Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

# Σχεδιασμός και Υλοποίηση Εργαλείου Προσομοίωσης Κατανεμημένων Συστημάτων Επεξεργασίας Συνεχών Ροών Δεδομένων Σε Πραγματικό Χρόνο

*Μελέτη, σχεδίαση και υλοποίηση*

---

Διπλωματικη Εργασια

των

**ΑΛΕΞΑΝΔΡΟΥ ΙΟΝΙΤΣΑ**
**ΕΜΜΑΝΟΥΗΛ ΕΜΜΑΝΟΥΗΛΙΔΗ**

**Επιβλέπων:** Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2024

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

# Σχεδιασμός και Υλοποίηση Εργαλείου Προσομοίωσης Κατανεμημένων Συστημάτων Επεξεργασίας Συνεχών Ροών Δεδομένων Σε Πραγματικό Χρόνο

*Μελέτη, σχεδίαση και υλοποίηση*

---

Διπλωματικη Εργασια

των

**ΑΛΕΞΑΝΔΡΟΥ ΙΟΝΙΤΣΑ**
**ΕΜΜΑΝΟΥΗΛ ΕΜΜΑΝΟΥΗΛΙΔΗ**

**Επιβλέπων:** Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 18η Οκτωβρίου 2024.

| *(Υπογραφή)* | *(Υπογραφή)* | *(Υπογραφή)* |
|---|---|---|
| ..... | ..... | ..... |
| Νεκτάριος Κοζύρης | Ιωάννης Κωνσταντίνου | Δημήτριος Τσουμάκος |
| Καθηγητής Ε.Μ.Π. | Αναπληρωτής Καθηγητής Π.Θ. | Αναπληρωτής Καθηγητής Ε.Μ.Π |

Αθήνα, Οκτώβριος 2024

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Copyright © – All rights reserved. Με την επιφύλαξη παντός δικαιώματος.
Αλέξανδρος Ιονίτσα, Εμμανουήλ Εμμανουηλίδης, 2024.

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.

Το περιεχόμενο αυτής της εργασίας δεν απηχεί απαραίτητα τις απόψεις του Τμήματος, του Επιβλέποντα, ή της επιτροπής που την ενέκρινε.

**ΔΗΛΩΣΗ ΜΗ ΛΟΓΟΚΛΟΠΗΣ ΚΑΙ ΑΝΑΛΗΨΗΣ ΠΡΟΣΩΠΙΚΗΣ ΕΥΘΥΝΗΣ**

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνουμε ενυπογράφως ότι είμαστε αποκλειστικοί συγγραφείς της παρούσας Πτυχιακής Εργασίας, για την ολοκλήρωση της οποίας κάθε βοήθεια είναι πλήρως αναγνωρισμένη και αναφέρεται λεπτομερώς στην εργασία αυτή. Έχουμε αναφέρει πλήρως και με σαφείς αναφορές, όλες τις πηγές χρήσης δεδομένων, απόψεων, θέσεων και προτάσεων, ιδεών και λεκτικών αναφορών, είτε κατά κυριολεξία είτε βάσει επιστημονικής παράφρασης. Αναλαμβάνουμε την προσωπική και ατομική ευθύνη ότι σε περίπτωση αποτυχίας στην υλοποίηση των ανωτέρω δηλωθέντων στοιχείων, είμαστε υπόλογοι έναντι λογοκλοπής, γεγονός που σημαίνει αποτυχία στην Πτυχιακή μας Εργασία και κατά συνέπεια αποτυχία απόκτησης του Τίτλου Σπουδών, πέραν των λοιπών συνεπειών του νόμου περί πνευματικών δικαιωμάτων. Δηλώνουμε, συνεπώς, ότι αυτή η Πτυχιακή Εργασία προετοιμάστηκε και ολοκληρώθηκε από εμάς προσωπικά και αποκλειστικά και ότι, αναλαμβάνουμε πλήρως όλες τις συνέπειες του νόμου στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μας ανήκει διότι είναι προϊόν λογοκλοπής άλλης πνευματικής ιδιοκτησίας.

*(Υπογραφές)*

| | |
|---|---|
| ..... | ..... |
| Αλέξανδρος Ιονίτσα | Εμμανουήλ Εμμανουηλίδης |
| Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π. | Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π. |

18 Οκτωβρίου 2024

# Περίληψη

Η επεξεργασία συνεχών ροών δεδομένων σε πραγματικό χρόνο (Stream Processing - SP) είναι μια εφαρμογή που τα τελευταία χρόνια έχει γίνει ιδιαίτερα δημοφιλής, χάρη στην ανάπτυξη των τεχνολογιών επεξεργασίας και αποθήκευσης δεδομένων μεγάλου όγκου (Big Data). Τεχνολογίες όπως τα Apache Storm, Flink, Spark Streaming και Kafka Streams παρέχουν την υποδομή για την κατανεμημένη επεξεργασία δεδομένων, προσφέροντας τη δυνατότητα παράλληλης εκτέλεσης και διαχείρισης μεγάλων ροών δεδομένων, που κατανέμονται σε υπολογιστικούς κόμβους σε διάφορα φυσικά ή εικονικά περιβάλλοντα.

Η απόδοση αυτών των συστημάτων επηρεάζεται από ποικίλους παράγοντες, οι οποίοι περιλαμβάνουν την κατανομή των δεδομένων στους διαθέσιμους υπολογιστικούς πόρους, την αρχιτεκτονική και τον τρόπο λειτουργίας αυτών των συστημάτων (true streaming vs micro-batching), το είδος των τελεστών (stateful vs stateless), τις τεχνικές παραθύρου (windowing techniques) και τις στρατηγικές κατανομής των κλειδιών, με στόχο τον καλύτερο διαμοιρασμό του φόρτου μεταξύ των κόμβων. Αυτοί οι παράγοντες μπορούν να οδηγήσουν σε σημαντικές διακυμάνσεις στην απόδοση του συστήματος, επηρεάζοντας την καθυστέρηση επεξεργασίας (latency), τον συνολικό χρόνο εκτέλεσης και τη δυνατότητα κλιμάκωσης.

Ένα από τα κύρια ερευνητικά ερωτήματα αφορά την επίδραση που έχουν διαφορετικές παραμετροποιήσεις αυτών των συστημάτων στην απόδοσή τους καθώς και το πώς μπορούν να εξεταστούν νέες υλοποιήσεις και τεχνικές στα εκάστοτε συστήματα. Ωστόσο, η προσομοίωση και η αξιολόγηση αυτών των συστημάτων με ταχύτητα, χαμηλό κόστος και χωρίς την ανάγκη φυσικής ανάπτυξης αποτελεί μια πρόκληση, καθιστώντας δύσκολη την έρευνα και τη δοκιμή διαφορετικών παραμετροποιήσεων.

Ο στόχος αυτής της διπλωματικής εργασίας είναι ο σχεδιασμός και η υλοποίηση ενός παραμετροποιήσιμου εργαλείου προσομοίωσης κατανεμημένων συστημάτων επεξεργασίας συνεχών ροών δεδομένων (Distributed Stream Processing Systems - DSPS). Το εργαλείο αυτό επιτρέπει τη διερεύνηση της επίδρασης διαφόρων παραμέτρων, εύκολα και γρήγορα. Μέσα από τη μελέτη διαφορετικών σεναρίων και τοπολογιών, η προσομοίωση προσφέρει μια εις βάθος κατανόηση του τρόπου λειτουργίας και βελτιστοποίησης των κατανεμημένων συστημάτων SP, επιτρέποντας τη λήψη αποφάσεων για τη βελτίωση της απόδοσης και της αποδοτικότητάς τους σε πραγματικές συνθήκες λειτουργίας.

## Λέξεις Κλειδιά

Επεξεργασία Συνεχών Ροών Δεδομένων, Κατανεμημένα Συστήματα, Προσομοίωση, Ανάλυση Επιδόσεων, SP, Big Data, Παράλληλη Επεξεργασία, Ισορροπία Φόρτου

# Abstract

Real-time stream processing (Stream Processing - SP) is an application that has become particularly popular in recent years, thanks to the development of technologies for processing and storing large volumes of data (Big Data). Technologies such as Apache Storm, Flink, Spark Streaming, and Kafka Streams provide the infrastructure for distributed data processing, offering the capability of parallel execution and management of large data streams distributed across computing nodes in various physical or virtual environments.

The performance of these systems is influenced by various factors, including the distribution of data across the available computational resources, the architecture and operational mode of these systems (true streaming vs. micro-batching), the type of operators used (stateful vs. stateless), windowing techniques, and key partitioning strategies, aiming for optimal load balancing across nodes. These factors can lead to significant fluctuations in system performance, affecting processing latency, overall execution time, and scalability.

One of the main research questions concerns the impact of different configurations on the performance of these systems, as well as how new implementations and techniques can be evaluated in each system. However, simulating and evaluating these systems efficiently, at low cost, and without the need for physical deployment presents a challenge, making it difficult to research and test different configurations.

The goal of this thesis is the design and implementation of a customizable simulation tool for Distributed Stream Processing Systems (DSPS). This tool allows the investigation of the impact of various parameters, such as data distribution, windowing techniques, the use of different operators, and load balancing techniques, on system performance. By studying different scenarios and topologies, the simulation offers an in-depth understanding of the operation and optimization of distributed SP systems, enabling decisions to improve their performance and efficiency in real-world operating conditions.

## Keywords

Real-time Stream Processing, Distributed Systems, Simulation, Performance Analysis, SP, Big Data, Parallel Processing, Load Balancing

*στους γονείς μας*

# Ευχαριστίες

Θα ήθελα καταρχήν να ευχαριστήσω τον καθηγητή κ. Νεκτάριο Κοζύρη, Καθηγητή Ε.Μ.Π., που μου έδωσε την ευκαιρία να εκπονήσω τη διπλωματική μου εργασία στο CS Lab. Επίσης ευχαριστώ ιδιαίτερα τον υποψήφιο διδάκτορα Νίκο Χαλβαντζή, ο οποίος υπήρξε βασικός πυλώνας της εκπόνησης αυτή της διπλωματικής εργασίας μέσω των εύστοχων συμβουλών του και την τροφή για σκέψη που υπήρξε για την διερεύνυνση πολλών πτυχών εμβάθυνσης στο αντίκειμενο αυτό. Επιπλέον, εκφράζω την ευγνωμοσύνη μου απέναντι στον συνεργάτη μου Μάνο με τον οποίο είχαμε μία άρτια συνεργασία, πιέζοντας ο ένας τον άλλον ώστε να γινόμαστε πάντα καλύτεροι. Τέλος, οφείλω να ευχαριστήσω γονείς και φίλους οι οποίοι με την ατέρμονη στήριξη τους με βοήθησαν στο να μην τα παρατήσω ποτέ και να αντιμετωπίσω κάθε δυσκολία με το κεφάλι ψηλά και πιο δυνατός από πριν. Δεν θα μπορούσα να το έκανα χωρίς εσάς.

Θα ήθελα, πρωτίστως, να ευχαριστήσω θερμά τον επιβλέποντα της διπλωματικής μου εργασίας, κ. Νεκτάριο Κοζύρη, Καθηγητή Ε.Μ.Π., που μου έδωσε την ευκαιρία να εκπονήσω τη διπλωματική μου εργασία στο CS Lab. Ευχαριστώ τον υποψήφιο διδάκτορα Νίκο Χαλβαντζή , που με ενέπνευσε να ασχοληθώ με ένα τόσο ενδιαφέρον και επίκαιρο θέμα, καθώς επίσης και για την καθοδήγηση, τη στήριξη και τις συμβουλές που μου παρείχε κατά την εκπόνηση της εργασίας, η ολοκλήρωση της οποίας δε θα ήταν εφικτή χωρίς τη συμβολή του. Ακόμη, θα ήθελα να εκφράσω βαθιά ευγνωμοσύνη προς τους φίλους μου Χρήστο, Κωνσταντίνο, Χρήστο και Άλεξ, για τις ανησυχίες, τα άγχη, τα όνειρα και τις φιλοδοξίες που μοιραστήκαμε όλα αυτά τα χρόνια. Τέλος, θα ήθελα να εκφράσω το μεγαλύτερο ευχαριστώ προς την οικογένειά μου, την αδερφή μου Σοφία, και ιδιαίτερα τους γονείς μου Αχιλλέα και Ειρήνη, για την απεριόριστη στήριξη, κατανόηση και αγάπη που μου έχουν προσφέρει σε κάθε στάδιο και απόφαση της ζωής μου. Αποτελούν πρότυπα για εμένα και τους αφιερώνω όσα έχω καταφέρει μέχρι σήμερα.

Αθήνα, Οκτώβριος 2024

*Αλέξανδρος Ιονίτσα*
*Εμμανουήλ Εμμανουηλίδης*

# Συνοδευτικό Υλικό

Η παρούσα διπλωματική εργασία συνοδεύεται από επιπλέον υλικό και τον πηγαίο κώδικα, τα οποία είναι διαθέσιμα στο αποθετήριο GitHub: Stream-Processing-Simulator-NTUA-Thesis[1].

Στο αποθετήριο βρίσκεται όλος ο κώδικας που αναπτύχθηκε στο πλαίσιο της εργασίας, καθώς και αναλυτικές οδηγίες για την εγκατάσταση και χρήση του λογισμικού. Επιπλέον, υπάρχουν πρόσθετα έγγραφα που παρέχουν λεπτομέρειες για την αρχιτεκτονική και τη λειτουργικότητα του συστήματος.

Εάν αντιμετωπίσετε οποιοδήποτε πρόβλημα ή έχετε ερωτήσεις σχετικά με το υλικό, μπορείτε να υποβάλετε ένα issue στο GitHub. Αυτό θα βοηθήσει στην παρακολούθηση και επίλυση των προβλημάτων.

[1] https://github.com/emsquared2/Stream-Processing-Simulator-NTUA-Thesis

# Περιεχόμενα

# Κατάλογος Σχημάτων

Κεφάλαιο 1

# Εισαγωγή

## 1.1 Γενικά

Η ταχύτατη επεξεργασία δεδομένων έχει καταστεί απαραίτητη στη σύγχρονη ψηφιακή εποχή, καθώς ο όγκος και η ταχύτητα των δεδομένων αυξάνονται με εκθετικούς ρυθμούς. Το μοντέλο που ανταποκρίνεται σε αυτές τις απαιτήσεις είναι τα Κατανεμημένα Συστήματα Επεξεργασίας Συνεχούς Ροής Δεδομένων (Distributed Stream Processing Systems – DSPSs). Σε αντίθεση με την παραδοσιακή επεξεργασία δεδομένων (batch processing), τα DSPSs επιτρέπουν την άμεση επεξεργασία δεδομένων σε πραγματικό χρόνο, καθιστώντας τα ιδανικά για εφαρμογές που απαιτούν χαμηλή καθυστέρηση και άμεση ανταπόκριση [1].

Η κύρια διαφορά μεταξύ της επεξεργασίας ροής (stream processing) και της επεξεργασίας κατά παρτίδες (batch processing) έγκειται στη φύση των δεδομένων και τον τρόπο διαχείρισής τους. Στα DSPSs, τα δεδομένα είναι συνεχώς εισερχόμενα και ατέρμονα (unbounded), απαιτώντας άμεση επεξεργασία καθώς φτάνουν στο σύστημα (compute on the go). Αντίθετα, στην παραδοσιακή επεξεργασία παρτίδων, τα δεδομένα είναι πεπερασμένα (bounded) και η επεξεργασία τους γίνεται περιοδικά, γεγονός που οδηγεί σε αυξημένη καθυστέρηση [2, 3]. Ενώ τα DSPSs στοχεύουν στη μείωση της καθυστέρησης για άμεση επεξεργασία δεδομένων σε πραγματικό χρόνο, τα συστήματα παρτίδων παρουσιάζουν μεγαλύτερη καθυστέρηση λόγω της ανάγκης συγκέντρωσης δεδομένων και επεξεργασίας τους σε καθορισμένα χρονικά διαστήματα. Αυτή η θεμελιώδης διαφορά έχει οδηγήσει στην ανάπτυξη εφαρμογών σε διάφορους τομείς, όπως πλέγματα αισθητήρων [4], IoT [5], χρηματοοικονομική ανάλυση [6], ανίχνευση απάτης [7, 8], ανάλυση κοινωνικών δικτύων [9] και άλλους.

Η αυξανόμενη ζήτηση για εφαρμογές που απαιτούν άμεση επεξεργασία, σε συνδυασμό με τις ανάγκες για μεγαλύτερη απόδοση και κλιμάκωση, οδήγησαν στην ανάπτυξη πληθώρας συστημάτων, όπως το Apache Storm, Apache Flink, Google Cloud Dataflow, Apache Samza, Amazon Kinesis [10, 11]. Κάθε σύστημα προσφέρει διαφορετικά χαρακτηριστικά, καθιστώντας την επιλογή της κατάλληλης πλατφόρμας για κάθε περίπτωση ιδιαίτερα απαιτητική.

Ακόμη, τα συστήματα επεξεργασίας ροών αντιμετωπίζουν διάφορες προκλήσεις όσον αφορά το φόρτο εργασίας τους, ο οποίος είναι δύσκολα προβλέψιμος εφόσον κατά κύριο λόγο σχετίζεται με εξωγενείς παράγοντες και όχι παράγοντες του συστήματος. Μια από τις κύριες προκλήσεις είναι ο υψηλός ρυθμός άφιξης δεδομένων, όπου τα δεδομένα εισέρχονται στο σύστημα με ταχύτατο ρυθμό, ασκώντας πίεση στην ικανότητά του να τα επεξεργάζε-

ται σε πραγματικό χρόνο. Επιπλέον, η ανισοκατανομή δεδομένων (data skew) προκαλεί προβλήματα, καθώς ορισμένα κλειδιά εμφανίζονται πολύ πιο συχνά σε σχέση με άλλα, οδηγώντας σε ανισορροπία στο φόρτο των κόμβων [12]. Τέλος, οι αιχμές δεδομένων (data spikes), που αναφέρονται σε ξαφνικές και απρόβλεπτες αυξήσεις στον όγκο των δεδομένων, μπορούν να υπερφορτώσουν το σύστημα, απαιτώντας άμεση προσαρμογή για να συνεχιστεί η ομαλή λειτουργία του.

Η ποικιλομορφία σε εφαρμογές, αρχιτεκτονικές και συνθήκες φόρτου εργασίας καθιστά δύσκολη τη μελέτη και την αξιολόγηση νέων προσεγγίσεων στα συστήματα επεξεργασίας ροών δεδομένων. Δεν υπάρχει κάποιο ενιαίο εργαλείο προσομοίωσης που να μπορεί να καλύψει γρήγορα και εύκολα όλες αυτές τις διαφορετικές περιπτώσεις. Στις περισσότερες περιπτώσεις, οι προσομοιώσεις απαιτούν την εγκατάσταση και τροποποίηση φυσικών συστημάτων, κάτι που είναι χρονοβόρο και κοστοβόρο, και περιορίζεται άμεσα από τις δυνατότητες του εκάστοτε συστήματος.

## 1.2 Αντικείμενο της Διπλωματικής

Αντικείμενο της παρούσας διπλωματικής εργασίας είναι ο σχεδιασμός και η υλοποίηση ενός εργαλείου προσομοίωσης για κατανεμημένα συστήματα επεξεργασίας συνεχών ροών δεδομένων σε πραγματικό χρόνο. Το εργαλείο αυτό αποσκοπεί στο να καλύψει το υπάρχον κενό, παρέχοντας τη δυνατότητα εύκολης, γρήγορης και ακριβούς προσομοίωσης διαφόρων σεναρίων λειτουργίας και συνθηκών φόρτου. Έτσι, θα επιτρέπει στους ερευνητές και στους μηχανικούς να αξιολογούν γρήγορα την απόδοση των συστημάτων σε διαφορετικές καταστάσεις, χωρίς την ανάγκη υλοποίησης φυσικών συστημάτων.

Επιπλέον, το εργαλείο προσομοίωσης έχει σχεδιαστεί με γνώμονα την εύκολη επεκτασιμότητα, διευκολύνοντας τη σταδιακή ενσωμάτωση νέων λειτουργιών και παραμέτρων. Αυτή η επεκτασιμότητα καθιστά το εργαλείο ιδανικό για την αξιολόγηση πιθανών βελτιώσεων στα συστήματα επεξεργασίας ροών, επιτρέποντας τη δοκιμή νέων τεχνικών και αλγορίθμων, καθώς και την ανάλυση της απόδοσής τους σε ποικίλες συνθήκες.

Με το προτεινόμενο εργαλείο προσομοίωσης, θα είναι δυνατή η δοκιμή διάφορων παραμέτρων, όπως η διανομή των κλειδιών, ο ρυθμός άφιξης δεδομένων, η κλιμάκωση του αριθμού των κόμβων, καθώς και η ανάλυση των δεδομένων σε διαφορετικά χρονικά παράθυρα, προσφέροντας μια ολοκληρωμένη μελέτη των συστημάτων επεξεργασίας ροών.

## 1.3 Συνεισφορά

Η παρούσα διπλωματική εργασία συμβάλλει ουσιαστικά στη μελέτη και ανάλυση των κατανεμημένων συστημάτων επεξεργασίας συνεχών ροών δεδομένων, μέσω της ανάπτυξης ενός ολοκληρωμένου εργαλείου προσομοίωσης που ανταποκρίνεται στις σύγχρονες ανάγκες. Το εργαλείο αυτό προσφέρει ένα παραμετροποιήσιμο περιβάλλον, όπου οι χρήστες μπορούν να προσαρμόζουν με ευελιξία κρίσιμες παραμέτρους.

Η ευκολία προσαρμογής και κλιμάκωσης καθιστά το εργαλείο ιδανικό για την προσομοίωση σύνθετων σεναρίων και τη μελέτη επιδόσεων των συστημάτων σε πραγματικό χρόνο,

χωρίς την ανάγκη φυσικής υλοποίησης. Επιπλέον, επιτρέπει την αξιολόγηση νέων αλγορίθμων και τεχνικών, παρέχοντας στους ερευνητές και τους μηχανικούς τη δυνατότητα να δοκιμάσουν και να βελτιώσουν την απόδοση των συστημάτων σε ποικίλες καταστάσεις. Συνεπώς, το εργαλείο συμβάλλει στη μελέτη των σύγχρονων προκλήσεων των συστημάτων επεξεργασίας ροών, προσφέροντας ένα πρακτικό και ισχυρό μέσο για την αξιολόγηση και βελτιστοποίηση διαφόρων προσεγγίσεων.

## 1.4 Δομή και Οργάνωση

Η εργασία είναι οργανωμένη σε 7 κεφάλαια:

- Στο **Κεφάλαιο 2**, αναλύεται το θεωρητικό υπόβαθρο των Συστημάτων Επεξεργασίας Συνεχών Ροών Δεδομένων σε Πραγματικό Χρόνο. Εξετάζονται τα χαρακτηριστικά, η δομή και η αρχιτεκτονική τους, ενώ παρουσιάζονται τεχνικές διαχείρισης των ροών δεδομένων και οι τεχνολογίες που υποστηρίζουν τη λειτουργία τους.

- Στο **Κεφάλαιο 3**, παρουσιάζονται οι σχετικές εργασίες που έχουν γίνει στον τομέα της προσομοίωσης κατανεμημένων συστημάτων επεξεργασίας συνεχών ροών δεδομένων.

- Στο **Κεφάλαιο 4**, περιγράφεται η ανάλυση και σχεδίαση του προτεινόμενου προσομοιωτή. Περιγράφονται τα βασικά συστατικά του συστήματος και εξηγείται ο τρόπος με τον οποίο σχεδιάστηκε για να καλύπτει τις απαιτήσεις της προσομοίωσης κατανεμημένων συστημάτων.

- Στο **Κεφάλαιο 5**, περιγράφεται η πειραματική διάταξη που χρησιμοποιήθηκε για την αξιολόγηση της απόδοσης του προσομοιωτή, με έμφαση στα σενάρια, τις τοπολογίες και τις παραμέτρους που τροποποιήθηκαν κατά τη διάρκεια των πειραμάτων.

- Στο **Κεφάλαιο 6**, παρουσιάζονται τα αποτελέσματα των πειραμάτων και η ανάλυσή τους. Συζητούνται τα συμπεράσματα που προκύπτουν από τις μετρήσεις της απόδοσης και αξιολογείται η λειτουργία του συστήματος σε διάφορες συνθήκες.

- Στο **Κεφάλαιο 7**, συνοψίζονται τα κύρια συμπεράσματα της εργασίας και προτείνονται κατευθύνσεις για μελλοντική έρευνα και ανάπτυξη.

# Μέρος Ι

# Θεωρητικό Μέρος

Κεφάλαιο 2

# Θεωρητικό Υπόβαθρο

Στο κεφάλαιο αυτό παρουσιάζεται αναλυτικά το θεωρητικό υπόβαθρο των Συστημάτων Επεξεργασίας Συνεχών Ροών Δεδομένων (Stream Processing Systems), με έμφαση στη σημασία τους, τα βασικά χαρακτηριστικά, την αρχιτεκτονική, και τις τεχνικές που χρησιμοποιούνται για τη διαχείριση και ανάλυση των ροών δεδομένων.

## 2.1 Εισαγωγή

Η επεξεργασία συνεχών ροών δεδομένων έχει εξελιχθεί σε έναν από τους πιο σημαντικούς τομείς της σύγχρονης πληροφορικής και ανάλυσης δεδομένων, καλύπτοντας την ανάγκη για άμεση ανάλυση τεράστιων όγκων δεδομένων σε πραγματικό χρόνο. Καθώς οι πηγές δεδομένων, όπως οι αισθητήρες IoT, τα κοινωνικά δίκτυα, οι πλατφόρμες streaming και οι χρηματοοικονομικές εφαρμογές, παράγουν συνεχώς αυξανόμενους όγκους πληροφοριών, η ανάγκη για συστήματα που επεξεργάζονται τα δεδομένα άμεσα γίνεται επιτακτική [13]. Σε αντίθεση με την παραδοσιακή επεξεργασία παρτίδων (batch processing), που απαιτεί τη συγκέντρωση και αποθήκευση των δεδομένων πριν την ανάλυση τους, τα συστήματα συνεχών ροών επιτρέπουν την επεξεργασία δεδομένων τη στιγμή που αυτά παράγονται [14].

Η κλιμάκωση αποτελεί μία από τις μεγαλύτερες προκλήσεις για τα σύγχρονα συστήματα επεξεργασίας συνεχών ροών [15]. Με την εκθετική αύξηση των δεδομένων, τα συστήματα πρέπει να είναι ικανά να διαχειρίζονται αυξημένα φορτία χωρίς απώλεια απόδοσης. Κατανεμημένα συστήματα, όπως το Apache Storm και το Apache Flink, είναι σχεδιασμένα για να κατανέμονται σε πολλαπλούς κόμβους επεξεργασίας (processing nodes), διασφαλίζοντας υψηλή ταχύτητα και αποτελεσματικότητα ακόμη και σε μεγάλη κλίμακα.

Επιπλέον, η αξιοπιστία και η ανοχή σε σφάλματα (fault tolerance) αποτελούν εξίσου κρίσιμες πτυχές των συστημάτων επεξεργασίας συνεχών ροών δεδομένων. Δεδομένης της αδιάλειπτης ροής δεδομένων, τα συστήματα πρέπει να είναι σε θέση συνεχίζουν να λειτουργούν απρόσκοπτα ακόμη και όταν προκύπτουν αστοχίες σε ορισμένα τμήματα του συστήματος ή του δικτύου. Συστήματα όπως το Apache Storm και το Apache Flink διαθέτουν μηχανισμούς που επιτρέπουν την αυτόματη επαναφορά σε περίπτωση αποτυχίας, διασφαλίζοντας την αδιάκοπη επεξεργασία των δεδομένων.

Πέρα από την άμεση επεξεργασία, τα σύγχρονα συστήματα προσφέρουν τη δυνατότητα διαχείρισης σύνθετων ροών δεδομένων, με υποστήριξη λειτουργιών όπως συγχωνεύσεις (joins), λειτουργίες παραθύρου (windowing) και ανίχνευση μοτίβων σε πραγματικό χρόνο.

