση, επομένως να διπλασιάζονται σε αριθμό και οι X3D και MPEG-7 περιγραφές τους. Για το σκοπό αυτό δημιουργήσαμε 6 φακέλους που ονομάστηκαν "*application x1*", "*application x2*", "*application x4*", "*application x8*", "*application x16*" και "*application x32*", οι οποίοι περιέχουν τα metadata των 3D αρχείων σε 1, 2, 4, 8, 16 και 32 αντίγραφα αντίστοιχα. Κάθε φάκελος "*application xNoCopies*", όπου *NoCopies* είναι 1, 2, 4, 8, 16 ή 32, περιέχει δύο

φακέλους, τους *mp7_out* και *x3d*, οι οποίοι περιέχουν αντίστοιχα τις MPEG-7 και X3D περιγραφές των 3D αρχείων. Στις μεθόδους που χρησιμοποιούν, σε κάποιο στάδιο της αναζήτησης, την MongoDB, έχουν δημιουργηθεί 6 databases, οι "*ipromx1*", "*ipromx2"*, "*ipromx4"*, "*ipromx8"*, "*ipromx16*" και "*ipromx32*", οι οποίες αποθηκεύουν τα metadata των 3D αρχείων όταν αυτά βρίσκονται σε 1, 2, 4, 8, 16 και 32 αντίγραφα αντίστοιχα. Κάθε database περιέχει τις τρεις collections που αναφέρθηκαν προηγουμένως, "*PathXML*", "*EmbeddedXML*" και "*KeyValue*".

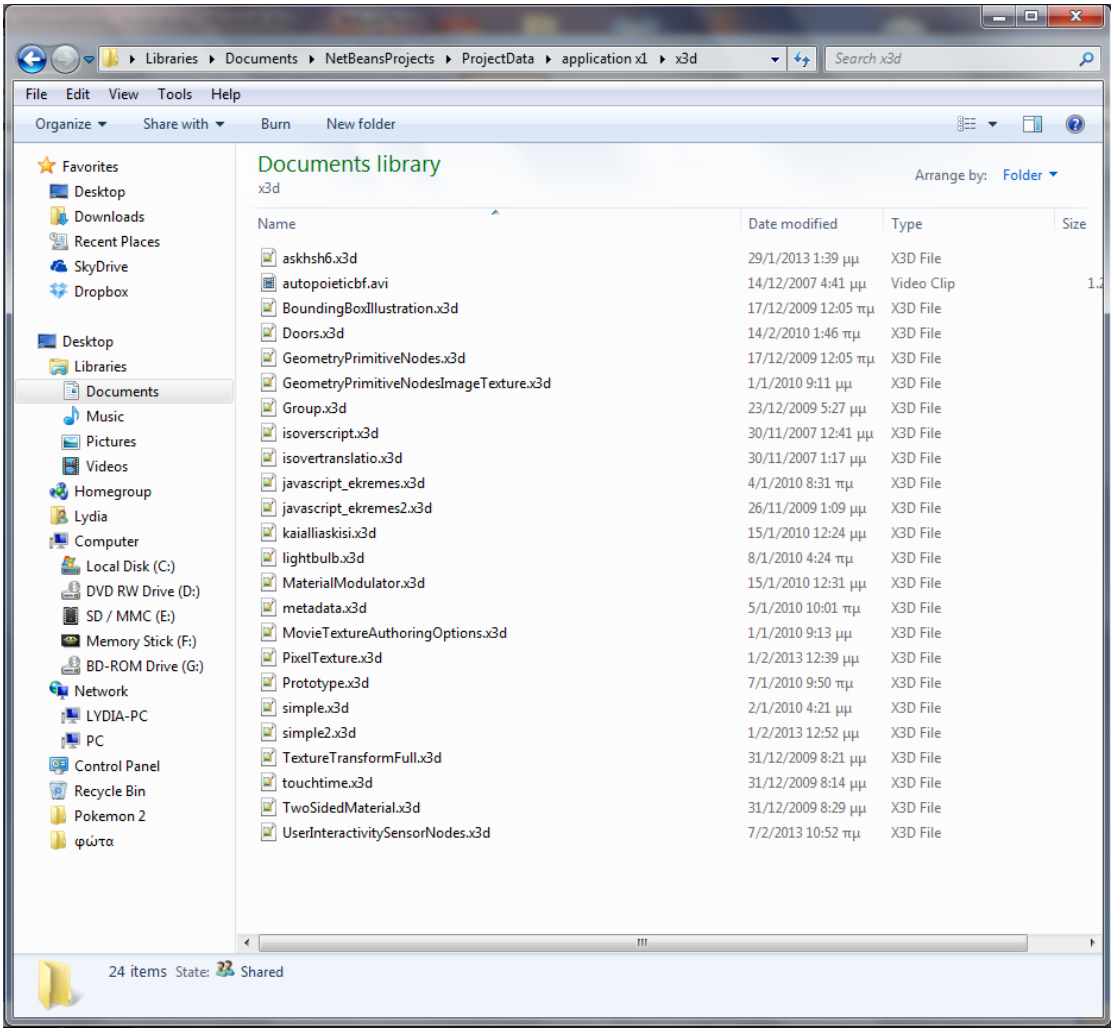
Η σύγκριση των μεθόδων πραγματοποιείται με υποβολή των 6 διαφορετικών Queries και υπολογισμού του χρόνου απόκρισης της αναζήτησης για όλα τα αντίγραφα των metadata. Ακολουθεί μια σχηματική αναπαράσταση του τρόπου ιεράρχησης των directories στην Εικόνα 22, καθώς και οι MPEG-7 και X3D περιγραφές των αρχείων 3D που χρησιμοποιήθηκαν στις Εικόνα 23 και Εικόνα 24 αντίστοιχα.



Εικόνα 22 ProjectData



Εικόνα 23 MPEG-7 files



Εικόνα 24 X3D files

Η αναζήτηση εκτελείται με κλήση της class *SearchCompareAll*. Σε αυτήν επιλέγεται κάθε φορά ένα Query προς υποβολή και μία προς μία όλες οι τεχνικές που εξετάζονται, με κλήση των classes *SearchLegacy*, *SearchPathXML*, *SearchEmbeddedXML*, *SearchKeyValueQueue* και *SearchKeyValuePath*. Για κάθε μέθοδο το Query εκτελείται στα αρχικά δεδομένα και όταν αυτά βρίσκονται σε 2, 4, 8, 16 και 32 αντίγραφα. Για διασταύρωση των αποτελεσμάτων και έλεγχο της ορθής λειτουργίας της αναζήτησης, καταγράφονται για κάθε μέθοδο οι απαντήσεις που επιστρέφονται. Το αρχείο στο οποίο αποθηκεύονται τα αποτελέσματα εκτέλεσης των Queries θα ονομάζεται *Αρχείο Αποτελεσμάτων*.

Καταγράφονται επίσης οι χρόνοι εκτέλεσης των Queries, ώστε να διαπιστωθεί το scalability της κάθε μεθόδου. Για την αποθήκευσή τους δημιουργείται στον φάκελο "*Duration*", αν δεν υπάρχει, ο φάκελος "*Query*". Σε αυτόν αποθηκεύονται 5 αρχεία για το κάθε Query, ένα για κάθε μέθοδο που χρησιμοποιείται στην αναζήτηση. Το όνομα του αρχείου είναι "*elementName attributeName attributeValue methodUsed*.txt", όπου *methodUsed* μπορεί να είναι *SearchLegacy*, *PathXML*, *EmbeddedXML*, *KeyValueQueue* ή *KeyValuePath*. Κάθε αρχείο περιέχει 6 μετρήσεις, μία για κάθε αριθμό αντιγράφων των metadata. Οι μετρήσεις υπολογίζονται με χρήση της μεθόδου *currentTimeMillis*, η οποία καλείται πριν την εκτέλεση της αναζήτησης και μετά την επιστροφή της. Η διαφορά των δύο χρόνων αποτελεί τον συνολικό χρόνο απόκρισης του Query και καταγράφεται στο αρχείο, το οποίο θα ονομάζεται *Αρχείο Χρόνου Απόκρισης*.

6.2.3.1 Legacy

Σε αυτήν την τεχνική τα metadata είναι αποθηκευμένα σε directories στο δίσκο. Πιο συγκεκριμένα χρησιμοποιούνται δύο φάκελοι, *mp7_out* και *x3d*, οι οποίοι περιέχουν τα XML αρχεία με τις MPEG-7 και X3D περιγραφές των 3D αρχείων αντίστοιχα. Η αναζήτηση των 3D αρχείων, αφορά στον εντοπισμό εκείνων που έχουν στα metadata τους τουλάχιστον έναν κόμβο με τα δοσμένα elementName, attributeName και attributeValue. Στην *Legacy* τεχνική, η αναζήτηση πραγματοποιείται με χρήση των διαθέσιμων εργαλείων διάσχισης και αναζήτησης πληροφορίας σε XML αρχεία. Το εργαλείο που θα χρησιμοποιηθεί από αυτήν την διπλωματική περιγράφεται στη συνέχεια.

Όπως έχει αναφερθεί στα XML αρχεία τα δεδομένα έχουν δενδρική δομή. Πολλά προγραμματιστικά περιβάλλονται διαθέτουν έτοιμα εργαλεία για την σύνδεσή τους με το περιεχόμενο των XML αρχείων. Η πρόσβαση σε αυτό γίνεται αφού τα δεδομένα μετασχηματιστούν σε μια δομή κατανοητή από το προγραμματιστικό περιβάλλον. Μια από τις δομές αυτές είναι το JDOM, Java Document Object Model, με το οποίο και μέσω του αντίστοιχου Java API είναι δυνατή η αναπαράσταση της δενδρικής μορφής των XML αρχείων με το αντίστοιχο, κατανοητό από τη Java, αντικειμενοστραφές δένδρο. Συνεπώς τα JDOM Documents αποτελούν την δομή που επιτρέπει την πρόσβαση και επεξεργασία των objects των XML αρχείων. Για την παραγωγή του JDOM, και κάθε άλλης αναπαράστασης XML αρχείων, είναι απαραίτητη η χρήση ενός parser. Ο parser είναι υπεύθυνος για την αντιστοίχηση της δενδρικής δομής του XML στη δομή που είναι κατανοητή από την χρησιμοποιούμενη γλώσσα προγραμματισμού. Ένας από τους ευρέως χρησιμοποιούμενους parsers, είναι ο SAX Parser, ο οποίος διαβάζει XML αρχεία και επεξεργάζεται τα τμήματά τους σειριακά. Η Java παρέχει την Class SAXBuilder η οποία χρησιμοποιεί έναν SAX parser, για την παραγωγή του JDOM Document που αντιστοιχεί στο XML αρχείο. Από αυτή τη διπλωματική θα χρησιμοποιεί για τη διάσχιση των XML αρχείων και για εντοπισμό των ζητούμενων κόμβων ο SAXBuilder και το παραγόμενο JDOM δένδρο.

Είναι φανερό πως τα δεδομένα βρίσκονται ήδη στην μορφή στην οποία θα εκτελεστεί η αναζήτηση, για αυτό το λόγο για υλοποίηση του *Legacy* απαιτείται μόνο η δημιουργία των Java classes που θα εφαρμόσουν την αναζήτηση. Ακολουθεί αναλυτική περιγραφή των classes που υλοποιήθηκαν και των μεθόδων που τις απαρτίζουν.

6.2.3.1.1 Αναζήτηση

Όπως αναφέρθηκε, για την αναζήτηση *Legacy* καλείται η class *SearchCompareAll*, η οποία μέσω της *SearchLegacy* θα εκτελέσει τα 6 διαφορετικά Queries που έχουν επιλεγεί για τις ανάγκες του προβλήματος. Κάθε Query υποβάλλεται στα αρχικά δεδομένα και στη συνέχεια σε 2, 4, 8, 16 και 32 αντίγραφά τους.

Ο Constructor της *SearchLegacy* δημιουργεί στον φάκελο "*application xNoCopies*", τον φάκελο "*Legacy*", αν δεν υπάρχει, και σε αυτόν έναν ακόμα που ονομάζεται "*results*". Στον φάκελο "*results*" θα αποθηκευτούν στο *Αρχείο Αποτελεσμάτων* με όνομα "*elementName attributeName attributeValue*.txt" τα αποτελέσματα του Query που εκτελέστηκε.

Η *SearchLegacy* έχει ως είσοδο τα *elementName, attributeName* και *attributeValue* που ορίστηκαν, καθώς και τους φακέλους *mp7_out* και *x3d* που περιέχουν τα metadata. Η *SearchLegacy* καλεί την μέθοδο *GetFiles*, για επιλογή των αρχείων στα οποία θα πραγματοποιηθεί η αναζήτηση, και στην συνέχεια την μέθοδο *SubmitQuery*, η οποία θα εκτελέσει το Query στα αρχεία που επέστρεψε η *GetFiles*.

Η *GetFiles* με χρήση δύο φίλτρων, των *FilterXMLFiles* και *FilterX3DFiles*, εντοπίζει από τα δοσμένα directories μόνο τα αρχεία που αποτελούν MPEG-7 και X3D περιγραφές αντίστοιχα. Με αυτόν τον τρόπο ελέγχεται η πραγματοποίηση της αναζήτησης σε valid αρχεία και όχι σε αρχεία που δεν αποτελούν metadata και που ενδεχομένως βρίσκονται

αποθηκευμένα στους *mp7_out* και *x3d* φακέλους. Ένα παράδειγμα τέτοιου αρχείου είναι το "*autopoieticbf.avi*" που βρίσκεται στο φάκελο *x3d*, όπως φαίνεται και στην Εικόνα 24.

Η μέθοδος *SubmitQuery* καλεί για κάθε ένα από τα αρχεία εισόδου της, την μέθοδο *QueryXMLFile* της class *NavigateXML*, με είσοδο τα *elementName, attributeName, attributeValue* καθώς και το XML αρχείο που εξετάζεται. Αν η *QueryXMLFile* επιστρέψει true, η *SubmitQuery* γράφει το όνομα του XML αρχείου στο *Αρχείο Αποτελεσμάτων*.

Η *NavigateXML* χρησιμοποιεί έναν SAX Parser και δημιουργεί το JDOM Document που αντιστοιχεί στο αρχείο εισόδου. Στη συνέχεια μέσω του Java API του JDOM πραγματοποιείται η διάσχιση του XML αρχείου και εντοπίζεται, αν υπάρχει, ο κόμβος με τα δοσμένα *elementName, attributeName* και *attributeValue*. Αν υπάρχει τέτοιος κόμβος, επιστρέφεται true, διαφορετικά επιστρέφεται false.

Τέλος, αφότου επιστρέψει η *SearchLegacy* καταγράφεται ο χρόνος εκτέλεσής της στο path "*Duration\Query*" στο αρχείο "*elementName attributeName attributeValue SearchLegacy*.txt".

Όταν η *SearchCompareAll* εκτελέσει την *SearchLegacy* για όλα τα αντίγραφα των δεδομένων, το *Αρχείο Χρόνου Απόκρισης* θα περιέχει 6 μετρήσεις. Η κάθε μία θα αφορά στον χρόνο εκτέλεσης της αναζήτησης στα δεδομένα όταν αυτά συναντώνται σε 1, 2, 4, 8, 16 και 32 αντίγραφα αντίστοιχα.

6.2.3.2 PathXML

Πολλές εφαρμογές χρησιμοποιούν εξειδικευμένα εργαλεία αναζήτησης XML αρχείων σε directories. Για τέτοιες περιπτώσεις επιλέχθηκε η αποθήκευση του path που βρίσκονται οι MPEG-7 και X3D περιγραφές σε αντίστοιχα key/value pairs στα JSON Documents της MongoDB. Εξετάστηκε κατά πόσον η λύση αυτή, θα βελτίωνε τον χρόνο απόκρισης μιας αναζήτησης, εντοπίζοντας γρηγορότερα τα ζητούμενα αρχεία, ενώ παράλληλα θα εκμεταλλευόταν τον υπάρχον εξοπλισμό διάσχισης XML αρχείων. Για αποτελεσματικότερη σύγκριση με την τεχνική *Legacy*, και σε αυτήν την μέθοδο αναζήτησης θα χρησιμοποιηθεί το JDOM δένδρο που παράγεται από τον SAXBuilder της Java.

Η παραπάνω τεχνική ονομάζεται *PathXML*. Σε αυτήν τα δεδομένα διατηρούνται στην database "*ipromxNoCopies*", όπου *NoCopies* είναι 1, 2, 4, 8, 16 ή 32, στην collection "*PathXML*". Τα αρχεία που θα αποθηκευτούν σε αυτή είναι JSON Documents, καθένα από τα οποία αφορά ένα συγκεκριμένο 3D αρχείο. Τα JSON Documents αποτελούνται από key/value pairs, όπως αυτά παρουσιάζονται στον Πίνακας 9. Στο key *_id* αποθηκεύεται το μοναδικό id που δίνει η MongoDB σε κάθε JSON Document που δημιουργείται. Στα keys *Mpeg7* και *X3D* αποθηκεύονται αντίστοιχα το *directoryPath* των metadata που αφορούν στην MPEG-7 και X3D περιγραφή του αρχείου 3D. Τέλος στο *name* διατηρείται το όνομα του 3D αρχείου το οποίο αφορούν τα metadata.

PathXML JSON Document

```
{
  "_id" : "ObjectID",
  "name" : "fileName",
  "X3D" : "ProjectData\\application xNoCopies\\x3d\\file.x3d",
  "Mpeg7" : "ProjectData\\application xNoCopies\\mp7_out\\file.xml"
}
```

Πίνακας 9 JSON Document in PathXML Collection

Για την αποθήκευση των αρχείων σύμφωνα με τον παραπάνω τρόπο είναι απαραίτητη η δημιουργία documents, συμβατών με τη βάση δεδομένων, που να περιέχουν τις ζητούμενες πληροφορίες των XML αρχείων με τα metadata. Για το λόγο αυτό η εκτέλεση της

αναζήτησης με την τεχνική *PathXML* απαιτεί πρώτα την δημιουργία των JSON Documents που θα αποθηκευτούν στην MongoDB. Στις επόμενες ενότητες περιγράφονται αναλυτικά οι διαδικασίες που ακολουθήθηκαν τόσο για την δημιουργία των JSON αρχείων, όσο και για την εκτέλεση της αναζήτησης.

6.2.3.2.1 Δημιουργία JSON Documents

Τα JSON Documents δημιουργούνται με χρήση της class *CreatePathXML*, η οποία εκτελείται για τα αρχικά δεδομένα όταν αυτά βρίσκονται σε 1, 2, 4, 8, 16 και 32 αντίγραφα. Αφότου επιστρέψει η *CreateFilesAll*, θα έχει εισαχθεί στις βάσεις "*ipromx1*", "*ipromx2*", "*ipromx4*", "*ipromx8*", "*ipromx16*" και "*ipromx32*", από μία collection "*PathXML*", που για κάθε 3D αρχείο δημιουργεί ένα JSON Document που περιέχει key/value pairs με τις X3D και MPEG-7 περιγραφές που το χαρακτηρίζουν. Ακολουθεί αναλυτική περιγραφή των βημάτων που ακολουθούνται για την δημιουργία και αποθήκευση των JSON Documents.

Η δημιουργία των αρχείων που θα αποθηκευτούν στην collection "*PathXML*", γίνεται με κλήση της class *CreateFilesAll*. Η *CreateFilesAll* καλεί την *CreatePathXML*, η οποία θα εξάγει από τα metadata, όταν αυτά βρίσκονται σε 1, 2, 4, 8, 16 και 32 αντίγραφα, τις κατάλληλες πληροφορίες για την δόμηση των JSON Documents που θα αποθηκευτούν στην MongoDB.

Ο Constructor της *CreatePathXML* παίρνει σαν είσοδο τον φάκελο "*application xNoCopies*" και τη βάση δεδομένων, "*ipromxNoCopies*". Από τον φάκελο "*application xNoCopies*" εξάγονται οι φάκελοι *mp7_out* και *x3d*, οι οποίοι διατηρούν τις MPEG-7 και X3D περιγραφές των 3D αρχείων. Ορίζεται ακόμα η collection "*PathXML*" της "*ipromxNoCopies*", στην οποία τελικά θα αποθηκευτούν τα παραγόμενα JSON Documents.

Κατ' αρχάς θεωρήθηκε πως κάθε φορά που απαιτείται εισαγωγή νέων δεδομένων σε μια collection της MongoDB, αυτή θα πρέπει να διαγραφεί και στην συνέχεια να δημιουργηθεί ξανά με τα νέα δεδομένα. Συνεπώς κατά την εκτέλεσή της, η *CreatePathXML* αρχικά διαγράφει την collection στην οποία θα εισαχθούν τα νέα δεδομένα. Στη συνέχεια χρησιμοποιούνται τα φίλτρα *FilterXMLFiles* και *FilterX3DFiles*, που αναλύθηκαν στο *Legacy*, για εντοπισμό των αρχείων των *mp7_out* και *x3d* φακέλων, που αποτελούν MPEG-7 και X3D περιγραφές αντίστοιχα. Το κάθε φίλτρο επιστρέφει μια List με τα αρχεία που ικανοποιούν τα ορίσματά του. Για την δημιουργία και εισαγωγή των JSON Documents στην "*PathXML*" collection, η *CreatePathXML* καλεί την μέθοδο *AddJSONFilesToMongoDB* με όρισμα την List που επέστρεψε το κάθε φίλτρο.

Η *AddJSONFilesToMongoDB* είναι υπεύθυνη για την δημιουργία των κατάλληλων key/value pairs που αφορούν στο κάθε 3D αρχείο και προσθήκη τους στο αντίστοιχο JSON Document. Συγκεκριμένα για κάθε αρχείο εισόδου, εξάγεται το όνομα του 3D αρχείου που περιγράφεται από τα metadata, ο τύπος, MPEG-7 ή X3D, της περιγραφής, και το *directoryPath* στο οποίο είναι αποθηκευμένη. Το όνομα του αρχείου ονομάζεται *fileName*, ο τύπος των metadata *key* και το *directoryPath value*. Στη συνέχεια με εφαρμογή του MongoDB-Query 2 εντοπίζεται, αν υπάρχει, το JSON Document με ένα key/value pairs με τιμές *name*/*fileName*. Από το query επιστρέφεται το πολύ ένα JSON Document.

```
db.PathXML.find( { name : fileName } );
```

MongoDB-Query 2 Search PathXML Collection for JSON with "name":"fileName"

Αν το query δεν επιστρέψει JSON Document, δεν έχει εισαχθεί ακόμα κάποια περιγραφή metadata για το 3D αρχείο που εξετάζεται. Σε αυτήν την περίπτωση χρησιμοποιείται το MongoDB-Query 3 το οποίο δημιουργεί ένα JSON Document με ένα key/value pairs με τιμές *name*/*fileName*. Από την MongoDB εισάγεται αυτόματα και το key/value pair με το *_id* που

θα χαρακτηρίζει μοναδικά το αρχείο. Στο JSON Document που μόλις δημιουργήθηκε θα αποθηκευτούν στη συνέχεια οι MPEG-7 και X3D περιγραφές που αφορούν το συγκεκριμένο 3D αρχείο. Για το σκοπό αυτό καλείται ξανά το MongoDB-Query 2, το οποίο αυτή τη φορά θα επιστρέψει το JSON Document που μόλις δημιουργήθηκε και σε αυτό θα εισάγει την περιγραφή των metadata με χρήση του update που αναλύεται στη συνέχεια.

```
db.PathXML.insert( { name : fileName } );
```

MongoDB-Query 3 Insert into PathXML Collection JSON with "name":"fileName"

Αν το query επιστρέψει JSON Document, τότε αρκεί ένα update για την εισαγωγή του *directoryPath* της metadata περιγραφής του 3D αρχείου. Το update γίνεται με χρήση του MongoDB-Query 4. Το *key* είναι *Mpeg7* ή *X3D*, ανάλογα με τον τύπο της περιγραφής, και το *value* είναι το αντίστοιχο *directoryPath* των metadata.

```
db.PathXML.update({name : fileName}, {$set : {key : value}});
```

MongoDB-Query 4 In PathXML Collection add to JSON with "name":"fileName" a "key":"value" field

6.2.3.2.2 Αναζήτηση

Επιγραμματικά η εκτέλεση της αναζήτησης με την τεχνική *PathXML* πραγματοποιείται με τα ακόλουθα βήματα. Για κάθε 3D αρχείο εξάγεται, μέσω των JSON Documents της "*PathXML*" collection της MongoDB, η τοποθεσία, δηλαδή το *directoryPath*, στο οποίο είναι αποθηκευμένα τα metadata του. Στη συνέχεια εντοπίζονται τα ζητούμενα XML αρχεία με τις MPEG-7 και X3D περιγραφές και σε αυτά εκτελείται η αναζήτηση σύμφωνα με το δοσμένο Query. Αν κάποια από τις περιγραφές απαντάει στο Query, συμπληρώνεται το όνομα του 3D αρχείου στο *Αρχείο Αποτελεσμάτων*.

Πιο αναλυτικά για την *PathXML* αναζήτηση χρησιμοποιείται, όπως και στην περίπτωση του *Legacy*, η *SearchCompareAll*. Αυτή με κλήση της class *SearchPathXM* εκτελεί τα 6 Queries που εξετάζονται για όλα τα αντίγραφα δεδομένων που διατηρούμε.

Ο Constructor της *SearchPathXML* δέχεται σαν όρισμα τα *elementName*, *attributeName* και *attributeValue* του εξεταζόμενου Query, το *directoryPath* στο οποίο θα αποθηκευτεί η επιστροφή του Query, καθώς και την database της MongoDB που διατηρεί τα JSON Documents με την τοποθεσία των metadata του κάθε 3D αρχείου. Στο δοσμένο *directoryPath* δημιουργείται ο φάκελος "*PathXML*" και μέσα σε αυτόν ο φάκελος "*results*". Σε αυτόν τον φάκελο θα αποθηκευτούν τα Αρχεία Αποτελεσμάτων για όλα τα Queries που θα εκτελεστούν. Θυμίζεται ότι στα Αρχεία Αποτελεσμάτων δίνεται το όνομα "*elementName attributeName attributeValue*.txt". Τέλος από τον Constructor επιλέγεται και η collection της database που θα χρησιμοποιηθεί, η "*PathXML*".

Στη συνέχεια καλείται η μέθοδος *Search* της class *SearchPathXML*. Η *Search* χρησιμοποιεί το MongoDB-Query 5 και εντοπίζει όλα τα JSON Documents της "*PathXML*". Από αυτά εξάγει τα paths των MPEG-7 και X3D περιγραφών και καλεί την μέθοδο *SubmitQuery* με όρισμα το κάθε *directoryPath*.

```
db.PathXML.find();
```

MongoDB-Query 5 Get all JSON from PathXML Collection

Η *SubmitQuery* εντοπίζει, σύμφωνα με το *directoryPath* που δίνεται, το ζητούμενο XML αρχείο, που αποτελεί μία από τις MPEG-7 ή X3D περιγραφές του 3D αρχείου. Στη συνέχεια

καλεί την μέθοδο *QueryXMLFile* της class *NavigateXML* για διάσχιση του XML αρχείου και εντοπισμό των ζητούμενων *elementName*, *attributeName* και *attributeValue*. Αν υπάρχει κόμβος του XML αρχείου, που να ικανοποιεί το Query, το όνομα του 3D αρχείου γράφεται στο *Αρχείο Αποτελεσμάτων*. Η *NavigateXML* είναι η ίδια class που περιεγράφηκε στο κομμάτι του *Legacy*.

Αφότου επιστρέψει η *SearchPathXML* καταγράφεται ο χρόνος εκτέλεσής της στο path "*Duration\Query*" στο *Αρχείο Χρόνου Απόκρισης* με όνομα "*elementName attributeName attributeValue PathXML*.txt". Όταν η *SearchCompareAll* εκτελέσει την *SearchPathXML* για όλα τα αντίγραφα των δεδομένων, το *Αρχείο Χρόνου Απόκρισης* θα περιέχει 6 μετρήσεις. Η κάθε μία θα αφορά στον χρόνο εκτέλεσης της αναζήτησης στα δεδομένα όταν αυτά συναντώνται σε 1, 2, 4, 8, 16 και 32 αντίγραφα αντίστοιχα.

6.2.3.3 EmbeddedXML

Για τις περιπτώσεις εφαρμογών που επιθυμείται η διατήρηση εξειδικευμένων εργαλείων αναζήτησης πληροφορίας σε XML αρχεία, ενώ ταυτόχρονα δεν καθίσταται απαραίτητη η αποθήκευση των MPEG-7 και X3D αρχείων σε directories, συστήνεται η τεχνική αποθήκευσης και αναζήτησης metadata *EmbeddedXML*.

Με την τεχνική αυτή οι MPEG-7 και X3D περιγραφές των multimedia δεδομένων αποθηκεύονται ως XML String στα αντίστοιχα key/value pairs των JSON Documents που περιγράφουν το κάθε 3D αρχείο. Είναι φανερό πως δεν απαιτείται η διατήρηση των directories που περιέχουν τα metadata από τη στιγμή που αυτά μεταφερθούν στην MongoDB. Παράλληλα με χρήση του υπάρχοντος εξοπλισμού είναι δυνατή η διάσχιση και η αναζήτηση πληροφορίας απ' ευθείας στα XML String που βρίσκονται στα key/value pairs των JSON Documents. Η αναζήτηση και εδώ θα πραγματοποιηθεί μέσω του JDOM δένδρου που παράγεται από τον SAXBuilder.

Με την *EmbeddedXML* τα δεδομένα αποθηκεύονται σε JSON Documents στην collection "*EmbeddedXML*" της database "*ipromxNoCopies*". Κάθε JSON Document αποτελείται από key/value pairs και συγκεκριμένα από το key *name*, στο οποίο διατηρείται το όνομα του 3D αρχείου που αφορά στο JSON Document, και από τα keys *Mpeg7* και *X3D*, που έχουν σαν τιμή το XML String της MPEG-7 και X3D περιγραφής του 3D αρχείου αντίστοιχα. Ανατίθεται αυτόματα από την MongoDB το *_id* key/value pair που ορίζει μοναδικά το κάθε JSON Document. Ένα παράδειγμα τέτοιου αρχείου δίνεται στον Πίνακας 10.

EmbeddedXML JSON Document

```
{
   "_id" : "ObjectID",
   "name" : "fileName",
   "X3D" : "<?xml version = "1.0" encoding = "UTF"?><X3D> … </X3D>",
   "Mpeg7" : "<?xml version = "1.0" encoding = "UTF"?>
                 <Mpeg7 xmlns="urn:mpeg:mpeg7:schema:2001">…</Mpeg7>"
}
```

Πίνακας 10 JSON Document in EmbeddedXML Collection

Όπως και για την *PathXML* τεχνική, είναι απαραίτητη η δημιουργία των αρχείων που θα αποθηκευτούν στην MongoDB. Αυτά θα πρέπει να είναι valid JSON Documents με τις απαραίτητες πληροφορίες για την εκτέλεση της αναζήτησης. Για το σκοπό αυτό υλοποιείται μια μέθοδος δημιουργίας των αρχείων και αποθήκευσης τους στη βάση δεδομένων. Η αναζήτηση θα εκτελεστεί στη συνέχεια στην collection που θα περιέχει τα δεδομένα, χωρίς ανάγκη χρήσης των directories που διατηρούσαν τα αρχικά XML αρχεία με τις MPEG-7 και X3D περιγραφές. Στις επόμενες ενότητες παρουσιάζονται αναλυτικά οι μέθοδοι που

χρησιμοποιήθηκαν για την δημιουργία των κατάλληλων JSON Documents και για την εκτέλεση της αναζήτησης σε αυτά.

6.2.3.3.1 Δημιουργία JSON Documents

Συνοπτικά για τη δημιουργία των JSON Documents χρησιμοποιείται η class *CreateEmbeddedXML*. Η *CreateEmbeddedXML* παίρνει σαν είσοδο τις MPEG-7 και X3D περιγραφές των 3D αρχείων, όταν αυτές βρίσκονται σε 1, 2, 4, 8, 16 και 32 αντίγραφα, από το directory "*application xNoCopies*". Μετά από κάθε εκτέλεσή της έχει εισαχθεί στην MongoDB, στην database "*ipromxNoCopies*", μία collection "*EmbeddedXML*" η οποία περιέχει JSON Documents. Τα JSON Documents διατηρούν τις πληροφορίες για τα 3D αρχεία, τα metadata των οποίων βρίσκονται στο directory "*application xNoCopies*". Ακολουθεί η περιγραφή των classes και μεθόδων που υλοποιήθηκαν για την δημιουργία και αποθήκευση των JSON Documents.

Για την δημιουργία των αρχείων χρησιμοποιείται η class *CreateFilesAll*, η οποία καλεί την class *CreateEmbeddedXML* με όρισμα το *directoryPath* στο οποίο είναι αποθηκευμένες, στους εσωτερικούς φακέλους *mp7_out* και *x3d* αντίστοιχα, οι MPEG-7 και X3D περιγραφές των 3D αρχείων. Ακολουθεί η δημιουργία ενός instance της *CreateEmbeddedXML* καθώς και η κλήση των απαραίτητων μεθόδων της, για το χτίσιμο και την αποθήκευση των JSON Documents στην MongoDB.

Ο Constructor της *CreateEmbeddedXML* class εντοπίζει τους *mp7_out* και *x3d* φακέλους με τα metadata, εντός του *directoryPath* που δέχτηκε σαν όρισμα. Ορίζει επίσης πως τα αρχεία που θα δημιουργηθούν, θα αποθηκευτούν στην collection "*EmbeddedXML*" της database "*ipromxNoCopies*".

Στη συνέχεια καλείται, από την *CreateFilesAll*, η μέθοδος *CreateAllJSONFiles* της class *CreateEmbeddedXML*. Κατά την κλήση της *CreateAllJSONFiles* διαγράφεται η collection "*EmbeddedXML*", για τους λόγους που εξηγήθηκαν νωρίτερα. Στη συνέχεια με χρήση των φίλτρων *FilterXMLFiles* και *FilterX3DFiles* επιλέγονται τα αρχεία των *mp7_out* και *x3d* φακέλων που θα χρησιμοποιηθούν για την δόμηση των JSON Documents, δηλαδή οι MPEG-7 και X3D περιγραφές. Τέλος, για την δημιουργία και αποθήκευση στη MongoDB των JSON, η *CreateAllJSONFiles* καλεί την μέθοδο *AddJSONFilesToMongoDB* με όρισμα την List που έχει επιστρέψει το κάθε φίλτρο.

Όπως έχει αναλυθεί κάθε 3D αρχείο χαρακτηρίζεται από ένα JSON Document, το οποίο περιέχει key/value pairs. Στην περίπτωση της *EmbeddedXML* αναζήτησης, τα key/value pairs με keys *Mpeg7* και *X3D* έχουν σαν value ολόκληρο το XML αρχείο, των MPEG-7 και X3D metadata αντίστοιχα, στη μορφή ενός String. Κάθε αρχείο εισόδου της *AddJSONFilesToMongoDB* αποτελεί ένα XML αρχείο που περιέχει κάποιο τύπο περιγραφής metadata του 3D αρχείου. Όμοια με την μεθοδολογία που περιεγράφηκε για την *PathXML* τεχνική, η *AddJSONFilesToMongoDB* εξάγει το όνομα του 3D αρχείου το οποίο αφορά το αρχείο εισόδου της. Με χρήση του MongoDB-Query 6 εντοπίζεται, αν υπάρχει, το JSON Document που αφορά σε αυτό το αρχείο.

```
db.EmbeddedXML.find( { name : fileName } );
```

MongoDB-Query 6 Search EmbeddedXML Collection for JSON with "name":"fileName"

Αν δεν υπάρχει τέτοιο JSON Document, αυτό δημιουργείται με χρήση του MongoDB-Query 7. Μετά την εκτέλεση του έχει εισαχθεί στην "*EmbeddedXML*" collection ένα JSON Document με ένα key/value pair με τιμές *name*/*fileName*, καθώς και το *_id* key/value pair που δίνεται αυτόματα από την MongoDB. Στην συνέχεια εκτελείται ξανά το MongoDB-Query 6, το οποίο επιστρέφει το JSON που μόλις δημιουργήθηκε και ακολουθεί το update, δηλαδή η

προσθήκη του key/value pair που αφορά στα metadata που δίνονται από το αρχείο εισόδου της *AddJSONFilesToMongoDB*. Η ακριβής λειτουργία του update αναλύεται αμέσως μετά.

```
db.EmbeddedXML.insert( { name : fileName } );
```

MongoDB-Query 7 Insert into EmbeddedXML Collection JSON with "name":"fileName"

Αν το MongoDB-Query 6 επιστρέψει ένα JSON Document, θα εφαρμοστεί σε αυτό ένα update. Συγκεκριμένα με χρήση του MongoDB-Query 8 προστίθεται στο JSON ένα key/value pair με key *Mpeg7* ή *X3D* ανάλογα με τον τύπο metadata που περιέχονται στο XML αρχείο εισόδου. Στο value αυτού του key αποθηκεύεται το XML αρχείο εισόδου στη μορφή ενός String.

```
db.EmbeddedXMl.update({name : fileName}, {$set : {key : value}});
```

MongoDB-Query 8 In EmbeddedXML Collection add to JSON with "name":"fileName" a "key":"value" field

6.2.3.3.2 Αναζήτηση

Όσον αφορά στην εκτέλεση της αναζήτησης με την τεχνική *EmbeddedXML* ακολουθούνται τα παρακάτω βήματα. Για κάθε JSON Document της collection "*EmbeddedXML*", ελέγχονται τα keys *Mpeg7* και *X3D*. Το value του key αποτελεί την MPEG-7 και X3D αντίστοιχα περιγραφή του 3D αρχείου που αφορά στο εξεταζόμενο JSON Document. Στο value εκτελείται η αναζήτηση, σύμφωνα με το δοσμένο Query, με χρήση των υπάρχοντων εργαλείων Parsing XML Strings. Αν η περιγραφή metadata ικανοποιεί το Query, το όνομα του 3D αρχείου γράφεται στο *Αρχείο Αποτελεσμάτων*.