Η σημασία αυτών των συστημάτων αυξάνεται διαρκώς και αναμένεται να συνεχίσει να αυξάνεται ακόμη περισσότερο καθώς οι τεχνολογίες δεδομένων συνεχίζουν να εξελίσσονται και να γίνονται πιο πολύπλοκες. Τα συστήματα επεξεργασίας συνεχών ροών παρέχουν στις επιχειρήσεις τη δυνατότητα ανάλυσης δεδομένων σε πραγματικό χρόνο, δίνοντάς τους ένα σημαντικό ανταγωνιστικό πλεονέκτημα σε έναν κόσμο όπου η ταχύτητα της πληροφορίας είναι καθοριστική για την επιτυχία.

## 2.2 Συστήματα Επεξεργασίας Συνεχών Ροών Δεδομένων

### 2.2.1 Κύρια Χαρακτηριστικά

Τα συστήματα επεξεργασίας συνεχών ροών δεδομένων χαρακτηρίζονται από τα ακόλουθα βασικά χαρακτηριστικά, τα οποία τα καθιστούν κατάλληλα για την ανάλυση δεδομένων σε πραγματικό χρόνο και την αντιμετώπιση των σύγχρονων αναγκών εφαρμογών μεγάλης κλίμακας:

- **Συνεχής Επεξεργασία**: Τα συστήματα επεξεργάζονται δεδομένα σε πραγματικό χρόνο, αναλύοντας τις ροές καθώς φτάνουν. Η επεξεργασία αυτή είναι αδιάκοπη [16], καθώς οι ροές δεδομένων ενδέχεται να είναι άπειρες ή να μην έχουν προκαθορισμένο τέλος. Αυτή η δυνατότητα επιτρέπει τη λήψη κρίσιμων αποφάσεων χωρίς καθυστερήσεις σε εφαρμογές όπως η παρακολούθηση συστημάτων ή η ανάλυση κοινωνικών μέσων [13], [14], [17].

- **Χαμηλή Καθυστέρηση**: Ένα από τα βασικά πλεονεκτήματα των συστημάτων αυτών είναι η δυνατότητα επεξεργασίας δεδομένων με εξαιρετικά χαμηλή καθυστέρηση, γνωστή και ως latency. Αυτό σημαίνει ότι μπορούν να ανταποκριθούν σε νέα δεδομένα σχεδόν άμεσα, καθιστώντας τα κατάλληλα για εφαρμογές όπου η άμεση ανταπόκριση είναι κρίσιμη, όπως χρηματοοικονομικές συναλλαγές, ανίχνευση απατών ή ανάλυση κινδύνου σε πραγματικό χρόνο [14], [18], [19], [13].

- **Κλιμακωσιμότητα**: Σχεδιάζονται με στόχο την οριζόντια επέκταση (horizontal scalability) [16], δηλαδή την ικανότητα να υποστηρίζουν την επεξεργασία μεγάλων όγκων δεδομένων με την προσθήκη περισσότερων κόμβων στο σύστημα. Η παράλληλη επεξεργασία σε κατανεμημένα συστήματα επιτρέπει την αποδοτική διαχείριση των αυξανόμενων απαιτήσεων για δεδομένα, ιδιαίτερα σε περιπτώσεις όπου οι ροές δεδομένων είναι συνεχείς και μεγάλης κλίμακας, όπως στις πλατφόρμες κοινωνικής δικτύωσης και στο Internet of Things (IoT) [17], [18], [19].

- **Ανοχή σε Σφάλματα**: Τα συστήματα επεξεργασίας συνεχών ροών δεδομένων ενσωματώνουν μηχανισμούς ανοχής σε σφάλματα [16], εξασφαλίζοντας την αδιάλειπτη λειτουργία τους, ακόμα και όταν προκύπτουν αποτυχίες στο υλικό ή στο λογισμικό. Αυτοί οι μηχανισμοί περιλαμβάνουν τεχνικές όπως τα στιγμιότυπα (checkpoints) και η επανεκτέλεση εργασιών, οι οποίες διασφαλίζουν ότι οι ροές δεδομένων συνεχίζουν να επεξεργάζονται σωστά και τα δεδομένα παραμένουν ακέραια [14], [18], [19].

- **Κατάσταση (Stateful Processing)**: Τα σύγχρονα συστήματα επεξεργασίας συνεχών ροών δεδομένων υποστηρίζουν επεξεργασία με κατάσταση, επιτρέποντας σε κάθε κόμβο του συστήματος να διατηρεί και να διαχειρίζεται πληροφορίες από προηγούμενες καταστάσεις. Αυτό είναι ιδιαίτερα χρήσιμο για σύνθετους υπολογισμούς, όπως η ανίχνευση προτύπων σε σειρές δεδομένων, η καταμέτρηση συμβάντων, ή η επεξεργασία δεδομένων με χρονικά παράθυρα (windowing) [13], [14], [18].

### 2.2.2 Τεχνολογίες

Τα συστήματα επεξεργασίας συνεχών ροών δεδομένων (stream processing systems) είναι εξειδικευμένες πλατφόρμες λογισμικού που έχουν σχεδιαστεί για να διαχειρίζονται και να επεξεργάζονται μεγάλα ποσά δεδομένων που ρέουν σε πραγματικό χρόνο. Σήμερα, υπάρχουν αρκετές δημοφιλείς τεχνολογίες που χρησιμοποιούνται ευρέως για την επεξεργασία συνεχών ροών, καθεμία από τις οποίες έχει μοναδικά χαρακτηριστικά και προσαρμοσμένες δυνατότητες που την καθιστούν κατάλληλη για συγκεκριμένες εφαρμογές.

#### Apache Storm

Το Apache Storm είναι ένα κατανεμημένο σύστημα επεξεργασίας ροών που σχεδιάστηκε για την επεξεργασία δεδομένων σε πραγματικό χρόνο με υψηλή ταχύτητα και χαμηλή καθυστέρηση. Αναπτύχθηκε αρχικά από την εταιρεία BackType [20] και αργότερα έγινε ανοιχτό από την Twitter το 2011. Το Storm είναι ικανό να επεξεργάζεται εκατομμύρια μηνύματα το δευτερόλεπτο ανά κόμβο και είναι γνωστό για την ευελιξία του στην υποστήριξη διαφόρων εφαρμογών σε κλίμακα [21].

Το Storm χρησιμοποιεί ένα μοντέλο κατανεμημένης υπολογιστικής βασισμένο σε γραφήματα κατευθυνόμενων ακυκλικών γραφημάτων (DAGs), όπου τα δεδομένα περνούν από διάφορους κόμβους επεξεργασίας (bolts) και παράγονται από πηγές δεδομένων (spouts). Το σύστημα είναι ιδιαίτερα αποτελεσματικό στην υποστήριξη εφαρμογών με απαιτήσεις σε πραγματικό χρόνο και παρέχει ανοχή σε σφάλματα μέσω μηχανισμών εγγύησης τουλάχιστον μιας φοράς (at-least-once semantics), εξασφαλίζοντας την επαναφορά μηνυμάτων σε περίπτωση αποτυχίας ενός κόμβου [22].

Το κύριο πλεονέκτημα του Storm είναι η ικανότητά του να χειρίζεται με χαμηλή καθυστέρηση μεγάλα δεδομένα, με χρήση μικρών κατανεμημένων cluster, καθιστώντας το ιδιαίτερα χρήσιμο για εφαρμογές ανάλυσης σε κοινωνικά δίκτυα, χρηματοοικονομικές εφαρμογές και ανίχνευση ανωμαλιών [21].

#### Apache Flink

Το Apache Flink είναι μια άλλη σημαντική τεχνολογία για την επεξεργασία συνεχών ροών, η οποία σχεδιάστηκε αρχικά για κατανεμημένη επεξεργασία δεδομένων και εξελίχθηκε σε έναν από τους πιο προηγμένους επεξεργαστές ροών σε πραγματικό χρόνο [23]. Ένα από τα κύρια χαρακτηριστικά του Flink είναι η δυνατότητα επεξεργασίας τόσο batch όσο και stream δεδομένων με το ίδιο API, γεγονός που το καθιστά ιδιαίτερα ευέλικτο για πολλαπλές περιπτώσεις χρήσης.

Το Flink είναι γνωστό για την παροχή εγγυήσεων ακριβώς μίας φοράς (exactly-once semantics), ένα σημαντικό χαρακτηριστικό για πολλές κρίσιμες εφαρμογές, όπως οι χρηματοοικονομικές συναλλαγές [23]. Έχει επίσης δυνατότητες παραθύρων χρόνου (windowing), οι οποίες επιτρέπουν την επεξεργασία δεδομένων μέσα σε συγκεκριμένα χρονικά πλαίσια. Αυτή η λειτουργία είναι ιδιαίτερα χρήσιμη για την επεξεργασία δεδομένων από IoT συσκευές, όπου τα δεδομένα πρέπει να αναλύονται εντός ορισμένων χρονικών παραθύρων.

Το Flink παρέχει επίσης εξαιρετική υποστήριξη για κατανεμημένες ροές δεδομένων και προσφέρει την ικανότητα ανάκτησης σε περίπτωση αποτυχίας, μέσω της χρήσης μηχανισμών checkpointing. Αυτή η δυνατότητα καθιστά το Flink ιδανικό για εφαρμογές μεγάλης κλίμακας που απαιτούν ανθεκτικότητα και αξιοπιστία. Επίσης, η χαμηλή καθυστέρηση σε συνδυασμό με την ισχυρή ανοχή σε σφάλματα το καθιστούν δημοφιλές σε μεγάλες επιχειρηματικές εφαρμογές [24].

#### Apache Kafka Streams

Το Apache Kafka Streams είναι ένα ακόμη σημαντικό εργαλείο επεξεργασίας συνεχών ροών δεδομένων. Το Kafka αρχικά σχεδιάστηκε ως ένα σύστημα διαμεσολάβησης μηνυμάτων (message broker) για τη συλλογή, αποθήκευση και διανομή δεδομένων σε πραγματικό χρόνο, αλλά με την επέκταση του μέσω της βιβλιοθήκης Kafka Streams, προσφέρει και δυνατότητες επεξεργασίας ροών [25, 26].

Ένα από τα βασικά χαρακτηριστικά του Kafka Streams είναι η ενσωμάτωσή του με το Kafka, καθιστώντας το ιδιαίτερα αποτελεσματικό για την επεξεργασία δεδομένων που έχουν ήδη παραχθεί από το Kafka. Επιπλέον, το Kafka Streams προσφέρει επίσης εγγυήσεις ακριβούς παράδοσης (exactly-once) και υποστηρίζει stateful επεξεργασία, κάτι που το καθιστά ιδανικό για εφαρμογές που απαιτούν την παρακολούθηση της κατάστασης [25, 26].

#### Apache Samza

Το Apache Samza είναι ένα σύστημα επεξεργασίας ροών που αναπτύχθηκε αρχικά από το LinkedIn και βασίζεται στο Apache Kafka και στο Apache Hadoop YARN για την εκτέλεση κατανεμημένων υπολογισμών σε ροές δεδομένων [27]. Το Samza είναι σχεδιασμένο για χαμηλή καθυστέρηση και υποστηρίζει τόσο την επεξεργασία σε πραγματικό χρόνο όσο και την επεξεργασία batch δεδομένων. Όπως το Kafka, το Samza παρέχει ανοχή σε σφάλματα και επεκτασιμότητα για την επεξεργασία μεγάλων ροών δεδομένων σε κατανεμημένα περιβάλλοντα.

Το κύριο πλεονέκτημα του Samza είναι η βαθιά ενσωμάτωσή του με το Kafka, που του επιτρέπει να αξιοποιεί τις δυνατότητες ανοχής σε σφάλματα και διαμερισματοποίησης (partitioning) που παρέχει το Kafka, ενώ χρησιμοποιεί το YARN για την κατανομή πόρων στο cluster [27]. Αυτό το καθιστά ιδιαίτερα κατάλληλο για εφαρμογές που απαιτούν κλιμακωτή επεξεργασία και αξιόπιστη εκτέλεση σε κατανεμημένα περιβάλλοντα.

### 2.2.3 Εφαρμογές

Τα συστήματα επεξεργασίας συνεχών ροών δεδομένων έχουν ευρεία χρήση σε πολλές βιομηχανίες και εφαρμογές που απαιτούν γρήγορη ανάλυση και επεξεργασία μεγάλων πο-

σοτήτων δεδομένων σε πραγματικό χρόνο. Οι πιο σημαντικές περιοχές εφαρμογών περιλαμβάνουν τα πλέγματα αισθητήρων και το Internet of Things (IoT), τις χρηματοοικονομικές αναλύσεις, τα κοινωνικά δίκτυα, την ασφάλεια δικτύων, καθώς και τον τομέα της υγείας και της επιστήμης των δεδομένων.

1. Πλέγματα Αισθητήρων και IoT

   Τα πλέγματα αισθητήρων και οι IoT συσκευές είναι εξαιρετικά σημαντικές εφαρμογές για τα συστήματα επεξεργασίας συνεχών ροών δεδομένων. Τα δεδομένα από αισθητήρες χρησιμοποιούνται σε διάφορους τομείς, όπως βιομηχανική αυτοματοποίηση, έξυπνες πόλεις, παρακολούθηση περιβαλλοντικών συνθηκών, γεωργία ακριβείας και παρακολούθηση κατασκευαστικών έργων. Οι αισθητήρες παράγουν τεράστιες ποσότητες δεδομένων συνεχούς ροής, οι οποίες απαιτούν άμεση ανάλυση και επεξεργασία για τη λήψη κρίσιμων αποφάσεων σε πραγματικό χρόνο.

   Συστήματα όπως το Apache Flink και το Apache Storm χρησιμοποιούνται ευρέως για τη διαχείριση δεδομένων IoT και αισθητήρων. Το Flink, με την υποστήριξή του για «παράθυρα χρόνου» (windowing), επιτρέπει την ανάλυση δεδομένων μέσα σε συγκεκριμένα χρονικά πλαίσια, κάτι που είναι ιδιαίτερα χρήσιμο για την παρακολούθηση αισθητήρων σε κατανεμημένα συστήματα [23]. Επίσης, η δυνατότητα ακριβούς παράδοσης (exactly-once semantics) του Flink το καθιστά ιδανικό για εφαρμογές που απαιτούν ακρίβεια, όπως η παρακολούθηση βιομηχανικών διαδικασιών και η αποφυγή σφαλμάτων στη συλλογή και επεξεργασία δεδομένων [24].

2. Χρηματοοικονομικές Αναλύσεις

   Στον χρηματοοικονομικό τομέα, οι επιχειρήσεις επεξεργάζονται τεράστιες ροές δεδομένων σε πραγματικό χρόνο για την παρακολούθηση αγορών, την εκτέλεση συναλλαγών και τη διαχείριση κινδύνου. Τα χρηματοοικονομικά συστήματα απαιτούν εξαιρετικά χαμηλή καθυστέρηση και υψηλή ακρίβεια, καθώς ακόμα και μικρές καθυστερήσεις μπορούν να προκαλέσουν σημαντικές ζημιές. Τα συστήματα όπως το Apache Kafka Streams και το Apache Flink χρησιμοποιούνται ευρέως στις χρηματοοικονομικές εφαρμογές λόγω της υποστήριξης εγγυήσεων ακριβούς παράδοσης (exactly-once semantics) και της δυνατότητας παρακολούθησης stateful δεδομένων [25].

   Χρηματοοικονομικές εταιρείες χρησιμοποιούν αυτά τα συστήματα για ανάλυση σε πραγματικό χρόνο των χρηματιστηριακών τιμών, διαχείριση χαρτοφυλακίων, ανάλυση κινδύνου, ανίχνευση απάτης και διαχείριση συναλλαγών υψηλής συχνότητας (high-frequency trading). Για παράδειγμα, το Apache Flink χρησιμοποιείται σε εφαρμογές trading για την ταχεία ανάλυση χρηματοοικονομικών δεδομένων, ενώ το Kafka Streams είναι ιδανικό για την επεξεργασία δεδομένων που παράγονται από αγορές και συναλλαγές σε πραγματικό χρόνο [23].

3. Κοινωνικά Δίκτυα

   Τα κοινωνικά δίκτυα είναι πηγές τεράστιων ποσοτήτων δεδομένων σε πραγματικό χρόνο, όπως tweets, δημοσιεύσεις, likes και shares. Τα δεδομένα αυτά χρειάζονται επεξεργασία άμεσα για την ανάλυση της συμπεριφοράς των χρηστών, την κατανόηση τάσεων,

τη στόχευση διαφημίσεων και την ανίχνευση κακόβουλης δραστηριότητας. Το Apache Storm, που χρησιμοποιείται από το Twitter, αποτελεί χαρακτηριστικό παράδειγμα εφαρμογής επεξεργασίας συνεχών ροών δεδομένων. Το Storm είναι ικανό να επεξεργάζεται εκατομμύρια μηνύματα το δευτερόλεπτο, καθιστώντας το ιδανικό για την ανάλυση των δεδομένων των κοινωνικών δικτύων [21].

Τα συστήματα συνεχούς ροής επιτρέπουν επίσης την ανίχνευση και ανάλυση συναισθημάτων (sentiment analysis), όπου οι εταιρείες μπορούν να αναλύσουν τη στάση των χρηστών απέναντι σε προϊόντα ή υπηρεσίες σε πραγματικό χρόνο, χρησιμοποιώντας τεχνολογίες όπως το Apache Flink. Τα δεδομένα μπορούν να συγχωνεύονται από διαφορετικές πηγές και να υποβάλλονται σε σύνθετη ανάλυση για να αποκαλυφθούν οι τάσεις και οι προτιμήσεις των χρηστών [23].

4. Ανίχνευση Ανωμαλιών και Ασφάλεια Δικτύων

   Η ασφάλεια δικτύων αποτελεί μια ακόμη κρίσιμη εφαρμογή των συστημάτων επεξεργασίας συνεχών ροών δεδομένων. Τα δεδομένα από firewalls, συστήματα παρακολούθησης δικτύου και log files παράγονται συνεχώς και πρέπει να αναλύονται σε πραγματικό χρόνο για την ανίχνευση απειλών και παραβιάσεων ασφαλείας. Το Apache Flink και το Apache Kafka χρησιμοποιούνται για την ανίχνευση ανωμαλιών σε πραγματικό χρόνο, επιτρέποντας στα συστήματα να εντοπίζουν ύποπτες δραστηριότητες και να αντιδρούν άμεσα [25].

   Η ανίχνευση ανωμαλιών βασίζεται στην παρακολούθηση συμπεριφορών και τη σύγκριση των δεδομένων με ιστορικά πρότυπα, ώστε να αναγνωρίζονται αποκλίσεις που ενδέχεται να αποτελούν ένδειξη κυβερνοεπιθέσεων ή άλλων απειλών. Συστήματα όπως το Apache Samza, το οποίο είναι σχεδιασμένο για χαμηλή καθυστέρηση και ανθεκτικότητα, είναι ιδανικά για εφαρμογές ασφαλείας που απαιτούν αξιόπιστη ανάλυση δεδομένων και επεκτασιμότητα [27].

5. Υγειονομικές Εφαρμογές

   Τα συστήματα υγειονομικής περίθαλψης παράγουν τεράστιες ροές δεδομένων από ιατρικές συσκευές, αρχεία ασθενών και συστήματα παρακολούθησης. Η επεξεργασία αυτών των δεδομένων σε πραγματικό χρόνο είναι απαραίτητη για την παρακολούθηση της υγείας των ασθενών, την άμεση διάγνωση και την παρέμβαση [28]. Τα συστήματα όπως το Apache Flink χρησιμοποιούνται για την ανάλυση δεδομένων υγείας, όπως οι μετρήσεις από ιατρικές συσκευές σε νοσοκομεία και κλινικές, επιτρέποντας την άμεση ανίχνευση προβλημάτων υγείας [23].

   Επίσης, οι ροές δεδομένων από συστήματα IoT χρησιμοποιούνται για την παρακολούθηση της υγείας ασθενών εκτός νοσοκομείου, όπως με συσκευές μέτρησης καρδιακών παλμών ή επίπεδων γλυκόζης. Οι εφαρμογές αυτές επιτρέπουν την πρόληψη κρίσεων υγείας μέσω της ανάλυσης δεδομένων σε πραγματικό χρόνο και τη λήψη μέτρων πριν εξελιχθεί μια κατάσταση σε κρίσιμη.

## 2.3 Βασικές Έννοιες

### 2.3.1 Δεδομένα και Ροές (Data and Streams)

Σε ένα σύστημα επεξεργασίας συνεχών ροών, η ροή (stream) αποτελεί το θεμελιώδες στοιχείο της αρχιτεκτονικής του. Αναπαριστά ένα ατελείωτο, συνεχώς εξελισσόμενο σύνολο δεδομένων που ανανεώνεται δυναμικά με την πάροδο του χρόνου. Είναι μια διατεταγμένη, επαναλήψιμη και ανθεκτική σε σφάλματα ακολουθία αμετάβλητων εγγραφών, αποθηκευμένων ως ζεύγη κλειδιού-τιμής (key-value pairs) [29].

### 2.3.2 Πηγές και Αποδέκτες (Sources and Sinks)

Οι πηγές και οι αποδέκτες αποτελούν τις βασικές εισόδους και εξόδους στα συστήματα επεξεργασίας συνεχών ροών, καθορίζοντας από πού προέρχονται τα δεδομένα και πού καταλήγουν μετά την επεξεργασία τους.

- **Πηγές (Sources)**: Οι πηγές είναι οι δημιουργοί των δεδομένων που τροφοδοτούν το σύστημα επεξεργασίας συνεχών ροών. Μπορεί να περιλαμβάνουν αισθητήρες IoT, αρχεία καταγραφής συστημάτων (logs), ροές μηνυμάτων από πλατφόρμες όπως το Apache Kafka, καθώς και δεδομένα από βάσεις δεδομένων ή συστήματα σε πραγματικό χρόνο. Οι πηγές αποτελούν τον πρώτο κρίκο της αλυσίδας επεξεργασίας, καθώς τροφοδοτούν το σύστημα με τα ακατέργαστα δεδομένα προς επεξεργασία [13].

- **Καταβόθρες (Sinks)**: Οι καταβόθρες (ή αποδέκτες αλλιώς), είναι τα συστήματα που καταναλώνουν τα επεξεργασμένα δεδομένα και αξιοποιούν τα αποτελέσματα που παράγει το σύστημα επεξεργασίας. Μπορεί να είναι εφαρμογές ελέγχου, συστήματα αναφορών και οπτικοποίησης, βάσεις δεδομένων, αποθηκευτικά μέσα ή άλλα συστήματα ανάλυσης που βασίζονται σε δεδομένα σε πραγματικό χρόνο. Οι καταβόθρες διασφαλίζουν την αξιοποίηση των αποτελεσμάτων της επεξεργασίας σε ποικίλες εφαρμογές, είτε για λήψη αποφάσεων είτε για περαιτέρω ανάλυση [13].

### 2.3.3 Επεξεργασία (Processing)

Η επεξεργασία σε συστήματα ροής δεδομένων αναφέρεται στη συνεχή ανάλυση και μετασχηματισμό δεδομένων καθώς αυτά φτάνουν στο σύστημα, χωρίς καθυστέρηση για την ολοκλήρωση ενός συνόλου. Τα δεδομένα, γνωστά και ως πλειάδες (tuples), δηλαδή ζεύγη κλειδιού-τιμής, προωθούνται στους κατάλληλους μηχανισμούς της εφαρμογής για επεξεργασία σε πραγματικό χρόνο [13]. Μια βασική προσέγγιση στην επεξεργασία ροών είναι η χρονικά-αγνωστική επεξεργασία (time-agnostic), στην οποία ο χρόνος δεν παίζει ουσιαστικό ρόλο, καθώς η επεξεργασία εξαρτάται από την άφιξη των δεδομένων και όχι από χρονικά όρια. Οι περισσότερες πλατφόρμες συνεχούς ροής υποστηρίζουν αυτήν την προσέγγιση εγγενώς, με μικρές διαφορές στις εγγυήσεις συνοχής που προσφέρουν [30]. Μια άλλη προσέγγιση είναι η παραθυροποίηση (windowing), όπου τα δεδομένα διασπώνται σε χρονικά περιορισμένα τμήματα, ώστε να είναι δυνατή η επεξεργασία τους σε διακριτά χρονικά παράθυρα [30].

Τέλος, υπάρχει και η τεχνική της μικρο-ομαδοποίησης (micro-batching), όπου τα δεδομένα οργανώνονται σε μικρότερες, διαχειρίσιμες παρτίδες, συνδυάζοντας την επεξεργασία παρτίδας και ροής. Μια γνωστή υλοποίηση αυτής της τεχνικής είναι οι D-Streams [31], οι οποίες αντιμετωπίζουν τα δεδομένα ροής ως μια σειρά μικρών παρτίδων που εκτελούνται ανά τακτά χρονικά διαστήματα.

### 2.3.4 Τελεστές (Operators)

Οι τελεστές αποτελούν τα βασικά λειτουργικά στοιχεία ενός συστήματος επεξεργασίας ροών. Κάθε τελεστής εκτελεί υπολογισμούς ή μετασχηματισμούς στα εισερχόμενα δεδομένα και μπορεί να είναι είτε stateless είτε stateful. Οι stateless τελεστές επεξεργάζονται κάθε δεδομένο ανεξάρτητα, ενώ οι stateful τελεστές διατηρούν την κατάσταση των προηγούμενων γεγονότων για πιο σύνθετους υπολογισμούς [13], [14]. Οι τελεστές συνδέονται μέσω ροών, σχηματίζοντας ένα γράφημα δεδομένων, γνωστό ως κατευθυνόμενος ακυκλικός γράφος (DAG) [13].

### 2.3.5 Ροή Δεδομένων και Αρχιτεκτονική

Η ροή δεδομένων σε ένα σύστημα επεξεργασίας συνεχών ροών ακολουθεί το μοντέλο του Κατευθυνόμενου Ακυκλικού Γράφου (Directed Acyclic Graph, DAG), στο οποίο οι κόμβοι (nodes) αντιπροσωπεύουν τελεστές και οι ακμές (edges) αντιπροσωπεύουν τις ροές δεδομένων μεταξύ τους [14]. Αυτή η αρχιτεκτονική επιτρέπει την παράλληλη επεξεργασία δεδομένων, εξασφαλίζοντας χαμηλή καθυστέρηση και υψηλή κλιμακωσιμότητα σε κατανεμημένα συστήματα [13].

### 2.3.6 Μετρικές (Metrics)

#### Καθυστέρηση (Latency)

Η καθυστέρηση (latency) εκφράζει τον χρόνο που απαιτείται για την επεξεργασία ενός γεγονότος. Ουσιαστικά, πρόκειται για το χρονικό διάστημα από τη λήψη ενός γεγονότος έως την εμφάνιση του αποτελέσματος της επεξεργασίας αυτού του γεγονότος στην έξοδο. Στα συστήματα ροής δεδομένων, η καθυστέρηση μετράται σε μονάδες χρόνου, όπως χιλιοστά του δευτερολέπτου, και μπορεί να μας ενδιαφέρει η μέση καθυστέρηση ή η μέγιστη καθυστέρηση. Χαμηλή καθυστέρηση είναι σημαντική για εφαρμογές πραγματικού χρόνου [14].

#### Ρυθμός Επεξεργασίας (Throughput)

Ο ρυθμός επεξεργασίας (throughput) μετρά την ικανότητα επεξεργασίας του συστήματος, δηλαδή πόσα γεγονότα μπορεί να επεξεργαστεί το σύστημα ανά μονάδα χρόνου. Ενώ θέλουμε η καθυστέρηση να είναι όσο το δυνατόν χαμηλότερη, συνήθως επιθυμούμε ο ρυθμός επεξεργασίας να είναι όσο το δυνατόν υψηλότερος [14].

Στα συστήματα ροής δεδομένων, πρέπει να εξασφαλίσουμε ότι το σύστημα μπορεί να αντέξει τον μέγιστο αναμενόμενο ρυθμό εισερχόμενων γεγονότων. Όταν ο ρυθμός εισόδου ξεπερνά την ικανότητα του συστήματος, τότε πρέπει να γίνεται προσωρινή αποθήκευση των γεγονότων, κάτι που ονομάζεται backpressure. Σε αυτή την περίπτωση, το σύστημα έχει

φτάσει τον μέγιστο ρυθμό επεξεργασίας του, και η περαιτέρω αύξηση του ρυθμού εισόδου θα οδηγήσει μόνο σε αύξηση της καθυστέρησης. Εάν η ροή δεδομένων συνεχιστεί με υψηλότερο ρυθμό από αυτόν που μπορεί να διαχειριστεί το σύστημα, οι προσωρινές αποθηκεύσεις ενδέχεται να μην επαρκούν, και υπάρχει κίνδυνος απώλειας δεδομένων [14].

#### Σχέση Μεταξύ Καθυστέρησης και Ρυθμού Επεξεργασίας

Είναι σημαντικό να σημειωθεί ότι η καθυστέρηση και ο ρυθμός επεξεργασίας δεν είναι ανεξάρτητες μετρικές. Αν τα γεγονότα καθυστερούν να διέλθουν από τον αγωγό επεξεργασίας, τότε είναι δύσκολο να επιτευχθεί υψηλός ρυθμός επεξεργασίας. Αντίστοιχα, αν η ικανότητα του συστήματος είναι περιορισμένη, τα γεγονότα θα πρέπει να περιμένουν πριν επεξεργαστούν, με αποτέλεσμα την αύξηση της καθυστέρησης [14].

## 2.4 Αρχιτεκτονική

### 2.4.1 Γενική Αρχιτεκτονική

Τα συστήματα επεξεργασίας συνεχών ροών βασίζονται σε μια αρχιτεκτονική που επιτρέπει την επεξεργασία δεδομένων σε πραγματικό χρόνο, χρησιμοποιώντας κατανεμημένους πόρους και παραλληλισμό. Η γενική αρχιτεκτονική περιλαμβάνει τα ακόλουθα στοιχεία [13, 14]:

1. **Πηγές Δεδομένων (Data Sources)**: Αποτελούν την είσοδο του συστήματος, παρέχοντας τις ροές δεδομένων από διάφορες πηγές.
2. **Κόμβοι Επεξεργασίας (Processing Nodes)**: Οι τελεστές που εκτελούν υπολογισμούς και μετασχηματισμούς πάνω στα δεδομένα.
3. **Ροές Δεδομένων (Data Streams)**: Τα κανάλια μέσω των οποίων τα δεδομένα μεταφέρονται μεταξύ των τελεστών.
4. **Αποδέκτες Δεδομένων (Data Sinks)**: Το τελικό σημείο όπου τα επεξεργασμένα δεδομένα αποθηκεύονται ή χρησιμοποιούνται.

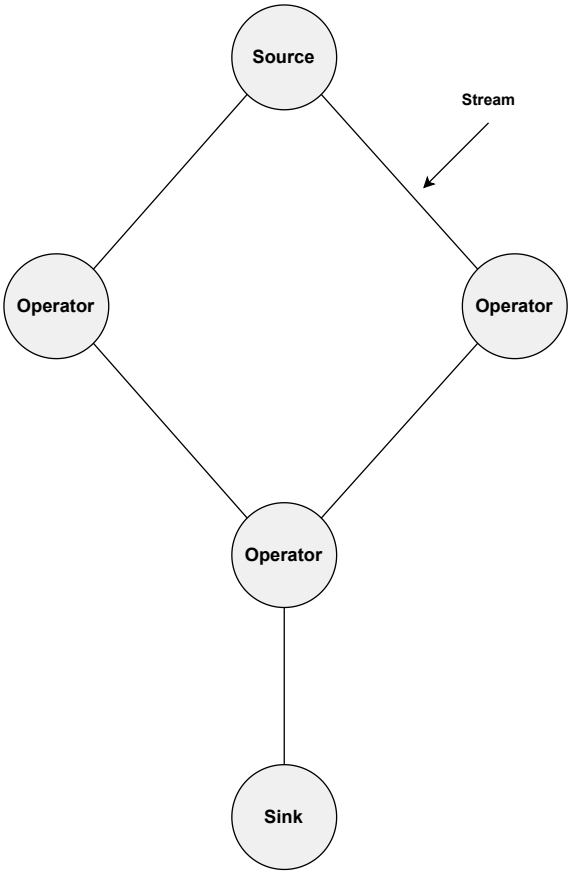
### 2.4.2 Ροή Δεδομένων

Η ροή των δεδομένων σε ένα σύστημα επεξεργασίας συνεχών ροών ακολουθεί τα εξής βήματα [14]:

- Τα δεδομένα παράγονται από τις πηγές και εισάγονται στο σύστημα ως ροές.
- Οι ροές δεδομένων κατευθύνονται μέσω των ακμών στους τελεστές, όπου υφίστανται επεξεργασία.
- Οι τελεστές μπορεί να μετασχηματίσουν, να φιλτράρουν ή να συσσωρεύσουν τα δεδομένα, δημιουργώντας νέες ροές.
- Τα επεξεργασμένα δεδομένα κατευθύνονται στους αποδέκτες για αποθήκευση ή περαιτέρω χρήση.

### 2.4.3 Παράδειγμα Αρχιτεκτονικής

Ένα παράδειγμα αρχιτεκτονικής παρουσιάζεται στο Σχήμα 2.1, όπου φαίνεται η ροή των δεδομένων από τις πηγές μέσω των τελεστών προς τους αποδέκτες.




Σχήμα 2.1: *Αρχιτεκτονική Συστημάτων Επεξεργασίας Συνεχών Ροών*

## 2.5 Τελεστές Συστημάτων Επεξεργασίας Συνεχών Ροών

Οι τελεστές (operators) αποτελούν τα δομικά στοιχεία των Συστημάτων Επεξεργασίας Συνεχών Ροών (Stream Processing Systems, SPS), υλοποιώντας τις βασικές λειτουργίες επεξεργασίας πάνω σε δεδομένα που ρέουν συνεχώς. Οι τελεστές μπορούν να διαχωριστούν σε δύο βασικές κατηγορίες: τους τελεστές χωρίς κατάσταση (stateless operators) και τους τελεστές με κατάσταση (stateful operators). Οι τελεστές χωρίς κατάσταση δεν απαιτούν καμία πληροφορία από προηγούμενα δεδομένα, ενώ οι τελεστές με κατάσταση διατηρούν πληροφορίες από παρελθόντα συμβάντα, τα οποία επηρεάζουν την επεξεργασία των επόμενων δεδομένων.

### 2.5.1 Τελεστές Χωρίς Κατάσταση

Οι τελεστές χωρίς κατάσταση (stateless operators) επεξεργάζονται κάθε εισερχόμενο συμβάν (event) ανεξάρτητα από τα υπόλοιπα, χωρίς να διατηρούν οποιαδήποτε πληροφορία για προηγούμενα δεδομένα. Αυτό σημαίνει πως η κάθε είσοδος αντιμετωπίζεται ως μεμονωμένο συμβάν, επιτρέποντας στους τελεστές αυτούς να είναι εύκολα παραλληλοποιήσιμοι και

αποδοτικοί.

Παραδείγματα τέτοιων τελεστών περιλαμβάνουν τον τελεστή προβολής (projection), ο οποίος προσθέτει, αφαιρεί ή ενημερώνει πεδία σε μια εισερχόμενη ροή δεδομένων, και τον τελεστή επιλογής (selection), ο οποίος φιλτράρει δεδομένα βάσει κάποιου λογικού κανόνα, όπως η επιλογή δεδομένων όπου η θερμοκρασία ξεπερνά έναν καθορισμένο αριθμό [13].

Η γενική λειτουργία ενός τελεστή χωρίς κατάσταση απεικονίζεται στο Σχήμα 2.2 [32], όπου φαίνεται πώς τα δεδομένα εισέρχονται στον τελεστή και επεξεργάζονται ανεξάρτητα από τα προηγούμενα συμβάντα.


Σχήμα 2.2: *Απεικόνιση της λειτουργίας ενός τελεστή χωρίς κατάσταση.*

Αυτοί οι τελεστές χρησιμοποιούνται ευρέως σε εφαρμογές όπου η ανάλυση των δεδομένων δεν απαιτεί γνώση προηγούμενων εισόδων. Για παράδειγμα, σε ένα σύστημα ανίχνευσης αλλαγών θερμοκρασίας από αισθητήρες, η επεξεργασία γίνεται στιγμιαία με κάθε νέα μέτρηση [14].

Το κύριο πλεονέκτημα των τελεστών χωρίς κατάσταση είναι η ευκολία παραλληλοποίησης και η απουσία συγχρονισμού, καθώς δεν διατηρούν πληροφορίες που απαιτούν συνεχή διαχείριση. Αυτό επιτρέπει την επανεκκίνηση των τελεστών χωρίς να απαιτείται περίπλοκη αποκατάσταση της κατάστασης, κάτι που βελτιώνει την απόδοσή τους σε περιβάλλοντα με πολλαπλά νήματα [13]. Ωστόσο, περιορίζονται στις απλές λειτουργίες και δεν είναι κατάλληλοι για σύνθετες επεξεργασίες, όπως συνενώσεις (joins) ή συσσωρεύσεις (aggregations) [33].

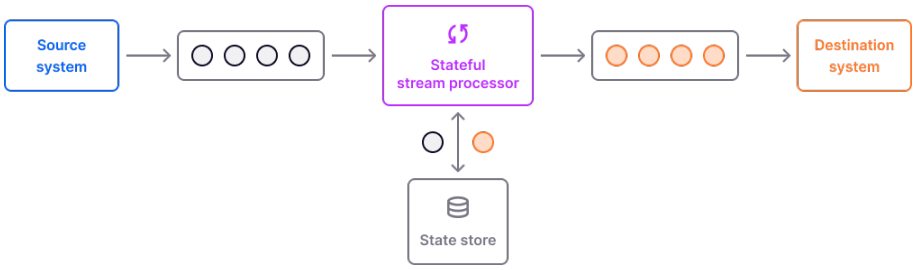
### 2.5.2 Τελεστές Με Κατάσταση

Οι τελεστές με κατάσταση (stateful operators) είναι υπεύθυνοι για τη διατήρηση μιας εσωτερικής κατάστασης κατά την επεξεργασία των δεδομένων. Αυτή η κατάσταση επιτρέπει στους τελεστές να ανατρέχουν σε προηγούμενα συμβάντα για να καθορίσουν τα επόμενα βήματα της επεξεργασίας. Τέτοιοι τελεστές είναι ιδανικοί για λειτουργίες όπως οι αθροιστικές υπολογιστικές διεργασίες και οι συνενώσεις δεδομένων.

Για παράδειγμα, ένας τελεστής αθροιστικής επεξεργασίας (aggregation) κρατά μια συσσωρευμένη τιμή, όπως το άθροισμα ή ο μέσος όρος, και την ενημερώνει κάθε φορά που φτάνουν νέα δεδομένα. Οι τελεστές συνένωσης (join) συσχετίζουν δύο διαφορετικές ροές δεδομένων με βάση ένα κοινό χαρακτηριστικό, όπως ένα μοναδικό αναγνωριστικό (key), για να παράγουν σύνθετα αποτελέσματα [13, 14].

Η βασική λειτουργία ενός τελεστή με κατάσταση φαίνεται στο Σχήμα 2.3 [32], όπου απεικονίζεται η ροή δεδομένων από το σύστημα προέλευσης προς τον τελεστή επεξεργασίας με κατάσταση, η αποθήκευση της κατάστασης σε έναν εξωτερικό αποθηκευτικό χώρο, και η συνέχιση της ροής των δεδομένων προς το σύστημα προορισμού.

Παρόλο που οι τελεστές με κατάσταση επιτρέπουν πιο σύνθετες και ισχυρές αναλύσεις, παρουσιάζουν σημαντικές προκλήσεις. Η διατήρηση κατάστασης απαιτεί περισσότερους




Σχήμα 2.3: *Απεικόνιση της λειτουργίας ενός τελεστή με κατάσταση.*

πόρους και είναι πιο δύσκολο να παραλληλοποιηθεί, καθώς η εσωτερική κατάσταση πρέπει να είναι συγχρονισμένη μεταξύ των διάφορων κόμβων του συστήματος. Επιπλέον, σε περίπτωση αποτυχίας του συστήματος, η αποκατάσταση της κατάστασης πρέπει να γίνεται με τη χρήση τεχνικών όπως τα checkpoints, τα οποία διασφαλίζουν ότι η επεξεργασία μπορεί να συνεχιστεί από το τελευταίο αποθηκευμένο σημείο.

Παραδείγματα τέτοιων τελεστών συναντώνται σε εφαρμογές όπως η ανίχνευση απάτης, όπου οι συναλλαγές ενός χρήστη αναλύονται και συγκρίνονται με προηγούμενες συναλλαγές για την ανίχνευση ύποπτων δραστηριοτήτων [33, 32, 34]. Επίσης, χρησιμοποιούνται ευρέως σε εφαρμογές παρακολούθησης συντήρησης, όπου η ανάλυση των δεδομένων εξαρτάται από τη διατήρηση και σύγκριση ιστορικών καταγραφών [33].

### 2.5.3 Διατήρηση Κατάστασης

Η κατάσταση (state) αποτελεί κρίσιμο στοιχείο για την επιτυχημένη επεξεργασία συνεχών ροών δεδομένων, ιδιαίτερα όταν απαιτούνται σύνθετοι υπολογισμοί που βασίζονται σε παρελθόντα δεδομένα. Η διατήρηση κατάστασης επιτρέπει σε τελεστές όπως οι αθροιστικές λειτουργίες ή οι χρονικές σειρές να διατηρούν προσωρινά δεδομένα για την παραγωγή ακριβών και συνεχιζόμενων υπολογισμών.

Η διαχείριση της κατάστασης περιλαμβάνει τη χρήση τεχνικών όπως τα checkpoints και τα savepoints, τα οποία αποθηκεύουν περιοδικά την κατάσταση του συστήματος για την αποκατάσταση της σε περίπτωση αποτυχίας. Τα checkpoints χρησιμοποιούνται για τη διασφάλιση της ανθεκτικότητας σε σφάλματα, επιτρέποντας στο σύστημα να συνεχίσει την επεξεργασία από το πιο πρόσφατο αποθηκευμένο σημείο [13, 14, 34].

Παρά τα πλεονεκτήματα της διατήρησης κατάστασης, υπάρχουν προκλήσεις, όπως η ανάγκη για αποδοτική αποθήκευση μεγάλων όγκων δεδομένων και η διαχείριση της κατάστασης σε κατανεμημένα συστήματα, όπου η συγχρονισμένη κατανομή της κατάστασης είναι απαραίτητη. Επιπλέον, η μεγάλη εξάρτηση από την κατάσταση μπορεί να επηρεάσει αρνητικά την απόδοση του συστήματος, καθώς αυξάνεται η πολυπλοκότητα της διαχείρισής της [33].

Παραδείγματα εφαρμογών που επωφελούνται από τη διατήρηση κατάστασης περιλαμβάνουν συστήματα ανίχνευσης απάτης και προγνωστικής συντήρησης, όπου η διατήρηση ιστορικών δεδομένων είναι κρίσιμη για την αναγνώριση μοτίβων και τη λήψη αποφάσεων σε πραγματικό χρόνο [32].

## 2.6 Παράθυρα στα Συστήματα Επεξεργασίας Ροών

Τα παράθυρα (windows) αποτελούν έναν από τους βασικότερους μηχανισμούς στα συστήματα επεξεργασίας συνεχών ροών δεδομένων (stream processing systems), καθώς επιτρέπουν τη μετατροπή μιας ατέρμονης ροής δεδομένων σε πεπερασμένα σύνολα. Αυτά τα σύνολα, γνωστά και ως «κουβάδες» (buckets), χρησιμοποιούνται για την εφαρμογή διαφόρων υπολογιστικών λειτουργιών, διευκολύνοντας έτσι την ανάλυση και την επεξεργασία της ροής δεδομένων σε πραγματικό χρόνο [35].

Η χρήση παραθύρων έχει ιδιαίτερη σημασία διότι επιτρέπει τη συλλογή και ομαδοποίηση γεγονότων που παράγονται συνεχώς και χωρίς σταματημό, δημιουργώντας «στιγμιότυπα» της ροής που μπορούν να επεξεργαστούν [36]. Με τον τρόπο αυτό, γίνεται δυνατή η διαχείριση και επεξεργασία μεγάλων όγκων δεδομένων, χωρίς να απαιτείται η αποθήκευση της συνολικής ροής, κάτι που θα ήταν εξαιρετικά δαπανηρό από άποψη υπολογιστικών πόρων.

Τα παράθυρα παρέχουν κρίσιμα πλεονεκτήματα, όπως η δυνατότητα εκτέλεσης αποδοτικών αθροιστικών και αναλυτικών λειτουργιών, όπως αθροίσματα, μέσοι όροι, και μέτρα κατανομής. Επιτρέπουν επίσης την παρακολούθηση τάσεων και την ανίχνευση γεγονότων σε σχεδόν πραγματικό χρόνο, καθιστώντας τα ιδανικά για εφαρμογές όπου η άμεση αντίδραση στα δεδομένα είναι κρίσιμη. Παραδείγματα τέτοιων εφαρμογών περιλαμβάνουν την ανάλυση κυκλοφορίας σε οδικά δίκτυα, την παρακολούθηση δεδομένων από αισθητήρες σε IoT περιβάλλοντα, την ανίχνευση ανωμαλιών σε χρηματοοικονομικές συναλλαγές, και την ανάλυση δεδομένων στα μέσα κοινωνικής δικτύωσης [36].

Η έννοια των παραθύρων συνδέεται στενά με την ανάγκη περιορισμού των δεδομένων που επεξεργάζονται σε κάθε στιγμή, καθώς η συνεχής φύση των ροών δεδομένων επιβάλλει τη χρήση τεχνικών που διαχειρίζονται τις ροές πιο αποδοτικά. Τα συστήματα επεξεργασίας συνεχών ροών αξιοποιούν ποικίλους τύπους παραθύρων, ανάλογα με τις ανάγκες της εφαρμογής, προκειμένου να διατηρούν την ισορροπία μεταξύ ακρίβειας, αποδοτικότητας και απόδοσης.

### 2.6.1 Τύποι Παραθύρων

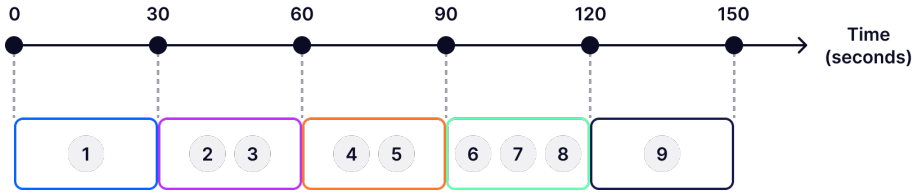
Τα συστήματα επεξεργασίας ροών χρησιμοποιούν διάφορους τύπους παραθύρων για διαφορετικά είδη ανάλυσης. Κάθε τύπος παραθύρου έχει συγκεκριμένα χαρακτηριστικά που τον καθιστούν κατάλληλο για διαφορετικές χρήσεις.

#### Αθροιστικά Παράθυρα (Tumbling Windows)

Τα αθροιστικά παράθυρα (tumbling windows) [14, 13, 37] δημιουργούν μη επικαλυπτόμενα, συνεχόμενα σύνολα δεδομένων στα οποία κάθε γεγονός ανήκει σε ένα μόνο παράθυρο. Το μέγεθος του παραθύρου είναι σταθερό και καθορίζεται από χρονικά ή ποσοτικά κριτήρια. Αυτά τα παράθυρα είναι κατάλληλα για σενάρια όπου θέλουμε να συγκρίνουμε σύνολα δεδομένων που δεν επικαλύπτονται, όπως η ανάλυση κυκλοφορίας ανά λεπτό ή οι χρηματοοικονομικοί υπολογισμοί σε σταθερά χρονικά διαστήματα.

Για παράδειγμα, ένα αθροιστικό παράθυρο με μέγεθος 30 δευτερολέπτων θα χωρίζει τη ροή σε διακριτά χρονικά διαστήματα όπως 0-30 δευτερόλεπτα, 30-60 δευτερόλεπτα, και ούτω

καθεξής, όπως φαίνεται στο Σχήμα 2.4. Σε κάθε χρονικό διάστημα, τα δεδομένα επεξεργάζονται ανεξάρτητα και εξάγονται τα αποτελέσματα για αυτό το διάστημα. Αυτά τα παράθυρα χρησιμοποιούνται συνήθως για την παραγωγή περιοδικών αναφορών ή την εκτέλεση υπολογισμών σε διακριτά χρονικά διαστήματα, όπως είναι οι οικονομικοί υπολογισμοί ανά ώρα ή οι μετρήσεις επισκεψιμότητας ιστοσελίδων ανά λεπτό.




Σχήμα 2.4: *Παράδειγμα αθροιστικού παραθύρου με μέγεθος 30 δευτερόλεπτα*

### Κυλιόμενα Παράθυρα (Sliding Windows)

Τα κυλιόμενα παράθυρα (sliding windows) [14, 13, 37] επιτρέπουν την ομαδοποίηση γεγονότων σε επικαλυπτόμενα σύνολα δεδομένων. Σε αντίθεση με τα αθροιστικά παράθυρα, τα γεγονότα μπορούν να ανήκουν σε πολλαπλά παράθυρα ταυτόχρονα. Τα κυλιόμενα παράθυρα ορίζονται με δύο βασικές παραμέτρους: το μήκος του παραθύρου και την ολίσθησή του (slide), η οποία καθορίζει το διάστημα στο οποίο δημιουργείται νέο παράθυρο.

Για παράδειγμα, το Σχήμα 2.5 δείχνει ένα κυλιόμενο παράθυρο διάρκειας 5 λεπτών. Αρχικά, το παράθυρο περιλαμβάνει τα γεγονότα 1 έως 9 (από 12:00:00 έως 12:05:00). Όταν ένα νέο γεγονός φτάνει στις 12:05:30, το παράθυρο ολισθαίνει ώστε να περιλαμβάνει πλέον τα γεγονότα 2 έως 10 (από 12:00:30 έως 12:05:30), και τα πιο παλιά δεδομένα (γεγονός 1) απομακρύνονται από το παράθυρο. Βέβαια, αυτό δεν είναι υποχρεωτικό. Σε πολλά συστήματα, αντί να προσαρμόζεται το υπάρχον παράθυρο, απλώς δημιουργείται ένα νέο που ξεκινάει από τη στιγμή άφιξης του νέου γεγονότος και λήγει μία στιγμή αργότερα σε σχέση με το προηγούμενο.

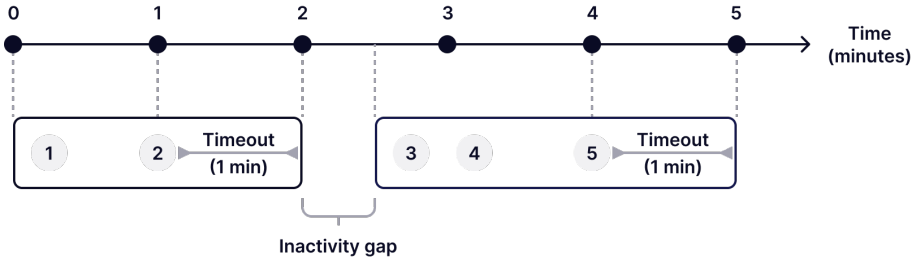
Η χρήση κυλιόμενων παραθύρων επιτρέπει τη συνεχή ενημέρωση της ανάλυσης με τα πιο πρόσφατα δεδομένα, καθιστώντας τα ιδανικά για εφαρμογές που απαιτούν συνεχή εποπτεία των δεδομένων σε κινούμενα χρονικά διαστήματα.


Σχήμα 2.5: *Παράδειγμα κυλιόμενου παραθύρου διάρκειας 5 λεπτών*

**Παράθυρα Συνεδριών (Session Windows)**

Τα παράθυρα συνεδριών (session windows) [14, 37, 30] ομαδοποιούν γεγονότα που προκύπτουν κατά τη διάρκεια μιας συνεδρίας ή περιόδου δραστηριότητας, και διαχωρίζονται από διαστήματα αδράνειας. Αυτά τα παράθυρα δεν έχουν προκαθορισμένη διάρκεια, αλλά βασίζονται στη διακοπή της δραστηριότητας. Για παράδειγμα, μια συνεδρία μπορεί να περιλαμβάνει διαδοχικές ενέργειες χρήστη μέχρι να υπάρξει ένα διάστημα αδράνειας, μετά το οποίο το παράθυρο κλείνει.

Το Σχήμα 2.6 απεικονίζει ένα παράδειγμα παραθύρου συνεδρίας, όπου τα γεγονότα ομαδοποιούνται κατά τη διάρκεια συνεχούς δραστηριότητας, ενώ το παράθυρο κλείνει μόλις υπάρξει ένα διάστημα αδράνειας. Τα παράθυρα συνεδριών είναι ιδιαίτερα χρήσιμα για την ανάλυση της συμπεριφοράς χρηστών, όπως η παρακολούθηση δραστηριότητας σε διαδικτυακές αγορές ή η ανάλυση περιόδων αλληλεπίδρασης με έναν ιστότοπο.




Σχήμα 2.6: *Παράδειγμα παραθύρου συνεδρίων*

### 2.6.2 Διαχείριση Παραθύρων

Η διαχείριση των παραθύρων είναι κρίσιμη για την αποδοτικότητα και ακρίβεια της επεξεργασίας δεδομένων. Κάθε τύπος παραθύρου έχει τις δικές του πολιτικές ενεργοποίησης (trigger policies) και εκκαθάρισης (eviction policies), οι οποίες καθορίζουν πότε τα δεδομένα στο παράθυρο θα αναλυθούν και πότε θα αφαιρεθούν.

- **Πολιτικές ενεργοποίησης (Trigger Policies)**: Ορίζουν πότε το περιεχόμενο του παραθύρου θα αξιολογηθεί. Για παράδειγμα, σε μια πολιτική με βάση τον αριθμό γεγονότων (count-based), το παράθυρο θα ενεργοποιηθεί μόλις φτάσει έναν προκαθορισμένο αριθμό γεγονότων, ενώ σε μια πολιτική με βάση το χρόνο (time-based), η ανάλυση μπορεί να γίνεται κάθε 30 δευτερόλεπτα ανεξάρτητα από το πλήθος των γεγονότων [38].

- **Πολιτικές εκκαθάρισης (Eviction Policies)**: Καθορίζουν πότε τα γεγονότα θα αφαιρεθούν από το παράθυρο. Για παράδειγμα, σε ένα παράθυρο με πολιτική εκκαθάρισης βάσει χρόνου, τα δεδομένα θα διαγράφονται μόλις περάσει το καθορισμένο χρονικό διάστημα. Αυτό επιτρέπει τη διαχείριση της μνήμης και την αποτροπή της υπερφόρτωσης του συστήματος [38].

Η ισορροπία μεταξύ της επιλογής του τύπου παραθύρου και των πολιτικών ενεργοποίησης και εκκαθάρισης μπορεί να επηρεάσει δραστικά την απόδοση του συστήματος, ιδιαίτερα

όταν γίνεται επεξεργασία μεγάλων ροών δεδομένων σε πραγματικό χρόνο. Συνεπώς, είναι απαραίτητο να προσαρμόζονται οι πολιτικές αυτές ανάλογα με τις απαιτήσεις της εφαρμογής.

## 2.7 Τεχνικές Διαμοιρασμού των Δεδομένων

### 2.7.1 Διαμερισμός Κλειδιών

Στα κατανεμημένα συστήματα επεξεργασίας συνεχών ροών δεδομένων (DSPSs), η αποδοτική κατανομή των δεδομένων παίζει καθοριστικό ρόλο στην επίτευξη κλιμακωσιμότητας, βέλτιστης απόδοσης και ισορροπίας φόρτου εργασίας μεταξύ των κόμβων επεξεργασίας. Οι στρατηγικές διαμερισμού κλειδιών καθορίζουν τον τρόπο με τον οποίο διανέμονται τα δεδομένα των συνεχών ροών στους κόμβους επεξεργασίας, προκειμένου να πραγματοποιηθεί παράλληλη επεξεργασία. Η κατανομή αυτή βασίζεται κατά κύριο λόγο στα κλειδιά των δεδομένων.