Όπως και στις προηγούμενες περιπτώσεις για την αναζήτηση χρησιμοποιείται η class *SearchCompareAll*. Συγκεκριμένα η *SearchCompareAll* καλεί την *SearchEmbeddedXML*, η οποία, με είσοδο τα δεδομένα σε όλα τα αντίγραφα που συναντώνται, θα εκτελέσει τα 6 Queries που έχουν οριστεί παραπάνω.

Κατά την δημιουργία ενός instance της *SearchEmbeddedXML* καλείται ο Constructor της, ο οποίο παίρνει σαν όρισμα τα *elementName*, *attributeName* και *attributeValue* που αντιστοιχούν στο εξεταζόμενο Query, την βάση δεδομένων που θα χρησιμοποιηθεί, το *directoryPath* στο οποίο θα αποθηκευτούν τα αποτελέσματα και το όνομα του *Αρχείου Αποτελεσμάτων*. Κατά την κλήση του, ο Constructor δημιουργεί το *Αρχείο Αποτελεσμάτων* με όνομα "*elementName attributeName attributeValue*.txt", καθώς και τους φακέλους *EmbeddedXML* και *results*, στους οποίους θα αποθηκευτούν οι αποκρίσεις του Query. Τέλος ορίζει πως η collection που θα χρησιμοποιηθεί για την αναζήτηση είναι η "*EmbeddedXML*".

Στη συνέχεια η *SearchEmbeddedXML* καλεί την μέθοδο *Search* για την πραγματοποίηση της αναζήτησης. Κατά την εκτέλεση του *Search* αρχικά με κλήση του MongoDB-Query 9, επιλέγονται όλα τα JSON Documents της collection "*EmbeddedXML*". Από αυτά εξάγονται οι τιμές των *Mpeg7* και *X3D* keys και δίνονται ως όρισμα στην μέθοδο *SubmitQuery*, η οποία θα πραγματοποιήσει τη διάσχιση του XML String και θα εντοπίσει αν υπάρχουν τους κόμβους που απαντούν στο Query.

```
db.EmbeddedXML.find();
```

MongoDB-Query 9 Get all JSON from EmbeddedXML Collection

Η *SubmitQuery* λαμβάνει ως όρισμα το XML String του *Mpeg7* ή *X3D* key/value pair του εξεταζόμενου JSON αρχείου. Για την υποβολή του Query καλείται η class *NavigateXML* και συγκεκριμένα η μέθοδός της *QueryXMLSrting*. Με αυτήν το XML String εισόδου μετατρέπεται σε XML αρχείο και σε αυτό εφαρμόζεται, με την μέθοδο που περιεγράφηκε σε προηγούμενη ενότητα, η διάσχιση και εντοπισμός κόμβων με τα ζητούμενα *elementName*, *attributeName* και *attributeValue*. Αν επιστραφεί true από την *QueryXMLSrting*, το όνομα του 3D αρχείου γράφεται στο *Αρχείο Αποτελεσμάτων*.

Όταν η *SearchEmbeddedXML* ολοκληρώσει την εκτέλεσή της γράφεται στο path "*Duration\Query*" στο *Αρχείο Χρόνου Απόκρισης*, με όνομα "*elementName attributeName attributeValue EmbeddedXML*.txt", ο χρόνος που χρειάστηκε για να πραγματοποιηθεί η αναζήτηση. Καθώς η *SearchEmbeddedXML* καλείται, για το κάθε Query, για όλα τα αντίγραφα δεδομένων που διατηρούμε, τελικά στο *Αρχείο Χρόνου Απόκρισης* θα υπάρχουν 6 μετρήσεις.

6.2.3.4 KeyValue

Τέλος υλοποιείται η τεχνική *KeyValue*. Η *KeyValue* απευθύνεται σε εφαρμογές που στόχο έχουν την πλήρη εκμετάλλευση των δυνατοτήτων που προσφέρει η MongoDB, όσον αφορά στην αποθήκευση δεδομένων στη μορφή key/value pairs. Για την εφαρμογή της απαιτείται χρήση νέων εργαλείων, με κάποια να προτείνονται σε αυτή τη διπλωματική, καθώς και καλή εξοικείωση των διαχειριστών με εργαλεία χρήσης της MongoDB.

Με χρήση της *KeyValue* τεχνικής για κάθε 3D αρχείο δημιουργείται ένα JSON Document το οποίο αποθηκεύεται στην MongoDB. Σε αυτό διατηρούνται τα metadata του 3D αρχείου με μετατροπή ολόκληρου του XML, που περιέχει τις MPEG-7 και X3D περιγραφές, σε key/value pairs. Ακολουθεί η αναζήτηση με διάσχιση των JSON Documents και εντοπισμό εκείνων που ικανοποιούν το Query που υποβλήθηκε.

Η μετατροπή των metadata σε key/value pairs πραγματοποιείται σύμφωνα με την μεθοδολογία που παρουσιάστηκε για την Εικόνα 20. Ο κάθε κόμβος του XML αρχείου μετατρέπεται σε ένα key/value pair με key το elementName του κόμβου και value ένα Embedded JSON Object. Σε αυτό το JSON Object αποθηκεύονται τα attributes του κόμβου με το κάθε attributeName και attributeValue να αντιστοιχίζονται σε ένα key/value pair. Στο Embedded JSON Object αποθηκεύονται επίσης τα child nodes του εξεταζόμενου κόμβου με χρήση της ίδια μεθόδου. Στην *KeyValue* τεχνική, μετά από την μετατροπή των XML περιγραφών σε valid JSON Documents, δεν είναι απαραίτητη η διατήρηση των metadata και σε directories στο δίσκο, καθώς η αναζήτηση πραγματοποιείται απ' ευθείας στην βάση δεδομένων με διάσχιση των key/value pairs των JSON Documents. Ακολουθεί ένα παραδείγματα μετατροπής ενός XML αρχείου, Πίνακας 11, στο αντίστοιχο JSON Document, Πίνακας 12.

XML Document

```
<messages>
  <note id="501">
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
  </note>
  <note id="502">
    <to>Jani</to>
    <from>Tove</from>
    <heading>Re: Reminder</heading>
    <body>I will not</body>
```

```
  </note>
</messages>
```

Πίνακας 11 Example XML Document

JSON Document

```
{
  "messages": {
    "note": [
      {
        "-id": "501",
        "to": "Tove",
        "from": "Jani",
        "heading": "Reminder",
        "body": "Don't forget me this weekend!"
      },
      {
        "-id": "502",
        "to": "Jani",
        "from": "Tove",
        "heading": "Re: Reminder",
        "body": "I will not"
      }
    ]
  }
}
```

Πίνακας 12 Converted JSON Document

Όμοια με τις τεχνικές που περιεγράφηκαν σε προηγούμενες ενότητες, τα JSON Documents με τα metadata, που χρησιμοποιούνται από την *KeyValue* τεχνική, αποθηκεύονται στην collection "*KeyValue*", η οποία ανήκει στην βάση δεδομένων "*ipromxNoCopies*". Κατά τα γνωστά *NoCopies* είναι 1, 2, 4, 8, 16 ή 32 και αποτελεί τον αριθμό των αντιγράφων που εμφανίζεται κάθε 3D αρχείο. Πιο αναλυτικά κάθε JSON Document αποτελείται από τα key/value pairs *_id*, που δίνεται από την MongoDB κατά τη δημιουργία κάθε νέου JSON, *name*, που είναι το όνομα του 3D αρχείου που περιγράφεται από το JSON Document, *Mpeg7* και *X3D*. Τα *Mpeg7* και *X3D* keys περιέχουν το Embedded JSON Document με όλη την MPEG-7 και X3D περιγραφή του multimedia αρχείο μετασχηματισμένη σε key/value pairs. Ένα τέτοιο JSON Document δίνεται στον Πίνακας 13.

KeyValue JSON Document

```
{
    "_id" : "ObjectID",
    "name" : "fileName",
    "Mpeg7": {
      "-xmlns": "urn:mpeg:mpeg7:schema:2001",
      "Description": {
        "-xmlns:xsi": "http://www.w3.org/2001/XMLSchema-instance",
        "-xsi:type": "ContentEntityType",
        "MultimediaContent": {
          "-xsi:type": "MultimediaCollectionType",
          "StructuredCollection": {
            "Collection": {
              "DescriptorCollection": {
                "-id": "Geometry_N10025",
                "Descriptor": {
                  "-xsi:type": "BoundingBox3DType",
```

```
                  "BoundingBox3DSize": {
                    "-BoxWidth": "-1",
                    "-BoxHeight": "-1",
                    "-BoxDepth": "-1"
                  }
                }
              },
              "DescriptorCollectionRef": { "-href": "#GROUND" }
            }
          }
        }
      }
    },
    "X3D": {
      "-profile": "Immersive",
      "head": {
        "meta": {
          "-name": "ExportTime",
          "-content": "17: 6:18"
        }
      },
      "Scene": {
        "Group": {
          "-DEF": "GROUND",
          "Transform": {
            "Shape": {
              "-DEF": "Box1",
              "-containerField": "children"
            }
          }
        },
        "ROUTE": {
          "-fromNode": "Sensor1",
          "-fromField": "touchTime",
          "-toNode": "MovieTexture_Box1",
          "-toField": "startTime"
        }
      }
    }
}
```


Πίνακας 13 JSON Document in KeyValue Collection

Στην περίπτωση της αναζήτησης πληροφορίας σε XML αρχεία, στόχος ήταν ο εντοπισμός ενός τουλάχιστον κόμβου με όνομα *elementName*, και ο οποίος να διαθέτει ένα attribute με όνομα *attributeName* και τιμή *attributeValue*. Η ίδια αναζήτηση μεταφράζεται, όσον αφορά στην αναζήτηση σε JSON Documents, σε εντοπισμό ενός key με όνομα *elementName*, που έχει ως value ένα Embedded JSON Object ή Array, το οποίο διαθέτει ένα key/value pair με τιμές *attributeName*/*attributeValue* αντίστοιχα.

Για την εκτέλεση της αναζήτησης αποτελεί προϋπόθεση η διάσχιση του JSON Document και ο εντοπισμός των ζητούμενων key/value pairs. Η παραπάνω διαδικασία απαιτεί δύο στάδια. Στο πρώτο στάδιο διασχίζεται το JSON Document και στο δεύτερο ελέγχεται αν στο περιεχόμενό του συναντάται το ζητούμενο key/value pair.

Για τον έλεγχο της ύπαρξης ή όχι ενός key χρησιμοποιείται από την MongoDB ο query operator "*$exists*". Ένα παράδειγμα χρήσης του δίνεται από το MongoDB-Query 10, το οποίο εντοπίζει τα JSON Documents της collection "*exampleCollection*" που περιέχουν ένα key με όνομα "*my_key*". Ωστόσο, αν το key αυτό βρίσκεται σε ένα Embedded Object, η MongoDB δεν θα το εντοπίσει και δεν θα επιστρέψει το εξεταζόμενο JSON Document. Για να εντοπιστεί ένα key, όταν αυτό ανήκει σε ένα Embedded Object ή Array, είναι απαραίτητος ο προσδιορισμός όλων των keys που πρέπει να διασχιστούν μέχρι το ζητούμενο. Αυτό πραγματοποιείται με χρήση του operator "." που παρέχεται από την MongoDB. Ακολουθεί ένα παράδειγμα χρήσης του operator "*$exists*" και τα αποτελέσματα που επιστρέφονται.

Κατά την εκτέλεση του MongoDB-Query 10 στην collection "*exampleCollection*" που περιέχει τα JSON Documents του Πίνακας 14, θα επιστραφεί μόνο το "*file1*" JSON Document. Για να επιστραφεί το JSON Document "*file2*" θα πρέπει να υποβληθεί το MongoDB-Query 11, με το οποίο εντοπίζονται αν υπάρχουν JSON Documents με key "*father_key*", το οποίο έχει ως value ένα Embedded JSON Object, το οποίο περιέχει ένα key με όνομα "*my_key*". Σημειώνεται πως με το MongoDB-Query 11 θα επιστραφεί μόνο το "*file2*" και όχι το "*file1*".

```
db.exampleCollection.find({ "my_key" : { $exists : true} });
```

MongoDB-Query 10 Select JSON from exampleCollection where "key" field exists

```
db.exampleCollection.find({"father_key.my_key" : {$exists : true}});
```

MongoDB-Query 11 Select JSON from exampleCollection where "father_key.key" field exists

file1	file2
{ "_id" : "ObjectID", "name" : "file1", "my_key" : " my_value" }	{ "_id" : "ObjectID", "name" : "file2", "father_key" : { " my_key": " my_value" } }

Πίνακας 14 JSON Documents in exampleCollection

Είναι φανερό πως για τον εντοπισμό key/value pairs, όταν αυτά βρίσκονται σε κάποιο από τα Embedded JSON Objects ή Arrays του JSON Document που εξετάζεται, είναι απαραίτητη η γνώση του ακριβές path από keys που πρέπει να προσπελαστούν μέχρι να βρεθεί η αναζήτηση στο ζητούμενο key/value pair.

Η MongoDB δεν διαθέτει ενσωματωμένο μηχανισμό για αυτόματο εντοπισμό keys που βρίσκονται σε Embedded Objects και Arrays . Για αυτό το λόγο, από την διπλωματική αυτή, υλοποιείται ένας μηχανισμός αναζήτησης keys σε όλο το βάθος του JSON Document, για τον εντοπισμό αυτών που ικανοποιούν το δοσμένο Query. Αυτός ο τρόπος αναζήτησης ονομάστηκε *KeyValueQueue*.

Ωστόσο καθώς τα δεδομένα αποτελούν metadata που έχουν δημιουργηθεί με συγκεκριμένο πρότυπο, το MPEG-7 και το X3D, σε πολλές περιπτώσεις η θέση του κάθε key, δηλαδή το μονοπάτι από keys που πρέπει να ακολουθηθούν μέχρι το ζητούμενο, είναι γνωστό. Για να εξεταστεί και αυτή η περίπτωση αναζήτησης πληροφοριών σε JSON Documents, υλοποιήθηκε ένας δεύτερος μηχανισμός διάσχισης τους, ο οποίος εντοπίζει αν υπάρχουν τα ζητούμενα keys, όταν αυτά βρίσκονται σε συγκεκριμένο path του JSON Document. Αυτός ο μηχανισμός ονομάστηκε *KeyValuePath*.

Για την εκτέλεση της αναζήτησης με την τεχνική *KeyValue* απαιτείται η ανάπτυξη δύο εργαλείων. Το πρώτο αφορά στην δημιουργία κατάλληλης δομής JSON Documents και αποθήκευσή τους στην collection "*KeyValue*". Το δεύτερο αφορά σε υλοποίηση των δύο εναλλακτικών μηχανισμών αναζήτησης που περιεγράφηκαν προηγουμένως, οι οποίοι ικανοποιούν διαφορετικές ανάγκες εφαρμογών. Σε επόμενη ενότητα ακολουθεί αναλυτική περιγραφή τόσο της δημιουργίας και αποθήκευσης των JSON Documents, καθώς και της εκτέλεσης της αναζήτησης.

6.2.3.4.1 Δημιουργία JSON Documents

Όπως και στις προηγούμενες τεχνικές, τα metadata των 3D αρχείων διατηρούνται σε 1, 2, 4, 8, 16 και 32 αντίγραφα στους φακέλους *mp7_out* και *x3d*. Τα δεδομένα, με χρήση της class *CreateKeyValue*, θα μετατραπούν στα κατάλληλα JSON Documents και θα αποθηκευτούν στην collection "*KeyValue*" της database "*ipromxNoCopies*", όπου *NoCopies* είναι 1, 2, 4, 8, 16 και 32 αντίστοιχα.

Πιο συγκεκριμένα αρχικά καλείται η *CreateFilesAll*, η οποία ορίζει την database που θα αποθηκευτούν τα JSON Documents. Στη συνέχεια εκτελείται η *CreateKeyValue* με είσοδο το *directoryPath* των metadata καθώς και την βάση δεδομένων που θα χρησιμοποιηθεί.

Ο Constructor της *CreateKeyValue* εντοπίζει του εσωτερικούς φακέλους του *directoryPath*, *mp7_out* και *x3d*, οι οποίοι περιέχουν τις MPEG-7 και X3D περιγραφές των 3D αρχείων αντίστοιχα, και ορίζει πως τα παραγόμενα JSON Documents θα αποθηκευτούν στην collection "*KeyValue*" της βάσης δεδομένων που ορίστηκε από την *CreateFilesAll*.

Ακολουθεί η κλήση της *CreateKeyValue* και συγκεκριμένα της μεθόδου της *CreateAllJSONFiles*. Η *CreateAllJSONFiles* διαγράφει την collection "*KeyValue*", εφαρμόζει τα φίλτρα *FilterXMLFiles* και *FilterX3DFiles*, για απομόνωση των αρχείων που αποτελούν MPEG-7 και X3D περιγραφές αντίστοιχα, και καλεί την μέθοδο *AddJSONFilesToMongoDB* με είσοδο της Lists που επέστρεψαν τα δύο φίλτρα.

Στην *AddJSONFilesToMongoDB* κάθε αρχείο εισόδου αποτελεί metadata του 3D αρχείου με όνομα *fileName*. Με χρήση του MongoDB-Query 12, εντοπίζεται το JSON Document της collection "*KeyValue*" με ένα key/value pair με τιμές *name*/*fileName* αντίστοιχα.

```
db.KeyValue.find( { name : fileName } );
```

MongoDB-Query 12 Search KeyValue Collection for JSON with "name":"fileName"

Αν δεν βρεθεί τέτοιο αρχείο, απαιτείται η δημιουργία του. Αυτό γίνεται με χρήση του MongoDB-Query 13. Με αυτό δημιουργείται ένα JSON Document με δύο key/value pairs. Το *_id* key/value pair που ανατίθεται αυτόματα από τη MongoDB και το *name*/*fileName* key/value pair που συμπληρώνεται από την AddJSONFilesToMongoDB. Μετά τη δημιουργία του JSON Document, καλείται ξανά το MongoDB-Query 12 και ακολουθεί το update, με το οποίο προστίθεται στο JSON Document που θα επιστραφεί, η περιγραφή των metadata που συναντώνται στο αρχείο εισόδου της *AddJSONFilesToMongoDB*.

```
db.KeyValue.insert( { name : fileName } );
```

MongoDB-Query 13 Insert into KeyValue Collection JSON with "name":"fileName"

Όταν βρεθεί το JSON Document με *name* το δοσμένο *fileName*, αρκεί η εφαρμογή ενός update για την προσθήκη της περιγραφής των metadata στο κατάλληλο key/value pair. Οι MPEG-7 και X3D περιγραφές διατηρούνται σε ένα field με key *Mpeg7* και *X3D* αντίστοιχα. Για την αποθήκευση των metadata απαιτείται μετατροπή του αρχείου εισόδου σε valid JSON Object. Η μετατροπή αυτή πραγματοποιείται με χρήση της *toJSONObject* μεθόδου της class *XML*. Στη συνέχεια με εφαρμογή του MongoDB-Query 14 προστίθεται το JSON Object που δημιουργήθηκε ως value στο κατάλληλο key, *Mpeg7* ή *X3D*, του JSON Document.

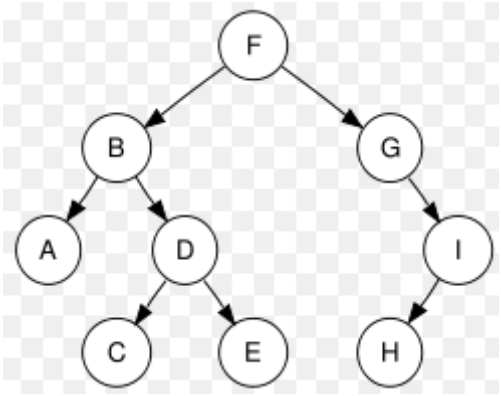
```
db.KeyValue.update({name : fileName}, {$set : {key : value}});
```

MongoDB-Query 14 In KeyValue Collection add to JSON with "name":"fileName" a "key":"value" field

6.2.3.4.2 Αναζήτηση KeyValueQueue

Όπως αναφέρθηκε στην προηγούμενη ενότητα, για την αναζήτηση με την *KeyValue* τεχνική υλοποιούνται δύο εναλλακτικοί τρόποι εκτέλεσής της ανάλογα με τις ανάγκες της εφαρμογής. Σε αυτήν την ενότητα θα αναπτυχθεί η τεχνική *KeyValueQueue* η οποία αφορά σε περιπτώσεις χρήσης όπου δεν είναι γνωστό το ακριβές path στο οποίο είναι αποθηκευμένα τα επιθυμητά key/value pairs.

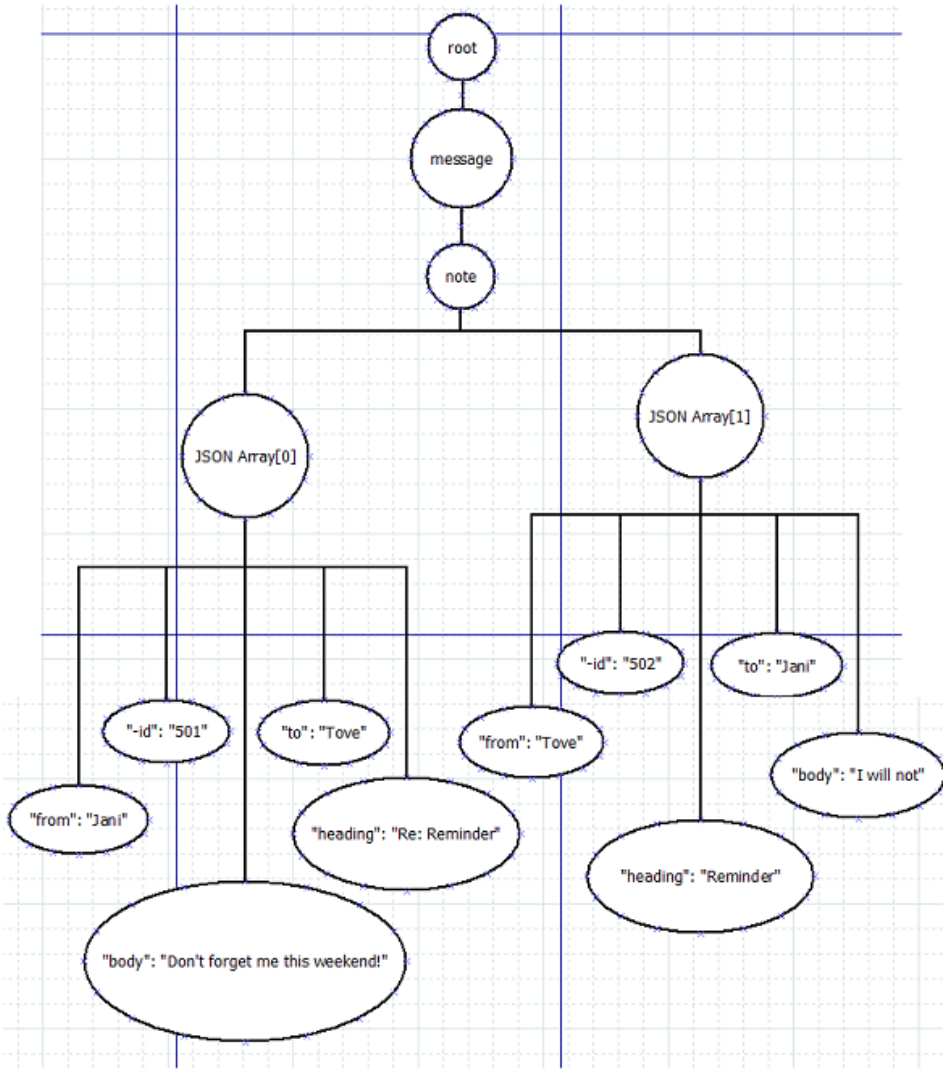
Στην επιστήμη των υπολογιστών συναντάται η αφηρημένη δομή δεδομένων tree, δέντρο. Σε αυτήν, τα δεδομένα δομούνται σε μια ιεραρχική μορφή δέντρου, με ρίζα και άλλα υπόδεντρα ως παιδιά, ως ένα σύνολο διασυνδεδεμένων κόμβων. Ένα παράδειγμα δέντρου φαίνεται στην Εικόνα 25, όπου ο κόμβος *F* αποτελεί την ρίζα του δέντρου, οι κόμβοι *B, G, D* και *I* τους εσωτερικούς κόμβους και οι κόμβοι *A, C, E* και *H* τα φύλλα. "*Ρίζα*" ονομάζονται οι κόμβοι που δεν έχουν πατέρα, δηλαδή δεν είναι παιδιά κανενός κόμβου, ως "*εσωτερικοί*" χαρακτηρίζονται οι κόμβοι που δεν είναι ρίζα και έχουν παιδιά, ενώ ως "*φύλλα*", οι κόμβοι που δεν έχουν παιδιά.



Εικόνα 25 Tree Data Structure

Η αναζήτηση πληροφοριών σε trees γίνεται με χρήση αλγορίθμων διάσχισής τους. Ένας από αυτούς είναι ο *Breadth-First Search, BFS*, κατά τον οποίο διασχίζεται ολόκληρο το δέντρο ανά επίπεδο μέχρι τον εντοπισμό του ζητούμενου κόμβου ή μέχρι να διασχιστούν όλοι οι κόμβοι. Το *επίπεδο*, ή *βάθος*, στα trees ορίζεται ως το πλήθος των ακμών που πρέπει να διασχιστούν από την ρίζα μέχρι τον εξεταζόμενο κόμβο. Στο παράδειγμα της Εικόνα 25 οι κόμβοι *B* και *G* βρίσκονται σε βάθος 1, δηλαδή στο $1^{ο}$ επίπεδο του δέντρου. Οι κόμβοι *A, D* και *I* σε βάθος 2, δηλαδή πρέπει να διασχιστούν 2 ακμές από την ρίζα για να φτάσουμε σε αυτούς. Τέλος οι κόμβοι *C, E* και *H* είναι σε βάθος 3.

Τα JSON Documents αποτελούνται από key/value pairs τα οποία στο value μπορεί να έχουν άλλα Embedded JSON Objects. Αυτή η ενσωμάτωση πληροφορίας εντός του αρχικού αρχείου θυμίζει σε δομή ένα δέντρο, όπου σαν ρίζα θεωρείται το αρχικό JSON Document και σαν κόμβοι τα key/value pairs του. Πιο συγκεκριμένα key/value pairs που στο value διαθέτουν ένα Embedded JSON Object ή Array θα αποτελούν ενδιάμεσους κόμβους, καθώς θα περιέχουν στο value τους τουλάχιστον ένα ακόμα key/value pair, το οποίο θα αποτελεί έναν κόμβο παιδί. Αντίθετα key/value pairs που στο value δεν έχουν ενσωματωμένο κάποιο άλλο JSON Object θα αποτελούν κόμβους φύλλα. Η αναπαράσταση του JSON Document του Πίνακας 12 σε tree δίνεται από την Εικόνα 26.



Εικόνα 26 JSON as Tree

Είναι φανερό πως με εφαρμογή *BFS* είναι δυνατή η διάσχιση ολόκληρου του JSON tree μέχρι τον εντοπισμό των ζητούμενων key/value pairs. Για αυτό το λόγο στην αναζήτηση *KeyValueQueue* επιλέχθηκε η χρήση ενός *BFS* και η υλοποίησή του με χρήση μιας ουράς, *Queue*.

Ο αλγόριθμος διάσχισης γράφων με *BFS* ακολουθά τα παρακάτω βήματα.

1. Στην αρχή της ουράς εισάγεται ο root node.
2. Εξάγεται ένας κόμβος από την ουρά.
    a. Αν βρεθεί ο ζητούμενος κόμβος, η αναζήτηση τελείωσε και επιστρέφει “found”.
    b. Διαφορετικά εισάγονται στο τέλος της ουράς όλα τα child nodes του εξεταζόμενου κόμβου.
3. Αν η ουρά είναι άδεια, έχει εξεταστεί ολόκληρο το δέντρο και δεν έχει βρεθεί ο ζητούμενος κόμβος. Η αναζήτηση επιστρέφει “not found”.
4. Αν η ουρά δεν είναι άδεια επαναλαμβάνουμε την αναζήτηση από το βήμα 2.

Πίνακας 15 BFS Algorithm

Στην ίδια λογική θα διεξαχθεί και η διάσχιση του JSON Document. Κάθε φορά εξάγεται από την ουρά ένας κόμβος, που θυμίζεται ότι αποτελείται από ένα key/value pair. Αν αυτό έχει ως key το δοσμένο *elementName*, ελέγχεται αν στο value του υπάρχει ένα key/value pair με τιμές *attributeName*/*attributeValue* αντίστοιχα. Αν υπάρχει τέτοιο key/value pair η αναζήτηση έχει τελειώσει και επιστρέφεται "found", διαφορετικά εισάγονται στο τέλος της ουράς τα Embedded JSON Objects του τρέχοντος κόμβου. Η αναζήτηση συνεχίζεται με τον ίδιο τρόπο μέχρι να βρεθεί κόμβος που να ταιριάζει στο ζητούμενο Query ή μέχρι να αδειάσει η ουρά. Αν η ουρά αδειάσει σημαίνει πως δεν υπάρχει κόμβος που να ικανοποιεί το Query και άρα επιστρέφεται "not found". Τα ονόματα των 3D αρχείων που απάντησαν "found" κατά την αναζήτηση γράφονται στο *Αρχείο Αποτελεσμάτων*. Ακολουθεί αναλυτική περιγραφή των μεθόδων και των classes που υλοποιήθηκαν για την *KeyValueQueue* αναζήτηση.

Η αναζήτηση εκτελείται με κλήση της *SearchCompareAll*, η οποία με τη σειρά της καλεί την *SearchKeyValueQueue* με ορίσματα το *directoryPath* στο οποίο θα αποθηκευτούν τα αποτελέσματα της αναζήτησης, η *database* που θα χρησιμοποιηθεί για την αναζήτηση, και τα *elementName*, *attributeName* και *attributeValue* που συνθέτουν το Query. Για την υποβολή του Query και εκτέλεση της αναζήτησης καλείται η μέθοδος *Search* της *SearchKeyValueQueue* class. Καταγράφεται επίσης ο χρόνος έναρξης της αναζήτησης και ο χρόνος ολοκλήρωσης της. Η διαφορά των δύο μετρήσεων, δηλαδή ο συνολικός χρόνος εκτέλεσης της αναζήτησης καταγράφεται στο path "*Duration**Query*\" στο *Αρχείο Χρόνου Απόκρισης* με όνομα "*elementName attributeName attributeValue KeyValueQueue*.txt".

Ο Constructor της *SearchKeyValueQueue* ορίζει πως για την αναζήτηση θα χρησιμοποιηθεί η collection "*KeyValue*" και πως τα αποτελέσματα του Query θα αποθηκευτούν μέσα στο *directoryPath* στο φάκελο "*KeyValue**Queue**results*\" στο *Αρχείο Αποτελεσμάτων* με όνομα "*elementName attributeName attributeValue*.txt".

Η μέθοδος *Search* χρησιμοποιεί το MongoDB-Query 15 για εντοπισμό όλων των JSON Documents της "*KeyValue*" collection. Το καθένα από αυτά μετατρέπεται σε JSONObject και περνάει ολόκληρο ως όρισμα στην μέθοδο *SubmitQuery*.

```
db.KeyValue.find();
```

MongoDB-Query 15 Get all JSON from KeyValue Collection

Η μέθοδος *SubmitQuery* εισάγει τα παιδιά του JSON Document, δηλαδή τα key/value pairs που εμφανίζονται σε βάθος 1 του JSON Document, στην *Queue*. Στη συνέχεια καλείται η μέθοδος *SearchQueue*, με την οποία διασχίζεται το JSON Document με τον αλγόριθμο που παρουσιάστηκε παραπάνω και επιστρέφεται το αποτέλεσμα της αναζήτησης. Αν επιστραφεί true από την *SearchQueue* το όνομα του 3D αρχείου εισόδου γράφεται στο *Αρχείο Αποτελεσμάτων*, διαφορετικά όχι. Η αναζήτηση συνεχίζει σε επόμενα JSON Documents της collection "*KeyValue*".

Όταν επιστρέψει η *SearchCompareAll* θα έχουν γραφεί 6 χρόνοι, ένας για κάθε database "*ipromxNoCopies*" στην οποία εκτελείται η αναζήτηση, στα *Αρχείο Αποτελεσμάτων* των 6 διαφορετικών Queries που υποβλήθηκαν.

6.2.3.4.3 Αναζήτηση KeyValuePath

Εξετάστηκε επίσης η περίπτωση της γνώσης του path από keys στο οποίο είναι αποθηκευμένο το ζητούμενο key/value pair. Η τεχνική αυτή αναζήτησης ονομάστηκε *KeyValuePath*.

Όπως έχει αναλυθεί για τον εντοπισμό JSON Documents της collection "*exampleCollection*" που διαθέτουν το key *my_key* χρησιμοποιείται το MongoDB-Query 10. Για εντοπισμό του *my_key* που βρίσκεται στο Embedded JSON Object του *father_key* χρησιμοποιείται το MongoDB-Query 11. Είναι φανερό πως θα πρέπει να χτιστεί χειροκίνητα ολόκληρο το path από keys μέχρι να καταλήξουμε στο ζητούμενο key/value pair.

Για τις ανάγκες του προβλήματος δεν αρκεί ο εντοπισμός ύπαρξης ενός key, αλλά και η ύπαρξη μιας συγκεκριμένης τιμής στο value του. Για το σκοπό αυτό η MongoDB χρησιμοποιεί την μέθοδο *find* με όρισμα το key/value pair που θέλουμε να διαθέτουν τα JSON Documents που θα επιστραφούν. Ένα παράδειγμα τέτοιου query είναι το MongoDB-Query 16, το οποίο επιστρέφει τα JSON Documents της collection "*exampleCollection2*", Πίνακας 16, που έχουν ένα key/value pair με τιμές αντίστοιχα *my_key*/*my_value*. Μετά την υποβολή του, θα επιστραφεί μόνο το JSON Document "*file1*".

```
db.exampleCollection2.find({"my_key" : "my_value"});
```

MongoDB-Query 16 Return JSON from myCollection with a "my_key" : "my_value" pair

file1	file2	file3

{
"_id":"ObjectID",
"name":"file1",
"my_key":"my_value"
}
	{
	"_id" : "ObjectID",
	"name" : "file2",
	"father_key" : {
	" my_key":"my_value"
	}
	}
		{
		"_id" : "ObjectID",
		"name" : "file3",
		"father_key" : {
		"my_key": "not_my_value"
		}
		}

Πίνακας 16 JSON Documents in exampleCollection2

Στις περιπτώσεις που το ζητούμενο key ανήκει σε ένα Embedded JSON Object του JSON Document που εξετάζεται, τότε απαιτείται ο ορισμός του path από keys μέχρι το ζητούμενο. Η διάσχιση των Embedded JSON Objects γίνεται με τον τελεστή ".", όπως φαίνεται από το MongoDB-Query 17. Με αυτό αναζητούνται στην collection "*exampleCollection2*", Πίνακας 16, τα JSON Documents που έχουν ένα key/value pair με key *father_key* και value ένα Embedded JSON Object που έχει ένα key/value pair με τιμές *my_key*/*my_value* αντίστοιχα. Μετά την υποβολή του θα επιστραφεί μόνο το JSON Document "*file2*".

```
db.exampleCollection2.find({"father_key.my_key":"my_value"});
```

MongoDB-Query 17 Search exampleCollection2 for "my_key" : "my_value" pair inside Embedded JSON Object of "father_key" key/value pair

Σύμφωνα με τα παραπάνω για κάθε Query που εξετάζεται είναι απαραίτητος ο προσδιορισμός του ακριβές path από keys που θα ακολουθηθεί καθώς και το value που πρέπει να έχει το τελευταίο key. Ακολουθεί ο Πίνακας 17 που δείχνει για κάθε Query ποιο είναι το ζητούμενα *keyPath*, που στο εξής θα ονομάζεται *key*, και *value*. Δίνεται επίσης και ένα παράδειγμα, MongoDB-Query 18, μετατροπής του Query Q5 στη μορφή που τελικά θα υποβληθεί στη βάση δεδομένων.

Query	path	value
Q1	Mpeg7.Description.MultimediaContent. StructuredCollection.Collection.id	Transformations
Q2	X3D.Scene.Group.Transform.Shape.Cylinder.radius	0.5
Q3	Mpeg7.Description.MultimediaContent.	Box

	StructuredCollection.Collection.DescriptorCollection. Descriptor.Geometry3D.ObjectType	
Q4	X3D.head.meta.name	title
Q5	X3D.Scene.Group.Transform.DEF	TRANS
Q6	X3D.profile	Immersive

Πίνακας 17 Examined Queries Path of Keys

```
db.KeyValue.find({"X3D.Scene.Group.Transform.DEF" : "TRANS"});
```

MongoDB-Query 18 Examined Query Q5 submitted in KeyValue Collection

Αναλυτικά η εκτέλεση της αναζήτησης πραγματοποιείται με τα ακόλουθα βήματα. Αρχικά καλείται η *SearchCompareAll*, η οποία ορίζει το *key* και το *value* που αντιστοιχούν στα δοσμένα *elementName*, *attributeName* και *attributeValue* και συνθέτει το *Query* που τελικά θα υποβληθεί στη βάση δεδομένων "*ipromxNoCopies*". Ορίζει επίσης το *directoryPath* "*Duration\Query*" του *Αρχείου Απόκρισης Χρόνου*, με όνομα "*elementName attributeName attributeValue KeyValuePath*.txt", στο οποίο θα αποθηκευτούν οι χρόνοι απόκρισης της αναζήτησης. Για την εκτέλεση της αναζήτησης καλείται η class *SearchKeyValuePath* και στη συνέχεια η μέθοδός της *Search*.

Ο Constructor της *SearchKeyValuePath* δέχεται σαν όρισμα την βάση δεδομένων που θα χρησιμοποιηθεί, το *Query* που θα υποβληθεί, τα *elementName, attributeName* και *attributeValue*, καθώς και το *directoryPath* στο οποίο θα αποθηκευτεί το *Αρχείο Αποτελεσμάτων*. Ο Constructor είναι υπεύθυνος για τον ορισμό του ονόματος του *Αρχείου Αποτελεσμάτων*, "*elementName attributeName attributeValue*.txt", του ακριβές path, εντός του *directoryPath*, στο οποίο θα αποθηκευτεί, "*KeyValue\results\Path*", καθώς και για τον ορισμό της collection στην οποία είναι αποθηκευμένα τα JSON Documents, "*KeyValue*".

Τέλος καλείται η μέθοδος Search, η οποία εφαρμόζει το *Query* στα δεδομένα και γράφει τα ονόματα των αρχείων που επιστράφηκαν στο *Αρχείο Αποτελεσμάτων*.

Η *SearchCompareAll* εκτελείται για όλα τα Queries που εξετάζονται από αυτήν την διπλωματική καθώς και για όλα τα αντίγραφα των δεδομένων που διατηρούμε. Αφότου επιστρέψει, θα έχει υποβληθεί κάθε Query σε όλα τα αντίγραφα δεδομένων που διατηρούνται και θα έχουν δημιουργηθεί 6 Αρχεία Χρόνου Απόκρισης με 6 τιμές, όπως έχει περιγραφεί και στις προηγούμενες μεθόδους που εξετάστηκαν.

6.2.4 Τομέας Μελέτης *MapReduce*

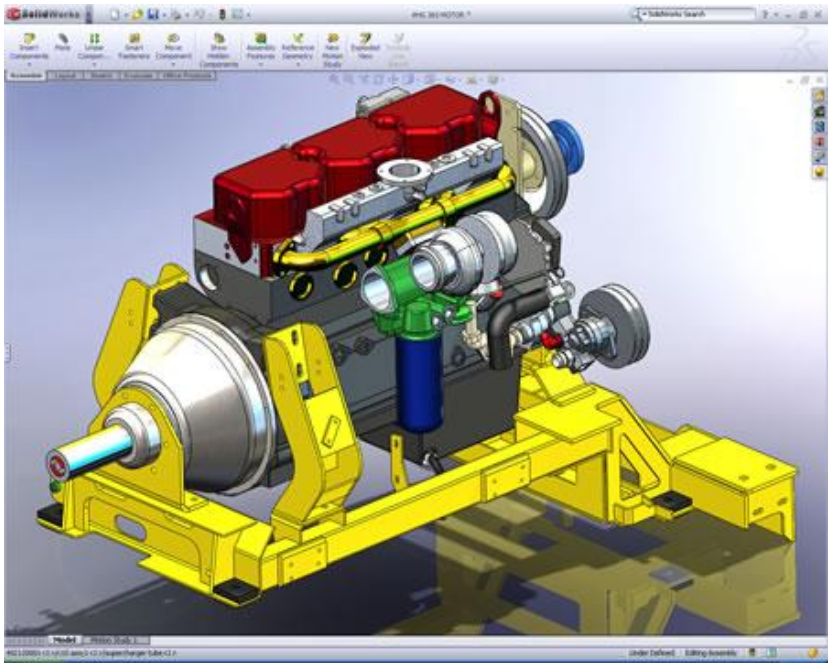
Όμοια με την προηγούμενη κατηγορία αναζήτησης που εξετάστηκε, και στον τομέα μελέτης *MapReduce* χρησιμοποιείται η γλώσσα προγραμματισμού Java και η NoSQL βάση δεδομένων MongoDB. Σε αυτήν την κατηγορία αναζήτησης για την εκτέλεση των queries απαιτείται κάποιο aggregation στα δεδομένα. Οι τεχνικές που υλοποιήθηκαν από αυτήν την διπλωματική στις οποίες μπορούν να υποβληθούν τέτοιου είδους queries είναι η *MapReduce* και η *NoMapReduce*. Ακολουθεί ανάλυση των δύο αυτών μεθόδων.

6.2.4.1 MapReduce

Παρατηρήθηκε πως στις πλατφόρμες αναζήτησης metadata οι χρόνοι απόκρισης του συστήματος οφείλονταν κατά κύριο λόγο στην επικοινωνία της εφαρμογής με το δίσκο, δηλαδή στα I/O, και δευτερευόντως στον απαιτούμενο επεξεργαστικό χρόνο για την εκτέλεση των επιμέρους πράξεων, δηλαδή στη CPU. Τα υψηλά I/O οφείλονται στο μεγαλύτερο ποσοστό στο μεγάλο αριθμό JOINS που απαιτείται να γίνουν, στη περίπτωση χρήσης βάσης

δεδομένων, ή στην διάσχιση ενθυλακωμένων αντικειμένων, στην περίπτωση αναζήτησης σε αρχεία που βρίσκονται στο δίσκο, προκειμένου να απαντηθούν τα υποβαλλόμενα ερωτήματα.

Ένα χαρακτηριστικό παράδειγμα query που απαιτεί πολλές φορές πρόσβαση στο δίσκο καθώς και την εκτέλεση πολλών απλών πράξεων, είναι η εύρεση του κυρίαρχου χρώματος σε μια εικόνα ή scene, σκηνή. Για την εκτέλεση αυτού του query απαιτείται η διάσχιση της βάσης δεδομένων για όλα τα επιμέρους αντικείμενα που συνθέτουν την σκηνή και ο αναδρομικός υπολογισμός του ποσοστού χώρου που καταλαμβάνει κάθε αντικείμενο, καθώς και του χρώματος του. Στις περιπτώσεις που η σκηνή αποτελείται από πολλές χιλιάδες αντικείμενα, όπως αυτά της Εικόνα 27, όπου κάθε αντικείμενο αποτελείται από πολλά μικρότερα, η εκτέλεση του query, με JOINS ή με αναδρομική διάσχιση των αρχείων που αναπαριστούν τα δεδομένα, θα απαιτούσε πάρα πολύ χρόνο.



Εικόνα 27 SolidWorks Scene

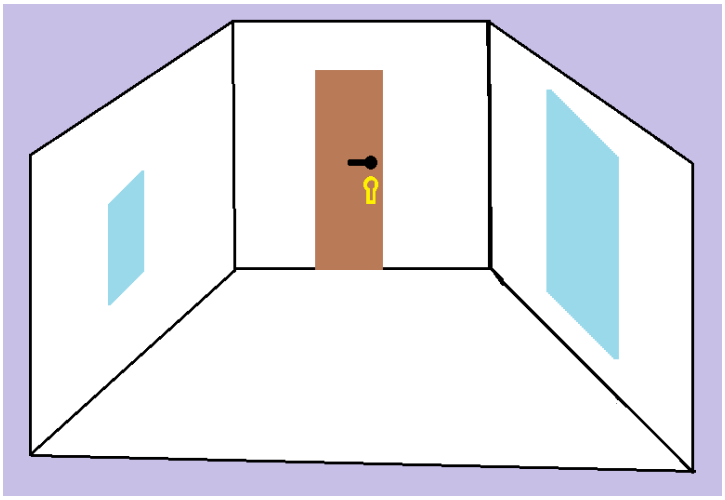
Για αυτές τις περιπτώσεις εξετάστηκε η εκτέλεση της αναζήτησης με χρήση του προγραμματιστικού μοντέλου MapReduce. Με αυτό επιτυγχάνεται ο διαχωρισμός των δεδομένων σε πολλά ανεξάρτητα κομμάτια, και συνεπώς ο καταμερισμός τόσο των απαιτούμενων I/O, όσο και της επεξεργαστικής ισχύς, με αποτέλεσμα την γρηγορότερη απόκριση των queries. Η τεχνική που υλοποιείται από αυτή τη διπλωματική ονομάζεται *MapReduce* και απευθύνεται σε διαχειριστές που είναι αρκετά εξοικειωμένοι με αυτό το προγραμματιστικό μοντέλο καθώς και του τρόπου εφαρμογής του στη MongoDB.

Με την *MapReduce* τεχνική η αναζήτηση θα πραγματοποιηθεί στα metadata, δηλαδή στις MPEG-7 και X3D περιγραφές, των αρχείων που θα εξεταστούν. Τα metadata διατηρούνται στη MongoDB, σε JSON Documents στη μορφή που παρουσιάστηκε για την *KeyValue* μέθοδο της προηγούμενης ενότητας. Το Query που υλοποιήθηκε είναι "*Return dominant color of scene*", με την αναζήτηση να πραγματοποιείται απ' ευθείας στα JSON Documents χωρίς να απαιτούνται μετατροπές στο schema των metadata. Από την αναζήτηση επιστρέφεται το χρώμα που εμφανίζεται σε μεγαλύτερο ποσοστό στα αρχεία που εξετάστηκαν, καθώς και τα ονόματα των αντικειμένων που έχουν το συγκεκριμένο χρώμα.

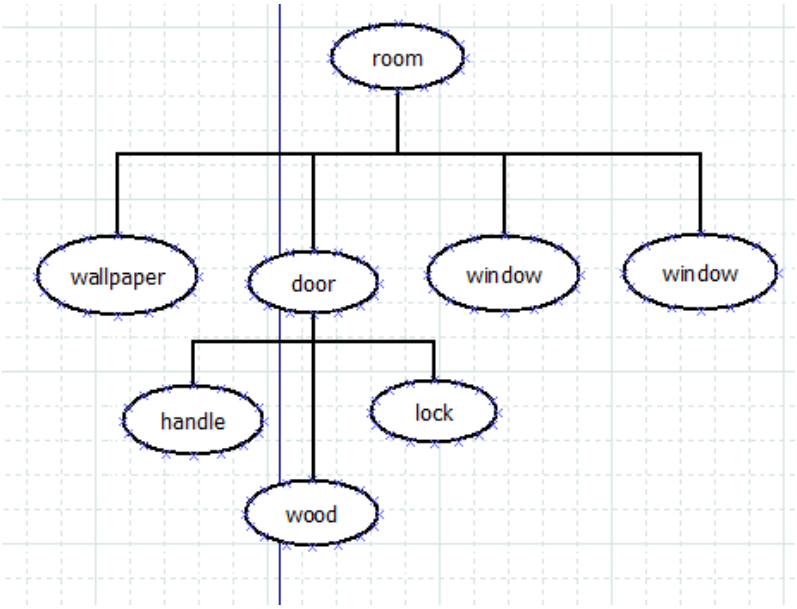
Για τις ανάγκες της διπλωματικής ήταν απαραίτητη η θεώρηση ορισμένων υποθέσεων για την δόμηση των δεδομένων, δηλαδή των σκηνών, στα οποία θα εκτελεστεί η αναζήτηση. Συγκεκριμένα θεωρήθηκε πως κάθε σκηνή αποτελείται από μικρά θεμελιώδη αντικείμενα,

δηλαδή αντικείμενα που δεν μπορούν να τμηματοποιηθούν σε άλλα μικρότερα. Τα θεμελιώδη αντικείμενα, leafObjects, χρησιμοποιούνται για τον σχηματισμό μεγαλύτερων αντικειμένων, fatherObjects. Αυτά με τη σειρά τους μπορούν να αποτελούν κομμάτια ακόμα μεγαλύτερων αντικειμένων της σκηνής, που επίσης ονομάζονται fatherObjects. Τέλος ορίζονται τα rootObjects, αντικείμενα που δεν συμμετέχουν στον σχηματισμό άλλων.

Με βάση την παραπάνω υπόθεση, είναι φανερό πως η σκηνή δομείται από leafObjects, τα οποία αναδρομικά δομούν τα rootObjects της σκηνής. Συνεπώς τα χρώματα της σκηνής προκύπτουν από τα χρώματα των leafObjects που την αποτελούν. Η δομή που περιεγράφηκε θυμίζει δένδρο ή δένδρα, ανάλογα με το πόσα rootObjects εμφανίζονται στη σκηνή. Κάθε δένδρο έχει σαν ρίζα ένα rootObject, ως ενδιάμεσους κόμβους τα παιδιά των rootObjects και fatherObjects και σαν φύλλα όλα τα leafObjects που τελικά δομούν την σκηνή. Ένα παράδειγμα της αναπαράστασης της δενδρικής μορφής της σκηνής "*sample*", Εικόνα 28, που αποτελείται από ένα rootObject "*room*" δίνεται από την Εικόνα 29.



Εικόνα 28 Scene "sample"



Εικόνα 29 Tree Structure of Scene "sample"

Για την σκηνή "*sample*", και σύμφωνα με το δένδρο που προέκυψε από αυτήν, εξάγουμε τα ακόλουθα συμπεράσματα. Η σκηνή διαθέτει ένα rootObject που ονομάζεται "*room*". Αυτό αποτελείται από τρία leafObjects, ένα "*wallpaper*" και δύο "*window*", και από ένα fatherObject με το όνομα "*door*". Το αντικείμενο "*door*" συντίθεται από τρία άλλα αντικείμενα, τα οποία είναι leafObjects και ονομάζονται "*handle*", "*wood*" και "*lock*". Είναι φανερό πως τα χρώματα της σκηνής "*sample*" προκύπτουν άμεσα από τα leafObjects που την αποτελούν, "*wallpaper*", "*window*", "*window*", "*handle*", "*wood*" και "*lock*", και όχι από τα ενδιάμεσα αντικείμενα που εμφανίζονται σε αυτήν, "*room*" και "*door*".

Σύμφωνα με τα παραπάνω το κυρίαρχο χρώμα της σκηνής υπολογίζεται με βάση το χρώμα των leafObjects σε συνδυασμό με το ποσοστό της σκηνής που καταλαμβάνει το καθένα. Στον υπολογισμό αυτό λαμβάνονται επίσης υπόψιν οι περιπτώσεις της μεγέθυνσης και της πιθανής επικάλυψης των leafObjects. Τελικά το ποσοστό που καταλαμβάνει σε μια σκηνή ένα leafObject υπολογίζεται από τρία διαφορετικά μεγέθη. Το ποσοστό του αντικειμένου που είναι ορατό, την μεγέθυνση ή σμίκρυνση που έχει και από το κανονικοποιημένο μέγεθός του. Με τον όρο κανονικοποιημένο μέγεθος ορίζουμε το ποσοστό της σκηνής που καταλαμβάνει ένα leafObject όταν συναντάται σε μεγέθυνση 1:1 και δεν επικαλύπτεται από άλλο, δηλαδή όταν είναι ολόκληρο ορατό.

Για την περιγραφή των rootObject και fatherObject, χρησιμοποιούνται metadata που υπακούν στα πρότυπα MPEG-7 και X3D. Σε αυτά, σύμφωνα με το schema των προτύπων, συμπληρώνονται οι απαραίτητες πληροφορίες για την αναπαράσταση της δενδρικής μορφής της σκηνής και των χαρακτηριστικών των αντικειμένων που τη συνθέτουν. Για τα leafObjects επιλέχθηκε η αναπαράστασή τους με αρχεία τύπου XML και ορισμού μόνο των χαρακτηριστικών τους που μας ενδιαφέρουν. Είναι φανερό πως η χρήση του προτύπου MPEG-7 είναι δυνατή, αλλά παράλληλα θα απαιτούσε πολλές εξειδικευμένες γνώσεις περιγραφής των αντικειμένων, που ξέφευγαν από τα πλαίσια αυτής της διπλωματικής.

Καθώς τα leafObjects αποτελούν XML αρχεία, τα μεγέθη που τα χαρακτηρίζουν δομούνται σε nodes. Διακρίνονται τρείς κόμβοι με ονόματα "*name*", "*normalizedForm*" και "*color*". Το element "*name*" προσδιορίζει το όνομα του leafObject, και το element "*normalizedForm*" δίνει το ποσοστό της σκηνής που καταλαμβάνει το αντικείμενο στο κανονικοποιημένο μέγεθός του. Στον κόμβο "*color*" διατηρούνται τρία childNodes, με ονόματα "*red*", "*green*" και "*blue*" που δίνουν αντίστοιχα τις RGB συνιστώσες που συνθέτουν το χρώμα του leafObject. Στον Πίνακας 18 δίνεται το XML αρχείο που περιγράφει το leafObject "*wallpaper*" της Εικόνα 28. Από αυτόν φαίνεται πως το αντικείμενο "*wallpaper*" έχει χρώμα λευκό, (255,255,255) στην κλίμακα RGB, και όταν βρίσκεται σε κανονικοποιημένο μέγεθος καταλαμβάνει το 50% της σκηνής.

wallpaper.xml

```
<Mpeg7>
    <name>
        wallpaper.xml
    </name>
    <normalizedForm>
        50%
    </normalizedForm>
    <color>
        <red>
            255
        </red>
        <green>
            255
        </green>
        <blue>
            255
        </blue>
```

```
    </color>
</Mpeg7>
```

Πίνακας 18 leafObject Example

Όσον αφορά στην απεικόνιση των fatherObjects και rootObjects, χρησιμοποιείται το πρότυπο MPEG-7 και συγκεκριμένα το MPEG-7 Part 5, Multimedia Description Schemes (MDS). Για κάθε fatherObject και rootObject προστίθενται, στο κομμάτι περιγραφής των Textures που συνθέτουν την σκηνή, σε τρία strings αντίστοιχα, το όνομα του κάθε childObject, η μεγέθυνση στην οποία συναντάται και το ποσοστό του που είναι τελικά ορατό στη σκηνή. Ένα παράδειγμα περιγραφής τέτοιου Object δίνεται από τον Πίνακας 19, ο οποίος δείχνει τα MPEG-7 metadata του rootObject "*room*" της Εικόνα 28.

room.xml

```
<?xml version="1.0"?>
<Mpeg7 xmlns="urn:mpeg:mpeg7:schema:2001">
  <Description xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                                        xsi:type="ContentEntityType">
    <MultimediaContent xsi:type="MultimediaCollectionType">
      <StructuredCollection>
        <Collection xsi:type="ContentCollectionType" id="Textures">
          <ContentCollection id="Textures_N10082" name=”Group_N10082”>
            <Content xsi:type="MultimediaType" id="ImageTexture_N100A0">
              <Multimedia>
                <MediaLocator>
                  <MediaUri>"wallpaper.xml" "100%" "90%"</MediaUri>
                </MediaLocator>
              </Multimedia>
            </Content>
          </ContentCollection>
          <ContentCollection id="Textures_N100BA"name="Transform_N100BA">
            <Content xsi:type="MultimediaType" id="ImageTexture_N100C5">
              <Multimedia>
                <MediaLocator>
                  <MediaUri>"door.xml" "150%" "100%"</MediaUri>
                </MediaLocator>
              </Multimedia>
            </Content>
          </ContentCollection>
          <ContentCollection id="Textures_N100CD" name="Transform_N100CD">
            <Content xsi:type="MultimediaType" id="ImageTexture_N100D8">
              <Multimedia>
                <MediaLocator>
                  <MediaUri>"window.xml" "100%" "100%"</MediaUri>
                </MediaLocator>
              </Multimedia>
            </Content>
          </ContentCollection>
          <ContentCollection id="Textures_N100CD" name="Transform_N100CD">
            <Content xsi:type="MultimediaType" id="ImageTexture_N100D8">
              <Multimedia>
                <MediaLocator>
                  <MediaUri>"window.xml" "200%" "100%"</MediaUri>
                </MediaLocator>
              </Multimedia>
            </Content>
          </ContentCollection>
        </Collection>
      </StructuredCollection>
    </MultimediaContent>
  </Description>
</Mpeg7>
```

Πίνακας 19 fatherObject Example

Από τον Πίνακας 19 φαίνεται πως το Object "*room*" της σκηνής "*sample*" απαρτίζεται από τέσσερα childObjects, "*wallpaper*", "*door*", "*window*" και "*window*". Το "*wallpaper*" έχει μεγέθυνση "*100%*", δηλαδή 1:1, και είναι σε ποσοστό "*90%*" ορατό στην ολική σκηνή. Το "*door*" και το ένα "*window*" συναντώνται στις κανονικοποιημένες διαστάσεις τους και είναι εξ' ολοκλήρου ορατά. Τέλος το δεύτερο "*window*" είναι σε μεγέθυνση "*200%*", δηλαδή διπλάσιο από το κανονικοποιημένο μέγεθος, και είναι "*100%*" ορατό στη σκηνή "*sample*".

Είναι φανερό πως για τον εντοπισμό του κυρίαρχου χρώματος απαιτείται διάσχιση όλων των δένδρων που ανήκουν σε μια σκηνή μέχρι τον εντοπισμό όλων των leafObjects. Κάθε φορά που εντοπίζεται ένα leafObject, υπολογίζεται το ποσοστό εμφάνισής του στη σκηνή σύμφωνα με την Εξίσωση 3. Όταν ολοκληρωθεί η διάσχιση των δένδρων αθροίζεται το ποσοστό εμφάνισης για κάθε leafObject και υπολογίζεται το τελικό ποσοστό εμφάνισης κάθε χρώματος. Από την αναζήτηση επιστρέφεται το χρώμα που εμφανίζεται σε μεγαλύτερο ποσοστό και τα ονόματα των leafObjects που συμμετείχαν στην ανάδειξή του σε κυρίαρχο χρώμα.

$$Total\_view = scale \cdot visible \cdot normalizedForm$$

Εξίσωση 3 Calculate Total_view of examined leafObject

Σε αναλογία με τις προηγούμενες τεχνικές, κάθε αντικείμενο που συνθέτει την σκηνή, leafObject, fatherObject και rootObject, αποθηκεύεται σε ένα JSON Document, το οποίο προκύπτει με μετασχηματισμό των MPEG-7 και X3D περιγραφών του αντικειμένου σε key/value pairs, όπως παρουσιάστηκε στην τεχνική *KeyValue*. Για τον έλεγχο του scalability της *MapReduce* τεχνικής η αναζήτηση εφαρμόστηκε όταν τα δεδομένα, δηλαδή τα rootObjects που συνέθεταν τις σκηνές, υπήρχαν σε 1, 2, 4, 8, 16, 32, 64, 128, 256, 512 και 1024 αντίγραφα. Κάθε σκηνή περιγράφεται από την collection "*KeyValue*" της database "*ScenexNoCopies*", όπου NoCopies ο αριθμός των αντιγράφων των rootObjects της. Διευκρινίζεται πως οι collections "*KeyValue*" διαθέτουν τα JSON Documents που περιγράφουν κάθε αντικείμενο τους, rootObject, fatherObject και leafObject, σε ένα μόνο αντίγραφο, όσες φορές και αν εμφανίζονται αυτά στη σκηνή.

Στην *MapReduce* τεχνική απαιτείται η μετατροπή των metadata των αντικειμένων που συνθέτουν τις σκηνές σε κατάλληλα JSON Documents και στη συνέχεια η εφαρμογή των map και reduce functions που θα διασχίσουν τα JSON Documents και θα εξάγουν το κυρίαρχο χρώμα για την εξεταζόμενη σκηνή. Το κομμάτι της δημιουργίας των αρχείων που θα αποθηκευτούν στην MongoDB έχει ήδη παρουσιαστεί και είναι το ίδιο με αυτό για την τεχνική *KeyValue*. Ακολουθεί αναλυτική περιγραφή των σκηνών που χρησιμοποιήθηκαν και της αναζήτησης που εκτελέστηκε.

6.2.4.1.1 Αναζήτηση

Για την αναζήτηση με χρήση της τεχνικής *MapReduce* χρησιμοποιήθηκε η σκηνή "*test case 3*" που περιέχει μια μηχανή με τα επιμέρους εξαρτήματά της. Στη συνέχεια η ίδια αναζήτηση εκτελέστηκε όταν η σκηνή διέθετε 2, 4, 8, 16, 32, 64, 128, 256, 512 και 1024 μηχανές ίδιες με αυτήν της αρχικής σκηνής. Στον Πίνακας 20 φαίνεται η δενδρική δομή των rootObjects, fatherObjects και leafObjects που απαρτίζουν την σκηνή "*test case 3*".

Scene "test case 3" Tree Structure
test case 3_engine test case 3_lubrication_system pump

```
            rod
      test case 3_starter_system
            driveshaft
            bumb
      test case 3_diagnostic_system
            test case 3_engineboard
                  cable
                  test case 3_motherboard
                        chipset
                        test case 3_memory
                              cooling
                              coolingfan
                              chipset
                        socket
            test case 3_powerboard
                  battery
                  fpu
      test case 3_flywhells
            disk
            crank
            cylinder
      crankshaft
      test case 3_control_system
            openclose
            startup
      test case 3_cooling_system
            air
            liquid
      test case 3_valves
            test case 3_piston_engine_valves
                  valve x 10
            test case 3_control_valves
                  turbo_valve     x 2
            test case 3_injection_valves
                  beck x 20
      test case 3_piston_engine_valves
            valve x 10
```

Πίνακας 20 Scene "test case 3" Tree Structure

Όπως έχει αναφερθεί το χρώμα της σκηνής δίνεται από τα leafObjects που την αποτελούν. Ωστόσο για τον προσδιορισμό του αριθμού των leafObjects που υπάρχουν και του ποσοστού συμμετοχής τους στην τελική σκηνή είναι απαραίτητη η διάσχιση των δένδρων που ορίζουν τα rootObjects της σκηνής. Για αυτή την αναζήτηση θα χρησιμοποιηθεί η λογική του αλγορίθμου BFS που παρουσιάστηκε στην προηγούμενη ενότητα, Πίνακας 15.

Πιο αναλυτικά ο αλγόριθμος αναζήτησης που χρησιμοποιείται αποτελείται από 3 στάδια. Στο πρώτο στάδιο εντοπίζονται τα rootObjects της σκηνής. Αυτό είναι απαραίτητο για την εκκίνηση της αναζήτησης με διάσχιση των δένδρων που ορίζονται από τα rootObjects. Παράλληλα δημιουργούνται δομές στις οποίες αποθηκεύονται πληροφορίες για το κάθε Object.

Στο δεύτερο στάδιο πραγματοποιείται η διάσχιση των δένδρων. Η αναζήτηση ξεκινάει από τα rootObjects και συνεχίζεται ανά επίπεδο. Τα rootObjects σημειώνουν για κάθε childObject τους, στις αντίστοιχες δομές που δημιουργήθηκαν στο προηγούμενο στάδιο, το πλήθος εμφάνισής τους καθώς και αθροιστικά, για τα όμοια childObjects, το ποσοστό

συμμετοχής τους στην κατασκευή της σκηνής. Ακολουθεί αναδρομικά η διάσχιση των υπόλοιπων επιπέδων των δένδρων. Κάθε φορά που εξετάζεται ένα Object από την αντίστοιχη δομή, που δημιουργήθηκε στο πρώτο στάδιο, αφαιρείται μία μονάδα από το πλήθος εμφάνισης του αντικειμένου. Στη συνέχεια το εξεταζόμενο Object υποβάλλει τα παιδιά του, συμπληρώνοντας το πλήθος εμφάνισής τους και το ποσοστό συμμετοχής τους στη σκηνή. Η αναδρομή συνεχίζεται μέχρι να διασχιστεί ολόκληρο το δένδρο.

Στο τρίτο στάδιο έχει ολοκληρωθεί η διάσχιση των δένδρων, οπότε αθροίζονται τα ποσοστά συμμετοχής των leafObjects και υπολογίζεται το ολικό ποσοστό συμμετοχής τους στη σκηνή, με χρήση του κανονικοποιημένου μεγέθους τους. Πραγματοποιείται μία συγχώνευση με βάση το χρώμα και επιστρέφεται το κυρίαρχο χρώμα της σκηνής και τα αντικείμενα που το έχουν.

Η εφαρμογή του προγραμματιστικού μοντέλου MapReduce απαιτεί την δημιουργία δύο functions, της *map*, η οποία θα μετατρέψει τα δεδομένα εισόδου της σε key/value pairs, και της *reduce*, η οποία θα ομαδοποιήσει τα key/value pairs της map σύμφωνα με το key τους και θα εφαρμόσει μια πράξη συγχώνευσης στις ομαδοποιημένες τιμές του κάθε key. Ακολουθεί αναλυτική περιγραφή των classes που υλοποιήθηκαν για εκτέλεση της αναζήτησης με την τεχνική *MapReduce* σύμφωνα με τον αλγόριθμο που παρουσιάστηκε προηγουμένως.

Με κλήση της class *SearchMapReduce* ορίζεται η σκηνή στην οποία θα εκτελεστεί η αναζήτηση. Η *SearchMapReduce* καλεί για όλα τα αντίγραφα που διατηρούνται για τα αντικείμενα της σκηνής την μέθοδο *MapReduce* της class *MapReduce*. Η *MapReduce* μέθοδος δέχεται ως ορίσματα την σκηνή στην οποία θα εκτελεστεί η αναζήτηση, "*Scene*", καθώς και το πλήθος των αντιγράφων των αντικειμένων που διατηρεί, "*NoCopies*".

Η μέθοδος *MapReduce* είναι υπεύθυνη για τον ορισμό της βάσης δεδομένων που θα χρησιμοποιηθεί, "*ScenexNoCopies*", και για τον ορισμό του *directoryPath* στο οποίο θα γραφούν τα αποτελέσματα της αναζήτησης. Χρησιμοποιώντας την μέθοδο *currentTimeMillis* γράφει τον χρόνο εκτέλεσης της αναζήτησης στο *Αρχείο Χρόνου Απόκρισης*, που αποθηκεύεται στο path "*Duration\MapReduce*" με όνομα "*MapReduce Scene*.txt". Το *Αρχείο Αποτελεσμάτων* ορίζεται να έχει όνομα "*Scene application xNoCopies*.txt" και αποθηκεύεται στο *path* "*MapReduceData\Results\MapReduce*". Ακολουθεί η κλήση της μεθόδου *CallMapReduce* με όρισμα την βάση δεδομένων στην οποία είναι αποθηκευμένα τα αντικείμενα που συνθέτουν την σκηνή "*Scene*".

Η *CallMapReduce* είναι η μέθοδος που καλεί τις απαραίτητες *map* και *reduce* functions για την εκτέλεση της αναζήτησης. Συνολικά καλούνται τρία MapReduce tasks, αντίστοιχα με τα στάδια εκτέλεσης του αλγορίθμου αναζήτησης.

Κατά το πρώτο MapReduce η map function, *Map1_FindFather*, δέχεται σαν είσοδο την collection "*KeyValue*" της database που δέχτηκε σαν όρισμα η *CallMapReduce*, ενώ η reduce function ,*Reduce1_FindFather*, αποθηκεύει τα δεδομένα εξόδου της σε μια νέα collection "*MapReduce*". Η map function στο πρώτο MapReduce στόχο έχει την εύρεση των rootObjects της collection "*KeyValue*". Για κάθε αντικείμενο της σκηνής δημιουργείται ένα JSON Document που αποθηκεύεται στην collection "*MapReduce*" το οποίο έχει ένα key/value pair με key *_id* και value το όνομα του αντικειμένου, και ένα δεύτερο key/value pair με key *value* και value ένα Embedded JSON Object. Στο Embedded JSON Objects περιλαμβάνονται key/value pairs που δείχνουν τα χαρακτηριστικά του κάθε αντικειμένου. Συγκεκριμένα διακρίνεται ένα key *count*, που δείχνει τον αριθμό που συναντάται το συγκεκριμένο αντικείμενο στη σκηνή, το key *is_child*, που δείχνει αν το αντικείμενο που εξετάζεται είναι childObject ενός άλλου και το key *cur_view* που δείχνει το ποσοστό συμμετοχής του αντικειμένου στη σκηνή. Διακρίνονται και εξειδικευμένα key/value pairs στο Embedded JSON Object ανάλογα με τη φύση των αντικειμένων που εξετάζονται. Για fatherObejcts και rootObjects αποθηκεύεται το key *child*, το value του οποίου είναι ένας JSON Array που περιέχει JSON Objects με τα χαρακτηριστικά των childObjects των αντικειμένων που εξετάζονται. Τα JSON Objects του JSON Array έχουν δύο key/value pairs, ένα *name*, που διατηρεί το όνομα του childObject και ένα *cur_view* που διαθέτει το ποσοστό

συμμετοχής του childObject στη σκηνή και υπολογίζεται σύμφωνα με την Εξίσωση 4. Όσον αφορά στα leafObjects προστίθενται τα key/value pairs *norma* και *color* που περιλαμβάνουν αντίστοιχα το κανονικοποιημένο μέγεθος και το χρώμα του αντικειμένου.

$$cur\_view = scale \cdot visible$$

Εξίσωση 4 Calculate cur_view of examined childObject

Με την map function για κάθε αντικείμενο της *KeyValue* collection εντοπίζονται αν υπάρχουν τα παιδιά του. Για κάθε παιδί γίνεται emit ένα key/value pair με key το όνομα του αντικειμένου και value ένα νέο key/value pair με τιμές *is_child/1* αντίστοιχα. Για τα αντικείμενα που διαθέτουν παιδιά συμπληρώνεται ο πίνακας *child* που περιεγράφηκε παραπάνω καθώς και ένα *is_child* key/value pair με value *0*. Όταν εξετάζονται leafObjects γίνεται emit ένα key/value pair με key το όνομα του αντικειμένου και value ένα JSON Object με τα *norma* και *color* key/value pairs. Ακολουθεί η reduce function η οποία για όλα τα όμοια keys, δηλαδή για όλες τις πληροφορίες που έχουν γίνει emit και αφορούν σε ένα αντικείμενο, αθροίζει τα values των *is_child* και *count* key/value pairs. Τα μόνα Objects που θα έχουν $count > 0$ και $is\_child = 0$ θα είναι τα rootObjects που συμμετέχουν στη σκηνή, καθώς όλα τα υπόλοιπα Objects είναι παιδιά κάποιου άλλου αντικειμένου και θα έχουν $is\_child \geq 1$. Για όλα τα αντικείμενα που δεν είναι rootObjects η reduce function μηδενίζει το *count* key/value pair καθώς θέλουμε η αναζήτηση να ξεκινήσει με δεδομένα μόνο τα rootObjects. Παραδείγματα για την μορφή των JSON Documents των rootObjects, fatherObjects και leafObjects, μετά την εκτέλεση της reduce function, δίνονται από τους Πίνακας 21, Πίνακας 22 και Πίνακας 23 αντίστοιχα.

rootObject

```
{
   "_id" : "rootObject_name",
   "value" : {
               "count" : "Number_of_occurrences",
               "is_child" : 0,
               "child" : [
                          {"name" : "child_1_name",
                           "cur_view" : "child_1_cur_view"},
                          …,
                          {"name" : "child_n_name",
                           "cur_view" : "child_n_cur_view"}
                         ]
            }
}
```

Πίνακας 21 rootObject as JSON Document after first MapReduce

fatherObject

```
{
   "_id" : "fatherObject_name",
   "value" : {
               "count" : "Number_of_occurrences",
               "cur_view" : 0,
               "is_child" : 1,
               "child" : [
                          {"name" : "child_1_name",
                           "cur_view" : "child_1_cur_view"},
                          …,
```

```
                                {"name" : "child_n_name",
                                 "cur_view" : "child_n_cur_view"}
                               ]
            }
}
```

Πίνακας 22 fatherObject as JSON Document after first MapReduce

leafObject

```
{
   "_id" : "leafObject_name",
   "value" : {
                "count" : "Number_of_occurrences",
                "cur_view" : 0,
                "norma" : "normalized_form",
                "color" : {
                             "r" : "red_value",
                             "g" : "green_value",
                             "b" : "blue_value"
                           }
              }
}
```

Πίνακας 23 leafObject as JSON Document after first MapReduce

Ακολουθεί το δεύτερο MapReduce task το οποίο εκτελεί την διάσχιση των δένδρων μέχρι να εξεταστούν όλα τα επίπεδα. Η είσοδος της map function, *Map2_CrossTree*, είναι η collection "*MapReduce*". Τα αποτελέσματα της reduce function, *Reduce2_CrossTree*, γράφονται επίσης στην collection "*MapReduce*" συγχωνεύοντας τα αποτελέσματα για JSON Documents με ίδιο *_id* key/value pair. Η map function εξετάζει όλα τα JSON Documents. Για αυτά που έχουν παιδιά, δηλαδή έχουν τον *child* Array, και το value του key *count* είναι μεγαλύτερο του 0, κάνουν emit count φορές τα childObjects του εξεταζόμενου αντικειμένου με key το όνομα του childObject και value τα *cur_view* και *count*, όπου η τιμή του *count* είναι 1, key/value pairs που χαρακτηρίζουν τα childObjects. Στη συνέχεια γίνεται emit το key/value pair που αφορά στο αντικείμενο που μόλις εξετάστηκε με key το όνομά του και value το key/value pair *count* με τιμή 0. Στην περίπτωση που δεν υπάρχει ο πίνακας *child*, το JSON Document αφορά σε leafObject, οπότε γίνεται emit το *color* και η *norma* του. Ακολουθεί η εκτέλεση της reduce function, η οποία αθροίζει τα values των *cur_view* και *count* των αντικειμένων με ίδιο key.