### 2.7.2 Ανισορροπία Κλειδιών

Η ανισορροπία (skewness) [39, 30] μεταξύ των κλειδιών, δηλαδή το κατά πόσο αυτά κατανέμονται ομοιόμορφα, επηρεάζει σημαντικά την απόδοση των στρατηγικών διαμερισμού. Στις περισσότερες περιπτώσεις, τα κλειδιά δεν διαμοιράζονται ισομερώς, γεγονός που επιβαρύνει την απόδοση των DSPSs και αυξάνει την πολυπλοκότητα των μηχανισμών που απαιτούνται για τη διαχείριση αυτής της ανισορροπίας.

Ένα κλασικό παράδειγμα είναι το word count [40, 26], στο οποίο στόχος είναι η καταμέτρηση των εμφανίσεων κάθε λέξης. Εάν οι λέξεις αποτελούν κλειδιά και τα δεδομένα κατανέμονται με βάση το αρχικό γράμμα της λέξης, τότε ένας κόμβος που επεξεργάζεται κλειδιά που ξεκινούν με το γράμμα 't' θα έχει σημαντικά μεγαλύτερο φόρτο σε σύγκριση με έναν κόμβο που αναλαμβάνει λέξεις που ξεκινούν με το γράμμα 'z'. Η συχνότητα εμφάνισης λέξεων που αρχίζουν με 't' είναι κατά πολύ υψηλότερη, οδηγώντας έτσι σε ανισόρροπη κατανομή φόρτου εργασίας. Αυτό μπορεί να επιφέρει καθυστερήσεις στον πρώτο κόμβο, επηρεάζοντας συνολικά την απόδοση του συστήματος.

Καθίσταται σαφές, ότι αυτή η μη ομοιόμορφη κατανομή των κλειδιών δημιουργεί την ανάγκη για πιο εξελιγμένες στρατηγικές διαμερισμού.

### 2.7.3 Τεχνικές Διαμεριμού Κλειδιών

Σε αυτή την ενότητα, εξετάζονται διάφορες δημοφιλείς στρατηγικές διαμερισμού που χρησιμοποιούνται για τον διαμερισμό των δεδομένων ροών σε κόμβους επεξεργασίας με βάση τα κλειδιά. Επίσης, γίνεται σύντομη αναφορά στα πλεονεκτήματα και τα μειονεκτήματα κάθε στρατηγικής.

#### Shuffle Grouping (Round-Robin)

Η στρατηγική Shuffle Grouping, βασίζεται στον αλγόριθμο Round-Robin [41], κατευθύνοντας τα δεδομένα σε κόμβους με κυκλικό τρόπο, χωρίς κάποιον προκαθορισμένο κανόνα. Το i-οστό κλειδί κατευθύνεται στον κόμβο i mod n, όπου n είναι το πλήθος των κόμβων

και mod είναι η πράξη modulo. Αυτή η μέθοδος εξασφαλίζει ομοιόμορφο φόρτο εργασίας, αποτρέποντας την υπερφόρτωση ενός συγκεκριμένου κόμβου, ενώ παράλληλα είναι πολύ εύκολη στην υλοποίηση.

Ωστόσο, η Shuffle Grouping δεν είναι ιδιαίτερα αποδοτική όταν απαιτείται η αποθήκευση κατάστασης. Για παράδειγμα, στο word count, κάθε κόμβος πρέπει να διατηρεί έναν μετρητή για κάθε λέξη που επεξεργάζεται. Η τυχαία κατανομή των λέξεων σημαίνει ότι κάθε κόμβος μπορεί να χρειαστεί να αποθηκεύσει μετρητές για όλα τα κλειδιά, αυξάνοντας το κόστος σε μνήμη και καθιστώντας αυτή τη στρατηγική μη αποδοτική σε εφαρμογές μεγαλύτερης κλίμακας, όπου απαιτείται η διατήρηση και ενημέρωση μεγάλυτερων δομών δεδομένων.

#### Κατανομή με Χρήση Κατακερματισμού (Key Hashing)

Στην κατανομή με χρήση κατακερματισμού, μία συνάρτηση κατακερματισμού h αναθέτει κάθε κλειδί σε έναν συγκεκριμένο κόμβο. Η στρατηγική αυτή εξασφαλίζει ότι κάθε κλειδί θα κατευθύνεται πάντα στον ίδιο κόμβο, επιλύοντας προβλήματα που αφορούν τη μνήμη, αφού κάθε κόμβος διαχειρίζεται μόνο τα συγκεκριμένα κλειδιά που του ανατίθενται.

Παρόλα αυτά, η κατανομή με κατακερματισμό δεν εξασφαλίζει ισομερή φόρτο εργασίας, ειδικά σε περιπτώσεις όπου τα κλειδιά εμφανίζονται με άνιση συχνότητα. Κάποιοι κόμβοι μπορεί να επιβαρυνθούν σημαντικά περισσότερο από άλλους, δημιουργώντας ανισορροπία.

#### Ομαδοποίηση Κλειδιών (Key Grouping)

Η στρατηγική Key Grouping αποτελεί μία πιο εξειδικευμένη μέθοδο διαμοιρασμού κλειδιών, όπου τα κλειδιά ομαδοποιούνται με βάση ένα κοινό χαρακτηριστικό πριν κατανεμηθούν στους κόμβους επεξεργασίας. Σε αντίθεση με την παραδοσιακή κατανομή μέσω κατακερματισμού (Key Hashing), όπου τα κλειδιά κατευθύνονται σε συγκεκριμένους κόμβους βάσει του αλγορίθμου κατακερματισμού, η Key Grouping διασφαλίζει ότι κλειδιά με κοινά χαρακτηριστικά διανέμονται στον ίδιο κόμβο. Αυτό το χαρακτηριστικό μπορεί να είναι, για παράδειγμα, το αρχικό γράμμα ή το πρόθεμα μιας λέξης.

Για να γίνει πιο κατανοητή η λειτουργία της στρατηγικής αυτής, μπορούμε να εξετάσουμε το παράδειγμα του word count. Στην κατανομή μέσω κατακερματισμού, κάθε λέξη θα ανατίθετο σε έναν κόμβο με βάση την τιμή που προκύπτει από τη συνάρτηση κατακερματισμού. Στην περίπτωση της Key Grouping, η κατανομή μπορεί να γίνει με βάση το πρόθεμα της λέξης, έτσι ώστε λέξεις με το ίδιο πρόθεμα να κατευθύνονται στον ίδιο κόμβο. Για παράδειγμα, οι λέξεις "apple" και "ability" μπορούν να κατευθυνθούν στον ίδιο κόμβο αν η συνάρτηση ομαδοποίησης λάβει υπόψη τα δύο πρώτα γράμματα.

Η Key Grouping παρέχει καλύτερη κατανομή του φόρτου εργασίας, ειδικά σε περιπτώσεις όπου τα κλειδιά εμφανίζονται με ανισοκατανομή. Με τον τρόπο αυτό, μειώνεται η υπερφόρτωση συγκεκριμένων κόμβων και επιτυγχάνεται μεγαλύτερη αποδοτικότητα.

Ωστόσο, η στρατηγική αυτή εξακολουθεί να υπόκειται στους περιορισμούς της ανισοκατανομής όταν τα χαρακτηριστικά βάσει των οποίων γίνεται η ομαδοποίηση δεν κατανέμονται ισομερώς. Παρά την προσφερόμενη ευελιξία στη διανομή των κλειδιών σε σχέση με την κατανομή μέσω κατακερματισμού, απαιτείται πιο σύνθετος μηχανισμός για την ομαδοποίηση και διαχείριση των κλειδιών, καθιστώντας την πιο πολύπλοκη στην υλοποίηση.

### The Power of Two Choices (PoTC)

Οι προηγούμενες στρατηγικές και κυρίως οι shuffle grouping και key grouping, αποτέλεσαν τη βάση για την ανάπτυξη πιο σύνθετων και αποδοτικών συστημάτων και τεχνικών (π.χ. [42, 43, 44, 45, 46]). Μία από αυτές είναι η "Power of two choices" [42].

Σε αυτή τη μέθοδο, για κάθε κλειδί, κατά την πρώτη εμφάνισή του, υπολογίζονται δύο υποψήφιοι κόμβοι επεξεργασίας. Αυτό συνήθως γίνεται με χρήση δύο διαφορετικών συναρτήσεων κατακερματισμού. Στη συνέχεια, υπολογίζεται ο φόρτος εργασίας των δύο κόμβων και επιλέγεται αυτός με το μικρότερο φόρτο. Από το σημείο αυτό και έπειτα, το κλειδί αυτό κατευθύνεται πάντα σε αυτόν τον κόμβο.

Αυτή η μέθοδος πετυχαίνει ακόμα καλύτερη κατανομή φόρτου εργασίας, αφού για κάθε κλειδί επιλέγονται δύο κόμβοι και στη συνέχεια αυτός με το λιγότερο φόρτο. Ωστόσο, εισάγει ακόμα περισσότερη πολυπλοκότητα, εφόσον αφενός πρέπει να διατηρείται κατάσταση με τις επιλογές που έχουν γίνει για τα κλειδιά και αφετέρου απαιτείται το σύστημα να γνωρίζει το φόρτο κάθε κόμβου, ώστε να κάνει την κατάλληλη επιλογή.

### Partial Key Grouping (PKG)

Μία άλλη σημαντική μέθοδος είναι η Partial Key Grouping (PKG), η οποία εισήχθηκε από τους Nasir et al. [43]. Αυτή η μέθοδος μοιάζει αρκετά με την προηγούμενη (PoTC), αλλά λειτουργεί με δυναμικό τρόπο. Πιο συγκεκριμένα, και πάλι, για κάθε κλειδί, υπολογίζονται δύο υποψήφιοι κόμβοι επεξεργασίας, αλλά αυτή τη φορά επιλέγονται και οι δύο. Κάθε φορά που εμφανίζεται το κλειδί, ελέγχονται οι δύο επιλεγμένοι κόμβοι και το κλειδί κατευθύνεται στον κόμβο που έχει το μικρότερο φόρτο εργασίας την εκάστοτε στιγμή. Συνεπώς, κάθε στιγμή το κλειδί μπορεί να πάει σε έναν από τους δύο κόμβους και δεν επιλέγεται στατικά και εξ 'αρχής ένας από τους δύο όπως στην προηγούμενη περίπτωση.

Γίνεται εύκολα αντιληπτό, ότι αυτή η δυναμική ιδιότητα αυτής της μεθόδου, πετυχαίνει ακόμα περισσότερο ομοιόμορφη κατανομή φόρτου μεταξύ των κόμβων. Ωστόσο, επιφέρει μεγαλύτερη πολυπλοκότητα και δυσκολία, αφού πρέπει κάθε φορά να υπολογίζεται ο φόρτος των επιλεγμένων κόμβων, ώστε να γίνεται κατάλληλη τελική επιλογή.

### Προηγμένες Στρατηγικές και Μελλοντικές Κατευθύνσεις

Πρόσφατες έρευνες έχουν εισαγάγει προηγμένες στρατηγικές για την καλύτερη αντιμετώπιση της ανισορροπίας των κλειδιών. Ορισμένες προσεγγίσεις, για παράδειγμα, εντοπίζουν "hot keys" (συχνά εμφανιζόμενα κλειδιά) [44, 45, 46] και εφαρμόζουν διαφορετικές στρατηγικές διαμερισμού σε αυτά τα κλειδιά για να βελτιώσουν τη συνολική απόδοση. Η κοινή πρακτική είναι μία τεχνική που ονομάζεται key splitting μέσω της οποίας ένα κλειδί «σπάει» και κατευθύνεται σε περισσότερους από έναν κόμβους επεξεργασίας. Αυτές οι μέθοδοι, συχνά αξιοποιώντας τεχνικές μηχανικής μάθησης [46], στοχεύουν στην καλύτερη διαχείριση των μοναδικών προκλήσεων που προκαλεί η ανισορροπία των κλειδιών.

Η υλοποίηση τέτοιων προχωρημένων μεθόδων ξεφεύγει από τα πλαίσια αυτής της διπλωματικής εργασίας. Ο κύριος στόχος της παρούσας εργασίας είναι η σχεδίαση και υλοποίηση

ενός προσομοιωτή, μέσω του οποίου θα είναι δυνατή η σύγκριση διαφόρων στρατηγικών διαμερισμού δεδομένων σε ένα κοινό περιβάλλον. Παράλληλα, ο προσομοιωτής θα επιτρέπει την εξαγωγή πληροφοριών και αποτελεσμάτων σχετικά με άλλες σημαντικές μετρικές, όπως ο ρυθμός άφιξης των κλειδιών και η υπολογιστική ισχύς των κόμβων επεξεργασίας.

Συνεπώς, για τους σκοπούς αυτής της διπλωματικής εργασίας, θα επικεντρωθούμε σε δύο θεμελιώδεις και ευρέως χρησιμοποιούμενες στρατηγικές, το key hashing και το shuffle grouping, καθώς αποτελούν τη βάση για τη μελέτη και κατανόηση πιο σύνθετων τεχνικών. Επιπλέον, θα μελετήσουμε τις στρατηγικές PoTC και PKG, καθώς η πρώτη υιοθετεί μια πιο ευφυή προσέγγιση και αποτελεί τη βάση για τη δεύτερη, η οποία εφαρμόζει την τεχνική του key splitting.

Κεφάλαιο 3

# Σχετικές Εργασίες

Η ανάπτυξη και η βελτίωση των συστημάτων επεξεργασίας συνεχών ροών δεδομένων έχει απασχολήσει έντονα την ερευνητική κοινότητα, οδηγώντας σε μια πληθώρα δημοσιεύσεων που εξετάζουν διάφορες πτυχές και προκλήσεις αυτού του πεδίου. Αυτή η ενότητα παρουσιάζει τρεις βασικές δημοσιεύσεις που αφορούν την προσομοίωση και αξιολόγηση συστημάτων επεξεργασίας ροής δεδομένων, επισημαίνοντας τις κύριες προκλήσεις και τις συνεισφορές κάθε προσέγγισης.

## 3.1 Σχετικές Εργασίες

Οι Alfred J. Park et al. [47] παρουσιάζουν τον Flow, μία πλατφόρμα σχεδιασμένη για την προσομοίωση συστημάτων επεξεργασίας ροής δεδομένων σε παράλληλο και κατανεμημένο περιβάλλον. Ο Flow προσφέρει ένα ευέλικτο περιβάλλον μοντελοποίησης για την ανάλυση εφαρμογών επεξεργασίας ροής, επιτρέποντας τη δημιουργία και σύνθεση διαφορετικών μοντέλων ροής και πυρήνων με πλήρη έλεγχο σε παραμέτρους όπως οι ρυθμοί εισόδου/εξόδου. Διευκολύνει την αυτόματη σύνθεση και διαμερισματοποίηση μεγάλων προσομοιώσεων, επιτρέποντας τη σύνθεση μικρότερων πραγματικών προγραμμάτων StreamIt και προσφέρει δυνατότητα αυτόματης παραλληλοποίησης των μοντέλων εντός κόμβου προσομοίωσης. Επιπλέον, μειώνει σχεδόν στο μηδέν τα κόστη συγχρονισμού για κατανεμημένες προσομοιώσεις με γραφήματα ροής δεδομένων που παρουσιάζουν συμπεριφορά προώθησης και υποστηρίζει διαφανώς την εκτέλεση σε κατανεμημένα περιβάλλοντα. Η απόδοση του Flow μελετήθηκε ως προς τις δυνατότητες κλιμακωσιμότητας, δείχνοντας εξαιρετική κλιμάκωση σε μεγάλες προσομοιώσεις. Ωστόσο, στα συμπεράσματα της μελέτης αναφέρεται η ανάγκη για περαιτέρω έρευνα, ιδίως στη βελτίωση της διαδικασίας διαμερισματοποίησης, στην υποστήριξη πιο πολύπλοκων σεναρίων και στην εξισορρόπηση φόρτου.

Οι Amarasinghe et al.[48] παρουσιάζουν το ECSNeT++, έναν προσομοιωτή σχεδιασμένο για την κατανομή και επεξεργασία ροής δεδομένων σε περιβάλλοντα edge και cloud. Το ECSNeT++ αποσκοπεί στο να παρέχει ένα ρεαλιστικό εργαλείο για τη μοντελοποίηση και ανάλυση κατανεμημένων συστημάτων επεξεργασίας ροής σε διάφορα υπολογιστικά περιβάλλοντα, επιτρέποντας την εκμετάλλευση πολλών συνδεδεμένων συσκευών όπως αυτές του Internet of Things (IoT). Οι κύριες δυνατότητες του ECSNeT++ περιλαμβάνουν:

1. **Εκτεταμένη Υποστήριξη για Edge και Cloud Περιβάλλοντα**: Παρέχει προσομοίωση συστημάτων σε περιβάλλοντα edge και cloud, επιτρέποντας την εξερεύνηση διαφορετικών αρχιτεκτονικών και κατανομών φόρτου.

2. **Αναλυτική Μοντελοποίηση Κατανεμημένων Συστημάτων**: Επιτρέπει τη μοντελοποίηση και ανάλυση κατανεμημένων εφαρμογών επεξεργασίας ροής με έμφαση σε χαρακτηριστικά όπως η κατανομή πόρων και η επικοινωνία μεταξύ κόμβων.

3. **Υποστήριξη Ποικιλίας Σενάριων**: Δυνατότητα προσομοίωσης διάφορων σεναρίων με διαφορετικούς τύπους φόρτου και πολιτικές κατανομής, ενισχύοντας την ευελιξία και την επεκτασιμότητα του προσομοιωτή.

4. **Εργαλεία Υπολογισμού Απόδοσης**: Παρέχει εργαλεία για τη μέτρηση και ανάλυση της απόδοσης, συμπεριλαμβανομένων των καθυστερήσεων επεξεργασίας και δικτύου, καθώς και της κατανάλωσης ενέργειας ανά γεγονός.

Η αξιολόγηση απόδοσης του ECSNeT++ δείχνει ότι ο προσομοιωτής είναι ικανός να διαχειρίζεται σύνθετα σενάρια σε κατανεμημένα περιβάλλοντα με υψηλή κλιμακωσιμότητα και ακρίβεια στις μετρήσεις. Οι εφαρμογές, κυρίως IoT, αξιολογήθηκαν χρησιμοποιώντας τέσσερις βασικές μετρήσεις: καθυστέρηση επεξεργασίας ανά γεγονός, καθυστέρηση μετάδοσης δικτύου ανά γεγονός, συνολική καθυστέρηση ανά γεγονός, και μέση κατανάλωση ισχύος σε κάθε edge συσκευή. Στα συμπεράσματα, η μελέτη επισημαίνει την ανάγκη για περαιτέρω βελτιώσεις, ιδίως στη βελτιστοποίηση των αλγορίθμων κατανομής πόρων και στην υποστήριξη πιο πολύπλοκων διατάξεων και συνθηκών φόρτου, ενώ τονίζει ότι το ECSNeT++ επικεντρώνεται κυρίως στη μέτρηση της απόδοσης του δικτύου και της υπολογιστικής καθυστέρησης, και όχι στις στρατηγικές διαμερισματοποίησης των δεδομένων.

Οι Karimov et al. [49] εισάγουν μια μεθοδολογία και εργαλεία για την αξιολόγηση των κατανεμημένων συστημάτων επεξεργασίας ροής δεδομένων (SDPS). Η δημοσίευσή τους επικεντρώνεται στην ανάπτυξη ενός ολοκληρωμένου πλαισίου για τη σύγκριση και ανάλυση των επιδόσεων διαφορετικών συστημάτων επεξεργασίας ροής. Οι βασικές συνεισφορές της μελέτης περιλαμβάνουν:

1. **Μηχανισμό Ακριβούς Μέτρησης Καθυστέρησης**: Εισάγεται μια μέθοδος για την ακριβή μέτρηση της καθυστέρησης (latency) των stateful operators σε SDPS, η οποία εφαρμόζεται σε διάφορες περιπτώσεις χρήσης.

2. **Μέτρηση Μέγιστης Βιώσιμης Ρυθμαπόδοσης (Sustainable Throughput)**: Το benchmarking framekwork διαχειρίζεται συγκεκριμένα χαρακτηριστικά συστήματος, όπως το backpressure, για τη μέτρηση της μέγιστης βιώσιμης ρυθμαπόδοσης.

3. **Εκτεταμένη Αξιολόγηση Μεγάλων SDPS**: Χρησιμοποιείται το benchmarking framework για εκτεταμένη αξιολόγηση τριών σημαντικών SDPS: Apache Storm, Apache Spark, και Apache Flink, με σενάρια πραγματικής χρήσης.

Η μελέτη αποδεικνύει ότι το προτεινόμενο πλαίσιο benchmarking είναι αποτελεσματικό για τη σύγκριση και ανάλυση των DSPSs, παρέχοντας ακριβείς και αξιόπιστες μετρήσεις

απόδοσης. Τα πειράματα αναδεικνύουν τα ιδιαίτερα χαρακτηριστικά και τις προκλήσεις κάθε συστήματος, προσφέροντας κατευθυντήριες γραμμές για τον καθορισμό των απαιτήσεων για κάθε χρήση. Ιδιαίτερη έμφαση δίνεται στη μέτρηση του throughput και του latency, παρέχοντας μια σαφή εικόνα των επιδόσεων των συστημάτων υπό διαφορετικά φορτία και σενάρια. Επιπλέον, το πλαίσιο που προτείνουν οι συγγραφείς διαχειρίζεται χαρακτηριστικά όπως το backpressure, επιτρέποντας τη μέτρηση της μέγιστης βιώσιμης ρυθμαπόδοσης, και παρέχει μετρήσεις καθυστέρησης (latency) που είναι ζωτικής σημασίας για την αξιολόγηση της απόδοσης των stateful operators. Επισημαίνεται ότι το πλαίσιο αυτό αφορά τη μέτρηση επιδόσεων σε πραγματικά συστήματα και ουσιαστικά λειτουργεί ως ένα πλήρες benchmarking suite.

## 3.2 Σύγκριση και Τοποθέτηση

Οι παραπάνω δημοσιεύσεις παρέχουν μια ικανοποιητική εικόνα των δυνατοτήτων και των προκλήσεων στην προσομοίωση συστημάτων επεξεργασίας ροής δεδομένων. Ο Flow επικεντρώνεται στην προσομοίωση μεγάλων και πολύπλοκων συστημάτων, παρέχοντας λεπτομερή ανάλυση των επιδόσεων δίνοντας έμφαση στην κλιμακωσιμότητα. Ο ECSNeT++ εξετάζει την κατανεμημένη επεξεργασία σε περιβάλλοντα ακμής και νέφους, μετρώντας την απόδοση των συστημάτων σε τέτοια περιβάλλοντα και εστιάζοντας κυρίως στις δικτυακές επιδόσεις τους και στην ακρίβεια προσομοίωσης τους. Η δημοσίευση των Karimov et al. [49] εισάγει ένα πλαίσιο αξιολόγησης (benchmarking) για τα κατανεμημένα συστήματα επεξεργασίας ροής δεδομένων, επικεντρώνοντας στη μέτρηση του throughput και του latency για την παροχή ακριβών και αξιόπιστων συγκρίσεων μεταξύ διαφορετικών συστημάτων όπως τα Apache Storm, Apache Spark, και Apache Flink.

Οι παραπάνω εργασίες συνεισφέρουν σημαντικά στην κατανόηση και τη βελτίωση συστημάτων επεξεργασίας ροής δεδομένων, παρέχοντας διάφορες προσεγγίσεις και εργαλεία για την προσομοίωση και αξιολόγηση επιδόσεων. Όπως είδαμε και παραπάνω, κάθε ένα σύστημα εστιάζει σε μία συγκεκριμένη πτυχή των DSPSs. Συνδυαστικά, προσφέρουν καθοδήγηση για τη σχεδίαση πιο αποδοτικών και ευέλικτων συστημάτων.

Παρά την σημαντική πρόοδο που έχουν επιφέρει αυτές οι συνεισφορές στην έρευνα για την επεξεργασία ροής δεδομένων, ο προσομοιωτής μας εισάγει νέα χαρακτηριστικά που τον διαφοροποιούν από τα υπάρχοντα εργαλεία. Αρχικά, έχει ως στόχο την προσομοίωση από άκρο σε άκρο των DSPSs. Για την καλύτερη εξέταση διαφορετικών σεναριών χρησιμοποιεί μια εξελιγμένη γεννήτρια κλειδιών (key generator) για την παραγωγή κλειδιών με ποικίλες κατανομές και χαρακτηριστικά. Όπως σημειώθηκε και νωρίτερα, μεγάλη σημασία έχει η επίτευξη του ισομοιρασμού του φόρτου στους κόμβους του συστήματος. Για το σκοπό αυτό, αναπτύχθηκε ένας επεκτάσιμος κατανεμητής (partitioner) που υλοποιεί ποικίλες στρατηγικές για τη διανομή των δεδομένων στους κόμβους. Επιπλέον, επιλέχθηκε η προσομοίωση κόμβων με κυλιόμενα παράθυρα, επιτρέποντας την ανάλυση ροών δεδομένων σε καθορισμένα χρονικά πλαίσια. Ακόμη, ο προσομοιωτής μας ακολουθεί μια πιο αφηρημένη προσέγγιση στην τομέα της επεξεργασίας, χωρίς να υλοποιεί συγκεκριμένες εργασίες. Ως εργασίες, θεωρούμε operators για κάθε στάδιο, οι οποίοι είναι αφηρημένοι και προσαρμοσμένοι ώστε η προσομοίωση να είναι πλήρως παραμετροποιήσιμη και να μη βασίζεται στην εκάστοτε εργα-

σία αλλά στην πολυπλοκότητα της. Έτσι, μπορούν να εισαχθούν συγκεκριμένοι operators που θέλει να υλοποιεί ο κάθε χρήστης.

Συγκεντρωτικά, μπορούν να παραμετροποιηθούν διάφορα χαρακτηριστικά, όπως η κατανομή των κλειδιών κατά τη γέννησή τους, η στρατηγική διαμερισμού των κλειδιών, το πλήθος των κόμβων του συστήματος, η ρυθμαπόδοση των κόμβων, ο ρυθμός άφιξης των κλειδιών, το μέγεθος και η ολίσθηση του παραθύρου, καθώς και ο τύπος τους κάθε σταδίου κόμβων. Αυτά τα χαρακτηριστικά προσφέρουν πιο ακριβείς και ευέλικτες προσομοιώσεις επεξεργασίας ροής, επιτρέποντας μια λεπτομερή εξέταση της απόδοσης και της συμπεριφοράς του συστήματος. Με την ενσωμάτωση αυτών των δυνατοτήτων, ο προσομοιωτής μας παρέχει μια πιο ολοκληρωμένη και προσαρμόσιμη πλατφόρμα για την έρευνα και την ανάπτυξη στην επεξεργασία ροής δεδομένων, καλύπτει κενά στις τρέχουσες μεθοδολογίες και ανοίγει το δρόμο για καινοτόμες λύσεις στον τομέα αυτό.

# Μέρος II

# Πρακτικό Μέρος

Κεφάλαιο 4

# Ανάλυση και Σχεδίαση

Σε αυτό το κεφάλαιο παρουσιάζεται η μελέτη που έγινε για την υλοποίηση του προσομοιωτή. Αρχικά, περιγράφεται η αρχιτεκτονική του προσομοιωτή και γίνεται διαχωρισμός στα επιμέρους συστατικά και τη λειτουργία τους, ενώ παράλληλα γίνεται και αναφορά σε παραδοχές και συμβάσεις που έχουν γίνει.

## 4.1 Ανάλυση και Περιγραφή Αρχιτεκτονικής

Στην ενότητα αυτή, παρουσιάζεται η ανάλυση του προσομοιωτή και ο χωρισμός του σε διάφορα συστατικά όσον αφορά την αρχιτεκτονική.