Είναι φανερό πως η αναζήτηση ολοκληρώνεται όταν έχουν εξεταστεί όλα τα rootObjects και fatherObjects, δηλαδή όταν *count* μεγαλύτερο του μηδενός έχουν μόνο τα leafObjects. Σε αυτό το σημείο εκτελείται η αναδρομή. Αν υπάρχει έστω και ένα fatherObject το οποίο έχει $count > 0$, εκτελείται ξανά το δεύτερο MapReduce, με την αναδρομή να σταματάει όταν όλα τα fatherObjects έχουν $count = 0$. Τότε η collection "*MapReduce*" θα έχει JSON Documents αντίστοιχα για rootObjects, fatherObjects και leafObjects της μορφής που παρουσιάζονται στους Πίνακας 24, Πίνακας 25 και Πίνακας 26.

rootObject

```
{
   "_id" : "rootObject_name",
   "value" : {
                "count" : 0,
                "is_child" : 0,
                "child" : [
```

```
                            {"name" : "child_1_name",
                             "cur_view" : "child_1_cur_view"},
                            …,
                            {"name" : "child_n_name",
                             "cur_view" : "child_n_cur_view"}
                           ]
            }
}
```

Πίνακας 24 rootObject as JSON Document after second MapReduce

fatherObject

```
{
   "_id" : "fatherObject_name",
   "value" : {
               "count" : 0,
               "cur_view" : 0,
               "is_child" : 1,
               "child" : [
                           {"name" : "child_1_name",
                            "cur_view" : "child_1_cur_view"},
                           …,
                           {"name" : "child_n_name",
                            "cur_view" : "child_n_cur_view"}
                          ]
            }
}
```

Πίνακας 25 fatherObject as JSON Document after second MapReduce

leafObject

```
{
   "_id" : "leafObject_name",
   "value" : {
               "count" : "Number_of_occurrences",
               "cur_view" : "Total_view_of_leafObject",
               "norma" : "normalized_form",
               "color" : {
                           "r" : "red_value",
                           "g" : "green_value",
                           "b" : "blue_value"
                          }
             }
}
```

Πίνακας 26 leafObject as JSON Document after second MapReduce

Όταν έχουν διασχιστεί όλα τα δένδρα που συνθέτουν την σκηνή πραγματοποιείται το τρίτο MapReduce task, το οποίο έχει σαν είσοδο την collection "*MapReduce*" και έξοδο την ίδια collection, διαγράφοντας τα προηγούμενα δεδομένα της. Η map function του, *Map3_CalcResult*, υπολογίζει το συνολικό ποσοστό συμμετοχής των leafObjects στη σκηνή και κάνει emit ένα key/value pair με key το χρώμα του αντικειμένου ως string και με value ένα JSON Object. Το JSON Object περιέχει το key/value pair με key *names* και value το όνομα του αντικειμένου και ένα δεύτερο key/value pair με key *whole* και value το συνολικό ποσοστό συμμετοχής του αντικειμένου στη σκηνή υπολογίζοντας και το κανονικοποιημένο μέγεθος του αντικειμένου σύμφωνα με την Εξίσωση 5. Η reduce function,

Reduce3_CalcResult, για τα όμοια keys, δηλαδή για το κάθε χρώμα που εμφανίζεται στη σκηνή, αθροίζει το ποσοστό εμφάνισής του, σύμφωνα με τις *whole* values που έγιναν emit από τα leafObjects, και αποθηκεύει στο value του key *names* έναν JSON Array με τα ονόματα των αντικειμένων που έχουν το εξεταζόμενο χρώμα. Συνεπώς μετά την ολοκλήρωση του τρίτου MapReduce η collection "*MapReduce*" θα περιέχει colorObjects της μορφής που παρουσιάζονται στον Πίνακας 27.

$$whole\_view = cur\_view \cdot norma$$

Εξίσωση 5 Calculate whole_view of leafObject with Reduce3 Task

colorObject

```
{
   "_id" : "red_value green_value blue_value",
   "value" : {
               "whole" : "Total_view_of_colorObject",
               "names" : [
                           leafObject_1_name,
                           …,
                           leafObject_k_name
                         ]
             }
}
```

Πίνακας 27 colorObject as JSON Document after third MapReduce

Η αναζήτηση ολοκληρώνεται με την εκτέλεση της μεθόδου *QueryOutput* της class *MapReduce*. Η *QueryOutput* με εκτέλεση του MongoDB-Query 19, ταξινομεί τα JSON Documents σε φθίνουσα σειρά σύμφωνα την τιμή "*value.whole*" και γράφει στο *Αρχείο Αποτελεσμάτων* το κυρίαρχο χρώμα της σκηνής και τα αντικείμενα που το διαθέτουν.

```
db.MapReduce.find(
     {}, {"value.whole":1, "value.color":1, "value.names":1}).
     sort({"value.whole":-1);
```

MongoDB-Query 19 Sort MapReduce Collection by value "value.whole" in descending order

Τελικά προκύπτει ένα *Αρχείο Αποτελεσμάτων* για κάθε μία από τις 11 εκτελέσεις της αναζήτησης. Αυτές αφορούν στον εντοπισμό του κυρίαρχου χρώματος όταν στη σκηνή συμμετέχουν αντικείμενα σε 1, 2, 4, 8, 16, 32, 64, 128, 256, 512 και 1024 αντίγραφα. Παράλληλα συμπληρώνεται με 11 μετρήσεις και το *Αρχείο Χρόνου Απόκρισης*.

6.2.4.2 NoMapReduce

Για την αξιολόγηση των αποτελεσμάτων της τεχνικής *MapReduce* ήταν απαραίτητη η ανάπτυξη ενός μηχανισμού που θα εκτελούσε το ίδιο query, στα ίδια δεδομένα, χωρίς όμως να γίνει χρήση των εργαλείων του προγραμματιστικού μοντέλου MapReduce και συγκεκριμένα των συναρτήσεων που προσφέρονται για το σκοπό αυτό από την MongoDB. Για το λόγο αυτό υλοποιήθηκε η τεχνική *NoMapReduce*., η οποία χρησιμοποιεί τα metadata της σκηνής "*test case 3"*, τα οποία είναι, όμοια με την μέθοδο *MapReduce*, αποθηκευμένα στην collection "*KeyValue*". Ακολουθεί περιγραφή του αλγορίθμου που χρησιμοποιήθηκε για εκτέλεση του query "*Return dominant color of scene*", χωρίς χρήση map και reduce

functions, και οι classes που αναπτύχθηκαν για εφαρμογή του στα metadata της σκηνής εισόδου, "*test case 3"*.

Σε αντιστοιχία με την μέθοδο *MapReduce*, ο αλγόριθμος αναζήτησης αποτελείται από 3 στάδια. Στο πρώτο εντοπίζονται οι ρίζες των δένδρων που περιέχουν τα αντικείμενα της σκηνής, δηλαδή τα rootObjects. Ταυτόχρονα δημιουργούνται δομές, για κάθε Object της σκηνής, που θα διατηρούν την πορεία εξέλιξης της αναζήτησης. Στη συνέχεια χρησιμοποιείται μια ουρά για διάσχιση των δένδρων και ενημέρωση των κατάλληλων δομών για το ποσοστό συμμετοχής τους στην κατασκευή της σκηνής. Αν το Object που εξετάστηκε από την ουρά αποτελεί fatherObject ή rootObject, υποβάλλονται τα παιδιά του στο τέλος της ουράς. Η αναζήτηση ολοκληρώνεται όταν έχουν εξεταστεί όλα τα δένδρα, δηλαδή όταν αδειάσει η ουρά. Τέλος, κατά το τρίτο στάδιο, συγχωνεύονται τα ποσοστά συμμετοχής των δομών ανάλογα με το χρώμα, ώστε να προκύψει αθροιστικά η συνολική εμφάνιση του κάθε ενός, και επιστρέφεται το κυρίαρχο χρώμα της σκηνής και τα leafObjects που το διαθέτουν.

Όσον αφορά στις classes που υλοποιήθηκαν, για την εκτέλεση της αναζήτησης καλείται η class *SearchNoMapReduce*, η οποία ορίζει την σκηνή στην οποία θα βρεθεί το κυρίαρχο χρώμα. Σε αντιστοιχία με την τεχνική *MapReduce* η αναζήτηση θα πραγματοποιηθεί στα metadata των αντικειμένων της ζητούμενης σκηνής όταν αυτά βρίσκονται σε 1, 2, 4, 8, 16, 32, 64, 128, 256, 512 και 1024 αντίγραφα. Στη συνέχεια καλείται η μέθοδος *NoMapReduce* της class *NoMapReduce*, με ορίσματα την σκηνή που θα χρησιμοποιηθεί "*Scene*" καθώς και το πλήθος των αντιγράφων των metadata, "*NoCopies*".

Όπως και στην τεχνική *MapReduce* που αναλύθηκε προηγουμένως, η μέθοδος *NoMapReduce* ορίζει τον φάκελο στον οποίο θα αποθηκευτούν τα αποτελέσματα της αναζήτησης, "*MapReduceData\Results\NoMapReduce*\", στο αρχείο με όνομα "*Scene application xNoCopies*.txt". Ορίζεται επίσης το αρχείο στο οποίο θα αποθηκευτεί ο χρόνος εκτέλεσης του query "*NoMapReduce Scene*.txt", στο directoryPath "*Duration\NoMapReduce*\". Τέλος σύμφωνα με το "*NoCopies*" επιλέγεται η βάση δεδομένων "*ScenexNoCopies*" στην οποία είναι αποθηκευμένα τα metadata που θα εξεταστούν. Ακολουθεί κλήση της μεθόδου *CallNoMapReduce*.

Η μέθοδος *CallMapReduce* δέχεται σαν είσοδο την βάση δεδομένων στην οποία είναι αποθηκευμένα τα metadata της σκηνής. Είναι υπεύθυνη για την κλήση των απαραίτητων συναρτήσεων που θα διασχίσουν τα δένδρα που ορίζονται από τα rootObjects της σκηνής, σύμφωνα με τον αλγόριθμο που παρουσιάστηκε προηγουμένως. Για το σκοπό αυτό ορίζεται κατ' αρχάς η collection "*KeyValue*", στην οποία διατηρούνται τα metadata της σκηνής. Στη συνέχεια καλούνται οι μέθοδοι *FindFather*, *AddFathersToQueue*, *CrossTree*, *CalcResult*, *SortResult* και *ReturnResult*, οι οποίες εκτελούν τα στάδια του αλγορίθμου αναζήτησης.

Πιο αναλυτικά η μέθοδος *FindFather* στόχο έχει τον εντοπισμό των rootObjects της σκηνής και της δημιουργίας των κατάλληλων δομών που θα διατηρούν πληροφορίες για την πρόοδο της αναζήτησης. Η κάθε δομή αφορά σε ένα αντικείμενο της σκηνής, όσα αντίγραφά του και αν παρουσιάζονται σε αυτήν, και αποτελεί ένα key/value pair, όπου στο key αποθηκεύεται το όνομα του Object που εξετάζεται και στο value ένα JSONObject. Τα key/value pairs, δηλαδή το σύνολο των δομών, όλων των Objects που συνθέτουν την σκηνή αποθηκεύονται σε ένα HashTable με όνομα *hm*. Σε αντιστοιχία με τη map function *Map1_FindFather* της τεχνικής *MapReduce*, η *FindFather* εξετάζει τα JSON Document της collection "*KeyValue*" της database "*ScenexNoCopies*". Ανάλογα με την φύση του Object που εξετάζεται από το JSON Document δημιουργούνται αντίστοιχα πεδία στο JSONObject που θα αποθηκευτεί στο HashTable *hm* στο κατάλληλο key/value pair. Όσον αφορά στα leafObjects, δημιουργείται ένα JSONObject με δύο key/value pairs, με keys το *norma* και το *color* με αντίστοιχα values το κανονικοποιημένο μέγεθος του αντικειμένου και το JSONObject με τις *red*, *green*, και *blue* συνιστώσες του χρώματος που έχει το Object. Το JSONObject αυτό αποθηκεύεται στο value του key/value pair του *hm* με key το όνομα του leafObject. Στην περίπτωση που εξετάζεται fatherObject, το JSONObject που θα αποθηκευτεί στον *hm* διαθέτει τρία key/value pairs με keys *count*, *is_child* και *child*. Τα rootObjects διατηρούν στο *count* το πλήθος εμφάνισης του αντικειμένου στη σκηνή, στο πεδίο *is_child*

την τιμή *0* και στο *child* έναν JSONArray που περιέχει όλα τα παιδιά του εξεταζόμενου Object. Ο JSONArray διατηρεί για το κάθε παιδί τα key/value pairs *name* και *cur_view* που έχουν ως τιμή αντίστοιχα το όνομα του childObject και το ποσοστό συμμετοχής του στη σκηνή. Οι τιμές *count* και *is_child* αλλάζουν όταν εξετάζεται fatherObject. Σε αυτήν την περίπτωση το *count* παίρνει την τιμή *0* και το *is_child* την τιμή *1*. Το *child* key/value pair του JSONObject που χαρακτηρίζει το fatherObject είναι όμοιο με αυτό που περιεγράφηκε για τα rootObjects. Αυτή η διάκριση στα key/value pairs *count* και *is_child* γίνεται για τον εντοπισμό των rootObjects από τα οποία θα ξεκινήσει η αναζήτηση. Στους Πίνακας 28, Πίνακας 29 και Πίνακας 30 δίνονται σχηματικά τα key/value pairs του *hm* αντίστοιχα όταν αφορούν σε rootObject, fatherObject και leafObject.

rootObject

```
"rootObject_name" : {
                      "count" : "Number_of_occurrences",
                      "is_child" : "0",
                      "child" : [
                            {"name" : "child_1_name",
                             "cur_view": "child_1_cur_view"},
                            …,
                            {"name" : "child_n_name",
                             "cur_view" : "child_n_cur_view"}
                            ]
                    }
```

Πίνακας 28 rootObject in *hm* HashTable

fatherObject

```
"fatherObject_name" : {
                      "count" : "0",
                      "is_child" : "1",
                      "child" : [
                            {"name" : "child_1_name",
                             "cur_view": "child_1_cur_view"},
                            …,
                            {"name" : "child_n_name",
                             "cur_view" : "child_n_cur_view"}
                            ]
                    }
```

Πίνακας 29 fatherObject in *hm* HashTable

leafObject

```
"leafObject_name" : {
                      "norma" : "normalized_form",
                      "color" : {
                                   "r" : "red_value",
                                   "g" : "green_value",
                                   "b" : "blue_value"
                                 }
                    }
```

Πίνακας 30 leafObject in *hm* HashTable

Ακολουθεί η κλήση της *AddFathersToQueue* η οποία θα εντοπίσει τα rootObjects και θα εκκινήσει την αναζήτηση. Για την αναζήτηση θα χρησιμοποιηθεί, όμοια με την μέθοδο

KeyValueQueue, μία ουρά, η οποία θα εκτελέσει έναν αλγόριθμο τύπου BFS. Όπως έχει αναφερθεί, Πίνακας 15, για την εκτέλεση του BFS εισάγονται στην ουρά οι root nodes, από τους οποίους ορίζονται τα δένδρα στα οποία θα εκτελεστεί η αναζήτηση. Στην περίπτωση της *NoMapReduce* τεχνικής οι root nodes αντιστοιχούν στα rootObjects που έχουν εντοπιστεί με βάση το key/value pair *is_child/0*. Για το κάθε ένα από αυτά εισάγονται στην ουρά, σύμφωνα με τον πίνακα *child* του πεδίου του *hm* που αφορά στο rootObject, όλα τα childObjects του. Συγκεκριμένα στην ουρά εισάγονται JSONObjects με διαφορετικά πεδία αν το childObject είναι leafObject ή όχι. Στην περίπτωση που το childObject είναι leafObject, εισάγεται στην αρχή της ουράς ένα JSONObject με πεδία *name* και *cur_view*. Το *cur_view* υπολογίζεται σύμφωνα με την Εξίσωση 6, όπου *count* το πλήθος εμφάνισης του rootObject και *child_cur_view* το *cur_view* όπως αυτό εμφανίζεται από το αντίστοιχο JSONObject του πίνακα *child* ου rootObject. Αντίστοιχα για childObjects που δεν είναι leafObjects εισάγεται στο τέλος της ουράς ένα JSONObject με πεδία *name, cur_view* και *count*, όπου *name* και *cur_view* αυτά ορίζονται από το αντίστοιχο JSONObject του πίνακα *child* και *count* το *count* του rootObject. Αυτή η ανάθεση του *count* γίνεται για λόγους εξοικονόμησης μνήμης, ώστε ίδια childObjects να διατηρούνται στην ουρά μία και όχι *count* φορές.

$$cur\_view = child\_cur\_view \cdot count$$

Εξίσωση 6 Calculate cur_view of leafObject

Η αναζήτηση εκκινείται με κλήση της *CrossTree* μεθόδου της class *NoMapReduce*. Με αυτήν εξετάζεται κάθε φορά το πρώτο αντικείμενο της ουράς. Αν το αντικείμενο είναι leafObject προστίθεται στο *cur_view* του αντίστοιχου key/value pair του *hm* το *cur_view* που ορίζεται από το αντικείμενο που εξάγεται από την ουρά. Αν το αντικείμενο της ουράς που εξετάζεται είναι fatherObject, προστίθενται τα παιδιά του στην ουρά ακριβώς με την ίδια διαδικασία που περιεγράφηκε για τα rootObjects. Προοδευτικά εξετάζονται όλα τα επίπεδα και όλοι οι κόμβοι των δένδρων που ορίζονται από τα rootObjects και προστίθενται στον *hm* στα key/value pairs που αφορούν σε leafObjects το συνολικό ποσοστό συμμετοχής του κάθε ενός στη σκηνή. Η αναδρομή ολοκληρώνεται όταν αδειάσει η ουρά. Τελικά στις δομές του *hm* που περιγράφουν τα leafObjects έχει προστεθεί ένα πεδίο που διατηρεί το ποσοστό εμφάνισης κάθε ενός στη σκηνή. Ένα τέτοιο παράδειγμα δίνεται στον Πίνακας 31.

leafObject

"leafObject_name" : {
"cur_view" : "Total_view_of_leafObject"
"norma" : "normalized_form",
"color" : {
"r" : "red_value",
"g" : "green_value",
"b" : "blue_value"
}
}

Πίνακας 31 Updated leafObject in *hm* HashTable

Αφού έχει ολοκληρωθεί η διάσχιση των δένδρων, έχουν εξεταστεί όλα τα Objects της σκηνής και έχουν υπολογιστεί τα ποσοστά συμμετοχής κάθε leafObject, καλείται η μέθοδος *CalcResult*. Με αυτήν και με τη χρήση δύο βοηθητικών HashTable *hmColor* και *hmSmall* ομαδοποιούνται όλα τα αντικείμενα του *hm* ανάλογα με το χρώμα τους και υπολογίζεται το συνολικό ποσοστό συμμετοχής κάθε χρώματος. Συγκεκριμένα για κάθε leafObject του *hm* αποθηκεύεται στον *hmColor* στο key/value pair, με key το χρώμα του αντικειμένου, ένα JSONObject με πεδία *whole*, όπου διατηρείται το συνολικό ποσοστό συμμετοχής του

εξεταζόμενου χρώματος και *names* όπου διατηρούνται σε πίνακα τα ονόματα των αντικειμένων που συνθέτουν αυτό το χρώμα. Στον *hmSmall* αποθηκεύεται στο key/value pair, με key το όνομα του χρώματος του αντικειμένου, το συνολικό ποσοστό συμμετοχής του.

Ακολουθεί η κλήση της *SortResult* η οποία ταξινομεί τον *hmSmall* σύμφωνα με το value των key/value pairs. Αυτός είναι και ο λόγος της διπλής αποθήκευσης που πραγματοποιήθηκε κατά την εκτέλεση της *CalcResult*.

Τέλος με κλήση της *ReturnResult* επιστρέφονται το χρώμα ή τα χρώματα που καταλαμβάνουν το μεγαλύτερο ποσοστό συμμετοχής στη σκηνή, από τον ταξινομημένο HashTable που προκύπτει από την *SortResult*, καθώς και τα ονόματα των αντικειμένων που τα συνθέτουν. Τα ονόματα αυτά λαμβάνονται από το HashTable *hmColor* από το key/value pair που στο key έχει το κυρίαρχο χρώμα της σκηνής. Τα αποτελέσματα γράφονται στο *Αρχείο Αποτελεσμάτων* και ο χρόνος που χρειάστηκε για την ολοκλήρωση της αναζήτησης καταγράφεται στο *Αρχείο Χρόνου Απόκρισης*. Τελικά προκύπτουν 11 χρόνοι για κάθε εκτέλεση της αναζήτησης, ένας για κάθε διαφορετικό πλήθος αντιγράφων των metadata που διατηρούσαμε, 1, 2, 4, 8, 16, 32, 64, 128, 256, 512 και 1024.

Κεφάλαιο 7
Συγκριτική Μελέτη

Στο κεφάλαιο αυτό γίνεται η ανάλυση και αξιολόγηση των αποτελεσμάτων των διαφορετικών τεχνικών που υλοποιήθηκαν. Θυμίζεται πως ανάλογα με τον τύπο και τη μορφή των queries που υποβάλλονται, διακρίνονται δύο κατηγορίες αναζήτησης, η *Search* και η *MapReduce*.

Η πρώτη κατηγορία αναζήτησης που υλοποιήθηκε ονομάστηκε *Search*. Τα queries που εκτελούνται με χρήση της *Search* έχουν την δομή των Queries του Πίνακας 8. Για την μελέτη της χρησιμοποιήθηκαν πέντε διαφορετικές τεχνικές, οι οποίες εκτέλεσαν το δοσμένο query στα πρότυπα δεδομένα όταν αυτά βρίσκονταν σε 1, 2, 4, 8, 16 και 32 αντίγραφα. Για την αξιολόγηση των μεθόδων χρησιμοποιήθηκε ο χρόνος εκτέλεσης του query, όπως θα εξηγηθεί στη συνέχεια. Στον τομέα μελέτης *Search* περιλαμβάνονται οι παρακάτω τεχνικές:

- *Legacy*
- *PathXML*
- *EmbeddedXML*
- *KeyValue*, η οποία χωρίζεται στις:
    - *KeyValueQueue*
    - *KeyValuePath*

Στη συνέχεια εξετάστηκε ο τομέας μελέτης *MapReduce*, με υποβολή του aggregation query "*Return dominant color of scene*". Οι τεχνικές που υλοποιήθηκαν για αυτόν τον τύπο αναζήτησης εκτέλεσαν το query στα πρότυπα δεδομένα, ίδιας μορφής με αυτά της κατηγορίας *Search*, όταν αυτά βρίσκονταν σε 1, 2, 4, 8, 16, 32, 64, 128, 256, 512 και 1024 αντίγραφα. Η κατηγορία αναζήτησης *MapReduce* περιλαμβάνει τις τεχνικές που αναλύθηκαν στο προηγούμενο κεφάλαιο:

- *MapReduce*
- *NoMapReduce*

Για τις ανάγκες τις διπλωματικής χρησιμοποιήθηκε ένα VM της υπηρεσίας *Cyclades* του συστήματος ~*okeanos*. Το μηχάνημα είχε λειτουργικό σύστημα Windows και διέθετε 4 CPUs, 4GB Ram και 60GB σκληρό δίσκο. Για την ανάπτυξη των τεχνικών που υλοποιήθηκαν χρησιμοποιήθηκε η 2.4 release της MongoDB και το JDK 7u25.

7.1 Αξιολόγηση Τομέα Μελέτης *Search*

Όπως αναφέρθηκε η αναζήτηση τύπου *Search* εκτελέστηκε με χρήση όλων των τεχνικών που υπάγονται σε αυτήν και στα δεδομένα όταν αυτά διατηρούνται σε διαφορετικό πλήθος αντιγράφων. Όσον αφορά στις μεθόδους αναζήτησης διακρίνονται δύο υποκατηγορίες. Η πρώτη αφορά στις τεχνικές που χρησιμοποιούν κάποιου είδους XML Parsing και είναι οι:

- *Legacy*
- *PathXML*
- *EmbeddedXML*

Η δεύτερη περιλαμβάνει εκείνες που δεν χρησιμοποιούν XML Parsing, αλλά ενσωματωμένα εργαλεία της Java και της MongoDB. Αυτές οι τεχνικές είναι οι:

- *KeyValueQueue*
- *KeyValuePath*

Για την αξιολόγηση της απόδοσης της κάθε τεχνικής χρησιμοποιήθηκε ο χρόνος που απαιτήθηκε για την εκτέλεση του query για όλα τα αντίγραφα δεδομένων. Συνεπώς για κάθε μέθοδο αναζήτησης προκύπτουν έξι χρόνοι απόκρισης του query, ένας για κάθε διαφορετικό αριθμό αντιγράφων των πρότυπων δεδομένων, 1, 2, 4, 8, 16 και 32.

Σε αυτήν την ενότητα θα παρουσιαστούν τα αποτελέσματα που προέκυψαν από την εκτέλεση ενός από τα Queries του Πίνακας 8, του Q3, τα χαρακτηριστικά του οποίου παρουσιάζονται στον Πίνακας 32. Δίνονται επίσης γραφικές παραστάσεις για σχηματική αναπαράσταση των αποτελεσμάτων, καθώς και αλγοριθμική επεξήγησή τους. Οι πίνακες αποτελεσμάτων και οι γραφικές παραστάσεις των υπόλοιπων Queries που υλοποιήθηκαν βρίσκονται στο Παράρτημα Α.

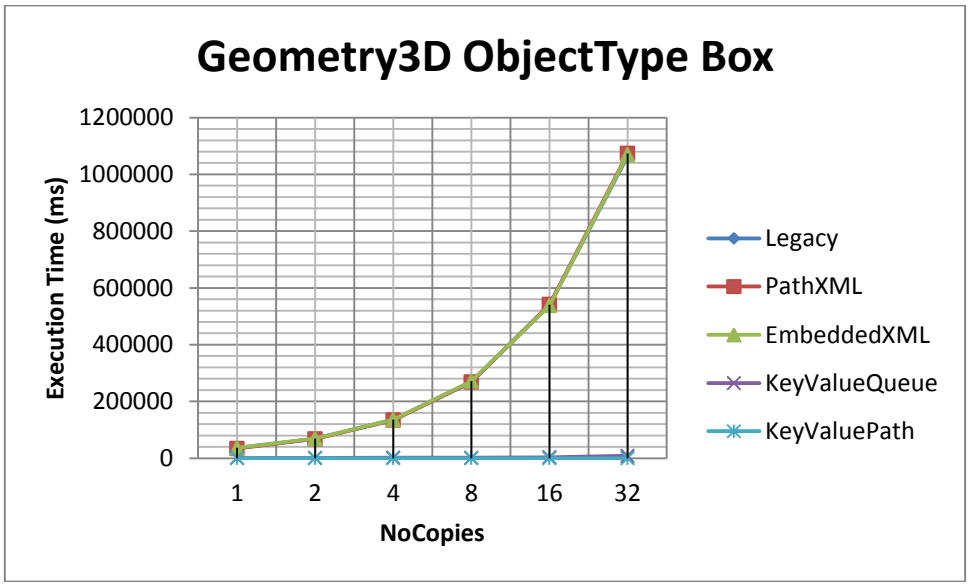
Query	elementName	attributeName	attributeValue
Q3	Geometry3D	ObjectType	Box

Πίνακας 32 Query 3

Το Q3 εκτελέστηκε με χρήση των τεχνικών *Legacy*, *PathXML*¸ *EmbeddedXML*, *KeyValueQueue* και *KeyValuePath*, στα δεδομένα όταν αυτά εμφανίζονταν σε 1, 2, 4, 8, 16 και 32 αντίγραφα. Για τις αναζητήσεις της υποκατηγορίας των τεχνικών που χρησιμοποιούν XML Parsing υπολογίστηκε ο χρόνος απόκρισής τους σε msec συμπεριλαμβανομένου του χρόνου αποθήκευσης των απαντήσεων των queries στο δίσκο. Αντίθετα στις τεχνικές της δεύτερης υποκατηγορίας, που δεν χρησιμοποιείται XML Parsing, ο χρόνος απόκρισής τους υπολογίστηκε μόνο για την εκτέλεση των queries και όχι για τον χρόνο εκτέλεσης των I/O στο δίσκο. Αυτή η διάκριση έγινε γιατί, όπως είναι γνωστό, το XML Parsing αποτελεί διαδικασία με μεγάλο overhead, με την αποθήκευση των δεδομένων στο δίσκο να αποτελεί αμελητέο ποσοστό του συνολικού χρόνου εκτέλεσης της εξεταζόμενης μεθόδου. Στις τεχνικές όμως που δεν χρησιμοποιούν XML Parsing, το overhead προκαλείται από τα I/O, δηλαδή η αποθήκευση των αποτελεσμάτων στο δίσκο αποτελεί πολύ μεγαλύτερο ποσοστό του συνολικού χρόνου εκτέλεσης της μεθόδου, από ότι η ίδια η αναζήτηση. Είναι φανερό πως αν στις μετρήσεις των τεχνικών που δεν χρησιμοποιούν XML Parsing συμπεριλαμβανόταν η αποθήκευση των δεδομένων στο δίσκο, δεν θα ήταν έγκυρη η σύγκριση του συνόλου των τεχνικών που μελετήθηκαν. Ακολουθούν όλες οι μετρήσεις που αφορούν στο Q3 ομαδοποιημένες σύμφωνα με την τεχνική που χρησιμοποιήθηκε, Πίνακας 33, καθώς και η αντίστοιχη γραφική παράσταση, Εικόνα 30.

Technique / NoCopies	Legacy	PathXML	EmbeddedXML	KeyValueQueue	KeyValuePath
1	35301	34305	36132	1139	22
2	68011	68056	69181	1054	22
4	136157	134033	136215	1753	13
8	268083	268549	271418	1645	21
16	539594	541018	537798	2946	20
32	1064142	1073735	1069459	8489	28

Πίνακας 33 Query3 Execution Time Results



Εικόνα 30 Query3 All Techniques

Στην γραφική παράσταση της Εικόνα 30 παρουσιάζονται συνολικά οι χρόνοι απόκρισης του Q3 με χρήση όλων των τεχνικών που ανήκουν στην κατηγορία αναζήτησης *Search*. Ένα πρώτο συμπέρασμα, στο οποίο καταλήγουμε με βάση αυτήν την γραφική παράσταση, είναι πως οι τεχνικές που χρησιμοποιούν XML Parsing έχουν χρόνους απόκρισης τάξεις μεγέθους μεγαλύτερους από τους αντίστοιχους των τεχνικών που δεν χρησιμοποιούν XML Parsing. Για την καλύτερη αξιολόγηση των τεχνικών ακολουθούν η Εικόνα 31 και η Εικόνα 32, οι οποίες αναπαριστούν αντίστοιχα τα αποτελέσματα των τεχνικών σύμφωνα με την υποκατηγορία στην οποία ανήκουν, εκείνων που χρησιμοποιούν XML Parsing και εκείνων που δεν το χρησιμοποιούν. Παράλληλα αναλύονται αλγοριθμικά οι μέθοδοι που χρησιμοποιήθηκαν για καλύτερη κατανόηση των αποτελεσμάτων που προκύπτουν.

Ανάλυση *Legacy*

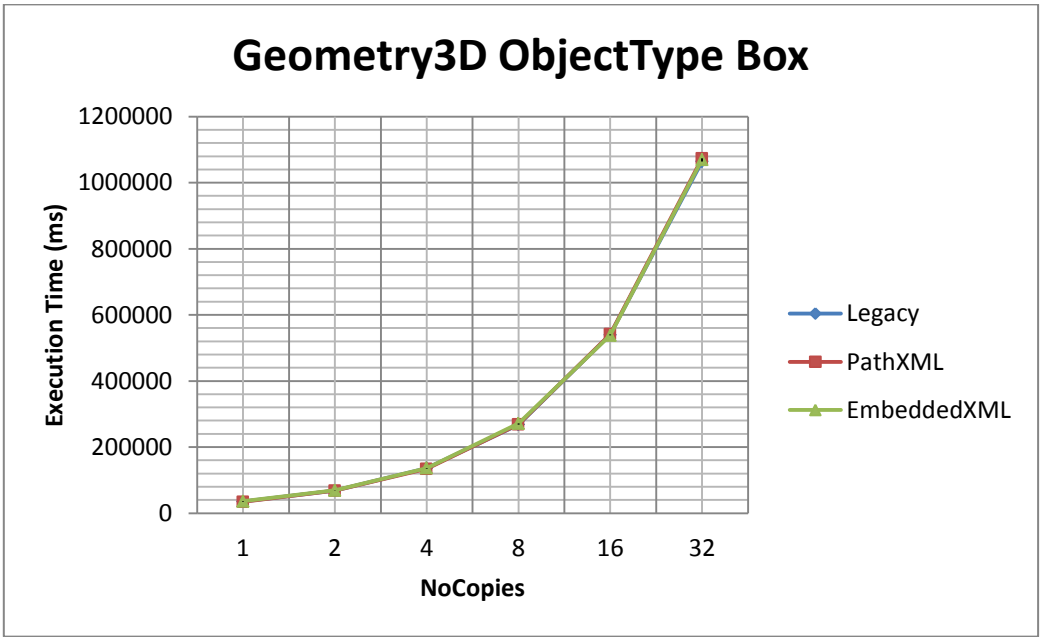
Η τεχνική *Legacy* χρησιμοποιεί τον κλασικό τρόπο διάσχισης XML αρχείων, κατά τον οποίο το XML αρχείο μετατρέπεται στο αντίστοιχο JDOM tree, επιλέγονται όλοι οι κόμβοι του και εντοπίζονται, αν υπάρχουν, εκείνοι με *elementName*, *attributeName* και *attributeValue* σύμφωνα με το δοσμένο query. Καθώς χρησιμοποιείται XML Parsing αναμένουμε οι χρόνοι απόκρισης αυτής της μεθόδου να είναι μεγάλοι.

Ανάλυση *PathXML*

Η τεχνική *PathXML* χρησιμοποιεί τα JSON Documents της MongoDB για τον εντοπισμό του *directoryPath* των XML αρχείων που περιέχουν τα metadata. Στη συνέχεια χρησιμοποιείται ο ίδιο XML Parser για τη διάσχισή τους και εντοπισμό εκείνων των 3D αρχείων που απαντούν στο δοσμένο query. Καθώς χρησιμοποιείται ο ίδιος XML Parser, στα ίδια XML αρχεία με αυτά της τεχνικής *Legacy*, αναμένεται ο χρόνος απόκρισης της αναζήτησης με την μέθοδο *PathXML* να είναι πολύ κοντά σε αυτόν της τεχνικής *Legacy*.

Ανάλυση *EmbeddedXML*

Τέλος η τεχνική *EmbeddedXML* χρησιμοποιεί μόνο τα JSON Documents της MongoDB, στα οποία είναι αποθηκευμένο ολόκληρο το XML αρχείο των metadata ως String στο αντίστοιχο key/value pair όταν αφορούν σε X3D ή MPEG7 περιγραφή. Για την εκτέλεση της αναζήτησης το XML String μετατρέπεται στο αντίστοιχο JDOM tree, το οποίο διασχίζεται όπως στην περίπτωση των δύο προηγούμενων τεχνικών προς εντοπισμό των κόμβων με τα ζητούμενα *elementName*, *attributeName* και *attributeValue*. Είναι φανερό πως και σε αυτήν την μέθοδο αναζήτησης αναμένεται οι χρόνοι απόκρισης των Queries να είναι πολύ κοντά σε αυτούς της τεχνικής *Legacy*.



Εικόνα 31 Query3 XML Techniques

Η γραφική παράσταση της Εικόνα 31 επιβεβαιώνει το παραπάνω συμπέρασμα. Παρατίθενται ακόμα ο Πίνακας 34 και Πίνακας 35, οι οποίοι παρουσιάζουν, αντίστοιχα για τις τεχνικές *PathXML* και *EmbeddedXML*, το συνολικό χρόνο απόκρισης του Q3, τον χρόνο που απαιτήθηκε μόνο για την εκτέλεση του XML Parsing και το ποσοστό αυτού του χρόνου επί του συνολικού. Όμοιοι πίνακες για όλα τα Queries που εξετάστηκαν παρουσιάζονται στο Παράρτημα Α. Ο πίνακας επίσης επιβεβαιώνει την θεώρηση πως όταν χρησιμοποιείται XML Parsing, ο χρόνος αποθήκευσης των αποτελεσμάτων του Q3 στο δίσκο αποτελεί αμελητέο ποσοστό του συνολικού χρόνου εκτέλεσης. Συγκεκριμένα φαίνεται πως η διάσχιση των XML αρχείων καταλαμβάνει σε όλες τις περιπτώσεις, δηλαδή για όλα τα αντίγραφα των

δεδομένων, και στις δύο τεχνικές πάνω από το 99% του συνολικού χρόνου εκτέλεσης της αναζήτησης.