### 4.1.1 Διαχωρισμός Συστατικών

Το σύστημα παρουσιάζει υψηλό βαθμό αφαιρετικότητας, αποτελούμενο από διακριτά συστατικά που ενσωματώνουν σημαντικά υποσυστήματα. Στην κορυφή της ιεραρχίας βρίσκεται ο Προσομοιωτής (Simulator), ο οποίος αποτελείται από δύο κύρια μέρη: τη Γεννήτρια Κλειδιών (KeyGenerator) και την Τοπολογία Προσομοίωσης (Topology). Η Τοπολογία αποτελείται από ένα μεταβλητό πλήθος Σταδίων (Stage), καθένα από τα οποία περιέχει έναν μεταβλητό αριθμό Κόμβων Επεξεργασίας (Node) και υλοποιεί μία διακριτή λειτουργία (Operation).

Οι Κόμβοι Επεξεργασίας διακρίνονται σε δύο κατηγορίες: αυτούς με εσωτερική κατάσταση (StatefulNode) και αυτούς χωρίς κατάσταση (StatelessNode). Οι Κόμβοι με εσωτερική κατάσταση είναι πιο σύνθετοι, καθώς διατηρούν μία εσωτερική κατάσταση (State) η οποία περιλαμβάνει και κυλιόμενα παράθυρα (Window) ως βασικό συστατικό της.

Αυτοί οι κόμβοι διακρίνονται περαιτέρω σε WorkerNode και AggregatorNode. Οι WorkerNodes εκτελούν επεξεργαστικές λειτουργίες, όπως ταξινόμηση, ενώ οι AggregatorNodes εκτελούν λειτουργίες συνάθροισης. Κάθε τύπος κόμβου διαθέτει τη δική του κλάση εσωτερικής κατάστασης: WorkerState για τους WorkerNodes και AggregatorState για τους AggregatorNodes.

Οι Κόμβοι χωρίς κατάσταση (StatelessNodes) λειτουργούν κυρίως ως Κατανεμητές Κλειδιών (KeyPartitioner), εφαρμόζοντας διαφορετικές στρατηγικές κατανομής κλειδιών. Τέλος, υπάρχει ο Εξαγωγέας (Exporter), ο οποίος είναι υπεύθυνος για την εξαγωγή πληροφοριών από το αρχείο παραμετροποίησης (Configuration File).

### 4.1.2 Περιγραφή Συστατικών

Παρακάτω δίνεται λεπτομερής περιγραφή για καθένα από τα προαναφερθέντα συστατικά.

#### Εξαγωγέας

Ο εξαγωγέας είναι μία απλή συνάρτηση, η οποία διαβάζει το αρχείο παραμετροποίησης του προσομοιωτή και εξάγει τις παραμέτρους του, ώστε να μπορούν να χρησιμοποιηθούν από τα υπόλοιπα συστατικά. Στο Σχήμα 4.1 παρουσιάζεται ένα παράδειγμα ενός τέτοιου αρχείου παραμετροποίησης.

```
{
    "keygen": {
        "streams": 1,
        "steps": 50,
        "number_of_keys": 10,
        "arrival_rate": 50,
        "spike_probability": 0,
        "spike_magnitude": 0,
        "distribution": {
            "type": "uniform"
        }
    },
    "topology": {
        "stages": [
            {
                "id": 0,
                "type": "stateless",
                "nodes": [
                    {
                        "id": 0,
                        "type": "key_partitioner",
                        "throughput": 1000,
                        "operation_type": "StatelessOperation",
                        "strategy": {
                            "name": "hashing"
                        }
                    }
                ]
            },
            {
                "id": 1,
                "type": "stateful",
                "key_splitting": false,
                "nodes": [
                    {
                        "id": 1,
                        "type": "stateful",
                        "throughput": 1000,
                        "operation_type": "Sorting",
                        "window_size": 5,
                        "slide": 2
                    },
```

```
                    {
                        "id": 2,
                        "type": "stateful",
                        "throughput": 1000,
                        "operation_type": "Sorting",
                        "window_size": 5,
                        "slide": 2
                    }
                ]
            },
            {
                "id": 2,
                "type": "stateless",
                "nodes": [
                    {
                        "id": 3,
                        "type": "key_partitioner",
                        "throughput": 1000,
                        "strategy": {
                            "name": "hashing"
                        }
                    },
                    {
                        "id": 4,
                        "type": "key_partitioner",
                        "throughput": 1000,
                        "strategy": {
                            "name": "hashing"
                        }
                    }
                ]
            },
            {
                "id": 3,
                "type": "stateful",
                "nodes": [
                    {
                        "id": 5,
                        "type": "stateful",
                        "throughput": 1000,
                        "operation_type": "Aggregation",
                        "window_size": 5,
                        "slide": 2
                    },
                    {
                        "id": 6,
                        "type": "stateful",
                        "throughput": 1000,
                        "operation_type": "Aggregation",
                        "window_size": 5,
                        "slide": 2
                    }
```

```
                ]
            }
        ]
    }
}
```

Listing 4.1: JSON Configuration File Example

Οι παράμετροι του αρχείου αυτού είναι οι ακόλουθες:

- `streams`: Το πλήθος των ροών, όπου κάθε ροή, χρησιμοποιεί τη γεννήτρια κλειδιών και παράγει κλειδιά.

- `steps` : Ο αριθμός των βημάτων, που αντιπροσωπεύουν χρονικές στιγμές άφιξης κλειδιών. Η πρώτη παραδοχή που κάνουμε είναι ως προς τον χρόνο, όπου δε χρησιμοποιούμε πραγματικό χρόνο. Αντίθετα, προσομοιώνουμε διακριτό χρόνο, με τη μορφή βημάτων. Ο κάθε ερευνητής, μπορεί να κάνει τη δική του αντιστοίχιση μεταξύ βήματος και χρόνου, σύμφωνα με τις ανάγκες της εκάστοτε προσομοίωσης. Για παράδειγμα, μπορούμε να θεωρήσουμε ότι κάθε βήμα αποτελεί ένα δευτερόλεπτο.

- `number_of_keys` : Το πλήθος των διαφορετικών κλειδιών που θα χρησιμοποιηθούν στην προσομοίωση. Ο στόχος είναι η υλοποίηση ενός αφηρημένου συστήματος, που δεν θα υλοποιεί συγκεκριμένη διεργασία, αλλά θα την προσομοιώνει. Έτσι, δεν μπορεί να υπάρχει μια καθολική μορφή για τα εισερχόμενα δεδομένα. Γι' αυτό το λόγο, χρησιμοποιούμε μόνο τα κλειδιά για να αναπαραστήσουμε τα δεδομένα, καθώς είναι το μοναδικό χαρακτηριστικό των δεδομένων που είναι σταθερό στα συστήματα επεξεργασίας ροών. Δηλαδή, τα δεδομένα έχουν πάντα ένα κλειδί. Ωστόσο, για να μπορούμε να αναπαραστήσουμε διαφορετικά είδη και μεγέθη δεδομένων και κυρίως για να μπορούμε να προσομοιώνουμε διαφορετικές λειτουργίες (για παράδειγμα άθροισμα, ταξινόμηση κ.ά.), χρησιμοποιούμε μία άλλη παράμετρο, την `operation_type`, που δίνει το κόστος σε πολυπλοκότητα για την επεξεργασία των κλειδιών.

- `arrival_rate` & `arrival_rate_ot` : Η παράμετρος `arrival_rate` ρυθμίζει τον ρυθμό άφιξης κλειδιών ανά βήμα προσομοίωσης ενώ η παράμετρος `arrival_rate_ot` ορίζει την ποσοστιαία αύξηση ή μείωση του ρυθμού άφιξης κλειδιών ανά βήμα (χρησιμοποιείται συνήθως για stress test του συστήματος). Σε αντίθεση με τις παραμέτρους σχετικές με το `spike` εδώ έχουμε συνεχόμενη αύξηση ή μείωση του ρυθμού αύξησης ανά βήμα και όχι με πιθανοτική μεταβολή, αλλά σε μια σταθερή ή προβλέψιμη μεταβολή του ρυθμού σε κάθε βήμα.

- `spike_probability` & `spike_magnitude` : Αυτές οι παράμετροι καθορίζουν την πιθανότητα και το μέγεθος μεταβολής του ρυθμού άφιξης των κλειδιών σε κάθε βήμα. Το spike_probability καθορίζει την πιθανότητα εμφάνισης ενός spike, ενώ το spike_magnitude καθορίζει το ποσοστό μεταβολής (αύξηση ή μείωση) του ρυθμού. Η αύξηση ή μείωση γίνεται με χρήση ομοιόμορφης κατανομής από `-spike_magnitude` έως `+spike_magnitude`. Τονίζουμε εδώ ότι, για να αποφευχθεί η υπερβολική μείωση του ρυθμού άφιξης μετά από ένα βήμα, θεωρούμε ότι η ελάχιστη τιμή που μπορεί να λάβει ο ρυθμός άφιξης

μετά την εμφάνιση ενός spike είναι ίση με την αρχική τιμή του arrival_rate που ορίζεται στο αρχείο παραμετροποίησης.

- `distribution` : Περιέχει πληροφορίες σχετικά με την κατανομή των κλειδιών κατά τη δημιουργία τους. Οι πληροφορίες είναι οι εξής:
  - `type`: Ο τύπος της κατανομής (ομοιόμορφη, κανονική, Poisson ή Zipf).
  - `mean`: Η μέση τιμή της κανονικής κατανομής, σε περίπτωση που έχει επιλεχθεί αυτή η κατανομή.
  - `stddev`: Η τυπική απόκλιση της κανονικής κατανομής, σε περίπτωση που έχει επιλεχθεί αυτή η κατανομή.
  - `lambda`: Η παράμετρος λ για την κατανομή Poisson, η οποία καθορίζει τον αναμενόμενο αριθμό γεγονότων σε μια σταθερή χρονική περίοδο ή ανά μονάδα χρόνου.
  - `alpha`: Η παράμετρος α για την κατανομή Zipf, η οποία καθορίζει την ανισοκατανομή των τιμών, δηλαδή πόσο απότομα μειώνονται οι συχνότητες. Χρησιμοποιείται κυρίως για να περιγράψει κατανομές όπου λίγες τιμές εμφανίζονται πολύ συχνά και πολλές τιμές σπάνια, όπως στις λέξεις μιας γλώσσας ή τη δημοτικότητα των αντικειμένων.

Αυτές οι παράμετροι ορίζουν πλήρως τη λειτουργία της γεννήτριας κλειδιών. Ακολουθούν οι παράμετροι που αφορούν την τοπολογία του συστήματος:

- `stages`: Τα στάδια της τοπολογίας. Κάθε στάδιο περιλαμβάνει:
  - `id`: Έναν μοναδικό προσδιοριστή.
  - `type`: Τον τύπο του σταδίου (με ή χωρίς εσωτερική κατάσταση / μνήμη). Ουσιαστικά αφορά τον τύπο των κόμβων επεξεργασίας που το συγκεκριμένο στάδιο περιέχει.
  - `key_splitting`: Μία προαιρετική boolean μεταβλητή που υποδεικνύει αν θα εφαρμοστεί η τεχνική διάσπασης κλειδιών (key splitting) για την αρχικοποίηση ενός κόμβου συνάθροισης (AggregatorNode), όπως θα εξηγηθεί παρακάτω.
  - `nodes`: Οι κόμβοι του κάθε σταδίου.
- node: Κόμβος επεξεργασίας, που διαθέτει τα παρακάτω χαρακτηριστικά:
  - `id`: Έναν μοναδικό προσδιοριστή.
  - `type`: Τον τύπο του κόμβου (με ή χωρίς εσωτερική κατάσταση / μνήμη).
  - `throughput`: Η ρυθμαπόδοση του κόμβου, δηλαδή ο αριθμός των κύκλων επεξεργασίας που μπορεί να εκτελεί σε κάθε βήμα.
  - `operation_type`: Ο τύπος της λειτουργίας που εκτελεί ο κόμβος, η οποία χρησιμοποιείται για τον υπολογισμό του κόστους επεξεργασίας των κλειδιών. Για παράδειγμα, αν εκτελείται ταξινόμηση με n κλειδιά, το κόστος θα είναι $O(n \log n)$ και η τιμή της παραμέτρου θα είναι "Sorting". Με αυτόν τον τρόπο, μπορούμε να προσομοιώσουμε διάφορες διεργασίες.

- `strategy`: Η στρατηγική διαμερισμού που ακολουθείται για την κατανομή των κλειδιών στους επόμενους κόμβους επεξεργασίας. Έχει δύο ιδιότητες:
  - `name`: Το όνομα της στρατηγικής.
  - `prefix_length`: Το μήκος προθέματος για την ομαδοποίηση των κλειδιών, σε περίπτωση χρήσης στρατηγικής KeyGrouping.
- `window_size`: Το μέγεθος του παραθύρου σε βήματα.
- `slide`: Η ολίσθηση του παραθύρου σε βήματα.

#### Γεννήτρια κλειδιών

Η γεννήτρια κλειδιών είναι το συστατικό (component) το οποίο παράγει τα κλειδιά τα οποία τροφοδοτούν την είσοδο του προσομοιωτή. Η βασική της λειτουργία είναι η δημιουργία κλειδιών που ακολοθούν μία συγκεκριμένη κατανομή για ένα πεπερασμένο αριθμό βημάτων προσομοίωσης.

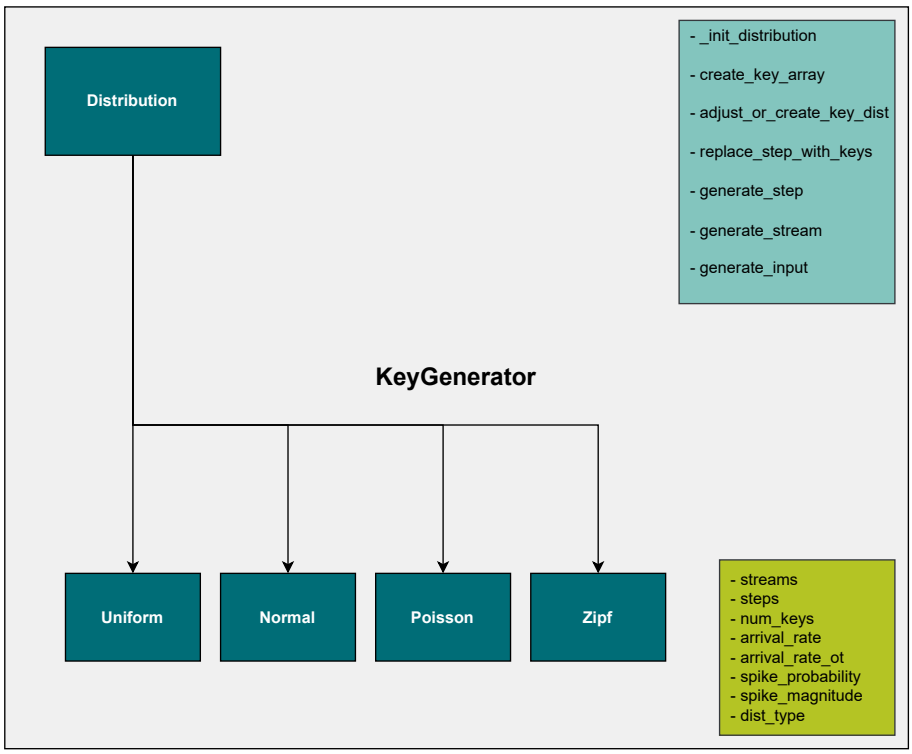
Πιο συγκεκριμένα, οι βασικές παραμετροποίησεις της γεννήτριας κλειδιών από τον χρήστη είναι οι εξής:

- **Ρυθμός Άφιξης:** Ο αριθμός των κλειδιών που παράγονται σε κάθε βήμα προσομοίωσης. Ορίζεται από το arrival_rate.
- **Βήματα Παραγωγής Κλειδιών:** Ορίζει το πόσες φορές η γεννήτρια θα παράξει κλειδιά.
  **Προσοχή!** Ουσιαστικά σε αυτό το σημείο ορίζονται τα βήματα προσομοίωσης (βλπ. steps στο αρχείο παραμετροποίησης).
- **Πλήθος Διαφορετικών Κλειδιών:** Τα διαφορετικά κλειδιά που θα παράγονται ανά βήμα προσομοίωσης. Αναπαρίστανται ως $\texttt{key}_i$ όπου $i$ από 0 έως το πλήθος των διαφορετικών κλειδιών. Όπως αναφέραμε και προηγουμένως μας ενδιαφέρουν μόνο τα διαφορετικά κλειδιά κατά την διάρκεια της προσομοίωσης και όχι η τιμή του κάθε κλειδιού. Περισσότερα για αυτή την επιλογή στην Ενότητα 4.3. Το σύνολο των διαφορετικών κλειδιών θεωρούμε ότι δεν αλλάζει κατά την διάρκεια της προσομοίωσης παρά μόνο η συχνότητα εμφάνισης τους όπως θα δούμε στη συνέχεια (βλπ. number_of_keys στο αρχείο παραμετροποίησης).
- **Κατανομή κλειδιών:** Η γεννήτρια κλειδιών μπορεί να παράγει κλειδιά με ομοιόμορφη, κανονική, Poisson ή Zipf κατανομή, προσαρμόζοντας τις αντίστοιχες παραμέτρους κάθε κατανομής. Η επιλογή αυτών των κατανομών επιτρέπει τη δημιουργία είτε ισοκατανεμημένου φόρτου εργασίας μέσω της ομοιόμορφης κατανομής, είτε στοχευμένου φόρτου εργασίας μέσω των υπολοίπων. Για περίσσότερες λεπτομέρειες αναφορικά με την κατανομή κλειδιών βλέπε distribution στο αρχείο παραμετροποίησης.

Η γεννήτρια κλειδιών πέρα από τις βασικές παραμετροποιήσεις που αναφέραμε και προηγουμένως παρέχει και την δυνατότητα πιο εξεζητημένων παραμέτρων ώστε να μπορούν να καταστρωθούν πιο ρεαλιστικές προσομοιώσεις.

- **Μεταβαλλόμενος Ρυθμός Αφίξης:** Μία σημαντική εκτίμηση όλων των συστημάτων συνεχούς ροής είναι να μπορεί να εκτιμηθεί το σημείο που το σύστημα υπερφορτώνεται. Για τον λόγο, μέσω του arrival_rate_ot, μπορούμε να ορίσουμε την ποσοστιαία αύξηση του ρυθμού άφιξης κλειδιών.
- **Spiked Παραγόμενα Δεδομένα:** Μία άλλη σημαντική μετρική για την απόδοση του συστήματος μέσω της προσομοίωσης είναι η αντιμετώπιση spike (αιχμών) στα δεδομένα. Για να εξετάσουμε την συμπεριφορά του προσομοιωμένου συστήματος σε ενδεχομένα spikes, δίνεται η δυνατότητα να οριστεί η πιθανότητα εμφάνισης ενός τέτοιου καθώς και το μέγεθος αυτού (βλπ. spike).

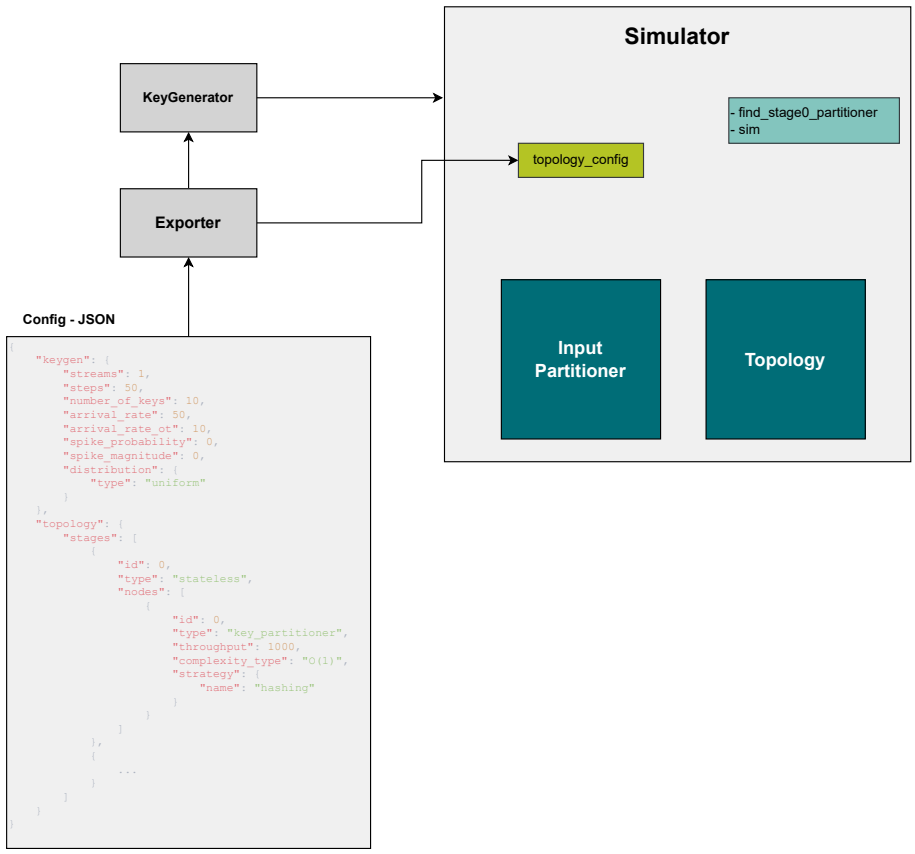
Το Σχήμα 4.1 παρουσιάζει τη δομή και τα βασικά στοιχεία της γεννήτριας κλειδιών.




Σχήμα 4.1: *Γεννήτρια Κλειδιών*

### Προσομοιωτής

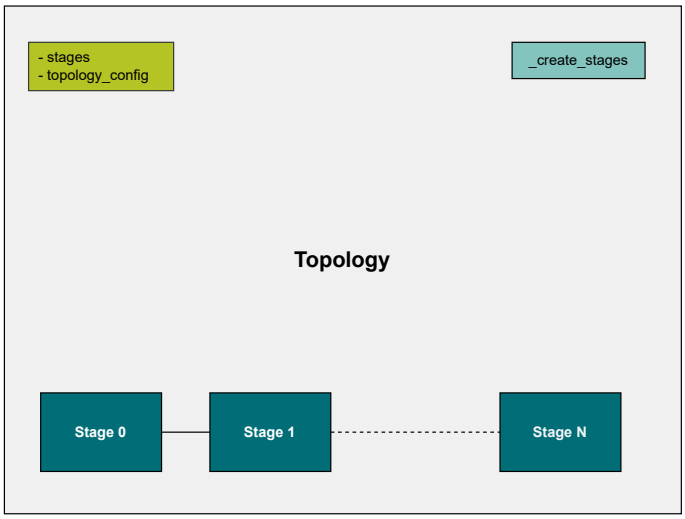
Ο προσομοιωτής αποτελεί το πιο αφηρημένο συστατικό. Αρχικά, εξάγει πληροφορίες από το αρχείο παραμετροποίησης προκειμένου στη συνέχεια να αρχικοποιήσει την τοπολογία με τα στάδια και τους κόμβους. Στην τοπολογία, το πρώτο στάδιο περιέχει πάντα έναν μόνο κόμβο χωρίς εσωτερική κατάσταση που ονομάζεται κατανεμητής κλειδιών (KeyPartitioner). Μέσω αυτού, ο προσομοιωτής λαμβάνει τα παραγόμενα από τη γεννήτρια κλειδιά και τα κατανέμει και τα στέλνει στα επόμενα στάδια και στους επόμενους κόμβους. Η υπόλοιπη εργασία γίνεται εσωτερικά στα στάδια και τους κόμβους. Συνεπώς, ο προσομοιωτής επιτελεί την αρχικοποίηση του συστήματος. Το Σχήμα 4.2 παρουσιάζει την αρχιτεκτονική του συστήματος του προσομοιωτή μαζί με τα διάφορα συστατικά του και τις διάφορες λειτουργίες του.




Σχήμα 4.2: *Προσομοιωτής*

### Τοπολογία

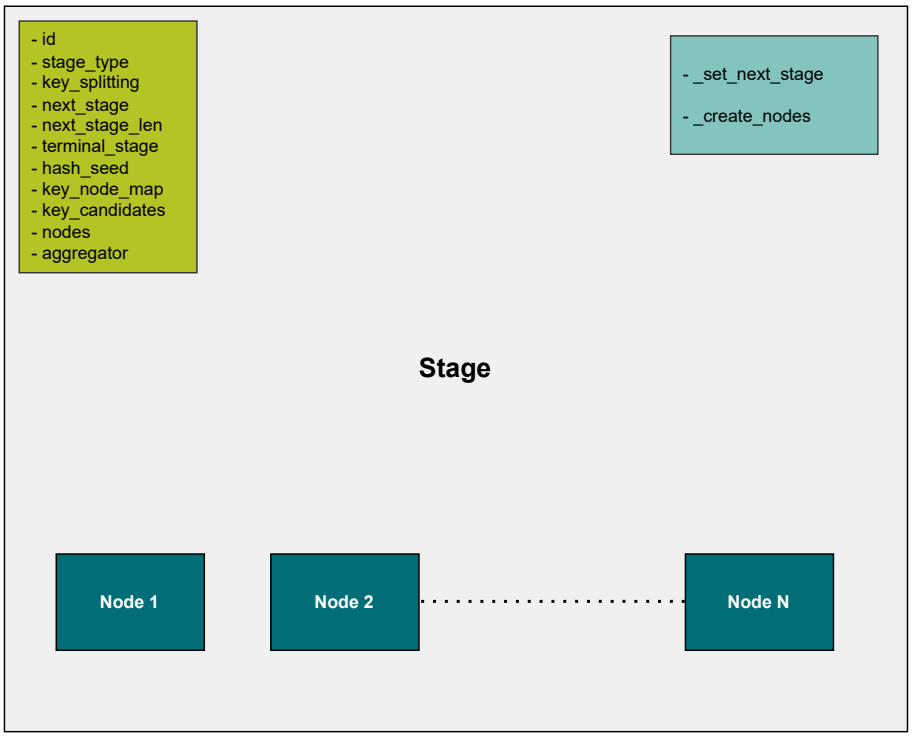
Η λειτουργία της τοπολογίας είναι απλή. Λαμβάνει όλες τις απαραίτητες πληροφορίες από τον προσομοιωτή και δημιουργεί τα διάφορα στάδια, παρέχοντάς τους κατάλληλες πληροφορίες σχετικά με τους κόμβους. Το Σχήμα 4.3 δίνει μία αφηρημένη απεικόνιση της τοπολογίας.




Σχήμα 4.3: *Τοπολογία*

### Στάδιο

Το στάδιο αντίστοιχα, λαμβάνει πληροφορίες από την τοπολογία και αρχικοποιεί τους κόμβους του, παρέχοντας λεπτομέρειες για την υλοποίηση τους (είδος, ρυθμαπόδοση κ.λπ.). Το Σχήμα 4.4 δίνει μία αφηρημένη απεικόνιση του Σταδίου.




Σχήμα 4.4: *Στάδιο*

### Κόμβος

Ο κόμβος έχει υλοποιηθεί αφηρημένα και υπάρχουν δύο είδη κόμβων: αυτός με εσωτερική κατάσταση και αυτός χωρίς:

### Κόμβος Χωρίς Εσωτερική Κατάσταση - Κατανεμητής Κλειδιών

Ο κατανεμητής κλειδιών είναι υπεύθυνος για την κατανομή των κλειδιών στους επόμενους κόμβους επεξεργασίας. Σε κάθε χρονικό βήμα, λαμβάνει κλειδιά που είτε παρήχθησαν από τη γεννήτρια (αν πρόκειται για τον εναρκτήριο κόμβο) είτε προήλθαν από έναν προηγούμενο κόμβο. Στη συνέχεια, τα διαμερίζει στους επόμενους κόμβους με βάση τη στρατηγική διαμερισμού που έχει καθοριστεί. Οι στρατηγικές που έχουν υλοποιηθεί είναι οι ακόλουθες:

- Hashing: Κάθε κλειδί περνάει από μία συνάρτηση κατακερματισμού η οποία δίνει ως έξοδο τον προσδιοριστή (id) του κόμβου στον οποίο θα κατευθυνθεί το κλειδί. Έτσι, κάθε μοναδικό κλειδί, κατευθύνεται πάντα στον ίδιο κόμβο.