Time Calculated / NoCopies	Total	XML Parsing	Percent
1	34305	34157	99,57%
2	68056	67797	99,62%
4	134033	133830	99,85%
8	268549	268244	99,89%
16	541018	540786	99,96%
32	1073735	1073290	99,96%

Πίνακας 34 Query3 Execution and XML Parsing Time Results PathXML Technique

Time Calculated / NoCopies	Total	XML Parsing	Percent
1	36132	35876	99,29%
2	69181	69045	99,80%
4	136215	135749	99,66%
8	271418	270951	99,83%
16	537798	537214	99,89%
32	1069459	1068561	99,92%

Πίνακας 35 Query3 Execution and XML Parsing Time Results Embedded XML Technique

Ανάλυση *KeyValueQueue*

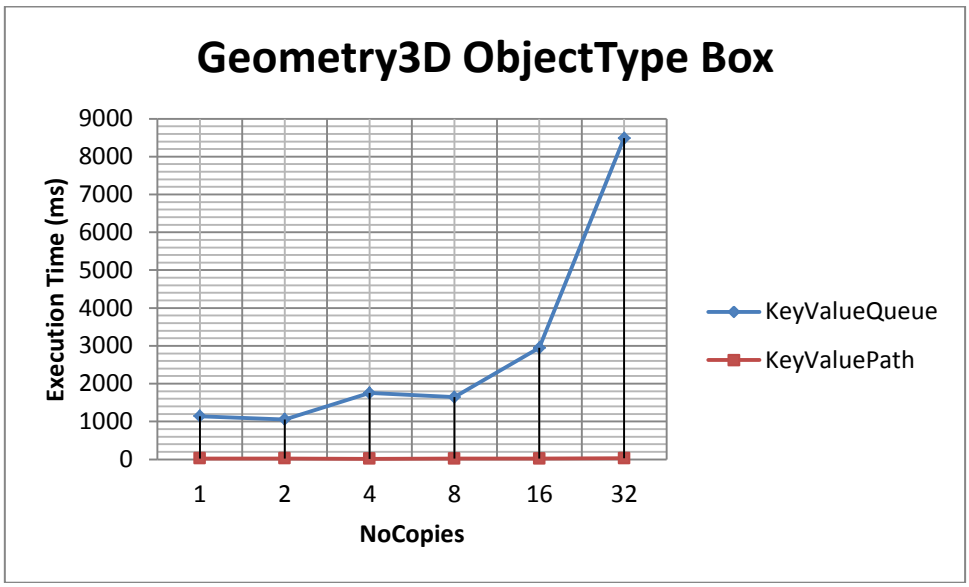
Η τεχνική *KeyValueQueue* χρησιμοποιεί τα JSON Documents της MongoDB τα οποία περιέχουν τα metadata των 3D αρχείων εξ' ολοκλήρου μετασχηματισμένα σε key/value pairs σύμφωνα με την μεθοδολογία που παρουσιάστηκε στο προηγούμενο κεφάλαιο. Τα JSON Documents περιέχουν Embedded JSON Objects, δομή που μπορεί να παραλληλιστεί με δένδρο. Για το λόγο αυτό για την αναζήτησης των ζητούμενων key/value pairs εφαρμόστηκε ο αλγόριθμος αναζήτησης BFS με χρήση μιας ουράς στην οποία αποθηκεύονται μέχρι να εξεταστούν όλα τα JSON Objects ενός επιπέδου του δένδρου. Καθώς η ουρά αποτελεί εσωτερική δομή της Java που αποθηκεύει τα δεδομένα της στη μνήμη, αναμένουμε η εκτέλεση της *KeyValueQueue* να είναι γρηγορότερη από την εκτέλεση τεχνικών που χρησιμοποιούν το XML Parsing, καθώς, όπως είναι γνωστό, αυτό χαρακτηρίζεται από υψηλό overhead.

Ανάλυση *KeyValuePath*

Με την *KeyValuePath* τεχνική η αναζήτηση πραγματοποιείται στα JSON Documents που περιέχουν στη μορφή key/value pairs τις περιγραφές των 3D αρχείων. Σε αντίθεση με την *KeyValueQueue* τεχνική το *keyPath* στο οποίο βρίσκονται, αν υπάρχουν, οι ζητούμενες πληροφορίες είναι εκ των προτέρων γνωστό. Για το λόγο αυτό χρησιμοποιείται απευθείας ένα MongoDB Query το οποίο επιστρέφει, με χρήση των ενσωματωμένων εργαλείων της MongoDB, τα JSON Documents που διαθέτουν τα ζητούμενα *elementName*, *attirbuteName* και *attributeValue*. Η τεχνική αυτή χρησιμοποιεί τους βελτιστοποιημένους μηχανισμούς της MongoDB που έχουν αναπτυχθεί ειδικά για την γρήγορη εκτέλεση queries σε JSON

Documents. Για το λόγο αυτό περιμένουμε η εκτέλεση της *KeyValuePath* να έχει αρκετά μικρότερο χρόνο απόκρισης σε σύγκριση με την *KeyValueQueue*.

Η Εικόνα 32 επιβεβαιώνει τις παραπάνω προβλέψεις, καθώς και την επιλογή του να μην συμπεριληφθεί στον χρόνο εκτέλεσης της αναζήτησης και ο χρόνος αποθήκευσης των αποτελεσμάτων στο δίσκο.



Εικόνα 32 Query3 KeyValue Techniques

7.2 Αξιολόγηση Τομέα Μελέτης *MapReduce*

Στο δεύτερο σκέλος αυτής της διπλωματικής υλοποιήθηκε η κατηγορία αναζήτησης *MapReduce*, στην οποία ανήκουν οι τεχνικές *MapReduce* και *NoMapReduce*. Το query που χρησιμοποιήθηκε είναι "*Return dominant color of scene*" και υποβλήθηκε στα metadata μιας σκηνής, όταν αυτά διατηρούνταν σε 1, 2, 4, 8, 16, 32, 64, 128, 256, 512 και 1024 αντίγραφα. Αντίστοιχα με την προηγούμενη κατηγορία αναζήτησης στην ενότητα αυτή θα παρουσιαστούν τα αποτελέσματα για μία σκηνή, την "*test case 3"*. Τα αποτελέσματα των υπόλοιπων σκηνών που εξετάστηκαν βρίσκονται στο Παράρτημα Β.

Για την αξιολόγηση και σύγκριση των δύο τεχνικών χρησιμοποιήθηκε ο χρόνος απόκρισης των queries για το ίδιο ερώτημα όταν αυτό εκτελείται σε όλα τα αντίγραφα των δεδομένων. Ακολουθεί περιγραφή των αναμενόμενων αποτελεσμάτων και αλγοριθμική επεξήγησή τους. Οι χρόνοι απόκρισης των αναζητήσεων παρουσιάζονται στον Πίνακας 36 σε msec και είναι ομαδοποιημένα με βάση την τεχνική που χρησιμοποιήθηκε, ενώ στην Εικόνα 33 παρουσιάζονται με γραφικό τρόπο οι χρόνοι απόκρισης τους query και για τις δύο μεθόδους που υλοποιήθηκαν.

Ανάλυση *MapReduce*

Το προγραμματιστικό μοντέλο MapReduce χρησιμοποιείται για τον καταμερισμό του φόρτου εργασίας των εφαρμογών σε πολλούς υπολογιστές, με τις μεγάλες δυνατότητές του να φαίνονται βέλτιστα όταν επεξεργάζεται TeraBytes δεδομένων. Για το λόγο αυτό στα λίγα δεδομένα που χρησιμοποιούνται από την διπλωματική αυτή δεν αναμένεται να φανούν οι

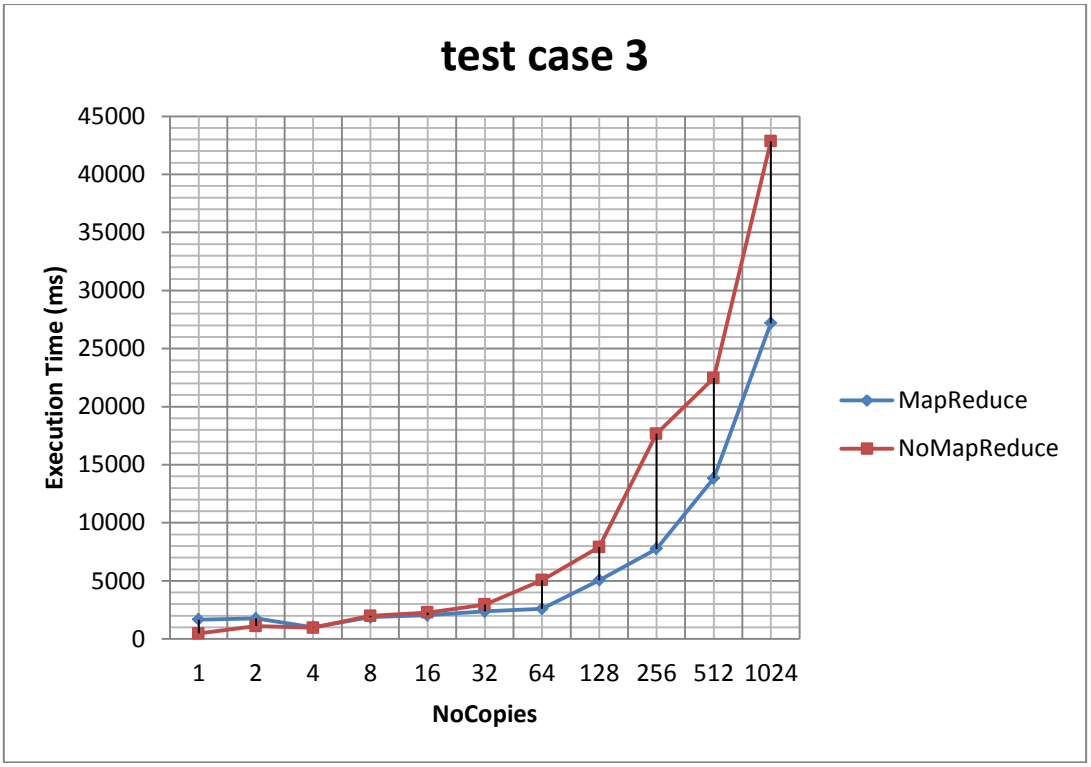
δυνατότητες υψηλής κλιμάκωσης του μοντέλου. Ταυτόχρονα καθώς χρησιμοποιείται μόνο ένα μηχάνημα για την επεξεργασία των δεδομένων, δεν περιμένουμε να δούμε ούτε τον καταμερισμό του φόρτου εργασίας που προσφέρεται εσωτερικά από το MapReduce. Λόγω του ότι στην τεχνική *MapReduce* χρησιμοποιούνται τα εργαλεία της MongoDB αναμένουμε γρηγορότερη εκτέλεση των queries συγκριτικά με μεθόδους που δεν τα χρησιμοποιούν, καθώς οι μηχανισμοί που προσφέρονται έχουν βελτιστοποιηθεί ειδικά για την παροχή υπηρεσιών MapReduce. Λόγω των εσωτερικών μηχανισμών της MongoDB αναμένεται ένα overhead κατά την εκτέλεση του MapReduce, που αφορά στην διασύνδεση των δεδομένων, την επικοινωνία των κόμβων του cluster και το aggregation των αποτελεσμάτων. Το overhead αυτό ωστόσο είναι αμελητέο όσο αυξάνεται ο όγκος των δεδομένων. Το πλεονέκτημα της μεθόδου *MapReduce* είναι η πολύ εύκολη τμηματοποίηση του προβλήματος και ο καταμερισμός των εργασιών με απλή προσθήκη μηχανημάτων στο cluster της MongoDB. Έτσι ανεξάρτητα από το πόσο αυξηθεί ο συνολικός όγκος των δεδομένων προς επεξεργασία, όσο προστίθενται κόμβοι, η τεχνική *MapReduce* θα εκτελείται και μάλιστα με αντιστρόφως ανάλογους χρόνους απόκρισης.

Ανάλυση *NoMapReduce*

Η αξιολόγηση της *MapReduce* τεχνικής έχει αξία όταν συγκρίνεται με την τεχνική *NoMapReduce*. Στην *NoMapReduce* χρησιμοποιούνται εργαλεία που έχουν αναπτυχθεί από την διπλωματική αυτή και απαντούν στο δοσμένο query στα ίδια δεδομένα χωρίς χρήση των MapReduce μηχανισμών της MongoDB. Καθώς χρησιμοποιείται ουρά, η οποία θυμίζεται αποθηκεύει τα δεδομένα της στη μνήμη, περιμένουμε ο χρόνος εκτέλεσης του query διαρκώς να αυξάνεται. Το μειονέκτημα αυτής της μεθόδου είναι ότι η παραλληλοποίηση δεν είναι εύκολη, με αποτέλεσμα και ο καταμερισμός του προβλήματος να μην γίνεται βέλτιστα και να απαιτούνται εξειδικευμένες γνώσεις για την υλοποίηση ενός τέτοιου μηχανισμού. Παράλληλα πρέπει να λαμβάνεται υπόψιν η επεξεργαστική ικανότητα της ουράς, δηλαδή η διαθέσιμη μνήμη της. Όσο ο όγκος των δεδομένων καταλαμβάνει μικρότερο αποθηκευτικό χώρο από όσο μπορεί να διαχειριστεί η ουρά, η τεχνική *NoMapReduce* θα επιστρέφει αποτέλεσμα. Όταν όμως ο όγκος των δεδομένων ξεπεράσει την διαθέσιμη μνήμη της ουράς η τεχνική δεν θα μπορεί να εκτελεστεί.

Technique / NoCopies	MapReduce	NoMapReduce
1	1668	470
2	1763	1104
4	1003	970
8	1880	1992
16	2041	2274
32	2367	2961
64	2586	5055
128	5064	7913
256	7758	17668
512	13840	22466
1024	27187	42851

Πίνακας 36 test case 3 MapReduce Query Execution Results



Εικόνα 33 test case 3 MapReduce Query Techniques

Από την Εικόνα 33 φαίνεται το αρχικό overhead της *MapReduce* τεχνικής που υλοποιήθηκε, το οποίο όσο αυξάνεται ο όγκος των δεδομένων γίνεται αμελητέο. Καθώς χρησιμοποιήθηκε μόνο ένα μηχάνημα για την εκτέλεση της αναζήτησης, ήταν αναμενόμενη η διαρκής αύξηση του χρόνου απόκρισης του query. Αν είχε προστεθεί ένα ακόμα μηχάνημα στο cluster της MongoDB οι χρόνοι θα ήταν μειωμένοι στο μισό. Αντίθετα καθώς η τεχνική *NoMapReduce* που υλοποιήθηκε δεν περιλαμβάνει μηχανισμό παραλληλοποίησής της, όταν ο όγκος των δεδομένων γινόταν μεγαλύτερος από την διαθέσιμη μνήμη της ουράς η τεχνική δεν θα μπορούσε να εκτελεστεί.

Κεφάλαιο 8
Συμπεράσματα

Στο προηγούμενο κεφάλαιο έγινε η παρουσίαση των αποτελεσμάτων που αφορούσαν στους χρόνους απόκρισης όλων των τεχνικών που αναπτύχθηκαν από τη διπλωματική αυτή και δόθηκε η αλγοριθμική επεξήγησή τους. Σε αυτό το κεφάλαιο καταγράφονται τα συμπεράσματα από την αξιολόγηση των μεθόδων και η συνεισφορά της παρούσας διπλωματικής εργασίας. Ακολουθούν δύο ενότητες για την παρουσίαση των συμπερασμάτων για τις τεχνικές που ανήκουν στους δύο τομείς μελέτης που εξετάστηκαν και μία ακόμα όπου αναλύονται συγκεντρωτικά τα συμπεράσματα στο οποία καταλήξαμε μετά την αξιολόγησή τους.

8.1 Συμπεράσματα Τομέα Μελέτης *Search*

Ο τομέας μελέτης *Search* αφορά σε εφαρμογές που επεξεργάζονται ερωτήματα με ορισμένα χαρακτηριστικά, *elementName, attributeName* και *attributeValue*, και καλούνται να επιστρέψουν τα ονόματα των multimedia αρχείων που τα ικανοποιούν. Υλοποιήθηκαν πέντε τεχνικές ανάλογα με τον διαθέσιμο εξοπλισμό και την εξειδίκευση του προσωπικού που διαχειρίζεται τέτοιες εφαρμογές αναζήτησης. Κάθε μία αφορά σε διαφορετικές ανάγκες και η υιοθέτησή της εξαρτάται από τις δυνατότητες μετάβασης από παλαιότερα μοντέλα στα νέα που αναπτύχθηκαν από την διπλωματική αυτή. Στον Πίνακας 37 παρουσιάζονται συγκεντρωτικά οι προτεινόμενοι τρόποι εκτέλεσης της αναζήτησης ανάλογα με τον τύπο του μεταβατικού σχεδίου.

Συγκεκριμένα, και σύμφωνα με την Εικόνα 31, όσον αφορά σε εφαρμογές που χρησιμοποιούν XML Parsing η εφαρμογή των τεχνικών *PathXML* και *EmbeddedXML* δεν φαίνεται να προσφέρει κανένα πλεονέκτημα συγκριτικά με την τεχνική *Legacy*. Ο λόγος είναι ο μεγάλος χρόνος που απαιτείται για την διάσχιση των XML αρχείων, με αποτέλεσμα η γρήγορη απόκριση της MongoDB να μην προσφέρει κανένα συγκριτικό πλεονέκτημα έναντι της απ' ευθείας αναζήτησης των ζητούμενων πληροφοριών στα XML αρχεία που είναι αποθηκευμένα σε directories. Συνεπώς για εφαρμογές που δεν επιθυμείται η μετάβαση σε νεότερες τεχνολογίες προτείνεται η διατήρηση των υπάρχοντων συστημάτων αναζήτησης πληροφοριών.

Από την Εικόνα 30 είναι φανερό το συντριπτικό πλεονέκτημα της εγκατάληψης της αναζήτησης πληροφοριών σε XML αρχεία και της μετάβασης σε μοντέλα αναζήτησης που χρησιμοποιούν JSON Documents, τα οποία διατηρούν το περιεχόμενο των XML αρχείων σε key/value pairs. Η μετάβαση αυτή απαιτεί τόσο την κατανόηση των νεότερων τεχνολογιών cloud computing και των NoSQL βάσεων δεδομένων όσο και την υψηλή εξοικείωση με τα ιδιαίτερα χαρακτηριστικά των εναλλακτικών που προσφέρουν, όπως τα διαφορετικά είδη NoSQL βάσεων δεδομένων, προκειμένου να γίνει εκμετάλλευση όλων των πλεονεκτημάτων τους. Για εφαρμογές που έχουν την δυνατότητα της ριζοσπαστικής αλλαγής στον τρόπο εκτέλεσης της αναζήτησης, προτείνεται η χρήση της τεχνικής *KeyValuePath* καθώς από την Εικόνα 32 είναι φανερό το πλεονέκτημά της έναντι της μεθόδου *KeyValueQueue*. Ωστόσο,

καθώς η τεχνική *KeyValuePath* απαιτεί εκ των προτέρων γνώση του *keyPath* στο οποίο βρίσκονται οι ζητούμενες πληροφορίες, στην περίπτωση που τέτοιες πληροφορίες δεν είναι γνωστές προτείνεται σαν δεύτερη καλύτερη εναλλακτική η τεχνική *KeyValueQueue*.

8.2 Συμπεράσματα Τομέα Μελέτης *MapReduce*

Εξετάστηκε επίσης η εφαρμογή του προγραμματιστικού μοντέλου MapReduce. Στόχος ήταν η παροχή της δυνατότητας υποβολής όλο και πιο πολύπλοκων queries τα οποία απαιτούν aggregation στα δεδομένα που εξετάζουν. Λόγω του μεγάλου overhead του XML Parsing η εφαρμογή MapReduce σε XML αρχεία θα ήταν απαγορευτική. Για το λόγο αυτό ο τομέας μελέτης *MapReduce* αφορούσε μόνο σε δεδομένα που διατηρούνται σε JSON αρχεία δομημένα σε key/value pairs. Οι χρήστες που επιθυμούν να το αναπτύξουν απαιτείται να έχουν εξειδικευμένες γνώσεις για τον τρόπο κλήσης του MapReduce μέσω της MongoDB.

Το πλεονέκτημα της τεχνικής *MapReduce* είναι φανερό από την Εικόνα 33, σε συνδυασμό με τον εύκολο κατακερματισμό του φόρτου εργασίας και της απαιτούμενης επεξεργαστικής ισχύς με απλή προσθήκη μηχανημάτων στο cluster της MongoDB. Η τεχνική *NoMapReduce* προτείνεται μόνο για πολύ μικρό όγκο δεδομένων, καθώς υπάρχει ο κίνδυνος της μη-εκτέλεσης της αναζήτησης όταν τα δεδομένα δεν χωράνε στην διαθέσιμη μνήμη της ουράς που χρησιμοποιείται.

8.3 Συγκεντρωτικά Συμπεράσματα

Στον Πίνακας 37 παρουσιάζονται συγκεντρωτικά τα συμπεράσματα στα οποία καταλήξαμε, σύμφωνα με την συγκριτική μελέτη των τεχνικών που υλοποιήθηκαν, και αφορούν σε μια πληθώρα απαιτήσεων, requirements, που χαρακτηρίζουν τις εφαρμογές.

Απαιτήσεις / Τεχνικές	**Μέγιστη Συμβατό-τητα**	**Εξειδι-κευμένη Χρήση**	**Εξειδι-κευμένο Λογισμικό**	**Ικανότητα Aggregation**	**Ασφάλεια**	**Απαιτούμενη Κατάρτιση**	**Επιδόσεις**
Legacy	-	Low	-	Low	Low	Low	Low
PathXML	High	Low	Medium	Low	Low	Medium	Low
EmbeddedXML	High	Low	Medium	Low	High	Medium	Low
KeyValueQueue	Medium	Low	Medium	Low	High	High	High
KeyValuePath	Medium	Medium	Medium	Low	High	High	High
MapReduce	Low	High	High	High	High	High	High
NoMapReduce	Low	High	Medium	High	High	High	Medium

Πίνακας 37 Πίνακας Συγκριτικής Μελέτης

Με τον όρο "*Μέγιστη Συμβατότητα*" εννοούμε την ευκολία μετάβασης από την τεχνική *Legacy* σε κάθε μία από τις νέες τεχνικές που προτείνονται. Η "*Εξειδικευμένη Χρήση*" δείχνει κατά πόσον γενική, generic, είναι η εφαρμογή της κάθε μεθόδου σε κάθε είδους αναζητήσεις. Οι τεχνικές *Legacy*, *PathXML*, *EmbeddedXML* και *KeyValueQueue* χαρακτηρίζονται από "*low*" "*Εξειδικευμένη Χρήση*", καθώς μόνο με την εισαγωγή των ζητούμενων *elementName*, *attirbuteName* και *attributeValue* είναι ικανές να απαντήσουν στο δοσμένο query. Αντίθετα η τεχνική *KeyValuePath* απαιτεί τον ακριβή προσδιορισμό του *keyPath* εντός του JSON Document, στο οποίο βρίσκονται οι ζητούμενες πληροφορίες, προκειμένου να είναι ικανή να

απαντήσει με ακρίβεια στο ερώτημα, συνεπώς χαρακτηρίστηκε ως "*medium*". Για τις τεχνικές *MapReduce* και *NoMapReduce* νομοτελειακά απαιτείται η υλοποίηση εξειδικευμένων τεχνικών προκειμένου να απαντήσουν στα aggregated queries που υποβάλλουν οι χρήστες και για το λόγο αυτό χαρακτηρίζονται από "*high*" "*Εξειδικειμένη Χρήση*". Η στήλη "*Εξειδικευμένο Λογισμικό*" δείχνει τις απαιτήσεις εγκατάστασης και συντήρησης επιπλέον λογισμικού συγκριτικά με την *Legacy* τεχνική προκειμένου να γίνει μετάβαση στις νέες τεχνικές αναζήτησης. Με το "*Δυνατότητα Aggregation*" φαίνεται η ικανότητα της κάθε τεχνικής στην εκτέλεση queries που απαιτούν τον συναδυασμό πληροφορίας για την παραγωγή αποτελεσμάτων. Καθώς το προγραμματιστικό μοντέλο MapReduce έχει δημιουργηθεί ειδικά για το σκοπό αυτό, οι τεχνικές *MapReduce* και *NoMapReduce* είναι εκείνες με υψηλές δυνατότητες εφαρμογής aggregation queries. Η "*Ασφάλεια*" αφορά στην εγγύηση της διατηρησης των δεδομένων ακόμα και μετά από αστοχία των κόμβων που τα διατηρούν. Καθώς η MongoDB διαθέτει ενσωματωμένων μηχανισμούς για το replication των δεδομένων, στις τεχνικές που τα δεδομένα διατηρούνται αποκλειστικά σε databases της MongoDB, δηλαδή στις τεχνικές *EmbeddedXML*, *KeyValueQueue*, *KeyValuePath*, *MapReduce* και *NoMapReduce* η ασφάλεια των δεδομένων είναι εγγυημένη. Αντίθετα στις τεχνικές *Legacy* και *PathXML* είναι στην ευχέρεια των σχεδιαστών των εφαρμογών η υιοθέτηση κατάλληλων μηχανισμών που να εγγυώνται την διατηρησιμότητα των δεδομένων σε περίπτωση αστοχίας του συστήματος. Για το λόγο αυτό οι δύο αυτές τεχνικές χαρακτηρίζονται "*low*" όσον αφορά στην ασφάλεια των δεδομένων, ενώ οι υπόλοιπες "*high*". Η "*Απαιτούμενη Κατάρτιση*" δείχνει τον απαιτούμενο βαθμό εξοικείωσης του ανθρώπινου δυναμικού που χειρίζεται τις εφαρμογές αναζήτησης με το εκάστοτε λογισμικό, προκειμένου να χρησιμοποιήσει τις τεχνικές που εξετάστηκαν από αυτή τη διπλωματική. Από την ανάλυση του προηγούμενου κεφαλαίου προκύπτει σαν άμεσο συμπέρασμα πως οι τεχνικές *MapReduce* και *NoMapReduce* χαρακτηρίζονται από τις μεγαλύτερες απαιτήσεις σε κατάρτιση των σχεδιαστών των εφαρμογών, προκειμένου να προσφέρουν αξιόπιστες υπηρεσίες. Τέλος το κομμάτι "*Επιδόσεις*" αφορά στο χρόνο απόκρισης της κάθε τεχνικής κατά την εκτέλεση της αναζήτησης.

8.4 Συνεισφορά της Διπλωματικής Εργασίας

Η διπλωματική αυτή εργασία ασχολήθηκε με την αναζήτηση πληροφοριών σε multimedia 3D αρχεία με βάση το περιεχόμενό τους. Για την παροχή έγκυρων απαντήσεων στα ερωτήματα που υποβάλλουν οι χρήστες είναι απαραίτητη η μελέτη των metadata των αρχείων, δηλαδή των δεδομένων που αφορούν στο περιεχόμενο των multimedia αρχείων. Η αναζήτηση εκτελείται σε αυτά και επιστρέφονται τα ονόματα των 3D αρχείων που ικανοποιούν το δοσμένο query. Διακρίνονται δύο τύποι queries, αυτά που εξετάζουν τα metadata και επιστρέφουν μεμονωμένες πληροφορίες για αυτά και σε αυτά που πραγματοποιούν ενοποίηση των δεδομένων και επιστροφή aggregated αποτελεσμάτων.

Υλοποιήθηκαν πέντε τεχνικές αναζήτησης για τον πρώτο τύπο queries. Από αυτές γρηγορότερη απόκριση έχει η τεχνική *KeyValuePath*, ενώ αμέσως επόμενη εναλλακτική αναδείχθηκε η *KeyValueQueue*. Για την υποβολή aggregation queries υλοποιήθηκαν δύο τεχνικές, μία που χρησιμοποιεί τα εργαλεία της MongoDB για εφαρμογή του προγραμματιστικού μοντέλου MapReduce και μία που χρησιμοποιεί προσαρμοσμένες στο ερώτημα Java classes. Από την συγκριτική μελέτη τους διαπιστώθηκε η υπεροχή της *MapReduce* τεχνικής, λόγω της χρήσης των βελτιστοποιημένων εργαλείων της MongoDB και της εύκολης κλιμάκωσής της.

Αποτελεί αντικείμενο μελλοντικής ερευνητικής εργασίας η επέκταση των τεχνικών που αναπτύχθηκαν, όπως για παράδειγμα η αυτόματη ενημέρωση των collections που διατηρούν τα δεδομένα, με εισαγωγή νέων metadata και διαγραφή εκείνων που αφορούν σε αρχεία που δεν υπάρχουν. Σε θετική κατεύθυνση θα ήταν επίσης η ανάπτυξη ενός εργαλείου αυτόματου εντοπισμού keys που ανήκουν σε Embedded JSONObject και JSONArrays ενός JSON

Document της MongoDB. Τότε η τεχνική *KeyValueQueue* θα μπορούσε να αντικατασταθεί από μία νέα τεχνική που θα δεχόταν σαν όρισμα τα ζητούμενα keys και θα επέστρεφε με υποβολή ενός MongoDB Query τα ονόματα των 3D αρχείων που διαθέτουν τα ζητούμενα key/value pairs. Η απόκριση της νέας αυτής τεχνικής αναμένεται να είναι ίδια με αυτή της τεχνικής *KeyValuePath*.

Βιβλιογραφία

[1] R. Lämmel, "Google's MapReduce Programming Model — Revisited".

[2] P. A. Bernstein, V. Hadzilacos and N. Goodman, "Concurrency Control and Recovery in Database Systems".

[3] S. Gilbert and N. A. Lynch, "Perspectives on the CAP Theorem".

[4] Y. Saito and M. Shapiro, "Optimistic Replication".

[5] J.-H. Kang, C.-S. Kim and E.-J. Ko, "An XQuery engine for digital library systems".

[6] D. Tjondronegoro and Y.-P. P. Chen, "Content-Based Indexing and Retrieval Using MPEG-7 and X-Query in Video Data Management Systems".

[7] N. Fatemi, M. Lalmas and T. Rölleke, "How to retrieve multimedia documents described by MPEG-7".

[8] M. Yoshikawa and T. Amagasa, "XRel: A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases".

[9] Y. Chu, L.-T. Chia and S. S. Bhowmick, "Looking at Mapping, Indexing & Querying of MPEG-7 Descriptors in RDBMS with SM3".

Παράρτημα Α:
Πίνακες και Γραφικές Παραστάσεις Αποτελεσμάτων Αναζήτησης Τομέα Μελέτης Search

Collection Id Transformations

Technique / NoCopies	Legacy	PathXML	EmbeddedXML	KeyValueQueue	KeyValuePath
1	35185	34747	34551	982	17
2	67828	68281	67319	1700	10
4	135319	134158	133944	1137	15
8	268999	269325	268924	2115	15
16	535303	529347	541019	2543	28
32	1162458	1055417	1072203	3052	37

Πίνακας 38 Query1 Execution Time Results

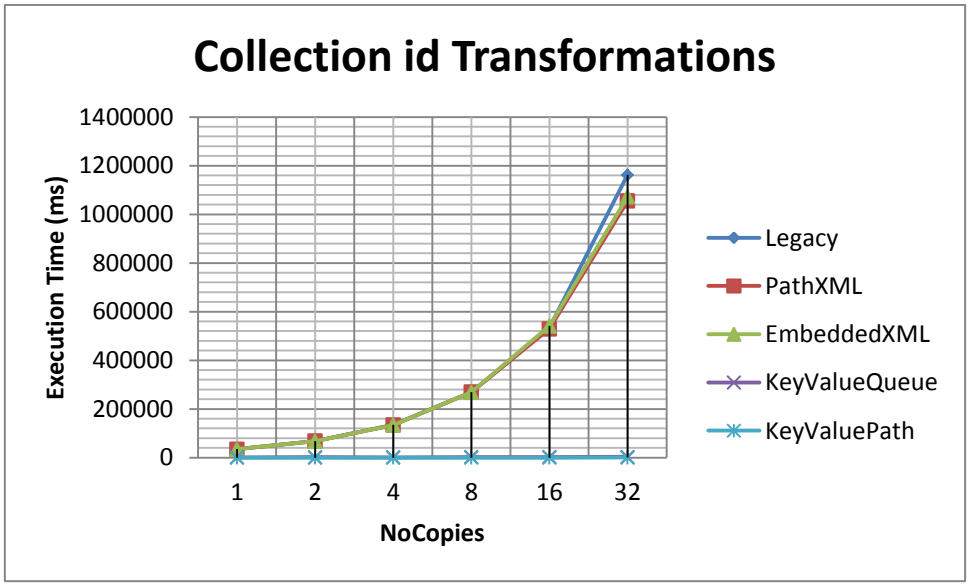
Time Calculated / NoCopies	Total	XML Parsing	Percent
1	34747	34572	99,50%
2	68281	68042	99,65%
4	134158	133937	99,84%
8	269325	268974	99,87%
16	529347	529088	99,95%
32	1055417	1054898	99,95%

Πίνακας 39 Query1 Execution and XML Parsing Time Results PathXML Technique

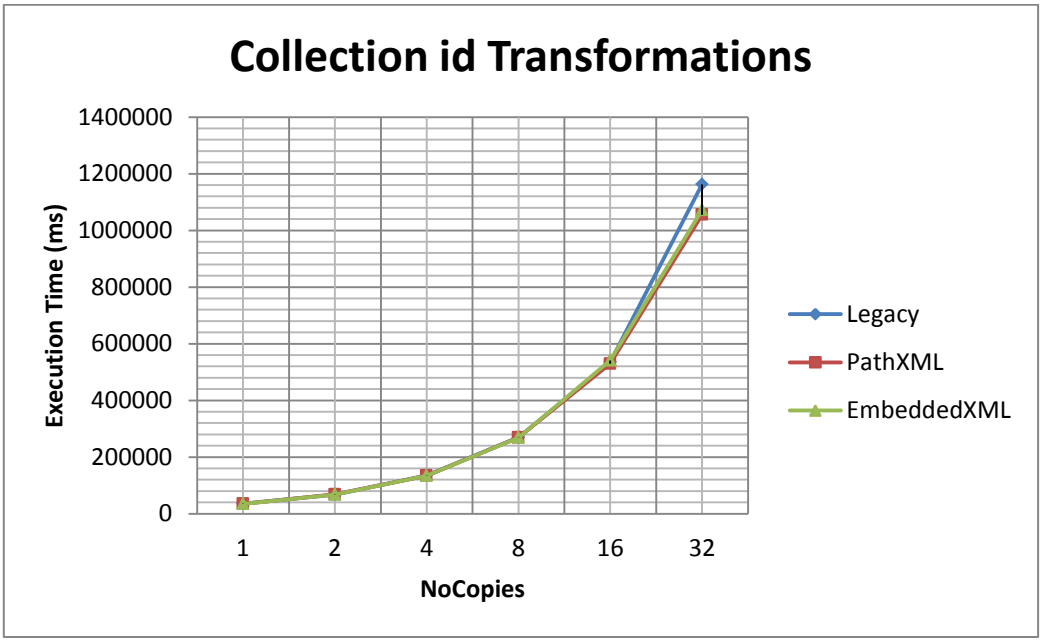
Time Calculated / NoCopies	Total	XML Parsing	Percent
1	34551	34344	99,40%
2	67319	67108	99,69%
4	133944	133690	99,81%
8	268924	268562	99,87%

16	541019	535527	98,98%
32	1072203	1071077	99,89%

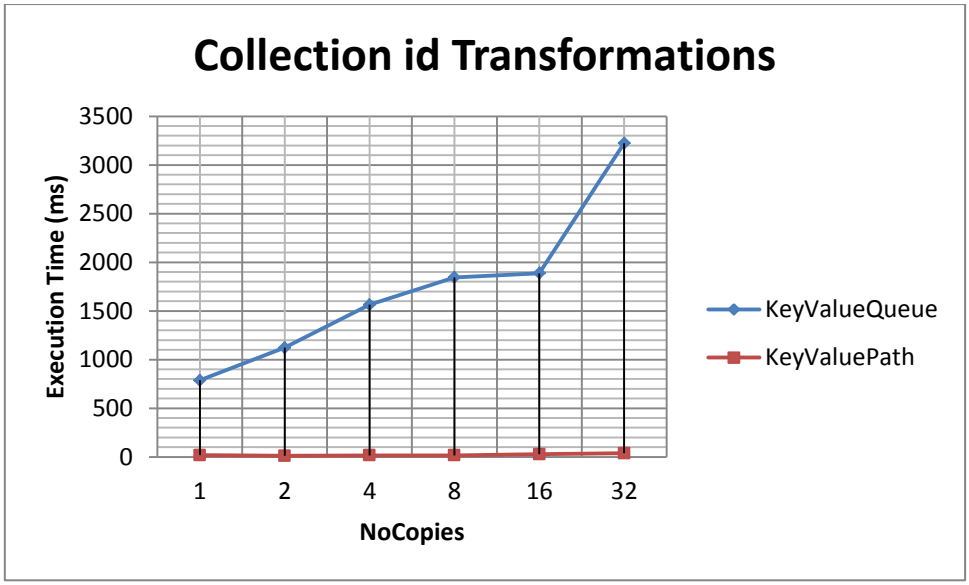
Πίνακας 40 Query1 Execution and XML Parsing Time Results Embedded XML Technique



Εικόνα 34 Query1 All Techniques



Εικόνα 35 Query1 XML Techniques



Εικόνα 36 Query1 KeyValue Techniques

Cylinder radius 0.5

Technique / NoCopies	Legacy	PathXML	EmbeddedXML	KeyValueQueue	KeyValuePath
1	34614	35093	35336	768	16
2	67558	67447	69690	1271	13
4	134362	134771	141317	1709	19
8	267342	268925	264386	1874	25
16	531580	537066	532846	2301	22
32	1069899	1095527	1180849	4258	23

Πίνακας 41 Query2 Execution Time Results

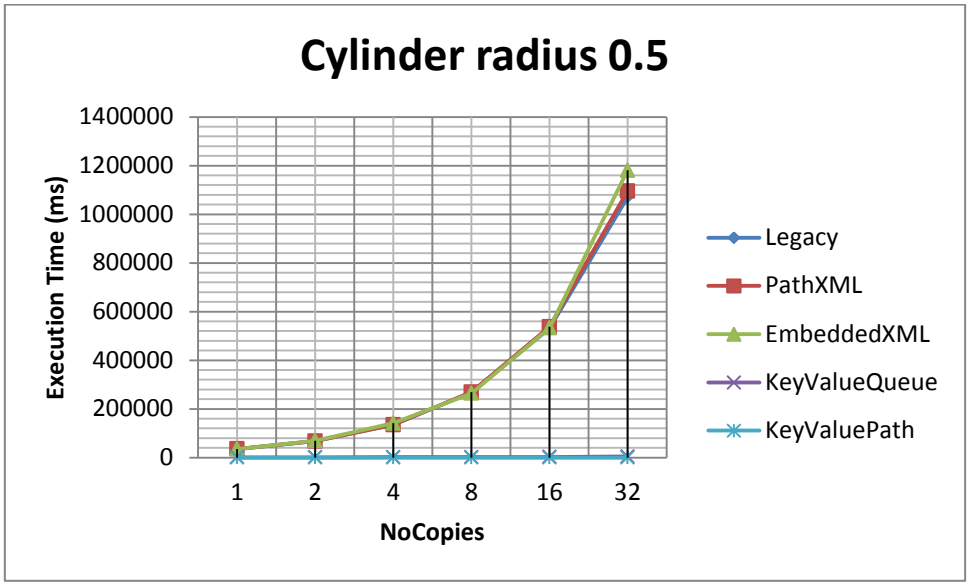
Time Calculated / NoCopies	Total	XML Parsing	Percent
1	35093	34814	99,20%
2	67447	67359	99,87%
4	134771	134467	99,77%
8	268925	268576	99,87%
16	537066	536641	99,92%
32	1095527	1094768	99,93%

Πίνακας 42 Query2 Execution and XML Parsing Time Results PathXML Technique

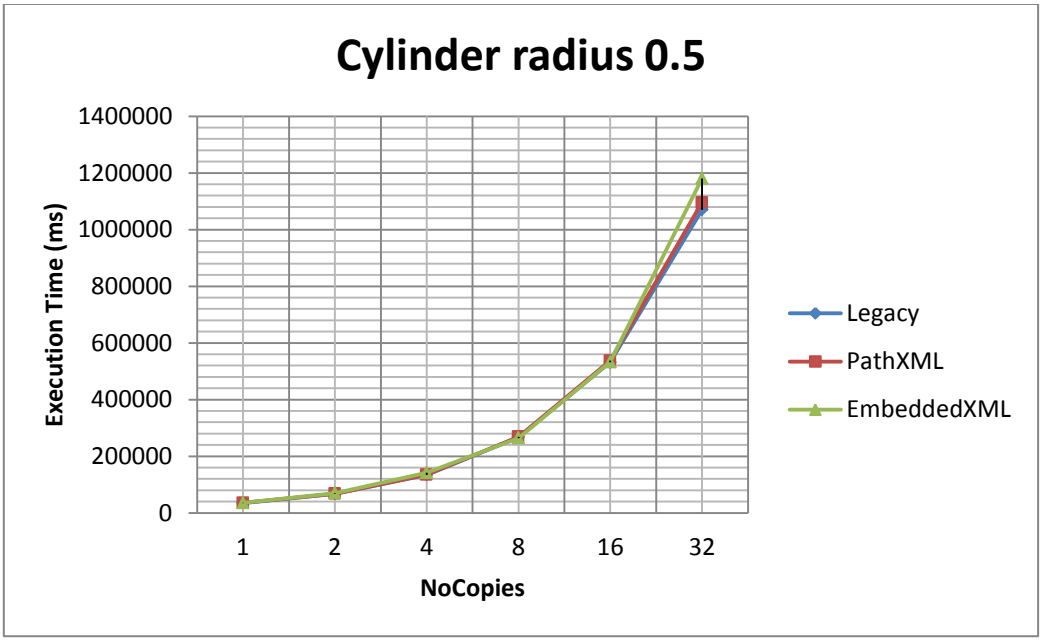
Time Calculated / NoCopies	Total	XML Parsing	Percent
1	35336	35030	99,13%
2	69690	69370	99,54%
4	141317	140764	99,61%

8	264386	263948	99,83%
16	532846	531437	99,74%
32	1180849	1174044	99,42%

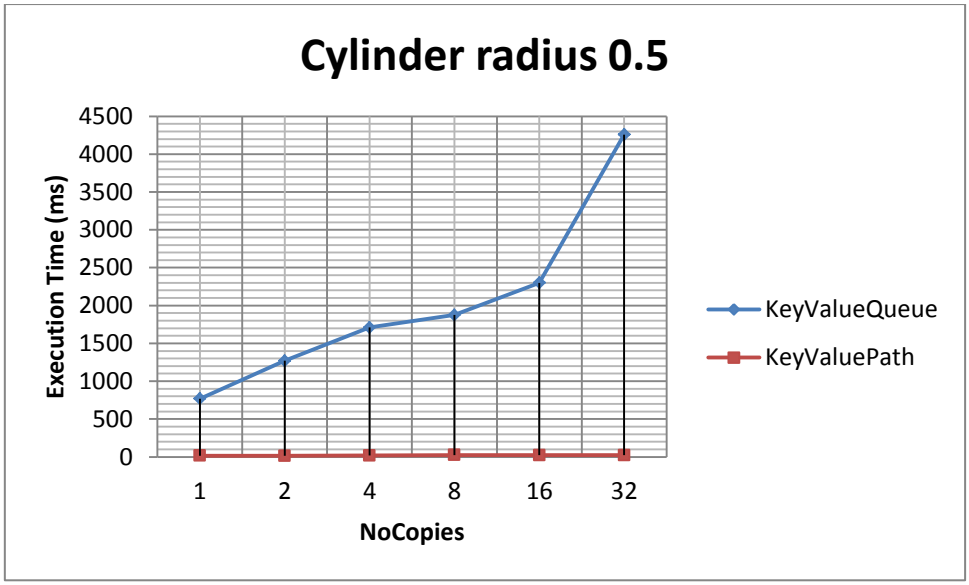
Πίνακας 43 Query2 Execution and XML Parsing Time Results EmbeddedXML Technique



Εικόνα 37 Query2 All Techniques



Εικόνα 38 Query2 XML Techniques



Εικόνα 39 Query2 KeyValue Techniques

meta name title

Technique / NoCopies	Legacy	PathXML	EmbeddedXML	KeyValueQueue	KeyValuePath
1	34038	35045	35192	764	23
2	68460	68657	68426	1350	16
4	135090	135817	135666	1847	23
8	268506	268344	269217	1806	13
16	534062	537127	536810	2629	19
32	1072864	1069861	1071681	3832	25

Πίνακας 44 Query4 Execution Time Results

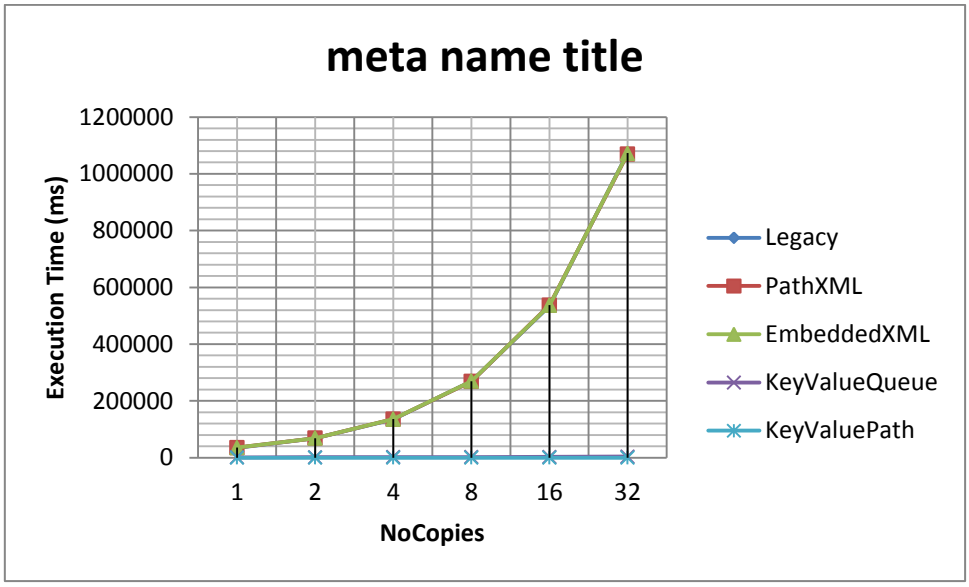
Time Calculated / NoCopies	Total	XML Parsing	Percent
1	35045	34822	99,36%
2	68657	68509	99,78%
4	135817	135537	99,79%
8	268344	268060	99,89%
16	537127	536777	99,93%
32	1069861	1069446	99,96%

Πίνακας 45 Query4 Execution and XML Parsing Time Results PathXML Technique

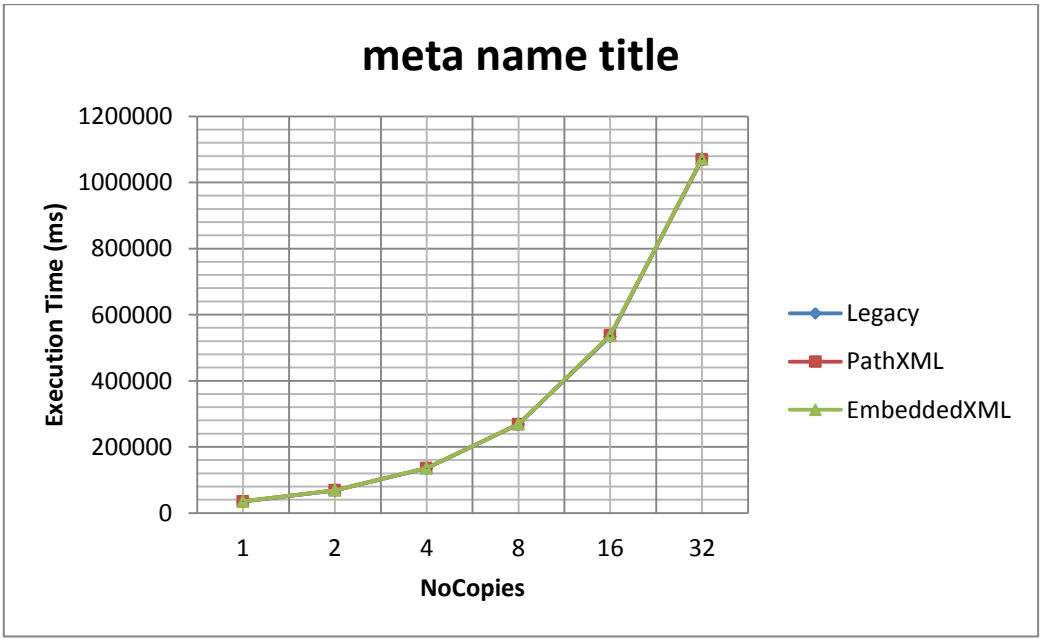
Time Calculated / NoCopies	Total	XML Parsing	Percent
1	35192	35024	99,52%
2	68426	68158	99,61%
4	135666	135217	99,67%
8	269217	268750	99,83%

16	536810	536259	99,90%
32	1071681	1070628	99,90%

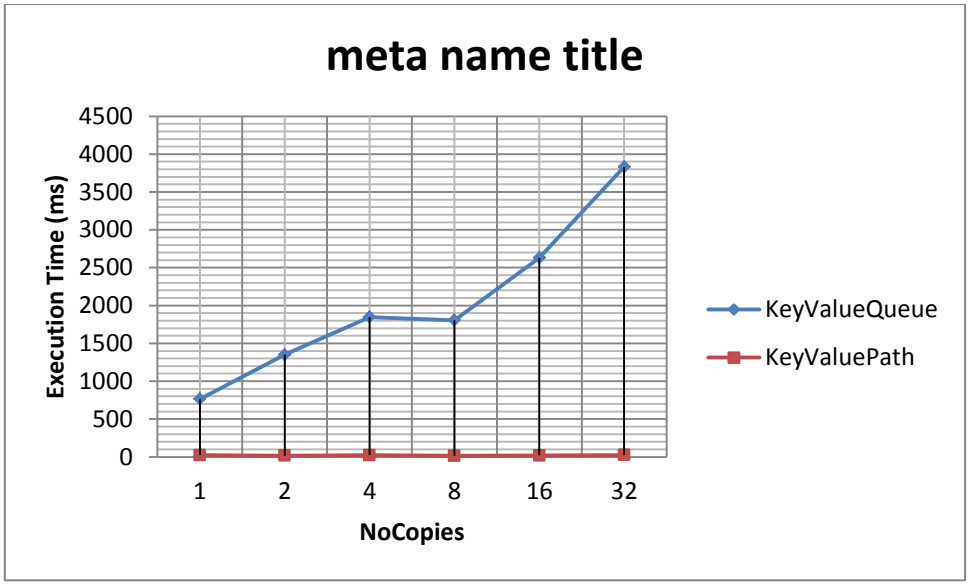
Πίνακας 46 Query4 Execution and XML Parsing Time Results EmbeddedXML Technique



Εικόνα 40 Query4 All Techniques



Εικόνα 41 Query4 XML Techniques



Εικόνα 42 Query4 KeyValue Techniques

Transform DEF TRANS

Technique / NoCopies	Legacy	PathXML	EmbeddedXML	KeyValueQueue	KeyValuePath
1	34316	34430	35003	716	10
2	68175	67105	68019	1418	24
4	136066	134085	132583	1276	23
8	266897	266220	263374	2076	19
16	538448	534212	534213	7507	11
32	1071507	1070870	1064983	4123	11

Πίνακας 47 Query5 Execution Time Results

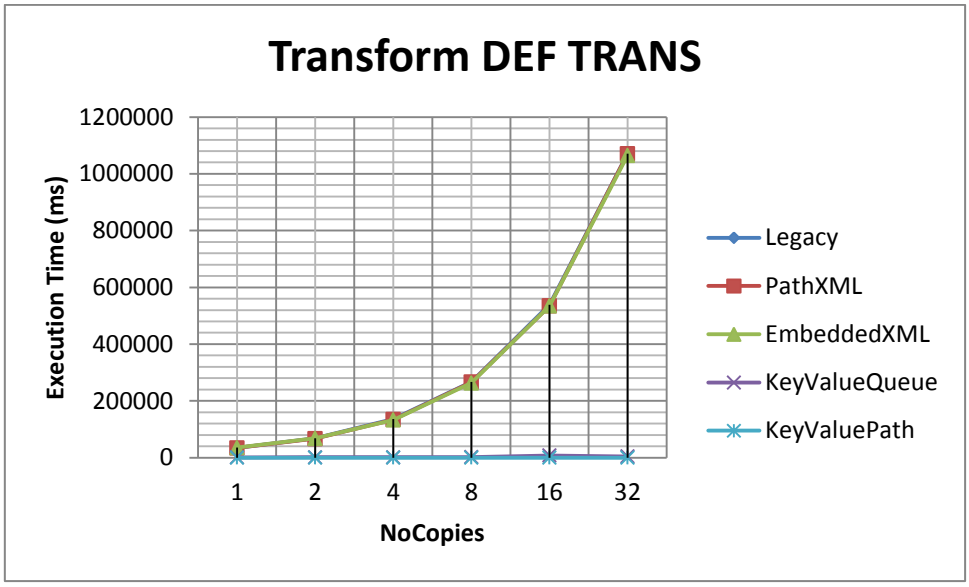
Time Calculated / NoCopies	Total	XML Parsing	Percent
1	34430	34200	99,33%
2	67105	66820	99,58%
4	134085	133852	99,83%
8	266220	265933	99,89%
16	534212	533861	99,93%
32	1070870	1070595	99,97%

Πίνακας 48 Query5 Execution and XML Parsing Time Results PathXML Technique

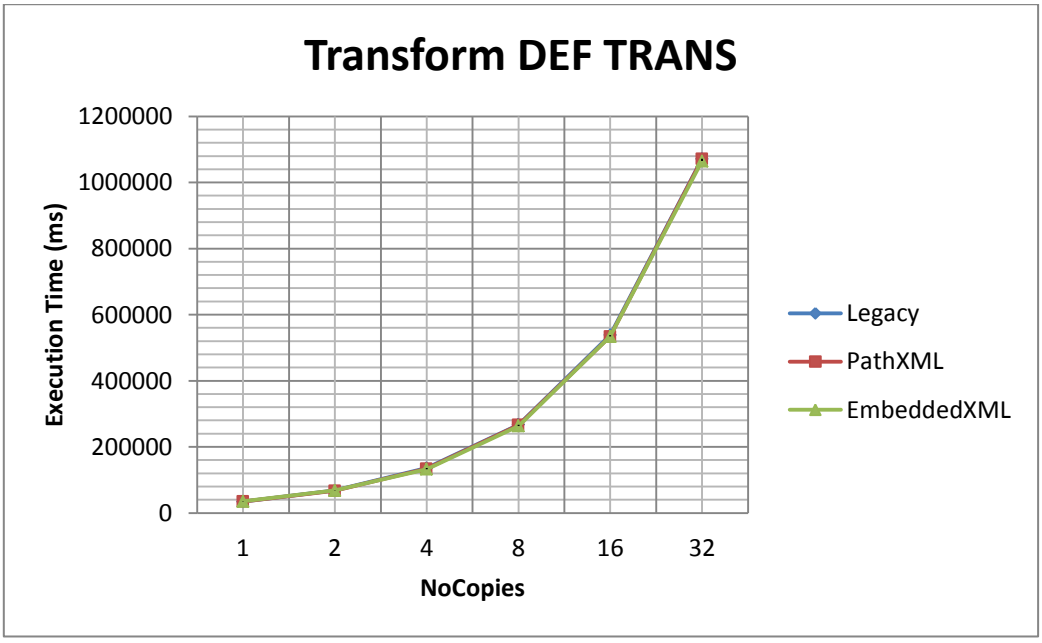
Time Calculated / NoCopies	Total	XML Parsing	Percent
1	35003	34794	99,40%
2	68019	67708	99,54%
4	132583	132363	99,83%
8	263374	263004	99,86%

16	534213	528826	98,99%
32	1064983	1063920	99,90%

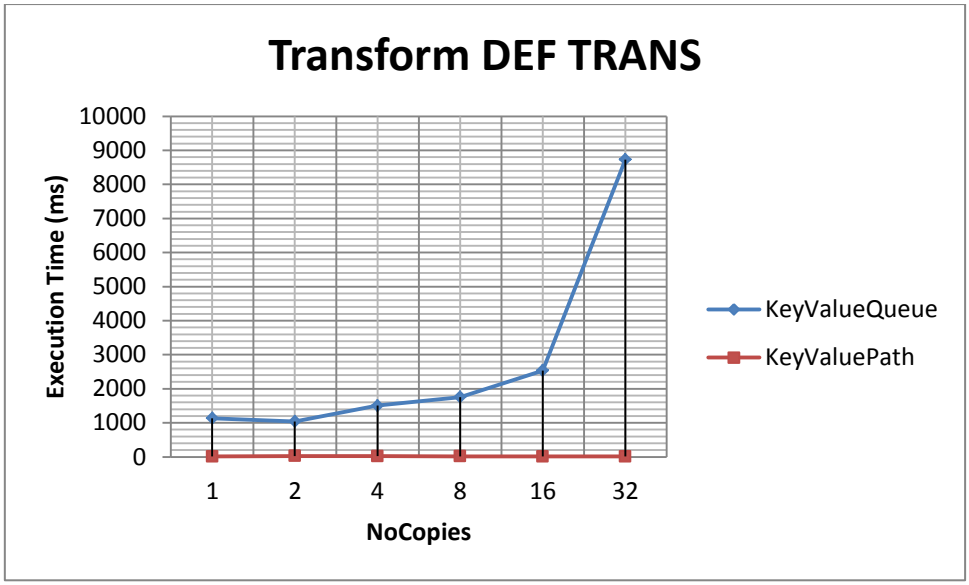
Πίνακας 49 Query5 Execution and XML Parsing Time Results EmbeddedXML Technique



Εικόνα 43 Query5 All Techniques



Εικόνα 44 Query5 XML Techniques



Εικόνα 45 Query5 KeyValue Techniques

X3D Profile Immersive

Technique / NoCopies	Legacy	PathXML	EmbeddedXML	KeyValueQueue	KeyValuePath
1	35305	42063	34933	899	21
2	68630	83963	68230	1091	26
4	135504	159177	136272	1689	14
8	270664	268958	269768	1857	22
16	538061	537705	536885	2178	13
32	1117859	1076233	1069693	3412	22

Πίνακας 50 Query6 Execution Time Results

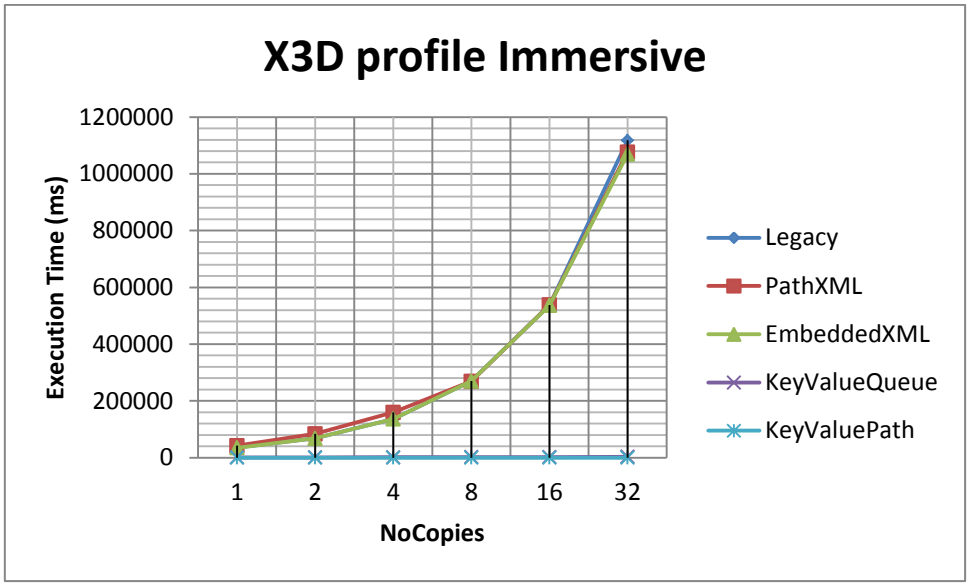
Time Calculated / NoCopies	Total	XML Parsing	Percent
1	42063	41806	99,39%
2	83963	83679	99,66%
4	159177	159039	99,91%
8	268958	268626	99,88%
16	537705	537286	99,92%
32	1076233	1075874	99,97%

Πίνακας 51 Query6 Execution and XML Parsing Time Results PathXML Technique

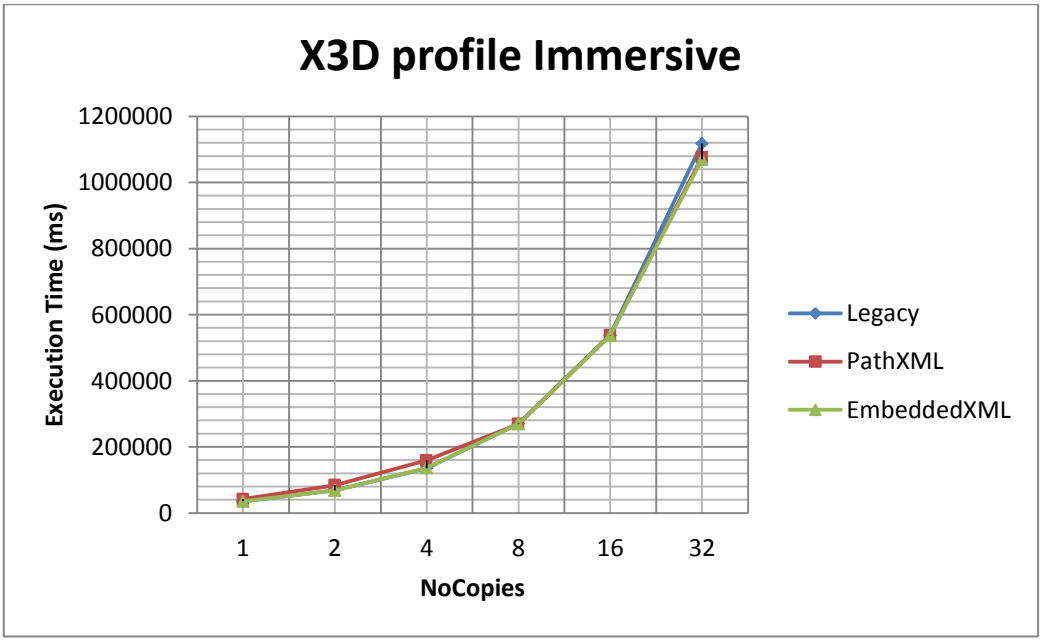
Time Calculated / NoCopies	Total	XML Parsing	Percent
1	34933	34820	99,68%
2	68230	68058	99,75%
4	136272	135864	99,70%
8	269768	269286	99,82%

16	536885	536516	99,93%
32	1069693	1068485	99,89%

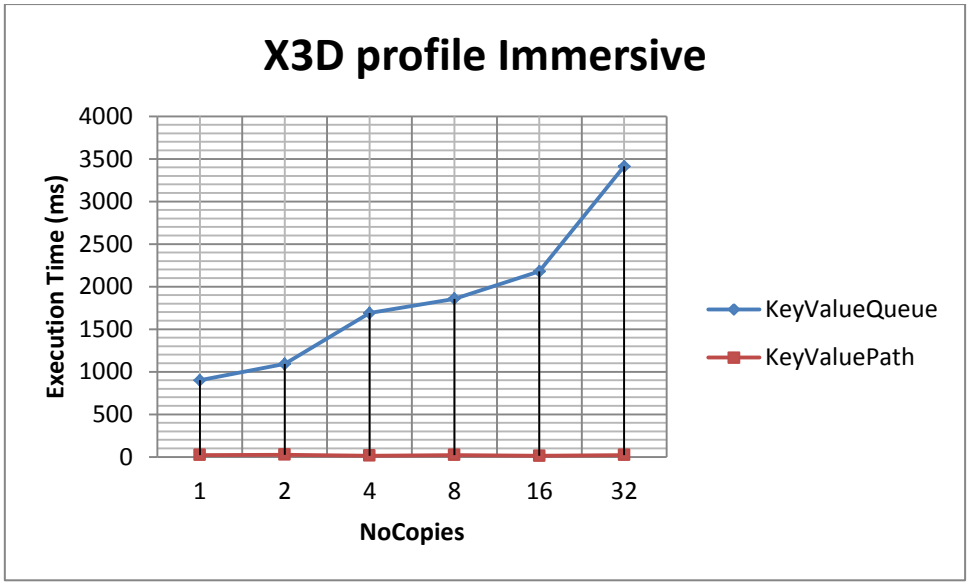
Πίνακας 52 Query6 Execution and XML Parsing Time Results EmbeddedXML Technique



Εικόνα 46 Query6 All Techniques



Εικόνα 47 Query6 XML Techniques



Εικόνα 48 Query6 KeyValue Techniques

Παράρτημα Β:

Πίνακες και Γραφικές Παραστάσεις Αποτελεσμάτων Αναζήτησης Τομέα Μελέτης *MapReduce*

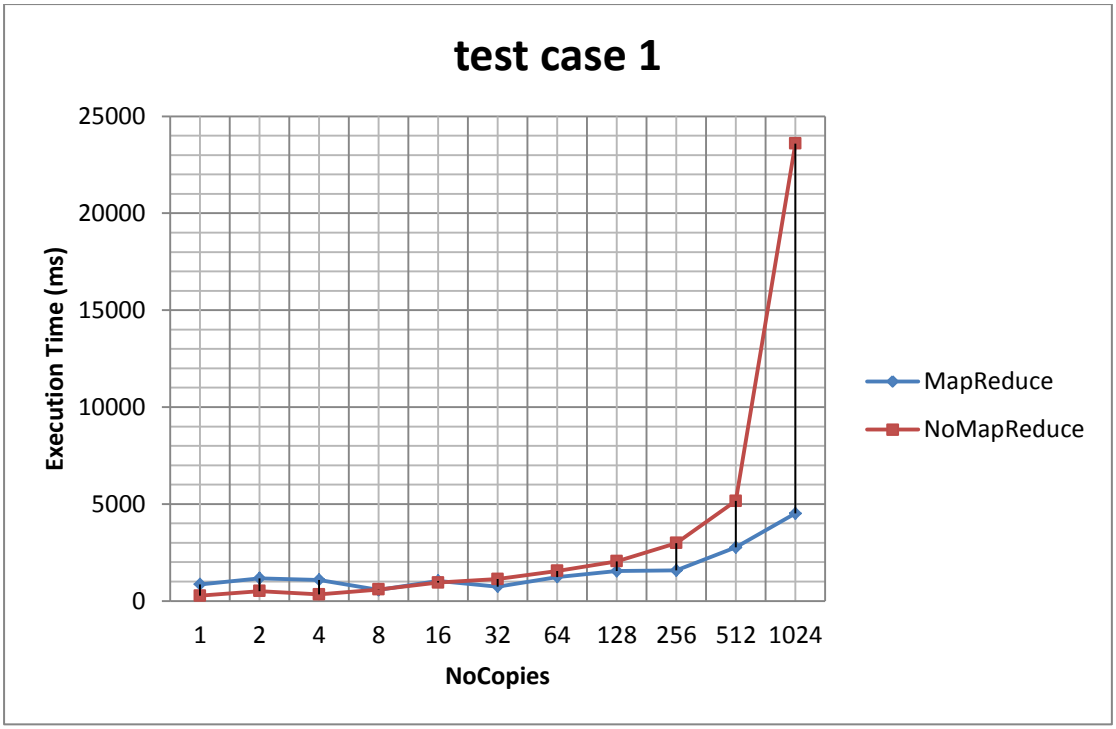
test case 1

Scene "test case 1" Tree Structure
test case 1_room wallpaper test case 1_door handle lock material window window window window window

Πίνακας 53 Scene "test case 1" Tree Structure

Technique / NoCopies	MapReduce	NoMapReduce
1	856	267
2	1168	507
4	1087	333
8	569	589
16	1037	948
32	743	1126
64	1232	1550
128	1540	2048
256	1573	2989
512	2763	5154
1024	4514	23593

Πίνακας 54 test case 1 MapReduce Query Execution Results



Εικόνα 49 test case 1 MapReduce Query Techniques

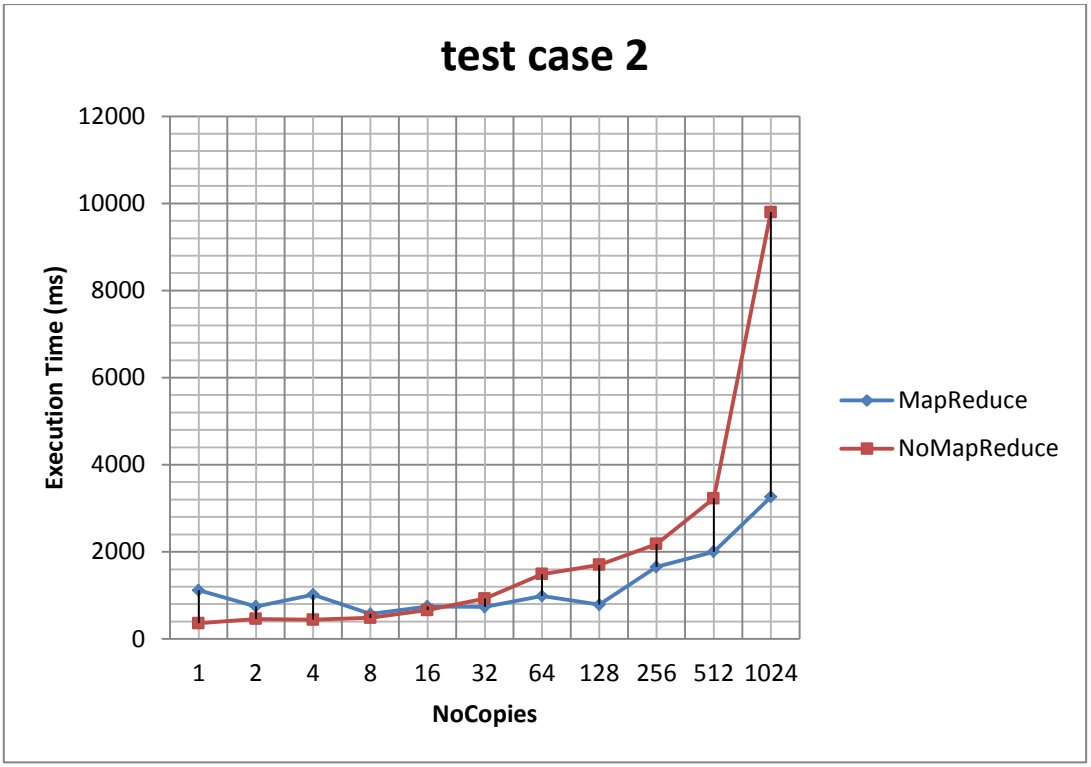
test case 2

Scene "test case 2" Tree Structure
test case 2_machine
gear
bolt
base
lid

Πίνακας 55 Scene "test case 2" Tree Structure

Technique / NoCopies	MapReduce	NoMapReduce
1	1119	362
2	747	460
4	1019	440
8	574	486
16	747	660
32	736	928
64	985	1488
128	787	1702
256	1655	2180
512	2004	3226
1024	3261	9799

Πίνακας 56 test case 2 MapReduce Query Execution Results



Εικόνα 50 test case 2 MapReduce Query Techniques

test case 4

Scene "test case 4" Tree Structure

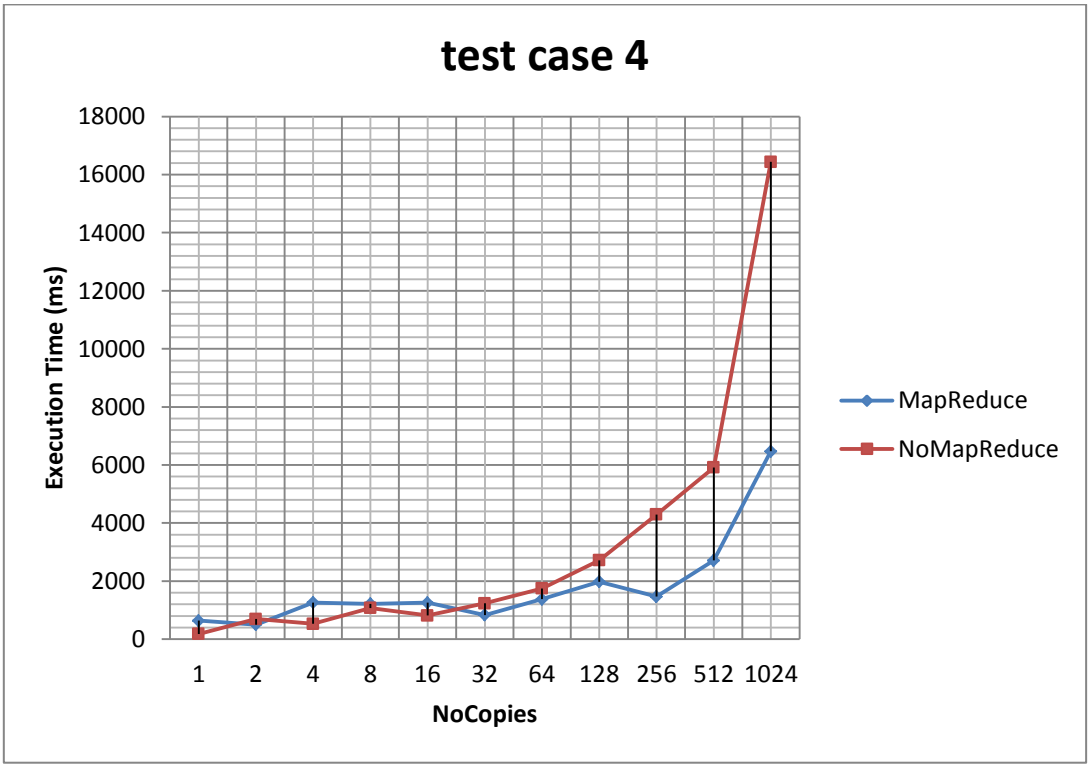
```
test case 4_room
      wallpaper
      test case 4_something
            wallpaper
            test case 4_door
                  handler
                  lock
                  material
      test case 4_something
            Wallpaper
            test case 4_door
                  Handler
                  Lock
                  Material
      test case 4_door
            pomolo
            kleidaria
            yliko
      window
      test case 1_porta
            handler
            lock
            material
      window
      window
```

window window

Πίνακας 57 Scene "test case 4" Tree Structure

Technique / NoCopies	MapReduce	NoMapReduce
1	638	184
2	503	694
4	1260	529
8	1216	1078
16	1255	817
32	835	1237
64	1381	1750
128	1980	2720
256	1467	4293
512	2714	5916
1024	6473	16439