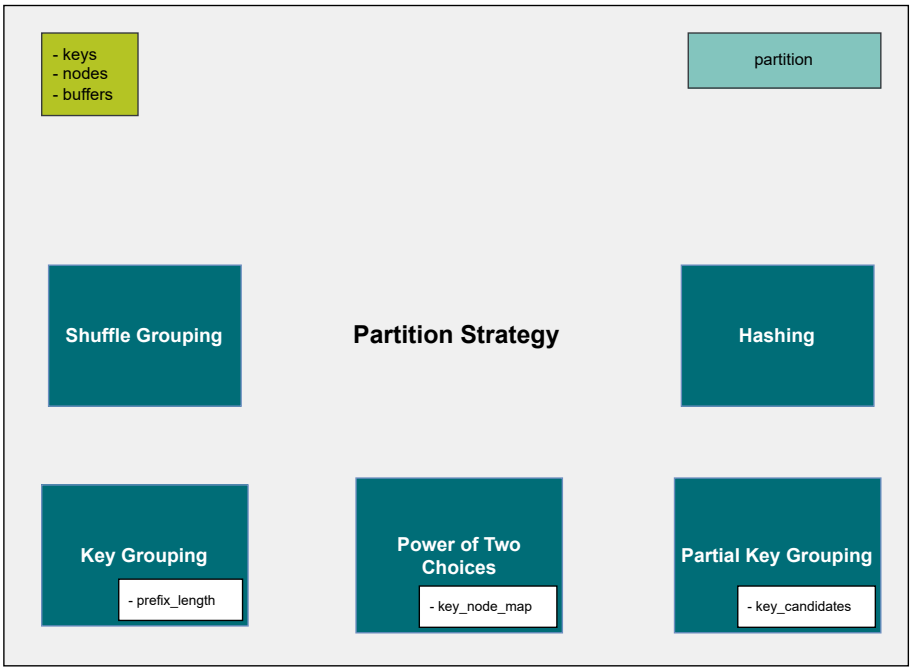
- Shuffle Grouping: Ακολουθεί λογική round robin στέλνωντας κάθε κλειδί σε κόμβους με κυκλική σειρά.

- Key Grouping: Ακολουθείται παρόμοια λογική με τη στρατηγική hashing, με τη διαφορά ότι εδώ επιλέγεται ένα μήκος προθέματος και περνάει μόνο το πρόθεμα από τη συνάρτηση κατακερματισμού και όχι ολόκληρο το κλειδί. Αυτό έχει ως αποτέλεσμα, κλειδιά με ίδιο πρόθεμα, να ομαδοποιούνται και να στέλνονται πάντα στον ίδιο κόμβο.

- Power of Two Choices (PoTC): Για κάθε κλειδί, κατά την πρώτη εμφάνισή του, υπολογίζονται δύο υποψήφιοι κόμβοι μέσω δύο διαφορετικών συναρτήσεων κατακερματισμού, και επιλέγεται ο κόμβος με το μικρότερο φόρτο, όπως αναλύθηκε στην Ενότητα 2.7.3.

- Partial Key Grouping (PKG): Για κάθε κλειδί, κατά την πρώτη εμφάνισή του, υπολογίζονται δύο κόμβοι μέσω δύο διαφορετικών συναρτήσεων κατακερματισμού, και επιλέγονται και οι δύο. Κάθε φορά που το κλειδί εμφανίζεται ξανά, αποστέλλεται στον κόμβο που έχει το μικρότερο φόρτο εκείνη τη στιγμή, όπως εξηγήθηκε στην Ενότητα 2.7.3.

Το Σχήμα 4.5, παρουσιάζει τις στρατηγικές διαμερισμού του KeyParitioner.

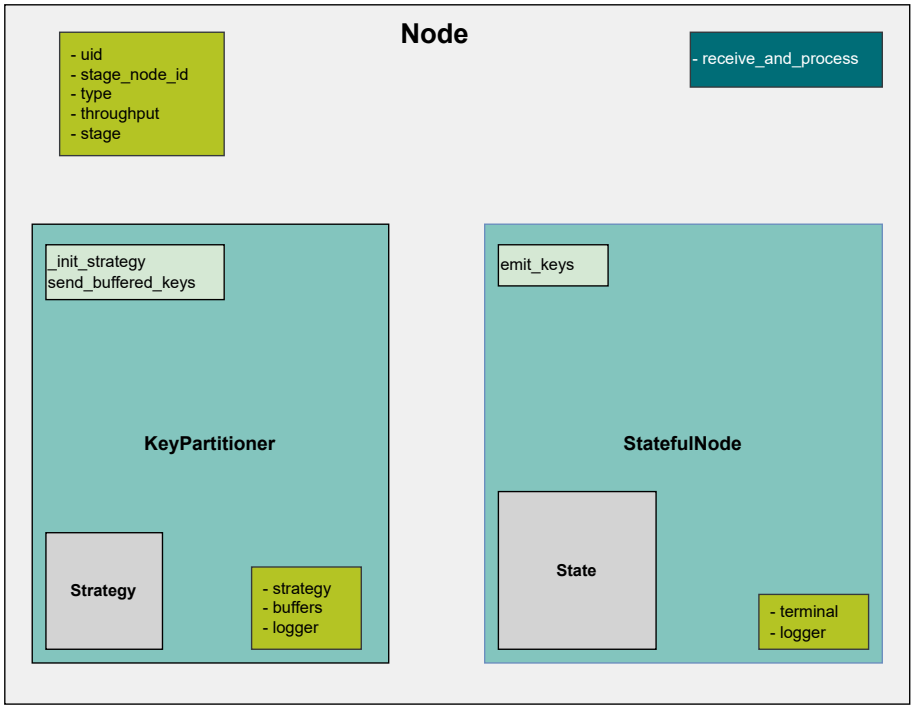



Σχήμα 4.5: *Στρατηγικές Διαμερισμού*

### Κόμβος Με Εσωτερική Κατάσταση

Ο κόμβος με εσωτερική κατάσταση είναι πιο σύνθετος και αφήνει την εσωτερική κατάσταση να εκτελέσει τις σημαντικότερες ενέργειες. Είναι υπεύθυνος για την αρχικοποίηση της εσωτερικής του κατάστασης και κάθε φορά που έρχονται κλειδιά σε αυτόν, χρησιμοποιεί τις λειτουργίες της εσωτερικής κατάστασης για να την ενημερώνει κατάλληλα και να επεξεργάζεται τα κλειδιά.

Το Σχήμα 4.6 απεικονίζει την αφηρημένη κλάση του Κόμβου (Node) καθώς και τις παραγώμενες κλάσεις του κόμβου με και χωρίς κατάσταση (StatefulNode και StatelessNode/KeyPartitioner) μαζί με τα πεδία και τις μεθόδους τους.




Σχήμα 4.6: *Κόμβοι*

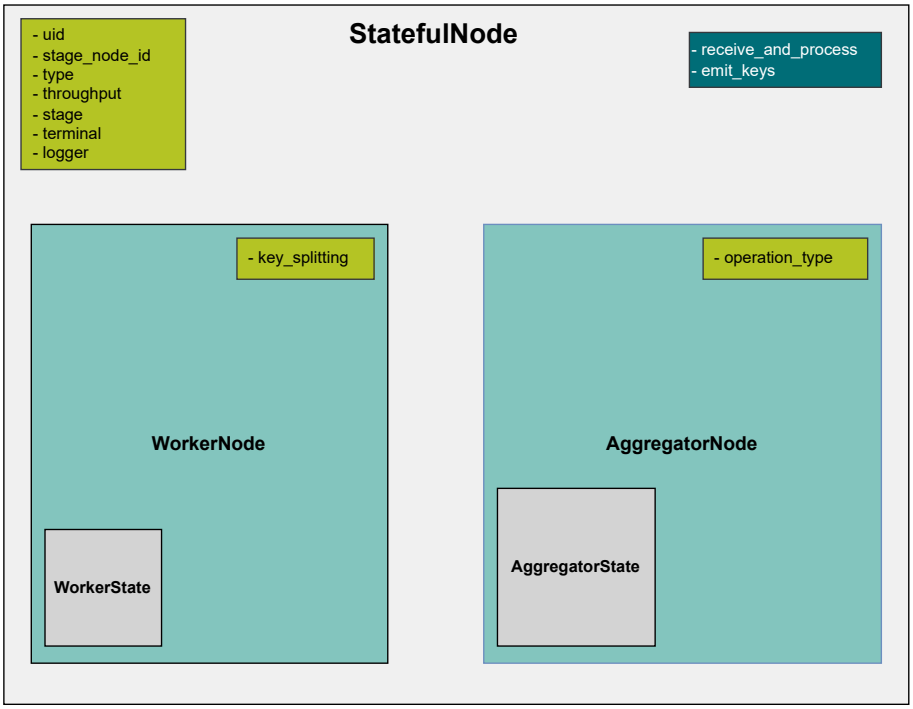
### Κόμβοι με Εσωτερική Κατάσταση – Worker & Aggregator Nodes

Όπως έχει ήδη αναφερθεί, οι κόμβοι με εσωτερική κατάσταση διακρίνονται σε δύο κατηγορίες. Ο πρώτος τύπος είναι ο κόμβος «εργάτης» (WorkerNode), ο οποίος συνήθως αναλαμβάνει τις βασικές και πιο χρονοβόρες λειτουργίες, όπως η ταξινόμηση. Ο δεύτερος τύπος είναι ο «συναθροιστής» (AggregatorNode), που εκτελεί λειτουργίες συνάθροισης.

Ο WorkerNode είναι απαραίτητος σε κάθε επεξεργασία και στο τέλος της διαδικασίας είτε μεταφέρει τα κλειδιά σε έναν Key Partitioner για να τα κατανείμει στους κόμβους του επόμενου σταδίου, είτε τα αποστέλλει σε έναν AggregatorNode, ο οποίος θα τα διαβιβάσει με τη σειρά του σε έναν Key Partitioner αφού ολοκληρώσει τις δικές του λειτουργίες.

Αντίθετα, ο AggregatorNode απαιτείται μόνο σε περιπτώσεις εφαρμογής key splitting, δηλαδή όταν χρησιμοποιούνται τεχνικές διαμερισμού όπως το shuffle grouping ή το partial key grouping. Σε αυτές τις περιπτώσεις, τα ίδια κλειδιά βρίσκονται σε περισσότερους από έναν κόμβους και απαιτείται η συγχώνευσή τους σε έναν κοινό κόμβο για να γίνει ο τελικός υπολογισμός. Συνεπώς, ο AggregatorNode δεν δηλώνεται άμεσα στην τοπολογία, αλλά έμμεσα μέσω της παραμέτρου key_splitting του σταδίου. Το στάδιο αυτό αναλαμβάνει την κατάλληλη αρχικοποίηση του AggregatorNode όποτε είναι αναγκαίο.

Το Σχήμα 4.7 απεικονίζει τις δύο κατηγορίες των κόμβων με εσωτερική κατάσταση, τον WorkerNode και τον AggregatorNode:




Σχήμα 4.7: *Κόμβοι με Εσωτερική Κατάσταση*

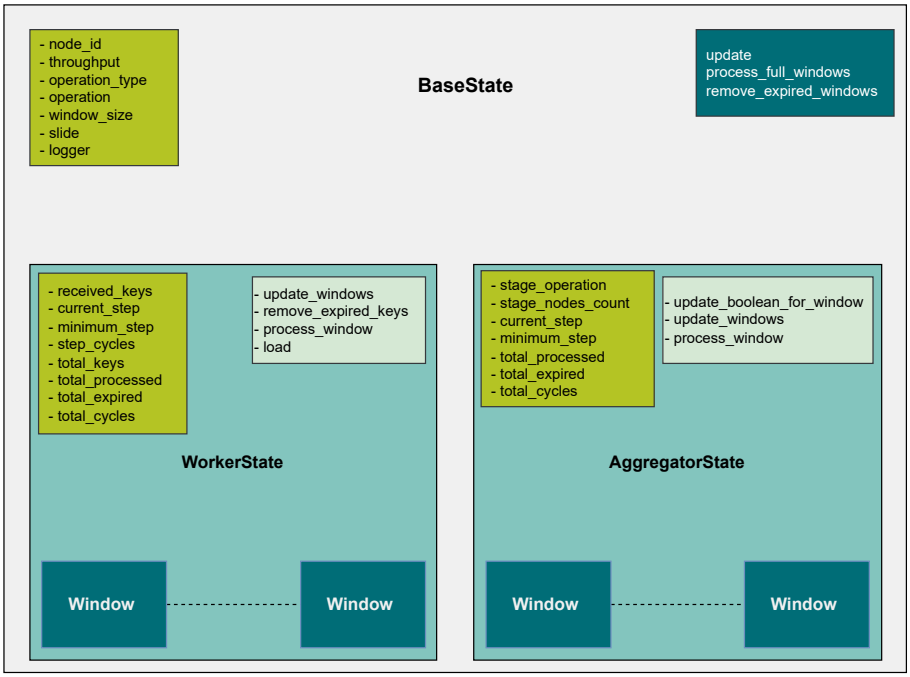
### Εσωτερική κατάσταση

Η εσωτερική κατάσταση είναι από τα σημαντικότερα συστατικά του συστήματος. Κάθε φορά που έρχονται νέα κλειδιά, η εσωτερική κατάσταση αναλαμβάνει την εκτέλεση των απαραίτητων ενεργειών. Πιο συγκεκριμένα, ενημερώνει τα τρέχοντα κυλιόμενα παράθυρα, προσθέτοντάς τους κλειδιά, ενώ παράλληλα δημιουργεί νέα. Ακόμη, διαγράφει τα παράθυρα που έχουν λήξει και επεξεργάζεται όσα έχουν γεμίσει. Τέλος, διατηρεί και ενημερώνει μετρικές όπως είναι το συνολικό πλήθος κλειδιών που έχουν ληφθεί, το συνολικό κόστος σε κύκλους επεξεργασίας που έχει δαπανηθεί για την επεξεργασία των κλειδιών, το πλήθος των τρεχόντων παραθύρων μαζί με τα κλειδιά τους και άλλα.

Παρόλο που τα δύο είδη Stateful κόμβων εκτελούν παρόμοιες λειτουργίες, υπάρχουν κάποιες σημαντικές διαφορές μεταξύ τους. Γι' αυτόν τον λόγο, έχουν οριστεί δύο διαφορετικές κλάσεις εσωτερικής κατάστασης: οι WorkerState και AggregatorState για τους WorkerNode και AggregatorNode, αντίστοιχα. Οι κλάσεις αυτές κληρονομούν από μία αφηρημένη κλάση, την BaseState. Το Σχήμα 4.8 απεικονίζει τις λειτουργίες και τα συστατικά αυτών των κλάσεων.

### Παράθυρα

Προκειμένου ο προσομοιωτής να είναι πιο ρεαλιστικός, έχουμε υλοποιήσει την έννοια του κυλιόμενου παραθύρου (sliding window). Κάθε παράθυρο έχει ένα μέγεθος (window size) που δηλώνει το πλήθος των χρονικών βημάτων για τα οποία παραμένει ενεργό και μία ολίσθηση (slide), η οποία καθορίζει πόσο συχνά το παράθυρο μετακινείται ή ανανεώνεται. Αυτό επιτρέπει την επεξεργασία των δεδομένων με συνεχή και επικαλυπτόμενο τρόπο, καλύπτοντας περισσότερα χρονικά διαστήματα και βελτιώνοντας την απόδοση του συστήματος.




Σχήμα 4.8: *Εσωτερικές Καταστάσεις*

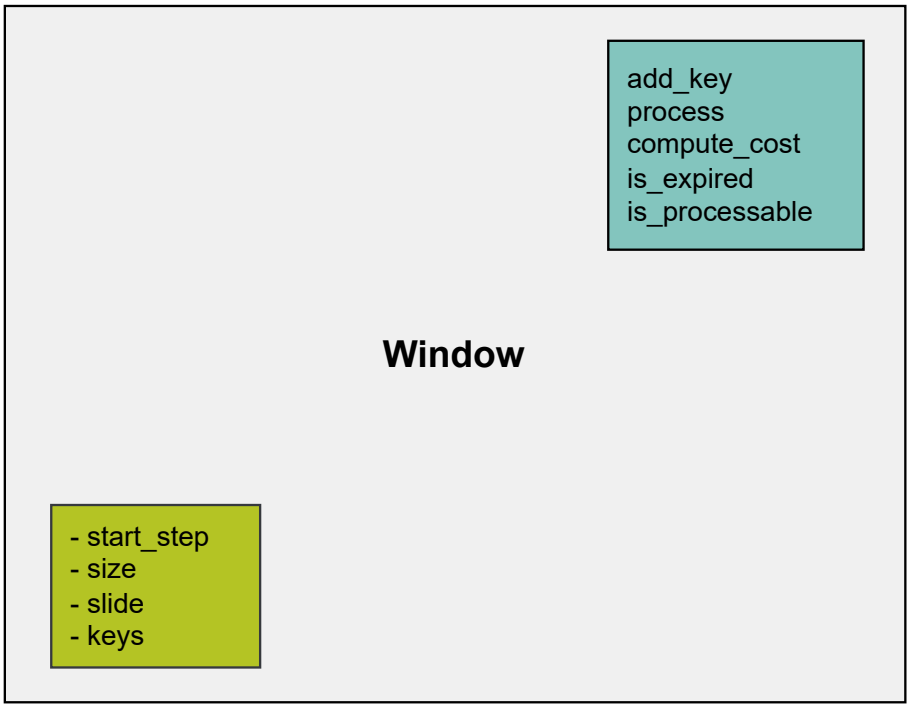
Από τα τρία είδη παραθύρων [14, 13, 37, 30], δηλαδή τα tumbling, session και sliding , αποφασίσαμε να υλοποιήσουμε το sliding window. Αυτό οφείλεται στο γεγονός ότι τα sliding windows προσφέρουν μεγαλύτερη ευελιξία, καθώς μπορούν να επεξεργάζονται τα δεδομένα συνεχώς, επιτρέποντας παράλληλα την επικαλυπτόμενη επεξεργασία διαφορετικών χρονικών βημάτων, κάτι που είναι κρίσιμο σε εφαρμογές συνεχούς ροής δεδομένων.

Σαν trigger policy, έχουμε επιλέξει κάθε παράθυρο να επεξεργάζεται τα δεδομένα του κάθε window size βήματα προσομοίωσης, γεγονός που καθιστά την πολιτική μας time-based. Αυτό σημαίνει ότι η επεξεργασία γίνεται σε συγκεκριμένα χρονικά διαστήματα, ανεξάρτητα από το αν τα δεδομένα φτάσουν με καθυστέρηση.

Όσον αφορά την eviction policy, επιλέξαμε να διαγράφεται κάθε παράθυρο είτε κατά τη στιγμή της επεξεργασίας του, είτε όταν περάσουν $window\_size + 3 \times slide$ βήματα προσομοίωσης, καθιστώντας την πολιτική μας time-based. Τα κλειδιά που δεν έχουν επεξεργαστεί εγκαίρως, δηλαδή κατά τα πρώτα window size βήματα από τη στιγμή άφιξής τους, θεωρούνται εκπρόθεσμα (overdue) και διατηρούνται για επιπλέον $3 \times slide$ βήματα. Εάν δεν καταφέρουν να επεξεργαστούν ούτε εντός αυτού του επιπλέον χρόνου, θεωρούνται ότι έχουν λήξει και το παράθυρο διαγράφεται. Το Σχήμα 4.9 απεικονίζει την κλάση Window.

## 4.2 Ροή και Λειτουργία του Προσομοιωτή

Στην ενότητα αυτή περιγράφεται η ροή και η λειτουργία του συστήματος του προσομοιωτή.




Σχήμα 4.9: *Παράθυρο*

### 4.2.1 Ροή

Αρχικά, διαβάζεται το αρχείο παραμετροποίησης και αρχικοποιείται η Γεννήτρια Κλειδιών, η οποία παράγει τα κλειδιά σύμφωνα με τις παραμέτρους που αναλύθηκαν προηγουμένως. Στη συνέχεια, ο Προσομοιωτής αρχικοποιείται σύμφωνα με την τοπολογία που έχει οριστεί. Ο Προσομοιωτής δημιουργεί την Τοπολογία, η οποία με τη σειρά της αρχικοποιεί τα διάφορα Στάδια. Τα Στάδια δημιουργούν και αρχικοποιούν τους Κόμβους επεξεργασίας τους. Οι Κόμβοι - Κατανεμητές Κλειδιών αρχικοποιούν τις στρατηγικές κατανομής των κλειδιών, ενώ οι Κόμβοι με Εσωτερική Κατάσταση αρχικοποιούν την εσωτερική τους κατάσταση. Αφού ολοκληρωθεί η αρχικοποίηση, ο Προσομοιωτής ξεκινά την προσομοίωση μέσω της μεθόδου `sim`, περνώντας ως παράμετρο τα παραχθέντα κλειδιά.

Κατά την έναρξη της προσομοίωσης, ο Προσομοιωτής εντοπίζει τον αρχικό Κατανεμητή Κλειδιών στο Στάδιο 0 και μέσω αυτού κατανέμει τα κλειδιά στους κόμβους των επόμενων σταδίων. Σε κάθε βήμα, όταν τα κλειδιά φτάνουν σε έναν κόμβο με εσωτερική κατάσταση, εκτελούνται οι ακόλουθες ενέργειες:

- Πρώτα, ενημερώνονται τα πεδία του κόμβου, όπως το συνολικό πλήθος κλειδιών (`total_keys`) και το τρέχον βήμα της προσομοίωσης (`current_step`).

- Στη συνέχεια, ενημερώνεται η εσωτερική κατάσταση με τα παράθυρα: είτε προσθέτοντας τα ληφθέντα κλειδιά σε υπάρχοντα παράθυρα, είτε δημιουργώντας νέα παράθυρα.

- Έπειτα, επεξεργάζονται τα γεμάτα παράθυρα, δηλαδή αυτά για τα οποία έχουν περάσει, από τη δημιουργία τους, τουλάχιστον τόσα βήματα όσα το μέγεθος του παραθύρου (`window_size`). Η επεξεργασία πραγματοποιείται με βάση την πολυπλοκότητα

που προσομοιώνεται σε υπολογιστικούς κύκλους, ενώ περιορίζεται από τη ρυθμαπόδοση (`throughput`), το οποίο καθορίζει το μέγιστο πλήθος κύκλων ανά βήμα.

- Αν τα παράθυρα δεν προλάβουν να επεξεργαστούν πλήρως, τα εναπομείναντα κλειδιά τους θεωρούνται εκπρόθεσμα και προστίθεται επιπλέον χρόνος, ο οποίος είναι συνήθως πολλαπλάσιο της ολίσθησης του παραθύρου, ώστε να ολοκληρωθεί η επεξεργασία σε επόμενα βήματα. Μετά το πέρας αυτού του χρονικού διαστήματος, τα παράθυρα και τα κλειδιά τους θεωρούνται ληγμένα και αφαιρούνται.

- Η επεξεργασία των κλειδιών γίνεται με τη σειρά που εμφανίζονται στο παράθυρο και ομαδοποιούνται ανάλογα με το κλειδί τους. Τα παραγόμενα κλειδιά μπορεί να είναι είτε όλα τα μοναδικά κλειδιά του παραθύρου είτε όλα τα κλειδιά ταξινομημένα. Αυτό εξαρτάται από τη λειτουργία του σταδίου: αν είναι συνάθροιση (aggregation), διατηρούνται μόνο τα μοναδικά κλειδιά, ενώ αν είναι ταξινόμηση (sorting), τα κλειδιά επιστρέφονται σε ταξινομημένη σειρά.

- Αφού επεξεργαστούν τα παράθυρα, αφαιρούνται τα ληγμένα παράθυρα και τα κλειδιά. Καθ' όλη τη διάρκεια, ενημερώνονται οι μετρικές του κόμβου και αποθηκεύονται σε αρχεία καταγραφής (`logs`).

- Τέλος, τα παραγόμενα κλειδιά αποστέλλονται στο επόμενο στάδιο, όπου κατανεμητές κλειδιών τα διανέμουν στους επόμενους κόμβους.

Η ροή των κλειδιών συνεχίζεται έως ότου φτάσουν σε τερματικούς κόμβους. Η προσομοίωση ολοκληρώνεται όταν παραχθούν και κατανεμηθούν όλα τα κλειδιά του τελευταίου βήματος από τη Γεννήτρια Κλειδιών.

## 4.3 Συμβάσεις και Παραδοχές

Στην ενότητα αυτή γίνεται αναφορά σε ορισμένες συμβάσεις και παραδοχές που υιοθετούνται στο πλαίσιο της παρούσας διπλωματικής εργασίας.

1. **Χρόνος:** Όπως αναφέρθηκε και νωρίτερα, για την προσομοίωση του χρόνου χρησιμοποιούμε διακριτά βήματα και όχι συνεχή χρόνο. Ο κάθε ερευνητής, μπορεί να κάνει τη δική του αντιστοίχιση μεταξύ βήματος και χρόνου, σύμφωνα με τις ανάγκες της εκάστοτε προσομοίωσης.

2. **Κλειδιά:** Σχετικά με την υλοποίηση των κλειδιών στο σύστημα μας κάναμε ορισμένες παραδοχές. Πρώτον, εφόσον υλοποιούμε έναν προσομοιωτή και όχι ένα πραγματικό σύστημα επεξεργασίας συνεχών ροών δεδομένων σε πραγματικό χρόνο, θεωρήσαμε την αναπαράσταση των τιμών των κλειδιών αδιάφορες εφόσον δεν στοχεύουμε στην αποτίμηση αποτελεσμάτων συγκεκριμένων διεργασιών αλλά στη μέτρηση της επίδοσης των εκάστοτε συστημάτων. Θεωρήσαμε ότι για τα πλαίσια του προσομοιωτή, μας ενδιαφέρει μόνο η πολυπλοκότητα του κάθε τελεστή σε σχέση με τον αριθμό των κλειδιών. Η δεύτερη παραδοχή που κάναμε αφορά την παραγωγή και την κατανάλωση των κλειδιών από το σύστημα. Θεωρήσαμε ότι όλα τα κλειδιά καταφθάνουν εντός σειράς, δηλαδή

δεν υπάρχει εκτέλεση εκτός σειράς σε κανέναν κόμβο. Η επιλογή αυτή έγινε λόγω των αυξημένων δυσκολιών που θα προέκυπταν από την υποστήριξη εκτέλεσης εκτός σειράς. Ωστόσο, σε μια μελλοντική αναβάθμιση του προσομοιωτή, αυτή η λειτουργία θα μπορούσε να αποτελεί μία από τις νέες δυνατότητες που θα εξεταστούν.

3. **Ιεραρχία Κλειδιών:** Η ιεραρχία εμφάνισης των κλειδιών ορίζεται αυθαίρετα κατά την αρχικοποίηση της γεννήτριας κλειδιών. Θεωρούμε ότι η ιεραρχία αυτή κατά την πρόοδο των βημάτων δεν αλλάζει αυθαίρετα, αλλά η ιεραρχία συχνότητας του κάθε κλειδιού μπορεί να αυξομειωθεί το πολύ κατά μία θέση.

4. **Τοπολογία - Κόμβοι:** Επιλέξαμε να θεωρήσουμε τους Κατανεμητές Κλειδιών (KeyPartitioners) ως κόμβους χωρίς εσωτερική κατάσταση (Stateless Nodes) ώστε να υπάρχει σαφής διαχωρισμός μεταξύ των κόμβων επεξεργασίας, δηλαδή αυτών με εσωτερική κατάσταση, και τους υπόλοιπους κόμβους που δεν έχουν εσωτερική κατάσταση. Ως αποτέλεσμα, η λειτουργία της κατανομής κλειδιών έχει αποσπαστεί πλήρως από τους Stateful Κόμβους και υλοποιείται ως ένα ξεχωριστό στάδιο, το οποίο ακολουθεί αμέσως μετά το στάδιο των Stateful Κόμβων.