Πίνακας 58 test case 4 MapReduce Query Execution Results



Εικόνα 51 test case 4 MapReduce Query Techniques

test case 5

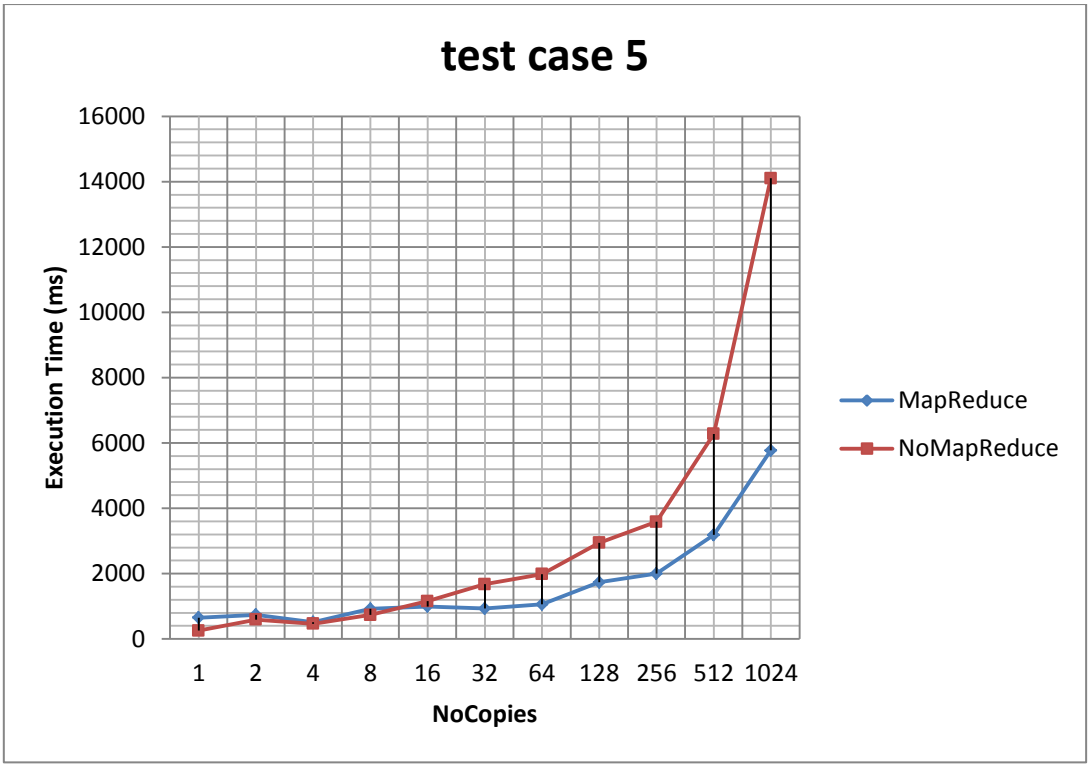
Scene "test case 5" Tree Structure
test case 5_room

```
wallpaper
test case 5_something
      wallpaper
      test case 5_door
            handler
            Lock
            material
test case 5_something
      wallpaper
      test case 5_door
            Handler
            Lock
            Material
test case 5_door
      pomolo
      kleidaria
      yliko
window
test case 1_porta
      handler
      lock
      material
window
window
window
window
```

Πίνακας 59 Scene "test case 5" Tree Structure

Technique / NoCopies	MapReduce	NoMapReduce
1	650	255
2	741	591
4	517	464
8	920	735
16	997	1159
32	930	1675
64	1058	1983
128	1736	2946
256	1999	3587
512	3183	6279
1024	5773	14104

Πίνακας 60 test case 5 MapReduce Query Execution Results



Εικόνα 52 test case 5 MapReduce Query Techniques

Παράρτημα Γ:
Κώδικας Υλοποίησης Τεχνικών Αναζήτησης Τομέα Μελέτης *Search*

Legacy

SearchLegacy.java

```
package Legacy;

import common.*;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import org.jdom2.JDOMException;

public class SearchLegacy {

    String elementName;
    String attributeName;
    String attributeValue;

    File resultsFile;

    /*
     * Scan mpeg7 folder for mpeg7 files.
     * Scan x3d folder for x3d files.
     * Add mpeg7 and x3d files to a Files Array allFiles.
     * Return allFiles.
     */

    private File[] GetFiles(File mpeg7Folder, File x3dFolder){

        File[] mpeg7Files = null;
        File[] x3dFiles = null;
        File[] allFiles;

        if(mpeg7Folder.exists() && x3dFolder.exists()){
            mpeg7Files = mpeg7Folder.listFiles(new FilterXMLFiles());
            x3dFiles = x3dFolder.listFiles(new FilterX3DFiles());
        }

        allFiles = new File[mpeg7Files.length + x3dFiles.length];
        int index=0;

        for(int j=0; j<mpeg7Files.length; j++){
            allFiles[index] = mpeg7Files[j];
            index++;
        }

        for(int j=0; j<x3dFiles.length; j++){
```

```
            allFiles[index] = x3dFiles[j];
            index++;
        }

        return allFiles;
    }

    /*
     * Search files from File Array allFiles.
     * Results written to resultsFile.
     */

    private void SubmitQuery(File[] allFiles) throws IOException,
JDOMException{

        boolean matchQuery;
        String filePath;
        File curFile;

        for(int i=0; i<allFiles.length; i++){

            filePath = allFiles[i].getPath();

            File xmlFile = new File(filePath);
            NavigateXML navigate = new NavigateXML(elementName,
attributeName, attributeValue);
            matchQuery = navigate.QueryXMLFile(xmlFile);

            /*
             * If query returns true write result to results file.
             */

            if(matchQuery){
                curFile = new File(filePath);
                try (FileWriter fileWriter = new FileWriter(resultsFile,
true)) {
                    fileWriter.write(new
ProcessFilename().RemoveSuffix(curFile.getName()) + "\n");
                }
            }
        }
    }

    /*
     * Get Files and Submit Query.
     */

    public void Search(File mpeg7Folder, File x3dFolder) throws IOException,
JDOMException{

        File[] allFiles = GetFiles(mpeg7Folder, x3dFolder);

        SubmitQuery(allFiles);

    }

    /*
     * Constructor
     */

    public SearchLegacy(String elName, String atName, String atValue, String
path, String resultsFileName) throws IOException{
        elementName = elName;
        attributeName = atName;
        attributeValue = atValue;

        /*
         * Results written to path + "Legacy\\results\\" folder.
```

```
         */

        String resultsSubPath = "Legacy\\";
        File jsonSubFolder = new File(path + resultsSubPath);

        if(!jsonSubFolder.exists())
            jsonSubFolder.mkdir();

        resultsSubPath = "Legacy\\results\\";
        String resultsFolderPath = path + resultsSubPath;

        File resultsFolder = new File(resultsFolderPath);

        if(!resultsFolder.exists()){
            resultsFolder.mkdir();
        }

        /*
         * Compose name of results file.
         * Each query creates its own file.
         * If a file with the same name exists, overwrite it.
         */

        resultsFileName += ".txt";
        String resultsFilePath = resultsFolderPath + resultsFileName;

        resultsFile = new File(resultsFilePath);

        if(!resultsFile.exists()){
            resultsFile.createNewFile();
        }
        else{
            try (FileWriter fileWriter = new FileWriter(resultsFile)) {
                fileWriter.flush();
            }
        }
    }
}
```

PathXML

SearchPathXML.java

```
package PathXML;

import common.*;
import com.mongodb.BasicDBObject;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import org.jdom2.JDOMException;

public class SearchPathXML {

    String elementName;
    String attributeName;
    String attributeValue;

    File resultsFile;
    DBCollection collection;
    DB db;


    private void SubmitQuery(String filePath) throws JDOMException,
```

```
IOException{

        boolean matchQuery;
        File xmlFile = new File(filePath);

        NavigateXML navigate = new NavigateXML(elementName, attributeName,
attributeValue);
        matchQuery = navigate.QueryXMLFile(xmlFile);

        /*
         * If query returns true write result to results file.
         */

        if(matchQuery){
            File curFile = new File(filePath);
            try (FileWriter fileWriter = new FileWriter(resultsFile, true)){
                    fileWriter.write(new
ProcessFilename().RemoveSuffix(curFile.getName()) + "\n");
                }
        }
    }

    /*
     * Get all documents of MongoDB collection "PathXML".
     * Get value of keys "Mpeg7" and "X3D" - filepath of XML Description.
     * Submit Query to filepath.
     */

    public void Search() throws IOException, JDOMException{

        String filePath;

        DBCursor cursor = collection.find();

        while (cursor.hasNext()) {
            BasicDBObject nextDoc = (BasicDBObject) cursor.next();

            if(nextDoc.containsField("Mpeg7")){
                filePath = (String) nextDoc.get("Mpeg7");
                SubmitQuery(filePath);
            }

            if(nextDoc.containsField("X3D")){
                filePath = (String) nextDoc.get("X3D");
                SubmitQuery(filePath);
            }

       }
    }

    /*
     * Constructor
     */

    public SearchPathXML(String elName, String atName, String atValue,
String path, String resultsFileName, DB database) throws IOException{
        elementName = elName;
        attributeName = atName;
        attributeValue = atValue;

        /*
         * Results written to path + "PathXML\\results\\" folder.
         */

        String resultsSubPath = "PathXML\\";
        File jsonSubFolder = new File(path + resultsSubPath);

        if(!jsonSubFolder.exists())
```

```
            jsonSubFolder.mkdir();

        resultsSubPath = "PathXML\\results\\";
        String resultsFolderPath = path + resultsSubPath;

        File resultsFolder = new File(resultsFolderPath);

        if(!resultsFolder.exists()){
            resultsFolder.mkdir();
        }

        /*
         * Compose name of results file.
         * Each query creates its own file.
         * If a file with the same name exists, overwrite it.
         */

        resultsFileName += ".txt";
        String resultsFilePath = resultsFolderPath + resultsFileName;

        resultsFile = new File(resultsFilePath);

        if(!resultsFile.exists()){
            resultsFile.createNewFile();
        }
        else{
            try (FileWriter fileWriter = new FileWriter(resultsFile)) {
                fileWriter.flush();
            }
        }

        String colName = "PathXML";
        db = database;
        collection = db.getCollection(colName);
    }
}
```

EmbeddedXML

SearchEmbeddedXML.java

```
package EmbeddedXML;

import com.mongodb.BasicDBObject;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodb.Bytes;
import common.NavigateXML;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import org.jdom2.JDOMException;

public class SearchEmbeddedXML {

    String elementName;
    String attributeName;
    String attributeValue;

    File resultsFile;
    DBCollection collection;
    DB db;

    private void SubmitQuery(String xmlString, String name) throws
JDOMException, IOException{
```

```
        boolean matchQuery;

        NavigateXML navigate = new NavigateXML(elementName, attributeName,
attributeValue);
        matchQuery = navigate.QueryXMLSrting(xmlString);

        /*
         * If query returns true write result to results file.
         */

        if(matchQuery){
            try (FileWriter fileWriter = new FileWriter(resultsFile, true)){
                    fileWriter.write(name + "\n");
                }
        }
    }

    /*
     * Get all documents of MongoDB collection "PathXML".
     * Get value of keys "Mpeg7" and "X3D" (the XML Description as String).
     * Submit Query to value.
     */

    public void Search() throws IOException, JDOMException{

        String xmlString;

        /*
         * QUERYOPTION_NOTIMEOUT is used so that the cursor doesn't timeout.
         * Happens in large databases.
         * Which one should be used?
         */

        DBCursor cursor = collection.find().addOption(new
Bytes().QUERYOPTION_NOTIMEOUT);

        while (cursor.hasNext()) {

            BasicDBObject nextDoc = (BasicDBObject) cursor.next();

            if(nextDoc.containsField("Mpeg7")){
                xmlString = (String) nextDoc.get("Mpeg7");
                SubmitQuery(xmlString, nextDoc.getString("name"));
            }

            if(nextDoc.containsField("X3D")){
                xmlString = (String) nextDoc.get("X3D");
                SubmitQuery(xmlString, nextDoc.getString("name"));
            }
       }
    }

    /*
     * Constructor
     */

    public SearchEmbeddedXML(String elName, String atName, String atValue,
String path, String resultsFileName, DB database) throws IOException{
        elementName = elName;
        attributeName = atName;
        attributeValue = atValue;

        /*
         * Results written to path + "EmbeddedXML\\results\\" folder.
         */

        String resultsSubPath = "EmbeddedXML\\";
        File jsonSubFolder = new File(path + resultsSubPath);
```

```
        if(!jsonSubFolder.exists())
            jsonSubFolder.mkdir();

        resultsSubPath = "EmbeddedXML\\results\\";
        String resultsFolderPath = path + resultsSubPath;

        File resultsFolder = new File(resultsFolderPath);

        if(!resultsFolder.exists()){
            resultsFolder.mkdir();
        }

        /*
         * Compose name of results file.
         * Each query creates its own file.
         * If a file with the same name exists, overwrite it.
         */

        resultsFileName += ".txt";
        String resultsFilePath = resultsFolderPath + resultsFileName;

        resultsFile = new File(resultsFilePath);

        if(!resultsFile.exists()){
            resultsFile.createNewFile();
        }
        else{
            try (FileWriter fileWriter = new FileWriter(resultsFile)) {
                fileWriter.flush();
            }
        }

        String colName = "EmbeddedXML";
        db = database;
        collection = db.getCollection(colName);

    }
}
```

KeyValue

SearchKeyValueQueue.java

```
package KeyValue;

import com.mongodb.BasicDBObject;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.LinkedList;
import java.util.Queue;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

public class SearchKeyValueQueue {
    String elementName;
    String attributeName;
    Object attributeValue;

    File resultsFile;
    DBCollection collection;
    DB db;
```

```
    Queue<JSONObject> myQueue = new LinkedList<>();

    /*
     * Gets a JSONObject. Takes all key/value pairs.
     * If key = attributeName checks the value.
     * If value is a String returns if value is equal to attributeValue.
     * If value is a JSONArray returns if it contains
     * a String equal to attributeValue.
     */

    private boolean CheckForAttributeMatchJSONObject(JSONObject json) throws
JSONException{

        JSONArray allKeys = json.names();
        Object curValue;

        for(int i=0; i<allKeys.length(); i++){
            if(allKeys.get(i).equals(attributeName)){

                curValue = json.get(attributeName);

                if(curValue.equals(attributeValue)){
                    return true;
                }

                else if(curValue instanceof JSONArray){

                    JSONArray arrayValues = (JSONArray) curValue;

                    for(int j=0; j<arrayValues.length(); j++){
                        if( arrayValues.get(j).equals(attributeValue)){
                            return true;
                        }
                    }
                }
            }
        }

        return false;
    }

    /*
     * Takes a JSON Array.
     * Returns if it contains a child JSON Object,
     * with key = attributeName and value = attributeValue.
     */

    private boolean CheckForAttributrMatchJSONArray(JSONArray jsonArray)
throws JSONException {

        JSONObject childJSON;

        for(int j=0; j<jsonArray.length(); j++){

            if (jsonArray.get(j) instanceof JSONObject){
                childJSON = jsonArray.getJSONObject(j);

                if(childJSON.has(attributeName)){
                    if (childJSON.get(attributeName).equals(attributeValue))
                        return true;
                }
            }
        }
        return false;
    }

    /*
```

```
     * Takes father JSON Object. Extracts child JSON Objects.
     * Adds child JSON Objects to Queue.
     */

    private void AddChildJSONObjsToQueue(JSONObject fatherJSON) throws
JSONException{

        JSONObject childJSON;

        String childKey;
        Object childValue;

        JSONArray allKeys = fatherJSON.names();

        for(int i=0; i<allKeys.length(); i++){
            childKey = (String) allKeys.get(i);
            childValue = fatherJSON.get(childKey);

            childJSON = new JSONObject();
            childJSON.put(childKey, childValue);

            myQueue.add(childJSON);
        }
    }

    /*
     * Gets a JSONObject with one key/value pair where value is a JSONArray.
     * Adds to Queue key/value pairs where key is the original key
     * and values all values found in JSON Array.
     *
     * {"key" : [JSONObject1, JSONObject2, JSONObject3]}
     *
     * New JSONObjects added to Queue :
     * {"key":JSONObject1} , {"key":JSONObject2} and {"key":JSONObject3}
     */

    private void AddChildJSONArrToQueue(JSONObject fatherJSON) throws
JSONException{

        String key = (String) fatherJSON.names().get(0);
        JSONArray array = fatherJSON.getJSONArray(key);

        JSONObject childJSON;
        String childKey = key;
        Object childValue;

        for(int i=0; i<array.length(); i++){

            childValue = array.get(i);

            childJSON = new JSONObject();
            childJSON.put(childKey, childValue);

            myQueue.add(childJSON);
        }
    }

    /*
     * Pulls all JSON Objects from Queue one at a time.
     * For each JSON Object gets key and value. If value
     * is a JSONObject or JSONArray, adds child JSONObjects to Queue.
     * If key = elementName and a match hasn't been found yet,
     * checks the value for a match.
     */

    private boolean SearchQueue() throws JSONException{

        JSONObject json;
```

```
        String key;
        Object value;

        boolean found = false;

        while(!myQueue.isEmpty()){
            json = myQueue.poll();

            key = (String) json.names().get(0);
            value = json.get(key);

            if(value instanceof JSONObject )
                AddChildJSONObjsToQueue((JSONObject) value);
            else if (value instanceof JSONArray)
                AddChildJSONArrToQueue(json);

            if((key.equalsIgnoreCase(elementName)) && (found == false)){

                if(value instanceof JSONArray){
                    found = CheckForAttributrMatchJSONArray((JSONArray)
value);
                    if (found == true){
                        myQueue.clear();
                        return true;
                    }
                }
                else if (value instanceof JSONObject){
                    found = CheckForAttributeMatchJSONObject((JSONObject)
value);
                    if (found == true){
                        myQueue.clear();
                        return true;
                    }
                }
            }
        }
        return false;
    }

    private void SubmitQuery(JSONObject json) throws IOException,
JSONException{
        boolean matchQuery;

        AddChildJSONObjsToQueue(json);
        matchQuery = SearchQueue();

        /*
         * If query returns true write result to results file.
         */

        if(matchQuery){
            try (FileWriter fileWriter = new FileWriter(resultsFile, true)){
                    fileWriter.write(json.getString("name") + "\n");
                }
        }
    }

    public void Search() throws JSONException, IOException{
        DBCursor cursor = collection.find();

        while (cursor.hasNext()) {

            BasicDBObject nextDoc = (BasicDBObject) cursor.next();

            JSONObject jDoc = new JSONObject(nextDoc.toString());

            SubmitQuery(jDoc);
        }
```

```
    }

    /*
     * Constructor
     */

    public SearchKeyValueQueue(String elName, String atName, Object atValue,
String path, String resultsFileName, DB database) throws IOException{
        elementName = elName;
        attributeName = atName;
        attributeValue = atValue;

        /*
         * Results written to path + "KeyValue\\results\\Queue\\" folder.
         */

        String resultsSubPath = "KeyValue\\";
        File jsonSubFolder = new File(path + resultsSubPath);

        if(!jsonSubFolder.exists())
            jsonSubFolder.mkdir();

        resultsSubPath = "KeyValue\\results\\";
        jsonSubFolder = new File(path + resultsSubPath);

        if(!jsonSubFolder.exists())
            jsonSubFolder.mkdir();

        resultsSubPath = "KeyValue\\results\\Queue\\";
        String resultsFolderPath = path + resultsSubPath;

        File resultsFolder = new File(resultsFolderPath);

        if(!resultsFolder.exists()){
            resultsFolder.mkdir();
        }

        /*
         * Compose name of results file.
         * Each query creates its own file.
         * If a file with the same name exists, overwrite it.
         */

        resultsFileName += ".txt";
        String resultsFilePath = resultsFolderPath + resultsFileName;

        resultsFile = new File(resultsFilePath);

        if(!resultsFile.exists()){
            resultsFile.createNewFile();
        }
        else{
            try (FileWriter fileWriter = new FileWriter(resultsFile)) {
                fileWriter.flush();
            }
        }

        String colName = "KeyValue";
        db = database;
        collection = db.getCollection(colName);
    }
}
```

SearchKeyValuePath.java

```
package KeyValue;

import com.mongodb.BasicDBObject;
```

```
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import org.json.JSONException;

public class SearchKeyValuePath {

    File resultsFile;
    DBCollection collection;
    DB db;
    BasicDBObject query;

    public void Search() throws JSONException, IOException{

        DBCursor cursor = collection.find(query);

        while (cursor.hasNext()) {

            BasicDBObject nextDoc = (BasicDBObject) cursor.next();

            try (FileWriter fileWriter = new FileWriter(resultsFile, true)){
                    fileWriter.write(nextDoc.get("name") + "\n");
                }
        }
    }

    /*
     * Constructor
     */

    public SearchKeyValuePath(String path, String resultsFileName, DB
database, BasicDBObject qry) throws IOException{

        /*
         * Results written to path + "KeyValue\\results\\Path\\" folder.
         */

        String resultsSubPath = "KeyValue\\";
        File jsonSubFolder = new File(path + resultsSubPath);

        if(!jsonSubFolder.exists())
            jsonSubFolder.mkdir();

        resultsSubPath = "KeyValue\\results\\";
        jsonSubFolder = new File(path + resultsSubPath);

        if(!jsonSubFolder.exists())
            jsonSubFolder.mkdir();

        resultsSubPath = "KeyValue\\results\\Path\\";
        String resultsFolderPath = path + resultsSubPath;

        File resultsFolder = new File(resultsFolderPath);

        if(!resultsFolder.exists()){
            resultsFolder.mkdir();
        }

        /*
         * Compose name of results file.
         * Each query creates its own file.
         * If a file with the same name exists, overwrite it.
         */

        resultsFileName += ".txt";
```

```
        String resultsFilePath = resultsFolderPath + resultsFileName;

        resultsFile = new File(resultsFilePath);

        if(!resultsFile.exists()){
            resultsFile.createNewFile();
        }
        else{
            try (FileWriter fileWriter = new FileWriter(resultsFile)) {
                fileWriter.flush();
            }
        }

        String colName = "KeyValue";
        db = database;
        collection = db.getCollection(colName);
        query = qry;
    }
}
```

common

NavigateXML.java

```
package common;

import java.io.File;
import java.io.IOException;
import java.io.StringReader;
import java.util.List;
import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.Attribute;
import org.jdom2.JDOMException;
import org.jdom2.filter.ElementFilter;
import org.jdom2.input.SAXBuilder;
import org.jdom2.util.IteratorIterable;
import org.xml.sax.InputSource;

public class NavigateXML {

    String elementName;
    String attributeName;
    String attributeValue;

    /*
     * Navigate through the xml file specified in filePath.
     * Look for elements with name = "elementName",
     * attribute = "attirbuteName" and attribute_value = "attributeValue".
     * If such element exists returns true, else false.
     */

    private boolean SubmitQueryInXMLFiles(Document doc) throws
JDOMException, IOException{

        String curAttributeName;
        String curAttributeValue;

        /*
         * Get root element.
         */

        Element root = doc.getRootElement();

        /*
         * Check root node for match.
         */
```

```
        if(root.getName().equals(elementName))
            if(root.getAttributeValue(attributeName).equals(attributeValue))
                return true;

        /*
         * Get all Descendants of root (all elements)
         * with element_name = elementName.
         */

        IteratorIterable<Element> allNodes = root.getDescendants(new
ElementFilter(elementName));

        while(allNodes.hasNext()){

            Element curNode = allNodes.next();

            /*
             * Get all attributes of filtered elements and check if
             * same with given attributeName and attributeValue.
             */


            List<Attribute> allAttributes = curNode.getAttributes();

            for(int i=0; i<allAttributes.size(); i++){

                curAttributeName = allAttributes.get(i).getName();
                curAttributeValue = allAttributes.get(i).getValue();

                if((attributeName.equals(curAttributeName)) &&
(attributeValue.equals(curAttributeValue)))
                    return true;
            }
        }
        return false;
    }

    public boolean QueryXMLSrting(String xmlString) throws JDOMException,
IOException{

        InputSource xmlFile = new InputSource(new StringReader(xmlString));

        /*
         * Builds a JDOM Document using a SAX Parser.
         */

        Document doc;
        SAXBuilder saxBuilder = new SAXBuilder();
        doc = saxBuilder.build(xmlFile);

        return SubmitQueryInXMLFiles(doc);
    }

    public boolean QueryXMLFile(File xmlFile) throws JDOMException,
IOException{

        /*
         * Builds a JDOM Document using a SAX Parser.
         */

        Document doc;
        SAXBuilder saxBuilder = new SAXBuilder();
        doc = saxBuilder.build(xmlFile);

        return SubmitQueryInXMLFiles(doc);
    }
```

```
    public NavigateXML(String elName, String atName, String atValue){
        elementName = elName;
        attributeName = atName;
        attributeValue = atValue;
    }

}
```

FilterX3DFiles.java

```
package common;

import java.io.File;
import java.io.FileFilter;

/*
 * Allows only files that end with ".x3d" or ".X3D"
 */

public class FilterX3DFiles implements FileFilter{

    @Override
    public boolean accept(File pathName) {

        if (pathName.toString().endsWith(".x3d") ||
pathName.toString().endsWith(".X3D")){
            return true;
        }

        return false;
    }
}
```

FilterXMLFiles.java

```
package common;

import java.io.File;
import java.io.FileFilter;

/*
 * Allows only files that end with ".xml" or ".XML"
 */

public class FilterXMLFiles implements FileFilter{

    @Override
    public boolean accept(File pathName) {

        if (pathName.toString().endsWith(".xml")||
pathName.toString().endsWith(".XML")){
            return true;
        }
        return false;
    }
}
```

ProcessFilename.java

```
package common;

public class ProcessFilename {
    /*
     * Remove extention from given filename.
     * "foo.txt" -> "foo"
     */
```

```
    public String RemoveSuffix (String fileName){
        String newFileName;
        int suffix = fileName.lastIndexOf(".");

        newFileName = fileName.substring(0, suffix);

        return newFileName;
    }

    /*
     * Return extention from given filename.
     * "foo.txt" -> ".txt"
     */

    public String GetSuffix (String fileName){
        String suffix;

        int index = fileName.lastIndexOf(".");
        suffix = fileName.substring(index);

        return suffix;
    }
}
```

Παράρτημα Δ:
Κώδικας Υλοποίησης Τεχνικών Αναζήτησης Τομέα Μελέτης *MapReduce*

MapReduce

MapReduce.java

```
package MapReduce;

import com.mongodb.BasicDBList;
import com.mongodb.BasicDBObject;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodb.DBObject;
import com.mongodb.MapReduceCommand;
import com.mongodb.MapReduceOutput;
import com.mongodb.MongoClient;
import common.WriteResult;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.net.UnknownHostException;
import java.util.Scanner;

public class MapReduce {

    File resultsFile;
    String pathOfRes;
    String pathOfFunctions;

    /*
     * Create resultsFile with path = "pathOfRes".
     * If resultFile exists, flush it.
     */

    private void CreateResultsFile () throws IOException{
        resultsFile = new File(pathOfRes);
        if(!resultsFile.exists()){
            resultsFile.createNewFile();
        }
        else{
            try (FileWriter fileWriter = new FileWriter(resultsFile)) {
                fileWriter.flush();
            }
        }
```

```
    }

    /*
     * Convert inFile into a String. Return String.
     */

    private String GetFunctionString(File inFile) throws
FileNotFoundException{

        String functionString;

        Scanner jsonContent = new Scanner(inFile);
        functionString = jsonContent.useDelimiter("\\Z").next();

        return functionString;
    }

    /*
     * Query documents in "MapReduce" collection.
     * Select those that contain "child" and "count" fields.
     * If count of at least one document is greater that 0,
     * repeat = true. Else repeat = false.
     */

    private boolean Done(DBCollection col){

        boolean repeat = false;

        BasicDBObject query = new BasicDBObject("value.child", new
BasicDBObject("$exists", true));
        query.put("value.count", new BasicDBObject("$exists", true));

        DBCursor cursor = col.find(query);
        BasicDBObject nextDoc;
        Object count;

        while((cursor.hasNext()) && (repeat == false)){
            nextDoc = (BasicDBObject) cursor.next();

            count = ((DBObject)nextDoc.get("value")).get("count");

            if((double) count != 0){
                repeat = true;
            }
        }
        return repeat;
    }

    /*
     * Query all docs in "MapReduce" collection.
     * Select fields "whole", "color" and "names".
     * Sort them by field "whole" in descending order.
     * Write to resultsFile with filePath = "pathRes"
     * those with max value "whole".
     */

    private void QueryOutput(DBCollection col) throws IOException{
        Object cur_whole, color, max, names;
        BasicDBList listNames;

        CreateResultsFile();

        BasicDBObject allDocs = new BasicDBObject();

        BasicDBObject fields = new BasicDBObject();
        fields.put("value.whole", 1);
        fields.put("value.color", 1);
        fields.put("value.names", 1);
```

```
        BasicDBObject orderBy = new BasicDBObject();
        orderBy.put("value.whole",-1);

        DBCursor cursor = col.find(allDocs, fields).sort(orderBy);
        BasicDBObject nextDoc;

        if (cursor.hasNext()){

            nextDoc = (BasicDBObject) cursor.next();

            max = ((DBObject)nextDoc.get("value")).get("whole");
            color = ((DBObject)nextDoc.get("value")).get("color");
            names = ((DBObject)nextDoc.get("value")).get("names");

            try (FileWriter fileWriter = new FileWriter(resultsFile, true))
{
                fileWriter.write("Dominant Color : " + color + " covering "
+ max + "% of Image. Objects : ");
                if(names instanceof BasicDBList){
                    listNames = (BasicDBList) names;
                    for(int i=0; i<((listNames.size()) - 1); i++){
                        fileWriter.write((String) listNames.get(i) + ", ");
                    }
                    fileWriter.write((String)
listNames.get((listNames.size() - 1)) + "\n");
                }
                else{
                    fileWriter.write(names.toString() + "\n");
                }
            }

            while (cursor.hasNext()) {
                nextDoc = (BasicDBObject) cursor.next();

                cur_whole = ((DBObject)nextDoc.get("value")).get("whole");
                color = ((DBObject)nextDoc.get("value")).get("color");
                names = ((DBObject)nextDoc.get("value")).get("names");

                if((double)cur_whole == (double)max){
                    try (FileWriter fileWriter = new FileWriter(resultsFile,
true)) {
                        fileWriter.write("Dominant Color : " + color + "
covering " + max + "% of Image. Objects : ");
                        if(names instanceof BasicDBList){
                            listNames = (BasicDBList) names;
                            for(int i=0; i<((listNames.size())-1); i++){
                            fileWriter.write((String) listNames.get(i) + ",
");
                            }
                            fileWriter.write((String)
listNames.get(listNames.size() - 1) + "\n");
                        }
                        else{
                            fileWriter.write(names.toString() + "\n");
                        }
                    }
                }
                else{
                    break;
                }
            }
        }
    }

    /*
     * Define map and reduce functions for FirstMapReduce.
     * Replace data of "MapReduce" collection with results.
```

```
     * First MapReduce is called to extract root objects
     * of "KeyValue" collection.
     * root objects are NOT leafs and are NOT children of any other object.
     */

    private DBCollection FirstMapReduce(String inputColString, String
outputColString, DB db) throws FileNotFoundException, IOException{

        // input collection
        DBCollection input_collection = db.getCollection(inputColString);
        // output collection
        String mapReduceColString = outputColString;
        DBCollection result_collection;

        // Create map String to be used as map function.
        File mapFile = new File(pathOfFunctions + "Map1_FindFather.js");
        String map = GetFunctionString(mapFile);

        // Create reduce String to be used as reduce function.
        File reduceFile = new File(pathOfFunctions+"Reduce1_FindFather.js");
        String reduce = GetFunctionString(reduceFile);

        // Build MapReduce command. Results will REPLACE previous data of
output collection.
        MapReduceCommand cmd = new MapReduceCommand(input_collection, map,
reduce,
                        mapReduceColString,
MapReduceCommand.OutputType.REPLACE, null);
        // Call MapReduce. Input from "KeyValue" collection,
        // output to "MapReduce" collection,
        // using "Map1_FindFather" and "Reduce1_FindFather" functions.
        MapReduceOutput callMapReduce = input_collection.mapReduce(cmd);
        result_collection = callMapReduce.getOutputCollection();

        return result_collection;
    }

    /*
     * Define map and reduce functions for SecondMapReduce.
     * Merge results with data from "MapReduce" collection.
     * Second MapReduce is used to cross one layer of the trees at a time.
     * Each time objects of current layer submit their children
     * updating their "count" and "cur_view" fields.
     * If at least one child BUT NOT LEAF object has count > 0,
     * SecondMapReduce will be called, examining next layer of the trees.
     */

    private DBCollection SecondMapReduce(String col, DB db) throws
FileNotFoundException, IOException{

        // input collection same as output collection
        DBCollection input_collection = db.getCollection(col);
        String outputCol = col;
        DBCollection result_collection;
        // Create map String to be used as map function.
        File mapFile = new File(pathOfFunctions + "Map2_CrossTree.js");
        String map = GetFunctionString(mapFile);
        // Create reduce String to be used as reduce function.
        File reduceFile = new File(pathOfFunctions+"Reduce2_CrossTree.js");
        String reduce = GetFunctionString(reduceFile);

        // Build MapReduce command.
        // Results will be MERGED with previous data of output collection.
        MapReduceCommand cmd = new MapReduceCommand(input_collection, map,
reduce,
                        outputCol, MapReduceCommand.OutputType.MERGE, null);

        // Call MapReduce. Input from "MapReduce" collection,
```

```
        // output to "MapReduce" collection,
        // using "Map2_CrossTree" and "Reduce2_CrossTree" functions.
        result_collection =
input_collection.mapReduce(cmd).getOutputCollection();

        return (result_collection);
    }

    /*
     * Define map and reduce functions for FinalMapReduce.
     * Replace data of "MapReduce" collection with results.
     * Final MapReduce is called to calculate
     * the "whole" field of all leaf objects.
     */

    private DBCollection FinalMapReduce(String col, DB db) throws
FileNotFoundException, IOException{

        // input collection same as output collection
        DBCollection input_collection = db.getCollection(col);
        String outputColString = col;
        DBCollection result_collection;

        // Create map String to be used as map function.
        File mapFile = new File(pathOfFunctions + "Map3_CalcResult.js");
        String map = GetFunctionString(mapFile);

        // Create reduce String to be used as reduce function.
        File reduceFile = new File(pathOfFunctions+"Reduce3_CalcResult.js");
        String reduce = GetFunctionString(reduceFile);

        // Build MapReduce command. Results will REPLACE
        // previous data of output collection.
        MapReduceCommand cmd = new MapReduceCommand(input_collection, map,
reduce,
                    outputColString, MapReduceCommand.OutputType.REPLACE,
null);

        // Call MapReduce. Input from "MapReduce" collection,
        // output to "MapReduce" collection,
        // using "Map3_CalcResult" and "Reduce3_CalcResult" functions.
        result_collection =
input_collection.mapReduce(cmd).getOutputCollection();

        return (result_collection);
    }

    /*
     * Calls FirstMapReduce.
     * Input Collection is "KeyValue", Output Collection is "MapReduce".
     * Calls SecondMapReduce until all layers of trees are accessed.
     * Input Collection is "MapReduce", Output Collection is "MapReduce".
     * Results are MERGED.
     * Calls FinalMapReduce. Input Collection is "MapReduce",
     * Output Collection is "MapReduce". Results are REPLACED.
     * Calls QueryOutput to extract max values of "MapReduce" collection.
     * Results written to file with filePath = pathOfRes.
     */

    private void CallMapReduce(DB db) throws IOException{

        String inputColString = "KeyValue";
        String mapReduceColString = "MapReduce";

        DBCollection result_collection;

        result_collection = FirstMapReduce(inputColString,
mapReduceColString, db);
```

```
        // use repeat to determine if SecondMapReduce should be recalled.
        boolean repeat;

        // Call SecondMapReduce.
        result_collection = SecondMapReduce(mapReduceColString, db);
        repeat = Done(result_collection);
        while(repeat == true){
            result_collection = SecondMapReduce(mapReduceColString,db);
            repeat = Done(result_collection);
        }

        // Call FinalMapReduce.
        result_collection = FinalMapReduce(mapReduceColString, db);
        // Call QueryOutput.
        QueryOutput(result_collection);
    }

    /*
     * Create files to write results and duration.
     * Call CallMapReduce method.
     */

    public void MapReduce(String demo, String times) throws
UnknownHostException, IOException{
        long startTime, endTime, duration;
        WriteResult writeResult = new WriteResult();

        // Path of Project data.
        String bigPath =
"C:\\Users\\Administrator\\Documents\\NetBeansProjects\\";

        // Path of Map/Reduce functions.
        pathOfFunctions =
"C:\\Users\\Administrator\\Documents\\NetBeansProjects\\Project\\src\\MapRed
uce\\";

        // Path of MapReduce Data.
        String dataPath = bigPath + "MapReduceData\\";

        // Subpath to write results (answers) of MapReduce.
        String resultsSubPath = dataPath + "Results\\";

        File tempFolder = new File(resultsSubPath);

        // Check if "Results" folder exists.
        if(!tempFolder.exists())
            tempFolder.mkdir();

        resultsSubPath += "MapReduce\\";

        tempFolder = new File(resultsSubPath);

        // Check if "Results" folder exists.
        if(!tempFolder.exists())
            tempFolder.mkdir();

        // Build path to write results (answers) of MapReduce.
        pathOfRes = resultsSubPath + "demo" + demo + " application x" +
times + ".txt";

        // Check if "Duration" folder exists.
        String durationSubPath = bigPath + "Duration\\";
        tempFolder = new File(durationSubPath);

        if(!tempFolder.exists())
            tempFolder.mkdir();

        // Check if "Duration//MapReduce" folder exists.
```

```
        String durationQueryPath = bigPath + "Duration\\MapReduce\\";
        tempFolder = new File(durationQueryPath);

        if(!tempFolder.exists())
            tempFolder.mkdir();

        // Path to write the duration of the MapReduce call.
        String durationQueryFilePath;
        // File to write the duration of the MapReduce call.
        File durationFile;

        String dbName = "demo" + demo + "x" + times;

        MongoClient mongoClient = new MongoClient("localhost", 27017);
        DB db = mongoClient.getDB(dbName);

        startTime =  System.currentTimeMillis();

        CallMapReduce(db);

        endTime =  System.currentTimeMillis();
        duration = (endTime - startTime);

        durationQueryFilePath = durationQueryPath + "MapReduce demo" + demo
+ ".txt";
        durationFile = new File(durationQueryFilePath);

        writeResult.WriteResult(duration, times, durationFile);

        mongoClient.close();
    }
}
```

Map1_FindFather.js

```
// Purpose of map function is to find the root JSON documents.
// Only JSON documents that have children will have count = 1
// and is_child = 0.
// Children objects, nodes that are not root and leafs,
// will have count = 0 and is_child = 1.

function map1_FindFather(){

    // whole name = "real_name - Copy ... ".
    // For this application we want only "real_name".
    var whole_name = this.name;
    // Split the String at " " and create an array of subStrings.
    var name_of_strings = whole_name.split(" ");
    // Extract real_name.
    var real_name = name_of_strings[0];

    // "depth_x" is used to visit embedded JSON documents.
    // For JSON documents that are NOT leaf Objects.
    var depth_1 = this.Mpeg7.Description;
    if(depth_1){
        for(var i=0; i<depth_1.length; i++){
            var depth_2 = depth_1[i].MultimediaContent;
            if(depth_2){
                var depth_3 = depth_2.StructuredCollection;
                if(depth_3){
                    var depth_4 = depth_3.Collection;
                    if(depth_4){
                        for(var i=0; i<depth_4.length; i++){
                            var element = depth_4[i];
                            var element_id = element.id;
                            if(element_id == "Textures"){
                                var depth_5 = element.ContentCollection;
                                if(depth_5){
```

```
// Create an array, where children
// of examined JSON document will
// be stored.
var children = [];

for(var j=0; j<depth_5.length; j++){
    // Take value of "depth_5[j].
    // Content.Multimedia.MediaLocator.
    // MediaUri" where the building
    // (child) objects are stored.
    var values_string =
          depth_5[j].Content.Multimedia.
                    MediaLocator.MediaUri;
    // Remove \" from the String.
    var newString =
       values_string.replace(/\"/g, "");
    // Split the String at " "
    // and create an array of subStrings
    var array_of_strings =
                  newString.split(" ");

    // Extract name of child object.
    var only_name =
    array_of_strings[0].indexOf(".xml");
    var temp_name =
                array_of_strings[0].
                substring(0,only_name);

    // Extract size of child object.
    var only_num =
                   array_of_strings[1].
                        indexOf("%");
    var temp_size =
           parseInt(array_of_strings[1].
                  substring(0,only_num));
    temp_size /= 100;

    // Extract visible of child object.
    only_num = array_of_strings[2].
                            indexOf("%");
    var temp_visible =
           parseInt(array_of_strings[2].
                  substring(0,only_num));
    temp_visible /= 100;

    // Calculate view of child object
    // with only two decimal places.
    var temp_view = Math.round(
            (temp_size * temp_visible) *
             Math.pow(10,2)
                    ) / Math.pow(10,2);

    // Create value of child object.
    // To separate from root object
    // add field is_child = 1.
    var value = {
        cur_view : 0,
        count : 0,
        is_child : 1
    };
    // Emit child object.
    emit (temp_name, value);

    // Create cur_child.
    // It contains the name of the
    // child object and the temp_view.
    var cur_child = {
        name : temp_name,
```

```
                                            cur_view : temp_view
                                        };
                                        // Add cur_child to children array.
                                        children.push(cur_child);
                                    }
                                    // Create father object.
                                    // Contains an array of child objects,
                                    // is_child = 0, since it is a father,
                                    // and count = 1.
                                    // count = 1 is added to determine how
                                    // many root objects exist in
                                    // collection.
                                    // In the reduce phase objects
                                    // who are NOT root objects will be
                                    // assigned with count = 0.
                                    var allChildren = {
                                        child : children,
                                        is_child : 0,
                                        count : 1
                                    };
                                    // Emit father object.
                                    emit(real_name, allChildren);
                                }
                            }
                        }
                    }
                }
            }

    }
    // For JSON documents that are leaf Objects.
    else {
        // Extract normalizedForm of examined JSON document.
        var only_num = this.Mpeg7.normalizedForm.indexOf("%");
        var temp_norm =
parseInt(this.Mpeg7.normalizedForm.substring(0,only_num));

        // Extract r,g,b values of examined JSON document.
        var temp_r = parseInt(this.Mpeg7.color.red);
        var temp_g = parseInt(this.Mpeg7.color.green);
        var temp_b = parseInt(this.Mpeg7.color.blue);

        // Build value to be emitted.
        // Contains norma and r,g,b values. count = 0,
        // since examined document is leaf.
        var value = {
            cur_view : 0,
            norma : temp_norm,
            count : 0,
            color : {
                r : temp_r,
                g : temp_g,
                b : temp_b
            }
        };
        emit(real_name, value);
    }
}
```

Reduce1_FindFather.js

```
// Purpose of reduce function is to find root JSON documents.
// Adds up all "is_child" values. If root object all "is_child" will be 0.
// If child element at least one "is_child" will be 1.
// For root objects sum_is_child = 0, for NOT root objects sum_is_child > 0.
// Adds up all "count" values to determine number of occurrence of examined
// object. Since we only want root objects to have count > 0,
```

```
// to start crossing the trees, if sum_is_child > 0, reduceVal.count = 0.

function reduce1_FindFather (keys,values){

    reduceVal = {
        cur_view : 0,
        count : 0,
        is_child : 0
    };

    for(var i=0; i<values.length; i++){
        // Sum "is_child" values.
        if(values[i].is_child){
            reduceVal.is_child += values[i].is_child;
        }

        // Sum "cur_view" values.
        if(values[i].cur_view){
            reduceVal.cur_view += values[i].cur_view;
        }

        // Sum "count" values.
        if(values[i].count){
            reduceVal.count += values[i].count;
        }

        // If objects has children, add child array.
        if(values[i].child){
            reduceVal.child = values[i].child;
        }

        // If object is a leaf add "norma" and "color" attributes.
        if(values[i].norma){
            reduceVal.norma = values[i].norma;

            reduceVal.color = {
                r : 0,
                g : 0,
                b : 0
            };

            reduceVal.color.r = values[i].color.r;
            reduceVal.color.g = values[i].color.g;
            reduceVal.color.b = values[i].color.b;
        }
    }

    // Update reduceVal.cur_view to have only two decimal places.
    reduceVal.cur_view = Math.round((reduceVal.cur_view) *
                                            Math.pow(10,2))/Math.pow(10,2);

    // If examined object is child, reduceVal.count = 0.
    if(reduceVal.is_child != 0){
        reduceVal.count = 0;
    }

    return reduceVal;
}
```

Map2_CrossTree.js

```
// Purpose of map function is to cross tree.

function map2_CrossTree(){

    var hasChild = this.value.child;
    var norma = this.value.norma;
    var count = this.value.count;
```

```
    // If examined object is NOT a leaf, it has children, procceed.
    if(hasChild){
        // Array of child objects.
        var children = this.value.child;
        // If count > 0, occurrence of examined object > 0,
        // emit count times all child objects.
        if(count != 0){
            for(var j=0; j<children.length; j++){
                var child_name = children[j].name;

                var value = {
                    cur_view : 0

                };

                value.cur_view = children[j].cur_view * count;
                value.count = count;
                emit(child_name, value);
            }
        }
        // Also emit examined object, to maintain child array.
        var new_value = {
            cur_view : 0,
            count : 0
        };

        new_value.child = children;
        emit(this._id, new_value);

    }
    // If examined object is leaf, emit it.
    if(norma){
        emit(this._id, this.value);
    }
}
```

Reduce2_CrossTree.js

```
// Purpose of reduce function id to sum up values of emitted objects.

function reduce2_CrossTree (keys,values){

    reduceVal = {
        cur_view : 0,
        count : 0
    };

    for(var i=0; i<values.length; i++){
        // Sum "cur_view" values.
        if(values[i].cur_view){
            reduceVal.cur_view += values[i].cur_view;
        }

        // Sum "count" values.
        if(values[i].count){
            reduceVal.count += values[i].count;
        }

        // If objects has children, add child array.
        if(values[i].child){
            reduceVal.child = values[i].child;
        }

        // If object is a leaf add "norma" and "color" attributes.
        if(values[i].norma){
            reduceVal.norma = values[i].norma;
```

```
            reduceVal.color = {
                r : 0,
                g : 0,
                b : 0
            };

            reduceVal.color.r = values[i].color.r;
            reduceVal.color.g = values[i].color.g;
            reduceVal.color.b = values[i].color.b;
        }
    }

    return reduceVal;
}
```

Map3_CalcResult.js

```
// Purpose of map function is
// whole = cur_view * norma

function map3_CalcResult(){

    var norma = this.value.norma;

    // If examined object is a leaf, it has norma, procceed.
    if(norma){
        var new_el = {
            whole : 0,
            color : {
                r : 0,
                g : 0,
                b : 0
            }
        };

        new_el.names = this._id;

        // Calculate whole, with only two decimal places.
        new_el.whole = Math.round(
                           (this.value.cur_view * this.value.norma) * 10)/10;

        // Add color attribute.
        new_el.color.r = this.value.color.r;
        new_el.color.g = this.value.color.g;
        new_el.color.b = this.value.color.b;

        // id is the color of the examined JSON.
        var id = "red: " + this.value.color.r + " green: " +
                        this.value.color.g + " blue: " + this.value.color.b;

        // Emit leaf object.
        emit(id, new_el);
    }
}
```

Reduce3_CalcResult.js

```
// Purpose of reduce function is to return total_view of colors
// Each key represents a color found in the image.
// For all colors add up "whole" values and
// an array "names" of child Objects with this color.

function reduce3_CalcResult (keys,values){

    reduceVal = {
        whole : 0,
        color : {
```

```
            r : 0,
            g : 0,
            b : 0
        }
    };

    var allNames = [];

    for(var i=0; i<values.length; i++){
        allNames.push(values[i].names);
        reduceVal.whole += values[i].whole;

        reduceVal.color.r = values[i].color.r;
        reduceVal.color.g = values[i].color.g;
        reduceVal.color.b = values[i].color.b;
    }

    reduceVal.names = allNames;

    return reduceVal;
}
```

NoMapReduce

NoMapReduce.java

```
package MapReduce;

import com.mongodb.BasicDBList;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodb.DBObject;
import com.mongodb.MongoClient;
import common.WriteResult;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.net.UnknownHostException;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Queue;
import java.util.Set;
import java.util.concurrent.LinkedBlockingDeque;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

public class NoMapReduce {
    File resultsFile;
    String pathOfRes;
    String pathOfFunctions;

    HashMap<String, JSONObject> hm;
    HashMap<String, JSONObject> hmColor;
    HashMap<String, Double> hmSmall;
    Map sortedMap;
    LinkedBlockingDeque<JSONObject> myQueue;

    /*
     * Create resultsFile with path = "pathOfRes".
```

```
     * If resultFile exists, flush it.
     */

    private void CreateResultsFile () throws IOException{
        resultsFile = new File(pathOfRes);
        if(!resultsFile.exists()){
            resultsFile.createNewFile();
        }
        else{
            try (FileWriter fileWriter = new FileWriter(resultsFile)) {
                fileWriter.flush();
            }
        }
    }

    private void PrintHashTable(HashMap hashMap){
        // Get hashmap in Set interface to get key and value
        Set s = hashMap.entrySet();

        // Move next key and value of HashMap by iterator
        Iterator it = s.iterator();

        while(it.hasNext())
        {
            // key=value separator this by Map.Entry to get key and value
            Map.Entry m = (Map.Entry) it.next();

            // getKey is used to get key of HashMap
            String key = (String) m.getKey().toString();

            // getValue is used to get value of key in HashMap
            Object value=(Object)m.getValue();

            System.out.println("Key :"+key);
            System.out.println("value :"+value);
        }
    }

    /*
     * HashTable hmColor has key/value pairs with key
     * the names of colors found in picture
     * and value a JSON Object with fields "whole" and "names"
     * where "whole" is the participation of color in the picture
     * and "names" a JSON Array with the names of the leaf objects
     * that construct the color.
     */

    private void AddToHashTableColor(String colorString, String name, double
whole, JSONObject color) throws JSONException{

        JSONObject json = new JSONObject();
        double prev_whole, new_whole;

        if(hmColor.containsKey(colorString)){
            json = hmColor.get(colorString);

            prev_whole = json.getDouble("whole");
            new_whole = prev_whole + whole;
            json.put("whole", new_whole);

            json.append("names", name);
        }
        else{
            json.put("whole", whole);
            json.append("names", name);
            json.put("color", color);
        }
```

```
        hmColor.put(colorString, json);
    }

    /*
     * key/value pairs with key all colors of the picture
     * and value whole participation of the color in the picture.
     */

    private void AddToHashTableSmall(String colorString, String name, double
whole, JSONObject color) throws JSONException{

        double prev_whole;

        if(hmSmall.containsKey(colorString)){
            prev_whole = hmSmall.get(colorString);
            whole += prev_whole;
        }

        hmSmall.put(colorString, whole);
    }

    /*
     * HashTable hm has key/value pairs with key the name of root,
     * child and leaf objects found in image examined,
     * and value a JSON Object with fields "count", "is_child",
     * "cur_view", "visited" and "child".
     * root objects have count > 0, is_child = 0
     * and child a JSON Array with the children of root objects.
     * child objects have count = 0, is_child = 1
     * and child a JSON Array with the children of child objects.
     * leaf objects have count = 0, is_child = 1, norma and color the
     * normalized_form and color values of the leaf object respectively.
     */

    private void AddToHashTable(String name, String key, Object value)
throws JSONException{

        JSONObject json = new JSONObject();
        Object new_value;

        if(hm.containsKey(name)){
            json = hm.get(name);

            switch (key){
                case ("count") : {
                    if((json.has("is_child")) &&
                                        (json.get("is_child").equals(1))){
                        json.put(key, 0);
                    }
                    else{
                        if(json.has(key)){
                            new_value = (int) json.get(key) + (int) value;
                            json.put(key, new_value);
                        }
                        else{
                            json.put(key, value);
                        }
                    }
                    break;
                }
                case ("is_child") : {
                    if(value.equals(1)){
                        json.put("count", 0);
                        json.put(key,value);
                    }
                    else if(json.has(key)){
                        new_value = (int) json.get(key) + (int) value;
                        json.put(key, new_value);
```

```
                    }
                    else{
                        json.put(key, value);
                    }
                    break;
                }
                case ("cur_view") : {
                    if(json.has(key)){
                        new_value = Math.round(
                                    ((double) json.get(key) + (double) value)
                                                * Math.pow(10,2)
                                                        ) / Math.pow(10,2);
                        json.put(key, new_value);
                    }
                    else{
                        json.put(key, value);
                    }
                    break;
                }
                case ("visited") : {
                    if(!json.has(key)){
                        json.put(key,value);
                    }
                    break;
                }
                case ("child") : {
                    if(!json.has("visited")){
                        json.append(key, value);
                    }
                    break;
                }
                default : {
                    json.put(key, value);
                    break;
                }
            }
        }
        else{
            json.put(key, value);
        }
        hm.put(name, json);
    }

    /*
     * Root Objects of HashTable hm are added to Queue.
     * BFS Search will start from these Objects,
     * which serve as root objects of the trees to be parsed.
     */

    private void AddFathersToQueue() throws JSONException{

        // Get hashmap in Set interface to get key and value
        Set set = hm.entrySet();
        Iterator it = set.iterator();

        while(it.hasNext())
        {
            // key=value separator this by Map.Entry to get key and value
            Map.Entry m =(Map.Entry)it.next();

            // getKey is used to get key of HashMap
            String name = (String)m.getKey();

            // getValue is used to get value of key in HashMap
            JSONObject json = (JSONObject)m.getValue();

            if((json.has("child")) && (json.get("is_child").equals(0))){
                int count = (int) json.get("count");
```

```
                JSONArray children = json.getJSONArray("child");

                for(int i=0; i<children.length(); i++){
                    JSONObject child = children.getJSONObject(i);
                    String childName = child.getString("name");

                    JSONObject hmJSON = hm.get(childName);

                    if(hmJSON.has("norma")){
                        double child_cur_view = child.getDouble("cur_view");
                        child_cur_view *= count;
                        child_cur_view = Math.round(
                                         child_cur_view * Math.pow(10,2)
                                                        )/Math.pow(10,2);
                        child.put("cur_view", child_cur_view);
                        myQueue.addFirst(child);
                    }
                    else{
                        child.put("count", count);
                        myQueue.add(child);
                    }
                }
            }
        }
    }

    /*
     * Is called to extract root objects of "KeyValue" collection.
     * root objects are NOT leafs and are NOT children of any other object.
     */

    private void FindFather(String inputColString, DB db) throws
IOException, JSONException {
        // input collection
        DBCollection input_collection = db.getCollection(inputColString);

        DBCursor cursor = input_collection.find();
        DBObject nextDoc;

        JSONObject jDoc1, jDoc2, jDoc3, jDoc4, jDoc5, jDoc6, jDoc7;
        JSONArray jArr1, jArr2, jArr3;

        String values_string, whole_name, real_name, child_name, temp_norma;
        String[] array_of_strings;

        int index;
        double temp_size, temp_visible, cur_view, norma;
        JSONObject color = new JSONObject();

        while (cursor.hasNext()){

            nextDoc = cursor.next();

            jDoc1 = new JSONObject(nextDoc.toString());

            whole_name = (String) jDoc1.get("name");
            real_name = (whole_name.split(" "))[0];

            if(jDoc1.has("Mpeg7")){
                jDoc2 = jDoc1.getJSONObject("Mpeg7");
                if(jDoc2.has("Description")){
                    jArr1 = jDoc2.getJSONArray("Description");

                    for(int i=0; i<jArr1.length(); i++){
                        jDoc3 = jArr1.getJSONObject(i);

                        if(jDoc3.has("MultimediaContent")){
```

```
                        jDoc4=jDoc3.getJSONObject("MultimediaContent");
                        if(jDoc4.has("StructuredCollection")){
                            jDoc5 = (JSONObject)
                                      jDoc4.get("StructuredCollection");

                            jArr2 = jDoc5.getJSONArray("Collection");

                            for(int j=0; j<jArr2.length(); j++){
                                jDoc6 = jArr2.getJSONObject(j);

                                if(jDoc6.get("id").equals("Textures")){
                                    jArr3 = (JSONArray)
                                         jDoc6.get("ContentCollection");

                                    AddToHashTable(real_name,"count",1);
                                    AddToHashTable(real_name,
                                                        "is_child" , 0);

                                    for(int k=0; k<jArr3.length(); k++){
                                        jDoc7 = jArr3.getJSONObject(k);

                                        values_string = (String)
                                    ((JSONObject)((JSONObject)
                                    ((JSONObject) jDoc7.get("Content")).
                                get("Multimedia")).get("MediaLocator")).
                                                        get("MediaUri");
                                        values_string = values_string.
                                                       replace("\"", "");

                                        array_of_strings =values_string.
                                               split(" ");
                                        index = array_of_strings[0].
                                                        indexOf(".xml");
                                        child_name =array_of_strings[0].
                                                     substring(0,index);

                                        index = array_of_strings[1].
                                                           indexOf("%");
                                        temp_size = Double.parseDouble(
                                array_of_strings[1].substring(0,index));
                                        temp_size /= 100;

                                        index = array_of_strings[2].
                                                           indexOf("%");
                                        temp_visible = Double.
                                                          parseDouble(
                                array_of_strings[2].substring(0,index));
                                        temp_visible /= 100;

                                        cur_view = Math.round(
                                            (temp_size * temp_visible)
                                                * Math.pow(10,2)
                                                     ) / Math.pow(10,2);

                                        AddToHashTable(child_name,
                                                       "cur_view", 0.0);
                                        AddToHashTable(child_name,
                                                        "count", 0);
                                        AddToHashTable(child_name,
                                                      "is_child", 1);

                                        JSONObject childJSON = new
                                                          JSONObject();
                                        childJSON.put("name",
                                                           child_name);
                                        childJSON.put("cur_view",
                                                             cur_view);
```

```
                                        AddToHashTable(real_name,
                                                "child", childJSON);

                                    }
                                    AddToHashTable(real_name,
                                                "visited", 1);
                                }
                            }
                        }
                    }
                }
                else if(jDoc2.has("normalizedForm")){
                    temp_norma = (String) jDoc2.get("normalizedForm");
                    index = temp_norma.indexOf("%");

                    norma = Double.parseDouble(
                                        temp_norma.substring(0,index));
                    AddToHashTable(real_name, "norma", norma);

                    color = (JSONObject) jDoc2.get("color");
                    AddToHashTable(real_name, "color", color);
                }
            }
        }
    }

    /*
     * Is used to cross one layer of the trees at a time.
     * Each time objects of current layer submit their children
     * updating their "count" and "cur_view" fields.
     * If at least one child BUT NOT LEAF object has count > 0,
     * CrossTree will be called again, examining next layer of the trees.
     */

    private void CrossTree() throws JSONException{
        JSONObject jsonQueue, jsonHM;

        while(!myQueue.isEmpty()){
            jsonQueue = myQueue.poll();

            String name = jsonQueue.getString("name");
            jsonHM = hm.get(name);

            JSONObject newJsonHM = new JSONObject();
            newJsonHM = jsonHM;

            if(newJsonHM.has("child")){
                JSONArray children = newJsonHM.getJSONArray("child");
                int count = jsonQueue.getInt("count");

                for(int i=0; i<children.length(); i++){
                    JSONObject child = children.getJSONObject(i);
                    String childName = child.getString("name");

                    JSONObject newChild = new JSONObject();

                    newChild.put("name", childName);

                    JSONObject childJsonHM = hm.get(childName);

                    if(childJsonHM.has("norma")){
                        double child_cur_view = child.
                                        getDouble("cur_view") * count;
                        newChild.put("cur_view", child_cur_view);
                        myQueue.addFirst(newChild);
                    }
                    else{
```

```
                    newChild.put("count", count);
                    myQueue.add(newChild);
                }
            }
        }
        else{
            double prev_cur_view = newJsonHM.getDouble("cur_view");
            double new_cur_view = Math.round((
                    (double) jsonQueue.get("cur_view") + prev_cur_view)
                                    * Math.pow(10,2)
                                            ) / Math.pow(10,2);

            jsonHM.put("cur_view", new_cur_view);
        }
    }
}

/*
 * Is called to calculate the "whole" field
 * of all colors found in picture.
 */

private void CalcResult() throws JSONException{
    // Get hashmap in Set interface to get key and value
    Set set = hm.entrySet();
    Iterator it = set.iterator();

    while(it.hasNext())
    {
        // key=value separator this by Map.Entry to get key and value
        Map.Entry m =(Map.Entry)it.next();

        // getKey is used to get key of HashMap
        String name = (String)m.getKey();

        // getValue is used to get value of key in HashMap
        JSONObject json = (JSONObject)m.getValue();

        if(json.has("norma")){

            double whole = Math.round(
                  (json.getDouble("norma") * json.getDouble("cur_view"))
                                * Math.pow(10,2)
                                    ) / Math.pow(10,2);
            JSONObject color = (JSONObject) json.get("color");
            String  colorString = "red: " + color.getInt("red") + "
   green: " + color.getInt("green") + " blue: " + color.getInt("blue");
            AddToHashTableColor(colorString, name, whole, color);
            AddToHashTableSmall(colorString, name, whole, color);
        }
    }
}

/*
 * HashTable hmSmall is used as input to sort
 * colors in descending order
 * according to "whole" field found in the value of hmSmall.
 * Result written to HashTable sortedMap.
 */

private void SortResult(){
    List list = new LinkedList(hmSmall.entrySet());

    // sort list based on comparator
   Collections.sort(list, new Comparator() {
        public int compare(Object o1, Object o2) {
            return (((Comparable) ((Map.Entry)
      (o1)).getValue()).compareTo(((Map.Entry) (o2)).getValue())) * -1;
```

```
            }
        });

        // put sorted list into map again
         //LinkedHashMap make sure order in which keys were inserted
        for (Iterator it = list.iterator(); it.hasNext();) {
             Map.Entry entry = (Map.Entry) it.next();
             sortedMap.put(entry.getKey(), entry.getValue());
        }
    }

    /*
     * Use HashTable sortedMap to find dominant color/colors of picture.
     * Use HashTable hmColor to get names of leaf objects
     * that participate in dominant color/colors.
     * Write result to resultsFile.
     */

    private void ReturnResult() throws IOException, JSONException{
        Object cur_whole, max, names;
        BasicDBList listNames;

        CreateResultsFile();

        Set set = sortedMap.entrySet();
        Iterator it = set.iterator();

        if(it.hasNext()){
            Map.Entry m =(Map.Entry)it.next();

            // getKey is used to get key of HashMap
            String colorString = (String) m.getKey();

            max = (double) m.getValue();

            names = hmColor.get(colorString).getJSONArray("names");

            try (FileWriter fileWriter = new FileWriter(resultsFile, true)){
                fileWriter.write("Dominant Color : {" + colorString + "}
                                  covering " + max + "% of Image. Objects : ");
                if(names instanceof BasicDBList){
                    listNames = (BasicDBList) names;
                    for(int i=0; i<((listNames.size()) - 1); i++){
                        fileWriter.write((String) listNames.get(i) + ", ");
                    }
                    fileWriter.write((String)
                              listNames.get((listNames.size() - 1)) + "\n");
                }
                else{
                    fileWriter.write(names.toString() + "\n");
                }
            }

            while (it.hasNext()) {
                m =(Map.Entry)it.next();

                // getKey is used to get key of HashMap
                colorString = (String) m.getKey();

                cur_whole = (double) m.getValue();
                names = hmColor.get(colorString).getJSONArray("names");

                if((double)cur_whole == (double)max){
                    try (FileWriter fileWriter = new FileWriter(resultsFile,
                                                                  true)) {
                        fileWriter.write("Dominant Color : {" + colorString
                          + "} covering " + max + "% of Image. Objects : ");
                        if(names instanceof BasicDBList){
```

```
                            listNames = (BasicDBList) names;
                            for(int i=0; i<((listNames.size())-1); i++){
                            fileWriter.write((String)listNames.get(i)+", ");
                            }
                            fileWriter.write((String)
                                listNames.get(listNames.size() - 1) + "\n");
                        }
                        else{
                            fileWriter.write(names.toString() + "\n");
                        }
                    }
                }
                else{
                    break;
                }
            }
        }
    }

    /*
     * Input Collection is "KeyValue".
     * Calls FindFather, AddFathersToQueue, CrossTree,
     * SortResult and ReturnResult.
     */

    private void CallNoMapReduce(DB db) throws IOException, JSONException{
        String inputColString = "KeyValue";

        FindFather(inputColString, db);

        AddFathersToQueue();

        CrossTree();

        CalcResult();

        SortResult();

        ReturnResult();
    }

    /*
     * Create files to write results and duration.
     * Call CallMapReduce method.
     */

    public void NoMapReduce(String demo, String times) throws
UnknownHostException, IOException, JSONException{
        long startTime, endTime, duration;
        WriteResult writeResult = new WriteResult();

        // Path of Project data.
        String bigPath =
                  "C:\\Users\\Administrator\\Documents\\NetBeansProjects\\";

        // Path of Map/Reduce functions.
        pathOfFunctions =
                   "C:\\Users\\Administrator\\Documents\\NetBeansProjects\\
                                              Project\\src\\MapReduce\\";

        // Path of MapReduce Data.
        String dataPath = bigPath + "MapReduceData\\";

        // Subpath to write results (answers) of MapReduce.
        String resultsSubPath = dataPath + "Results\\";

        File tempFolder = new File(resultsSubPath);
```

```
        // Check if "Results" folder exists.
        if(!tempFolder.exists())
            tempFolder.mkdir();

        resultsSubPath += "NoMapReduce\\";

        tempFolder = new File(resultsSubPath);

        // Check if "Results" folder exists.
        if(!tempFolder.exists())
            tempFolder.mkdir();

        // Build path to write results (answers) of MapReduce.
        pathOfRes = resultsSubPath + "demo" + demo + " application x" +
                                                              times + ".txt";

        // Check if "Duration" folder exists.
        String durationSubPath = bigPath + "Duration\\";
        tempFolder = new File(durationSubPath);

        if(!tempFolder.exists())
            tempFolder.mkdir();

        // Check if "Duration//MapReduce" folder exists.
        String durationQueryPath = bigPath + "Duration\\NoMapReduce\\";
        tempFolder = new File(durationQueryPath);

        if(!tempFolder.exists())
            tempFolder.mkdir();

        // Path to write the duration of the MapReduce call.
        String durationQueryFilePath;
        // File to write the duration of the MapReduce call.
        File durationFile;

        String dbName = "demo" + demo + "x" + times;

        MongoClient mongoClient = new MongoClient("localhost", 27017);
        DB db = mongoClient.getDB(dbName);

        startTime =  System.currentTimeMillis();

        CallNoMapReduce(db);

        endTime =  System.currentTimeMillis();
        duration = (endTime - startTime);

        durationQueryFilePath = durationQueryPath + "NoMapReduce demo" +
                                                               demo + ".txt";
        durationFile = new File(durationQueryFilePath);

        writeResult.WriteResult(duration, times, durationFile);

        mongoClient.close();

        hm.clear();
        hmColor.clear();
        hmSmall.clear();
        sortedMap.clear();

    }

    public NoMapReduce(){
        hm = new HashMap<>();
        hmColor = new HashMap<>();
        hmSmall = new HashMap<>();
        sortedMap = new LinkedHashMap();
        myQueue = new LinkedBlockingDeque<>();
```

```
        }

}
```

Παράρτημα Ε:
Κώδικας Δημιουργίας και Αποθήκευσης Αρχείων – Συμπληρωματικός Κώδικας

CreatePathXML.java

```
package PathXML;

import com.mongodb.BasicDBObject;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import common.*;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import org.json.JSONException;

public class CreatePathXML {

    File mpeg7Folder;
    File x3dFolder;
    DBCollection collection;

    /*
     * Take all files from list and add content in key/value pairs with
     * key = "Mpeg7" or "X3D" and value the filepath
     * of the Mpeg7 or X3D description as a String.
     */

    private void AddJSONFilesToMongoDB(File[] listOfFiles) throws
IOException, FileNotFoundException, JSONException {

        String fileName, rawFileName;
        String key, value;

        for(int i=0; i<listOfFiles.length; i++){

            fileName = listOfFiles[i].getName();
            rawFileName = new ProcessFilename().RemoveSuffix(fileName);

            key = new ProcessFilename().GetSuffix(fileName);

            if(key.equalsIgnoreCase(".xml"))
                key = "Mpeg7";
            else if(key.equalsIgnoreCase(".x3d"))
                key = "X3D";
            value = listOfFiles[i].getPath();
```

```
            BasicDBObject doc = new BasicDBObject();
            doc.put("name",rawFileName);

            DBCursor cursor = collection.find(doc);
            BasicDBObject nextDoc;

            if (!cursor.hasNext()){

                BasicDBObject newDoc = new BasicDBObject();
                newDoc.put("name", rawFileName);
                collection.insert(newDoc);
            }

            cursor = collection.find(doc);
            nextDoc = (BasicDBObject) cursor.next();

            BasicDBObject newField = new BasicDBObject(key,value);
            BasicDBObject newDoc = new BasicDBObject("$set", newField);

            collection.update(doc, newDoc);
        }
    }

    /*
     * Take an xml folder and an x3d folder.
     * Add key/value pairs to JSON files in a json folder.
     * If JSON file doesn't exists create a unique "name" key/value pair,
     * where key = "name" and value = "filename".
     */

    public void CreateAllJSONFiles () throws IOException,
FileNotFoundException, JSONException{

        collection.drop();

        if (mpeg7Folder.exists() && x3dFolder.exists()){

            /*
             * Create list of files in xml and x3d directories
             * that have .xml and .x3d extentions.
             */

            File[] listOfxmlFiles = mpeg7Folder.listFiles(new
                                                    FilterXMLFiles());
            File[] listOfx3dFiles = x3dFolder.listFiles(new
                                                    FilterX3DFiles());

            AddJSONFilesToMongoDB(listOfxmlFiles);
            AddJSONFilesToMongoDB(listOfx3dFiles);

        }
    }

    /*
     * Constructor.
     */

    public CreatePathXML(String path, DB db){

        mpeg7Folder = new File(path + "mp7_out\\");
        x3dFolder = new File(path + "x3d\\");

        String colName = "PathXML";
        collection = db.getCollection(colName);
    }
}
```

CreateEmbeddedXML.java

```
package EmbeddedXML;

import com.mongodb.BasicDBObject;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import common.*;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Scanner;
import org.json.JSONException;

public class CreateEmbeddedXML {

    File mpeg7Folder;
    File x3dFolder;
    DBCollection collection;

    /*
     * Take all files from list and add content in key/value pairs with
     * key = "Mpeg7" or "X3D" and value the description of Mpeg7 and X3D
     * as a string.
     */

    private void AddJSONFilesToMongoDB(File[] listOfFiles) throws
IOException, FileNotFoundException, JSONException {

        String fileName, rawFileName;
        String key, value;

        for(int i=0; i<listOfFiles.length; i++){

            fileName = listOfFiles[i].getName();
            rawFileName = new ProcessFilename().RemoveSuffix(fileName);

            key = new ProcessFilename().GetSuffix(fileName);

            if(key.equalsIgnoreCase(".xml"))
                key = "Mpeg7";
            else if(key.equalsIgnoreCase(".x3d"))
                key = "X3D";

            Scanner xmlContent = new Scanner(listOfFiles[i]);
            String xmlString = xmlContent.useDelimiter("\\Z").next();

            value = xmlString;

            BasicDBObject doc = new BasicDBObject();
            doc.put("name",rawFileName);

            DBCursor cursor = collection.find(doc);
            BasicDBObject nextDoc;

            if (!cursor.hasNext()){

                BasicDBObject newDoc = new BasicDBObject();
                newDoc.put("name", rawFileName);
                collection.insert(newDoc);
            }

            cursor = collection.find(doc);
            nextDoc = (BasicDBObject) cursor.next();

            BasicDBObject newField = new BasicDBObject(key,value);
            BasicDBObject newDoc = new BasicDBObject("$set", newField);
```

```
            collection.update(doc, newDoc);
        }
    }

    /*
     * Take an xml folder and an x3d folder.
     * Add key/value pairs to JSON files in a json folder.
     * If JSON file doesn't exists create a unique "name" key/value pair,
     * where key = "name" and value = "filename".
     */

    public void CreateAllJSONFiles () throws IOException,
FileNotFoundException, JSONException{

        collection.drop();

        if (mpeg7Folder.exists() && x3dFolder.exists()){

            /*
             * Create list of files in xml and x3d directories
             * that have .xml and .x3d extentions.
             */

            File[] listOfxmlFiles = mpeg7Folder.listFiles(new
                                                    FilterXMLFiles());
            File[] listOfx3dFiles = x3dFolder.listFiles(new
                                                    FilterX3DFiles());

            AddJSONFilesToMongoDB(listOfxmlFiles);
            AddJSONFilesToMongoDB(listOfx3dFiles);

        }
    }

    /*
     * Constructor.
     */

    public CreateEmbeddedXML(String path, DB db){

        mpeg7Folder = new File(path + "mp7_out\\");
        x3dFolder = new File(path + "x3d\\");

        String colName = "EmbeddedXML";
        collection = db.getCollection(colName);
    }
}
```

CreateKeyValue.java

```
package KeyValue;

import com.mongodb.BasicDBObject;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodb.util.JSON;
import common.*;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Scanner;
import org.json.JSONException;
import org.json.JSONObject;
import org.json.XML;

public class CreateKeyValue {
```

```
    File mpeg7Folder;
    File x3dFolder;
    DBCollection collection;

    /*
     * Take all files from list and add content in key/value pairs with
     * key = "Mpeg7" or "X3D" and value the description of Mpeg7 and X3D
     * as JSON Object.
     */

    private void AddJSONFilesToMongoDB(File[] listOfFiles) throws
IOException, FileNotFoundException, JSONException{

        String fileName, rawFileName;
        String key;
        Object value;

        for(int i=0; i<listOfFiles.length; i++){

            fileName = listOfFiles[i].getName();
            rawFileName = new ProcessFilename().RemoveSuffix(fileName);

            key = new ProcessFilename().GetSuffix(fileName);

            if(key.equalsIgnoreCase(".xml"))
                key = "Mpeg7";
            else if(key.equalsIgnoreCase(".x3d"))
                key = "X3D";

            Scanner fileContent = new Scanner(listOfFiles[i]);
            String fileString = fileContent.useDelimiter("\\Z").next();

            /*
             * Convert the XML file in a valid JSON Object.
             */

            JSONObject file_json = XML.toJSONObject(fileString);

            value = JSON.parse(file_json.get(key).toString());

            BasicDBObject doc = new BasicDBObject();
            doc.put("name",rawFileName);

            DBCursor cursor = collection.find(doc);
            BasicDBObject nextDoc;

            if (!cursor.hasNext()){
                BasicDBObject newDoc = new BasicDBObject();
                newDoc.put("name", rawFileName);
                collection.insert(newDoc);
            }

            cursor = collection.find(doc);
            nextDoc = (BasicDBObject) cursor.next();

            BasicDBObject newField = new BasicDBObject(key,value);
            BasicDBObject newDoc = new BasicDBObject("$set", newField);

            collection.update(doc, newDoc);
        }
    }

    /*
     * Take an xml folder and an x3d folder.
     * Add key/value pairs to JSON files in a json folder.
     * If JSON file doesn't exists create a unique "name" key/value pair,
     * where key = "name" and value = "filename".
     */
```

```
    public void CreateAllJSONFiles () throws IOException,
FileNotFoundException, JSONException{

        collection.drop();

        if (mpeg7Folder.exists() && x3dFolder.exists()){

            File[] listOfmpeg7Files = mpeg7Folder.listFiles(new
                                                            FilterXMLFiles());
            File[] listOfx3dFiles = x3dFolder.listFiles(new
                                                            FilterX3DFiles());

            AddJSONFilesToMongoDB(listOfmpeg7Files);
            AddJSONFilesToMongoDB(listOfx3dFiles);
        }
    }

    /*
     * Constructor.
     */

    public CreateKeyValue(String path, DB db){

        mpeg7Folder = new File(path + "mp7_out\\");
        x3dFolder = new File(path + "x3d\\");

        String colName = "KeyValue";
        collection = db.getCollection(colName);

    }
}
```