5. **Κατανεμητής Κλειδιών - Στάδιο 0 Τοπολογίας:** Στον προσομοιωτή μας θεωρήσαμε ότι το πρώτο στάδιο της τοπολογίας μας είναι πάντα ένας Κατανεμητής Κλειδιών. Η επιλογή αυτή είναι μία εύλογη παραδοχή με βάση τα πραγματικά συστήματα στα οποία πάντα το πρώτο στάδιο είναι το στάδιο εισαγωγής δεδομένων στο σύστημα. Εμείς διαχωρίσαμε την γεννήτρια κλειδιών από τους κόμβους τους συστήματος και θεωρήσαμε ότι επικοινωνεί απευθείας με το πρώτο στάδιο (Στάδιο 0) της τοπολογίας.

6. **Κυλιόμενο Παράθυρο - Εκπρόθεσμα Κλειδιά:** Σε κάθε κόμβο έχουμε καθορίσει τη μέγιστη ρυθμαπόδοση που μπορεί να επιτύχει. Σε περίπτωση που αυτή ξεπεραστεί λόγω αυξημένης ροής κλειδιών, θεωρούμε τα υπολειπόμενα κλειδιά εκπρόθεσμα εφόσον δεν μπορούν να επεξεργαστούν στο τρέχον βήμα. Τα εκπρόθεσμα κλειδιά διατηρούνται για ένα επιπλέον διάστημα ίσο με $3 \times slide$ βήματα. Εάν δεν καταφέρουν να επεξεργαστούν ούτε εντός αυτού του διαστήματος, τότε θεωρούνται ότι έχουν λήξει και το παράθυρο στο οποίο ανήκουν διαγράφεται. Αυτή η παραδοχή είναι λογική, καθώς σε πραγματικά συστήματα υπάρχει συνήθως αυτός ο επιπλέον χρόνος, ο οποίος συνήθως είναι πολλαπλάσιο είτε του παραθύρου είτε της ολίσθησης και χρησιμοποιείται για την επεξεργασία κλειδιών εκτός σειράς (out of order execution).

Κεφάλαιο 5

# Πειραματική Διάταξη

Στο κεφάλαιο αυτό περιγράφεται λεπτομερώς η πειραματική διάταξη που ακολουθήθηκε για την αξιολόγηση της απόδοσης του εργαλείου προσομοίωσης κατανεμημένων συστημάτων επεξεργασίας συνεχών ροών δεδομένων σε πραγματικό χρόνο. Η πειραματική διάταξη χωρίζεται σε δύο βασικές κατηγορίες: τη διαμόρφωση των παραμέτρων της προσομοίωσης και τα διαφορετικά σενάρια λειτουργίας που εξετάστηκαν.

## 5.1 Εισαγωγή

Τα πειράματα που πραγματοποιήθηκαν στο πλαίσιο αυτής της διπλωματικής εργασίας είχαν ως στόχο τον έλεγχο της ορθότητας του προσομοιωτή και την εξέταση διαφόρων συνθηκών λειτουργίας ενός κατανεμημένου συστήματος επεξεργασίας συνεχών ροών δεδομένων. Μέσα από διαφορετικές παραμετροποιήσεις του συστήματος, αξιολογείται η απόδοση του προσομοιωμένου περιβάλλοντος σε σενάρια που προσομοιώνουν πραγματικές καταστάσεις, όπως η αυξανόμενη άφιξη δεδομένων, η ανισοκατανομή κλειδιών, οι ξαφνικές αυξομειώσεις φόρτου εργασίας (spikes) η χρήση διαφορετικών τεχνικών διαμερισμού των κλειδιών και η κλιμάκωση του συστήματος με την προσθήκη επιπλέον κόμβων επεξεργασίας.

## 5.2 Διαμόρφωση των Παραμέτρων Προσομοίωσης

Η διαμόρφωση της προσομοίωσης πραγματοποιήθηκε μέσω του αρχείου παραμετροποίησης, το οποίο περιέχει όλες τις βασικές παραμέτρους για τη δημιουργία δεδομένων, τον ρυθμό άφιξης, την κατανομή τους, καθώς και την τοπολογία του συστήματος, όπως περιγράφεται αναλυτικά στην Ενότητα 4.1.2. Αυτές οι παράμετροι καθορίζουν τη συμπεριφορά του προσομοιωμένου συστήματος σε κάθε πείραμα, με τις περισσότερες από αυτές να προσαρμόζονται ανάλογα με το εκάστοτε σενάριο.

Οι βασικές παράμετροι που χρησιμοποιούνται στην προσομοίωση είναι οι εξής:

- **Ρυθμός Άφιξης (`arrival_rate`)**: Ο ρυθμός άφιξης καθορίζει πόσα κλειδιά παράγονται ανά χρονικό βήμα. Για παράδειγμα, η τιμή 50 σημαίνει ότι παράγονται 50 κλειδιά σε κάθε βήμα, τα οποία στη συνέχεια διανέμονται στους κόμβους επεξεργασίας. Η παράμετρος αυτή προσαρμόζεται ανάλογα με το σενάριο που εξετάζεται, όπως στα πειράματα που προσομοιώνουν αυξανόμενο ρυθμό άφιξης ή ξαφνικές αυξήσεις (spikes) στον ρυθμό.

- **Πλήθος Κλειδιών (`number_of_keys`)**: Αυτή η παράμετρος καθορίζει τον αριθμό των διαφορετικών κλειδιών που θα χρησιμοποιηθούν στην προσομοίωση. Ο αριθμός αυτός προσαρμόζεται ώστε να προσομοιωθούν διαφορετικά φορτία εργασίας, καθώς διαφορετικός αριθμός κλειδιών μπορεί να δημιουργήσει άνισο φόρτο στους κόμβους επεξεργασίας.

- **Κατανομή (`distribution`)**: Αυτή η παράμετρος ρυθμίζει τον τρόπο διανομής των κλειδιών στη ροή δεδομένων. Ορίζεται είτε ως ομοιόμορφη, με όλα τα κλειδιά να έχουν ίση πιθανότητα εμφάνισης, είτε ως κανονική, Poisson, ή Zipf κατανομή. Οι τελευταίες τρεις επιλογές είναι ιδανικές για την προσομοίωση ανισοκατανομής των κλειδιών με συγκεκριμένα χαρακτηριστικά, προσαρμόζοντας τις παραμέτρους τους (mean, stddev, lambda, alpha), για να μελετηθούν διάφορα επίπεδα skewness.

- **Παράμετροι Spike**: Οι παράμετροι, `spike_probability` και `spike_magnitude`, καθορίζουν την πιθανότητα και το μέγεθος της ξαφνικής αλλαγής στον ρυθμό άφιξης κλειδιών. Το spike χρησιμοποιείται για να προσομοιωθούν ακραίες καταστάσεις, όπου το σύστημα δέχεται αυξομειώσεις στην άφιξη δεδομένων. Για παράδειγμα, σε εφαρμογές κοινωνικών δικτύων, όπως το Twitter, μπορεί να υπάρξει μια απότομη αύξηση των δεδομένων σε μια περίοδο αιχμής.

- **Τοπολογία (`topology`)**: Η τοπολογία καθορίζει τη διαμόρφωση των σταδίων επεξεργασίας και των κόμβων σε κάθε στάδιο. Κάθε στάδιο μπορεί να αποτελείται από κόμβους με ή χωρίς εσωτερική κατάσταση, με διαφορετική πολυπλοκότητα επεξεργασίας και ρυθμαπόδοση (throughput).

Όπως αναφέρθηκε, οι περισσότερες από αυτές τις παραμέτρους προσαρμόζονται ανάλογα με τις απαιτήσεις κάθε πειράματος, προκειμένου να εξεταστούν διαφορετικά σενάρια λειτουργίας του συστήματος. Η μοναδική παράμετρος που παραμένει σταθερή, κατά τη διάρκεια ενός πειράματος, είναι η ρυθμαπόδοση (throughput) των κόμβων, καθώς η υπολογιστική ισχύς των κόμβων των πραγματικών συστημάτων δεν μεταβάλλεται εύκολα.

Κάθε ένα από τα πειραματικά σενάρια σχεδιάστηκε για να ελέγξει συγκεκριμένες πτυχές της λειτουργίας του συστήματος, και τα αποτελέσματα συγκρίθηκαν με στόχο την αξιολόγιση της ορθότητας του προσομοιωτή και την εξαγωγή αποτελεσμάτων για τη βελτιστοποίηση πραγματικών συστημάτων.

## 5.3 Σενάρια Δοκιμών

Τα σενάρια που εξετάστηκαν στο πλαίσιο των πειραμάτων περιλαμβάνουν ποικίλες συνθήκες λειτουργίας του συστήματος και όπως αναφέρθηκε νωρίτερα, χωρίζονται σε δύο κατηγορίες:

- Επαλήθευση της σωστής λειτουργίας του προσομοιωτή μέσω σεναρίων που εξασφαλίζουν ότι η συμπεριφορά του είναι η αναμενόμενη και

- Αξιολόγηση της επίδοσης των DSPSs σε διάφορες συνθήκες, ενισχύοντας την κατανόηση και τη βελτίωση των λειτουργιών τους.

### Επαλήθευση Ορθότητας

Η επαλήθευση της σωστής συμπεριφοράς του προσομοιωτή περιλαμβάνει πειράματα που ελέγχουν αν η λειτουργία του συστήματος είναι η αναμενόμενη. Περιλαμβάνονται τα εξής σενάρια:

- **Κανονική Λειτουργία**: Σε αυτό το σενάριο, το σύστημα λειτουργεί με σταθερό ρυθμό άφιξης κλειδιών, τα οποία κατανέμονται ομοιόμορφα. Η συχνότητα εμφάνισης των κλειδιών παραμένει σταθερή καθ' όλη τη διάρκεια του πειράματος, και κάθε κλειδί εμφανίζεται με ίση πιθανότητα. Αυτό το πείραμα αντιπροσωπεύει την τυπική λειτουργία του συστήματος, επιτρέποντας την αξιολόγηση της απόδοσής του υπό ομαλές και ελεγχόμενες συνθήκες. Τα αποτελέσματα αυτού του σεναρίου λειτουργούν ως σημείο αναφοράς για συγκρίσεις με πιο απαιτητικά ή σύνθετα πειραματικά σενάρια.

- **Αυξανόμενος Ρυθμός Άφιξης Κλειδιών**: Σε αυτό το σενάριο, προσομοιώνεται η σταδιακή αύξηση του ρυθμού άφιξης κλειδιών, αντικατοπτρίζοντας ένα συνεχώς αυξανόμενο φορτίο εργασίας. Το πείραμα δοκιμάζει την αντοχή του συστήματος κατά τη διάρκεια αυτών των συνθηκών, εστιάζοντας στην ανίχνευση σημείων κορεσμού και στην αξιολόγηση της απόκρισης του συστήματος υπό συνθήκες υπερφόρτωσης.

- **Ανισοκατανομή Κλειδιών**: Τα κλειδιά κατανέμονται άνισα, ακολουθώντας από τις διαθέσιμες κατανομές είτε την κανονική κατανομή με υψηλή τυπική απόκλιση είτε Zipf κατανομή με alpha μεγαλύτερο του 1. Αυτό σημαίνει ότι ορισμένα κλειδιά εμφανίζονται πολύ συχνότερα από άλλα, προσομοιώνοντας πραγματικές συνθήκες όπου ορισμένα δεδομένα είναι πιο συχνά. Το πείραμα αυτό εξετάζει την ικανότητα του συστήματος να διαχειριστεί έναν ανισοκατανεμημένο φόρτο εργασίας, αναμένοντας αυξημένη επιβάρυνση σε συγκεκριμένους κόμβους και μειωμένη σε άλλους, δεδομένης της χρήσης στρατηγικής διαμερισμού μέσω hashing.

- **Μεταβαλλόμενο Spike**: Σε αυτό το σενάριο, παρατηρούνται αυξομειώσεις στον ρυθμό άφιξης των κλειδιών, προσομοιώνοντας απότομες αυξήσεις ή μειώσεις στον φόρτο εργασίας. Στόχος του πειράματος είναι να αξιολογηθεί η ικανότητα του συστήματος να ανταποκρίνεται σε ξαφνικές αλλαγές στην εισερχόμενη ροή δεδομένων και να διατηρεί σταθερή απόδοση. Παρόμοια φαινόμενα παρατηρούνται σε εφαρμογές όπως το Twitter, ιδιαίτερα σε περιόδους αιχμής, όπως κατά τη διάρκεια των Ολυμπιακών Αγώνων [50]. Αναμένεται να εμφανιστούν σύντομα χρονικά διαστήματα με απότομη αύξηση του φόρτου στους κόμβους, τα οποία στη συνέχεια επανέρχονται σε φυσιολογικά επίπεδα.

### Αξιολόγηση Επίδοσης

Στην κατηγορία αυτή, τα πειράματα εστιάζουν στην παρατήρηση της απόδοσης του συστήματος κατά την εφαρμογή διαφορετικών στρατηγικών και σεναρίων που είναι πιθανό να αντιμετωπίσει σε πραγματικές συνθήκες. Τα σενάρια περιλαμβάνουν:

- **Κλιμακωσιμότητα**: Ένα από τα πιο κρίσιμα χαρακτηριστικά των κατανεμημένων συστημάτων επεξεργασίας συνεχών ροών δεδομένων, καθώς και των κατανεμημένων

συστημάτων γενικότερα, είναι η κλιμακωσιμότητα. Αυτή επιτυγχάνεται με την προσθήκη επιπλέον πόρων (κόμβων) και την αξιοποίηση της παραλληλίας. Σε αυτό το σενάριο, μεταβάλλεται ο αριθμός των κόμβων επεξεργασίας σε στάδια της τοπολογίας που παρουσιάζουν υψηλό φόρτο έργασιας, με σκοπό να αξιολογηθεί η ικανότητα του συστήματος να κλιμακώνεται αποτελεσματικά και να διαχειρίζεται τον φόρτο εργασίας καλύτερα.

- **Διαφορετικές Στρατηγικές Διαμερισμού Κλειδιών**: Σε αυτό το σενάριο, αξιολογούνται οι αποδόσεις διαφόρων στρατηγικών κατανομής κλειδιών, με στόχο να εντοπιστεί ποιά προσέγγιση κατανέμει καλύτερα το φόρτο εργασίας στους κόμβους, διασφαλίζοντας βέλτιστη απόδοση και κατανομή πόρων.

### 5.3.1 Ανάλυση Παραμετροποίησης

Η ρύθμιση των παραμέτρων προσομοίωσης προσαρμόστηκε σε κάθε πείραμα, με στόχο την κατάλληλη μελέτη των διαφορετικών σεναρίων. Συγκεκριμένα, οι τιμές των παραμέτρων `arrival_rate`, `distribution` και `spike` προσαρμόστηκαν ανάλογα με το φόρτο και τις απαιτήσεις κάθε προσομοίωσης. Αυτές οι αλλαγές επέτρεψαν την προσομοίωση ποικίλων συνθηκών λειτουργίας του συστήματος, όπως αυξημένος ρυθμός άφιξης δεδομένων, άνιση κατανομή κλειδιών και αιφνίδιες αυξήσεις φόρτου (spikes).

Για λόγους σαφήνειας και πληρότητας, οι ακριβείς τιμές των παραμέτρων που χρησιμοποιήθηκαν σε κάθε πείραμα θα παρουσιαστούν στο επόμενο κεφάλαιο, όπου θα αναλυθούν τα αποτελέσματα και οι παρατηρήσεις από την προσομοίωση. Επιπλέον, σε όλα τα πειράματα διατηρήθηκε σταθερή η τιμή της ρυθμαπόδοσης (`throughput`) στους κόμβους επεξεργασίας. Αυτή η προσέγγιση υιοθετήθηκε για να προσομοιωθούν ρεαλιστικές συνθήκες, όπου η υπολογιστική ισχύς των φυσικών κόμβων παραμένει σταθερή, εξασφαλίζοντας ακριβή αξιολόγηση των επιπτώσεων από τις μεταβολές των άλλων παραμέτρων.

## 5.4 Τοπολογίες

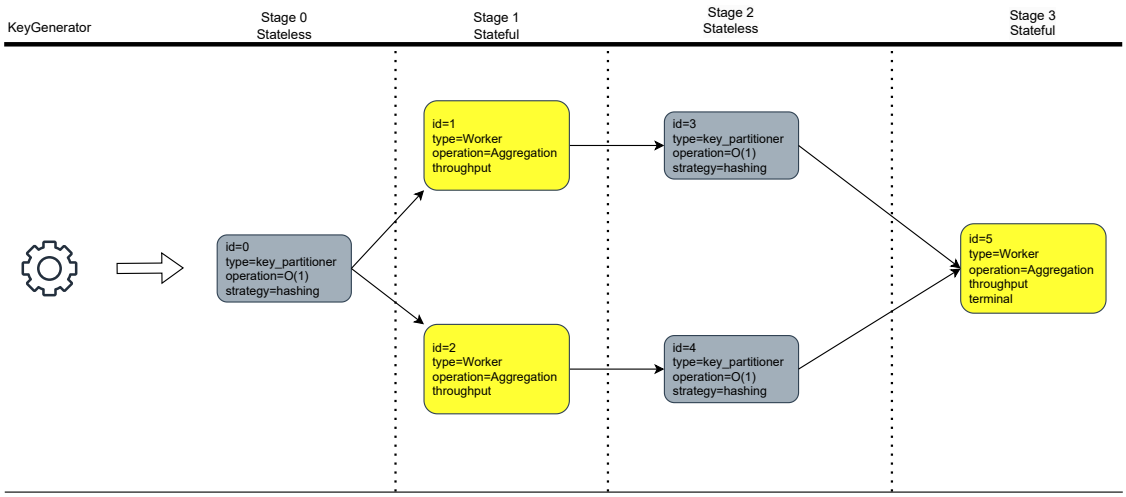
Για την εκτέλεση των παραπάνω σεναρίων και για την εξασφάλιση της πληρότητας των αποτελεσμάτων, χρησιμοποιήθηκαν δύο διαφορετικές τοπολογίες. Οι τοπολογίες αυτές σχεδιάστηκαν με στόχο την προσομοίωση πραγματικών εφαρμογών, όπως η διαδικασία word count για τον εντοπισμό hot tweets στην πρώτη τοπολογία, καθώς και την επεξεργασία δεδομένων από κοινωνικά δίκτυα [51] ή τον υπολογισμό ποσοστιαίων κατανομών για την ανάλυση συμπεριφοράς καταναλωτών [47] στη δεύτερη τοπολογία. Κάθε τοπολογία προσφέρει την ευελιξία να προσαρμόζεται σε διαφορετικά σενάρια επεξεργασίας.

Συγκεκριμένα, για τα πειράματα ορθότητας του εργαλείου χρησιμοποιήσαμε μόνο την πρώτη, πιο απλή τοπολογία ώστε να μπορέσουμε να εξακριβώσουμε ευκολότερα αν τα αποτελέσματα είναι τα αναμενόμενα. Αντίθετα, για τα πειράματα επίδοσης, εξετάσαμε και τις δύο τοπολογίες ώστε να μπορέσουμε να αξιολογήσουμε καλύτερα διαφορετικές συμπεριφορές και επιδόσεις των διαφορετικών παραμετροποιήσεων σε τοπολογίες με διακριτά χαρακτηριστικά.

### Τοπολογία 1

Η πρώτη τοπολογία, όπως φαίνεται στο Σχήμα 5.1, αποτελείται από τέσσερα διαδοχικά στάδια:

- **Στάδιο 0 (stateless)**: Περιλαμβάνει έναν key partitioner κόμβο που λαμβάνει τα κλειδιά από τη γεννήτρια κλειδιών και τα κατανέμει στους κόμβους του επόμενου σταδίου μέσω στρατηγικής hashing. Το στάδιο αυτό λειτουργεί ως το σημείο εκκίνησης της ροής δεδομένων, εξασφαλίζοντας τη διανομή τους για περαιτέρω επεξεργασία.
- **Στάδιο 1 (stateful)**: Αποτελείται από δύο stateful κόμβους, οι οποίοι εκτελούν την κύρια επεξεργασία των δεδομένων με μια απλή λειτουργία συνάθροισης. Τα αποτελέσματα από αυτό το στάδιο προωθούνται στο επόμενο.
- **Στάδιο 2 (stateless)**: Περιλαμβάνει δύο key partitioner κόμβους που διανέμουν τα αποτελέσματα του προηγούμενου σταδίου, εξασφαλίζοντας τη σωστή ροή δεδομένων στους επόμενους κόμβους.
- **Στάδιο 3 (stateful)**: Το τελικό στάδιο αποτελείται από έναν stateful κόμβο, ο οποίος ολοκληρώνει την επεξεργασία με μια λειτουργία συνάθροισης και παράγει τα τελικά αποτελέσματα της προσομοίωσης.



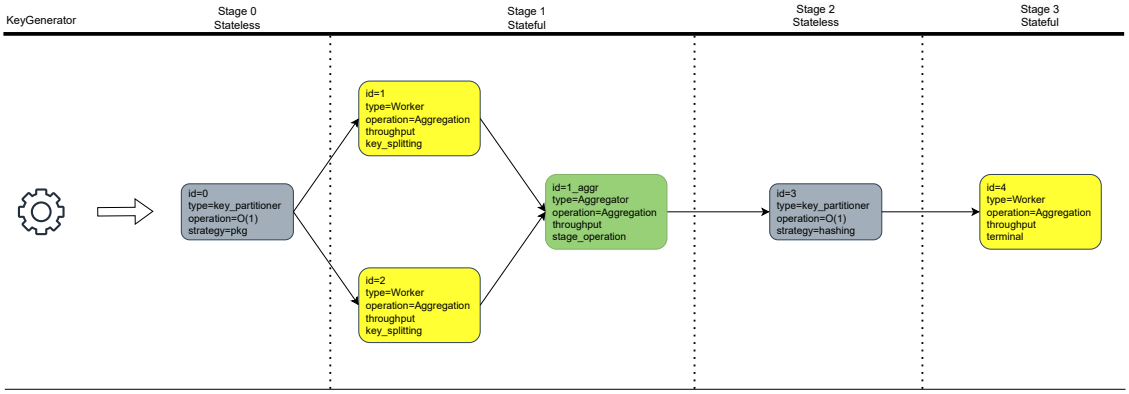

Σχήμα 5.1: *Τοπολογία 1: Απλή Τοπολογία*

Αυτή η τοπολογία προσομοιώνει μια σχετικά απλή διεργασία, όπως η καταμέτρηση λέξεων (word count), όπου οι λειτουργίες κάθε σταδίου είναι απλές και με γραμμική πολυπλοκότητα.

Στις περιπτώσεις που χρησιμοποιείται διάσπαση κλειδιών (key splitting), όπως στις στρατηγικές shuffle grouping και partial key grouping, η τοπολογία μετατρέπεται ως εξής:

- Οι stateful κόμβοι ενός σταδίου, αφού ολοκληρώσουν την επεξεργασία τους, δεν στέλνουν τα δεδομένα σε Key Partitioner, αλλά σε έναν εσωτερικό AggregatorNode, ο οποίος συνενώνει τα διασπασμένα κλειδιά και εκτελεί επιπλέον επεξεργασία. Στη συνέχεια,

ο AggregatorNode στέλνει τα παραγόμενα κλειδιά σε έναν KeyPartitioner. Έτσι, η τοπολογία τροποποιείται αυτόματα σε αυτή του Σχήματος 5.2.




Σχήμα 5.2: *Τοπολογία 1: Απλή Τοπολογία με Διάσπαση Κλειδιών και Συναθροιστή*

### Τοπολογία 2

Η δεύτερη τοπολογία, όπως φαίνεται στο Σχήμα 5.3, είναι πιο πολυεπίπεδη και προσαρμοσμένη στην επεξεργασία μεγαλύτερων ροών δεδομένων και πιο σύνθετων εφαρμογών. Αποτελείται από έξι στάδια:

- **Στάδιο 0 (stateless)**: Όπως και στην Τοπολογία 1, το πρώτο στάδιο περιλαμβάνει έναν key partitioner, ο οποίος διανέμει τα κλειδιά στους επόμενους κόμβους μέσω στρατηγικής hashing.

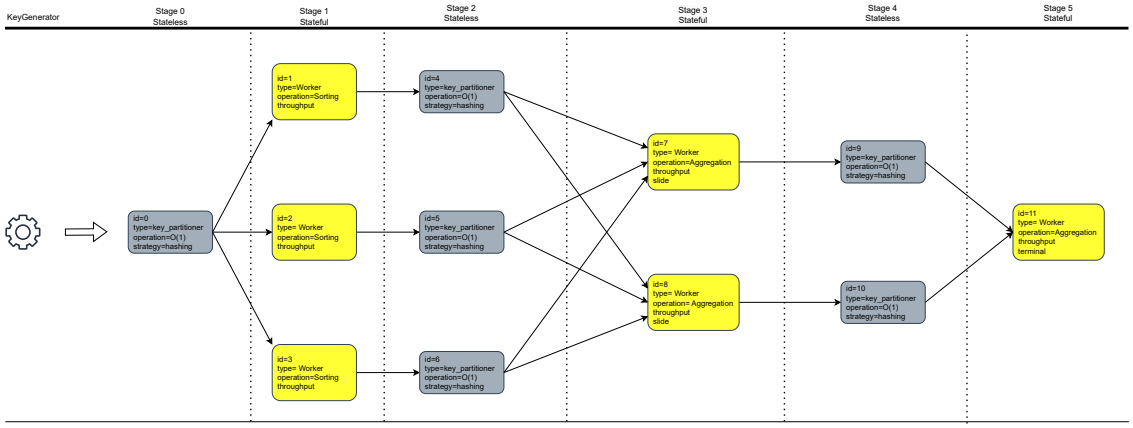
- **Στάδιο 1 (stateful)**: Αυτό το στάδιο περιλαμβάνει τρεις stateful κόμβους που λειτουργούν παράλληλα για να επεξεργαστούν τα δεδομένα, αυξάνοντας την απόδοση. Και οι τρεις κόμβοι εκτελούν λειτουργίες ταξινόμησης.

- **Στάδιο 2 (stateless)**: Οι τρεις key partitioners σε αυτό το στάδιο κατανέμουν τα αποτελέσματα από τους stateful κόμβους στους επόμενους κόμβους επεξεργασίας.

- **Στάδιο 3 (stateful)**: Περιλαμβάνει δύο stateful κόμβους, οι οποίοι λαμβάνουν τα ταξινομημένα δεδομένα και εκτελούν επεξεργασία συνάθροισης πάνω σε αυτά.

- **Στάδιο 4 (stateless)**: Οι δύο key partitioners κατανέμουν τα δεδομένα στον τελικό κόμβο για την ολοκλήρωση της επεξεργασίας.

- **Στάδιο 5 (stateful)**: Στο τελευταίο στάδιο, ένας stateful κόμβος ολοκληρώνει την τελική επεξεργασία των δεδομένων, εκτελώντας λειτουργία συνάθροισης.




Σχήμα 5.3: *Τοπολογία 2: Σύνθετη Τοπολογία*

Η Τοπολογία 2 προσφέρει μια πιο πολύπλοκη δομή σε σύγκριση με την Τοπολογία 1, επιτρέποντας την προσομοίωση πιο σύνθετων διεργασιών με αυξημένη πολυπλοκότητα. Οι τοπολογίες αυτές προσαρμόζονται στις ανάγκες κάθε πειράματος, καθώς οι παράμετροι, όπως ο τύπος λειτουργίας και η στρατηγική κατανομής (partition strategy), τροποποιούνται ανάλογα με το σενάριο της δοκιμής.

Κεφάλαιο 6

# Αποτελέσματα και Ανάλυση

Στο κεφάλαιο αυτό παρουσιάζονται και αναλύονται τα αποτελέσματα που προέκυψαν από τα πειράματα που διεξήχθησαν με τον προσομοιωτή μας. Όπως αναφέρθηκε στο προηγούμενο κεφάλαιο, τα πειράματα χωρίζονται σε δύο βασικές κατηγορίες:

- Πειράματα που στοχεύουν στην επαλήθευση της ακρίβειας και της σωστής λειτουργίας του προσομοιωτή.
- Πειράματα που εξετάζουν την επίδραση διαφόρων στρατηγικών και τοπολογιών στη συμπεριφορά του συστήματος.

Για την οπτικοποίηση των αποτελεσμάτων, χρησιμοποιούμε τέσσερις κύριους τύπους γραφικών παραστάσεων:

1. **Διάγραμμα Φόρτου Κόμβου**: Το συγκεκριμένο γράφημα απεικονίζει τον φόρτο εργασίας ενός κόμβου σε κάθε χρονικό βήμα της προσομοίωσης, συνοδευόμενο από μια καμπύλη που καταδεικνύει τη μέση εξέλιξη του φόρτου για ένα παράθυρο βημάτων. Πιο συγκεκριμένα, σε ένα παράθυρο δύο βημάτων εκατέρωθεν του τρέχοντος βήματος υπολογίζεται ο μέσος φόρτος του κόμβου (χρησιμοποιώντας γκαουσιανό φίλτρο – Gaussian Filter – με $\sigma = 2$ [52]) εξομαλύνοντας τις διακυμάνσεις και διατηρώντας παράλληλα τις βασικές τάσεις της αλλαγής του φόρτου.

   Ο φόρτος του κόμβου υπολογίζεται βάσει της ρυθμαπόδοσης (throughput) και του συνολικού αριθμού των κύκλων επεξεργασίας για τα κλειδιά που διαχειρίζεται ο κόμβος τη δεδομένη στιγμή, με τον ακόλουθο τύπο:

   $$\text{load} = \frac{\text{total cycles}}{\text{throughput}} \times 100$$

   όπου το load αντιπροσωπεύει τον φόρτο του κόμβου, το total cycles είναι ο συνολικός αριθμός κύκλων επεξεργασίας για τα κλειδιά εκείνη τη χρονική στιγμή, και το throughput είναι η ρυθμαπόδοση του κόμβου. Ένα παράδειγμα αυτής της γραφικής παράστασης φαίνεται στο Σχήμα 6.1.

2. **Εναλλακτικό Διάγραμμα Φόρτου Κόμβου (boxplot)**: Επιλέξαμε να αναπαραστήσουμε τον φόρτο σε διάγραμμα κουτιού (box plot) ώστε να φαίνεται το συχνότερο εύρος φόρτου του κάθε κόμβου καθ'όλη την διάρκεια του πειράματος (τιμές εσωτερικά του

κουτιού). Επιπλέον, μπορούμε να δούμε και την διάμεσο (median) του φόρτου του κάθε κόμβου (κόκκινη γραμμή). Ουσιαστικά αυτή θα είναι και η μετρική που θα δείχνει ποιος κόμβος είχε συνολικά μικρότερο φόρτο στο πείραμα. Τέλος, οι εξωτερικές γραμμές (whiskers) δείχνουν την μέγιστη και ελάχιστη τιμή (εφόσον δεν υπάρχουν outlier τιμές). Σε περίπτωση που υπάρχουν outlier τιμές, αυτές απεικονίζονται ως "+" στο διάγραμμα. Ένα παράδειγμα αυτής της παράστασης παρουσιάζεται στο Σχήμα 6.15.

3. **Διάγραμμα φόρτου Κόμβου σταδίου**: Σε αυτή την παράσταση παρουσιάζεται η καμπύλη του προηγούμενου διαγράμματος για όλους τους κόμβους ενός σταδίου, επιτρέποντας τη σύγκριση της επίδοσης μεταξύ των κόμβων και παρέχοντας μια συνολική εικόνα της απόδοσης του σταδίου. Ένα παράδειγμα αυτής της παράστασης παρουσιάζεται στο Σχήμα 6.2.

4. **Μετρικές επεξεργασίας ανά κόμβο**: Αυτή η γραφική παράσταση παρουσιάζει, για κάθε κόμβο, τον συνολικό αριθμό των επεξεργασμένων (processed), εκπρόθεσμων (overdue) και ληγμένων (expired) κλειδιών[1] σε κάθε χρονικό βήμα. Ένα σχετικό παράδειγμα φαίνεται στο Σχήμα 6.4.

Υπενθυμίζουμε ότι για τα πειράματα επαλήθευσης της ορθότητας του προσομοιωτή, χρησιμοποιήσαμε την τοπολογία 5.1, καθώς είναι πιο απλή και τα αναμενόμενα αποτελέσματα μπορούν να εκτιμηθούν ευκολότερα. Στη συνέχεια, κατά τα πειράματα αξιολόγησης της επίδοσης των συστημάτων, συνεχίζουμε να εκτελούμε ορισμένα πειράματα με την ίδια τοπολογία, ώστε να συγκρίνουμε με τα προηγούμενα αποτελέσματα. Παράλληλα, εισάγουμε και τη δεύτερη τοπολογία (5.3), στην οποία επαναλαμβάνουμε ορισμένα από τα προηγούμενα πειράματα, για να μελετήσουμε πώς συμπεριφέρεται μια μεγαλύτερη τοπολογία υπό διάφορες συνθήκες και πώς μπορεί να βελτιωθεί η απόδοση του συστήματος σε αυτήν.

## 6.1 Πειράματα Επαλήθευσης Ορθότητας

Σε αυτή την ενότητα αναλύονται τα αποτελέσματα των πειραμάτων επαλήθευσης της ορθότητας, με στόχο την αξιολόγηση της ορθής λειτουργίας του προσομοιωτή.

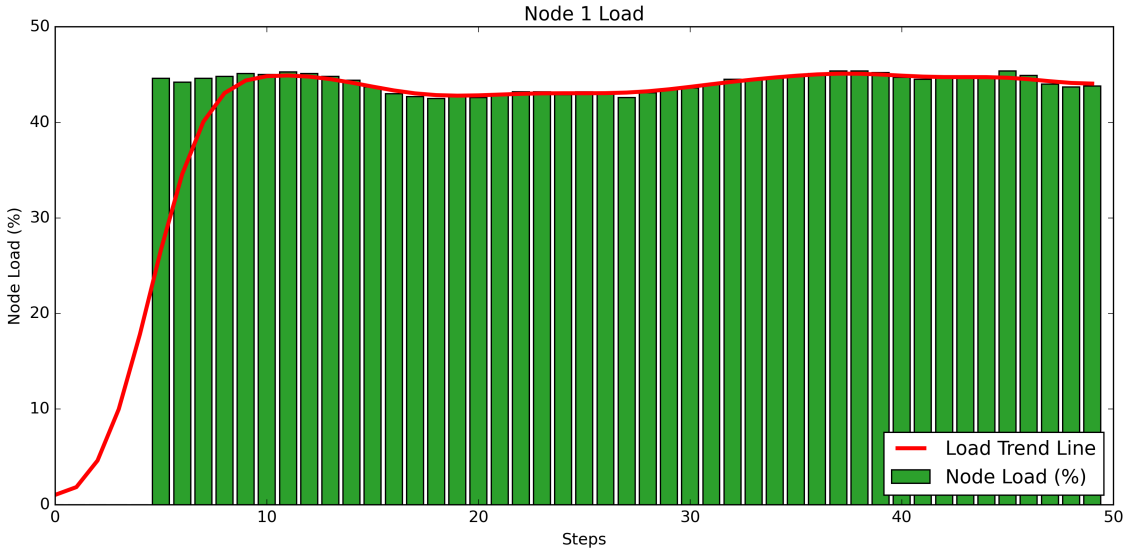
### 6.1.1 Κανονική Λειτουργία

Στο πείραμα αυτό εξετάζεται η επίδοση του συστήματος για έναν πεπερασμένο αριθμό βημάτων με σταθερό ρυθμό άφιξης και ομοιομόρφα κατανεμημένα κλειδιά.

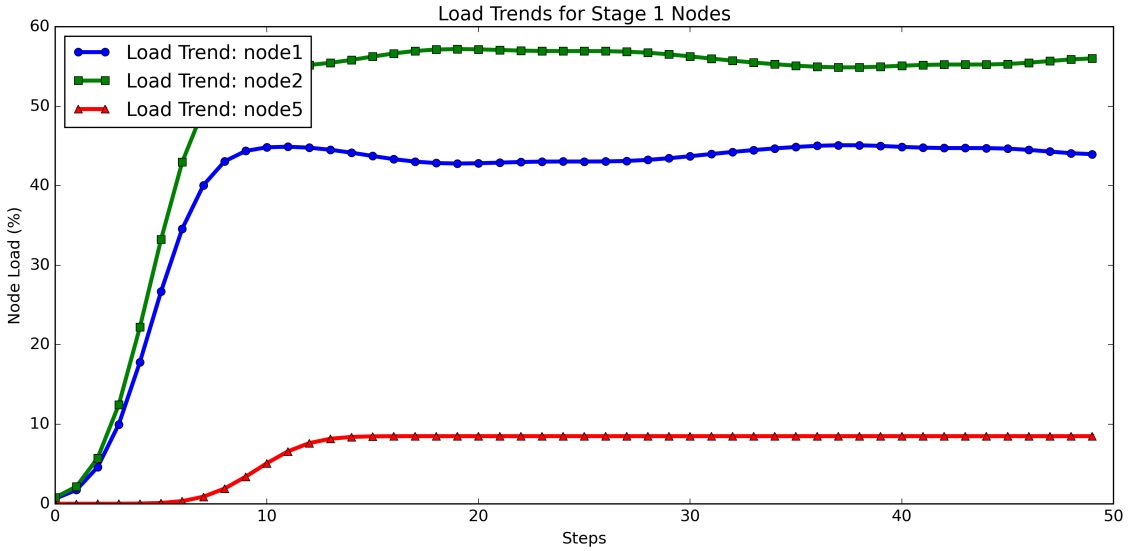
Στο Σχήμα 6.1 φαίνεται ο φόρτος του κόμβου 1 της τοπολογίας ανά βήμα της προσομοίωσης. Παρατηρούμε ότι στα αρχικά βήματα (0 έως 4) ο φόρτος είναι μηδαμινός. Υπενθυμίζουμε ότι ορίσαμε το κυλιόμενο παράθυρο να έχει μέγεθος ίσο με 5 βήματα στο τρέχον πείραμα. Το αποτέλεσμα είναι λογικό με βάση τις παραμέτρους που ορίστηκαν δεδομένου ότι ο κόμβος αναμένει την συμπλήρωση του παραθύρου πριν ξεκινήσει την επεξεργασία του. Μόλις συμπληρωθεί το πρώτο παράθυρο γίνεται και η πρώτη επεξεργασία στον κόμβο και

[1] Καθ'όλη την διάρκεια αυτής της διπλωματικής εργασίας χρησιμοποιούμε τον όρο κλειδιά (keys) και εγγραφές (records) αλληλένδετα.

εφόσον η τιμή της ολίσθησης είναι ίση με 1 σε κάθε βήμα αναμένουμε ένα παράθυρο να συμπληρώνεται και να υπολογίζεται. Ακριβώς αυτό παρατηρείται και στο γράφημα. Στα μετέπειτα βήματα ο φόρτος του κόμβου κυμαίνεται περίπου στο 45%. Αναλυτικότερα ο φόρτος όλων των κόμβων φαίνεται στο σχήμα 6.2.




Σχήμα 6.1: *Κανονική Λειτουργία - Φόρτος Κόμβου 1*




Σχήμα 6.2: *Κανονική Λειτουργία - Εξέλιξη Φόρτου Κόμβων*

Οι κόμβοι 1 και 2 είναι οι κόμβοι του 1ου σταδίου και υλοποιούν μία λειτουργία συνάθροισης των κλειδιών. Ο φόρτος τους δεν παρουσιάζει μεγάλες διακυμάνσεις καθ'όλη την διάρκεια της προσομοίωσης, κάτι που αντικατοπτρίζει και τα εισαγόμενα δεδομένα που διατηρούν σταθερό ρυθμό άφιξης. Αντίθετα παρατηρούμε, ότι υπάρχει απόκλιση μεταξύ του φόρτου των κόμβων του σταδίου. Αυτό οφείλεται κυρίως σε δύο λόγους. Πρώτον, η γεννήτρια κλειδιών προφανώς θα εμφανίζει ελαφρές αποκλίσεις από την ιδανική ομοιόμορφη κατανομή των κλειδιών και κατά δεύτερον, ως στρατηγική διαμέρισης των κλειδιών χρησιμοποιείται

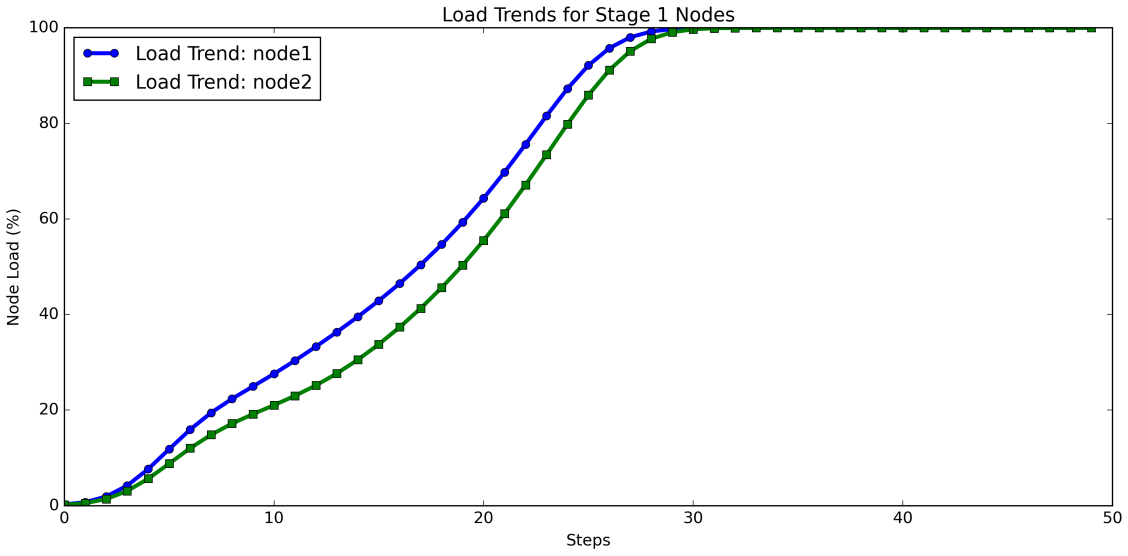
ο κατακερματισμός που δεν εξασφαλίζει τέλεια ομοιόμορφη διαμέριση των κλειδιών στους κόμβους.

Αναφορικά με τον κόμβο 5, ο οποίος είναι ο κόμβος του 2ου και τελευταίου σταδίου της τοπολογίας, παρατηρείται ένα αισθητά μικρότερο ποσοστό φόρτου λειτουργίας σε σχέση με τους κόμβους του προηγούμενου σταδίου. Το πρώτο στάδιο, εφόσον υλοποιεί μία λειτουργία συνάθροισης αναμένεται να μειώνει τον αριθμό κλειδιών σημαντικά στο δεύτερο στάδιο. Φυσικά αυτό βρίσκεται σε πλήρη αντιστοιχία με τα πραγματικά συστήματα που υλοποιούν μία εργασία συνάθροισης.

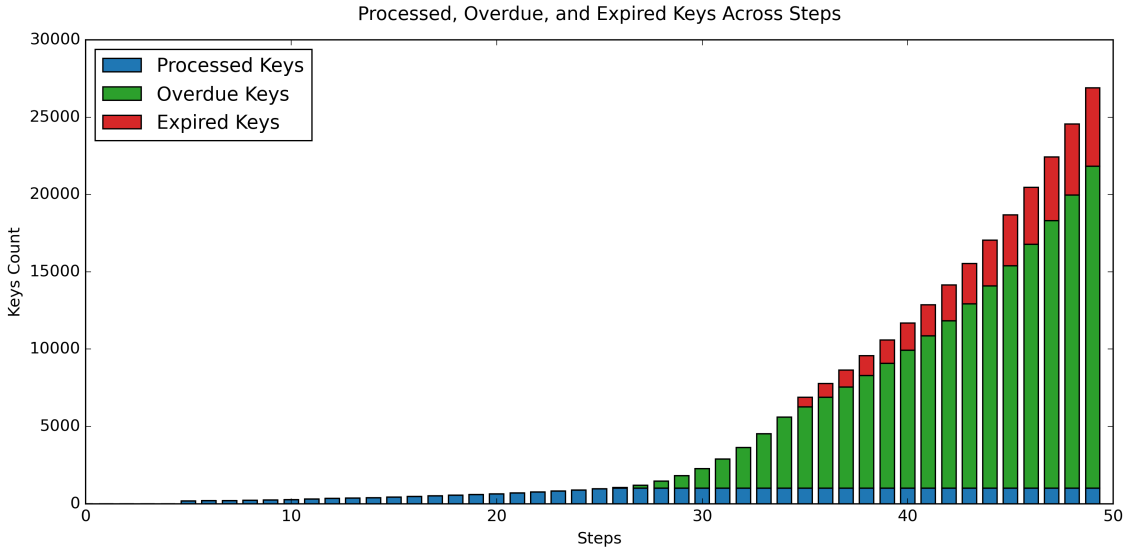
### 6.1.2 Αυξανόμενος Ρυθμός Άφιξης Κλειδιών

Σε αυτό το σενάριο χρησιμοποιείται η παράμετρος `arrival_rate_ot` με τιμή 8%, για τη συνεχή αύξηση του ρυθμού άφιξης των κλειδιών, οδηγώντας το σύστημα σε κατάσταση κορεσμού. Σε αυτή την κατάσταση, οι κόμβοι υπερφορτώνονται και πολλά κλειδιά λήγουν πριν προλάβουν να επεξεργαστούν.

Με την εκτέλεση του παραπάνω σεναρίου στην τοπολογία 5.1, προκύπτουν τα Σχήματα 6.3 και 6.4. Στο πρώτο, παρατηρούμε ότι ο φόρτος στους κόμβους του πρώτου stateful σταδίου αυξάνεται απότομα στο 100% και παραμένει εκεί μέχρι το τέλος της προσομοίωσης. Στο δεύτερο σχήμα, που αφορά τον κόμβο 1, αρχικά ο κόμβος επεξεργάζεται όλα τα κλειδιά που του ανατίθενται. Καθώς αυξάνεται ο ρυθμός άφιξης, εμφανίζονται εκπρόθεσμα κλειδιά τα οποία είτε θα επεξεργαστούν επιτυχώς σε επόμενα βήματα είτε θα λήξουν αν ξεπεραστεί το παράθυρο επεξεργασίας. Από τα μέσα της προσομοίωσης και μετά, παρατηρούμε την εμφάνιση ληγμένων κλειδιών, ο αριθμός των οποίων αυξάνεται σε κάθε βήμα. Αυτό δείχνει ότι το σύστημα έχει φτάσει σε σημείο κορεσμού, με τον ρυθμό άφιξης των δεδομένων να είναι τόσο υψηλός που δεν μπορεί να υποστηριχθεί από την ρυθμαπόδοση, ενώ συνεχίζει να αυξάνεται.




Σχήμα 6.3: *Αυξανόμενος Ρυθμός Άφιξης - Εξέλιξη Φόρτου Κόμβων Σταδίου 1*




Σχήμα 6.4: *Αυξανόμενος Ρυθμός Άφιξης - Μετρικές Επεξεργασίας Κόμβου 1*

### 6.1.3 Ανισοκατανομή Κλειδιών

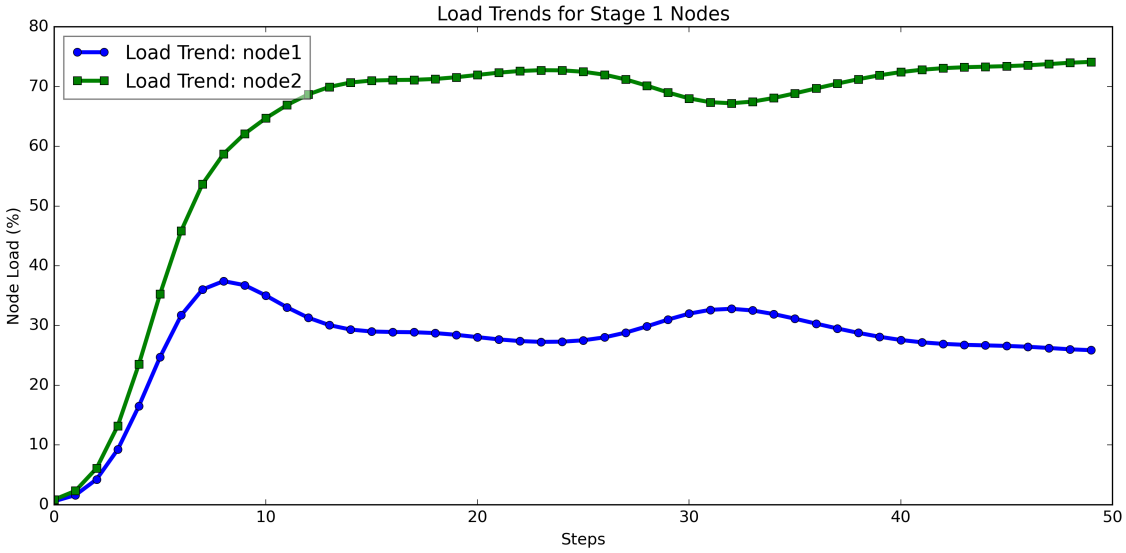
Για την προσομοίωση της ανισοκατανομής δεδομένων (data skewness), εκτελούμε δύο πειράματα: ένα με μέτριο βαθμό ανισοκατανομής και ένα με υψηλό βαθμό ανισοκατανομής. Στην πρώτη περίπτωση, χρησιμοποιούμε την κατανομή Zipf με παράμετρο $a = 1.25$. Αυτή η κατανομή προσομοιώνει τη συχνότητα εμφάνισης των λέξεων στη φυσική γλώσσα, σύμφωνα με τον Zipf's Law [53], όπου η συχνότερα εμφανιζόμενη λέξη εμφανίζεται διπλάσιες φορές από τη δεύτερη συχνότερη, η οποία εμφανίζεται τριπλάσιες φορές από την τρίτη πιο συχνή κ.ο.κ. Με αυτή την παραμετροποίηση, και δεδομένου ότι οι κόμβοι της πρώτης τοπολογίας εκτελούν λειτουργίες συνάθροισης, μπορούμε να παρομοιάσουμε το παράδειγμα με την εκτέλεση ενός word count αλγορίθμου.

Το Σχήμα 6.5 παρουσιάζει τον φόρτο εργασίας των κόμβων του πρώτου σταδίου (nodes 1 και 2). Παρατηρούμε ότι ο κόμβος 2 έχει σημαντικά μεγαλύτερο φόρτο συγκριτικά με τον κόμβο 1. Αυτό οφείλεται στο γεγονός ότι χρησιμοποιούμε στρατηγική διαμερισμού hashing, με αποτέλεσμα τα πιο συχνά εμφανιζόμενα κλειδιά (hot keys) να ανατίθενται στον κόμβο 2, αυξάνοντας τον φόρτο του. Ωστόσο, δεν παρατηρείται ακόμα σημείο κορεσμού.

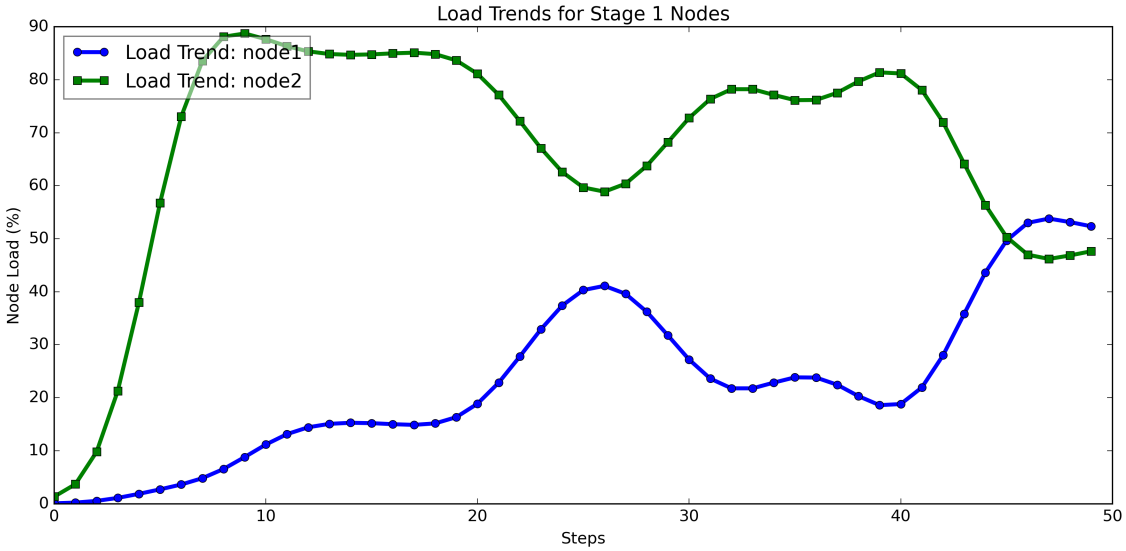
Στη δεύτερη περίπτωση, που αφορά υψηλή ανισοκατανομή, χρησιμοποιούμε κανονική κατανομή με μικρή τυπική απόκλιση και μέση τιμή κοντά στο μισό του πλήθους των διαφορετικών κλειδιών. Αυτό έχει ως αποτέλεσμα την αυξημένη εμφάνιση των κλειδιών γύρω από τη μέση τιμή και τη μειωμένη εμφάνιση των υπολοίπων. Στο πείραμα παρατηρούμε (Σχήμα 6.6) ότι ο κόμβος 2 έχει πολύ υψηλό φόρτο εργασίας, αγγίζοντας το 90% και πλησιάζοντας επίπεδα κορεσμού, ενώ ο κόμβος 1 έχει σημαντικά χαμηλότερο φόρτο, κυμαινόμενο μεταξύ 20% και 40% στο μεγαλύτερο διάστημα του πειράματος.

### 6.1.4 Μεταβαλλόμενο Spike

Στο πείραμα με μεταβαλλόμενο Spike εξετάζουμε την συμπεριφορά του προσομοιωτή σε μεγάλες διαβαθμίσεις του ρυθμού άφιξης των κλειδιών. Ποσοτικοποιείται το πόσο ανο-




Σχήμα 6.5: *Μέση Ανισοκατανομή - Εξέλιξη Φόρτου Κόμβων Σταδίου 1*




Σχήμα 6.6: *Υψηλή Ανισοκατανομή - Εξέλιξη Φόρτου Κόμβων Σταδίου 1*

χή έχει ο προσομοιωτής σε αυξημένους ρυθμούς αύξησης και το κατά πόσο επιστρέφει σε κατάσταση κανονικής λειτουργίας έπειτα από έλευση πεπερασμένων αριθμών βημάτων της προσομοίωσης.

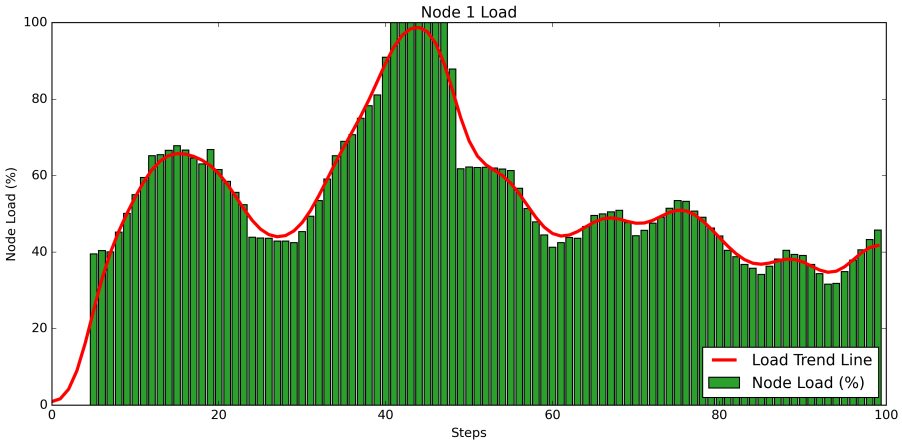
Συγκεκριμένα εξετάζουμε δύο τύπους αιχμών (spikes):

1. Medium Spike: Πιθανότητα μεταβολής ρυθμού αύξησης ίση με 25% ανά βήμα και ποσοστιαία μεταβολή ρυθμού άφιξης ίση με 60%.

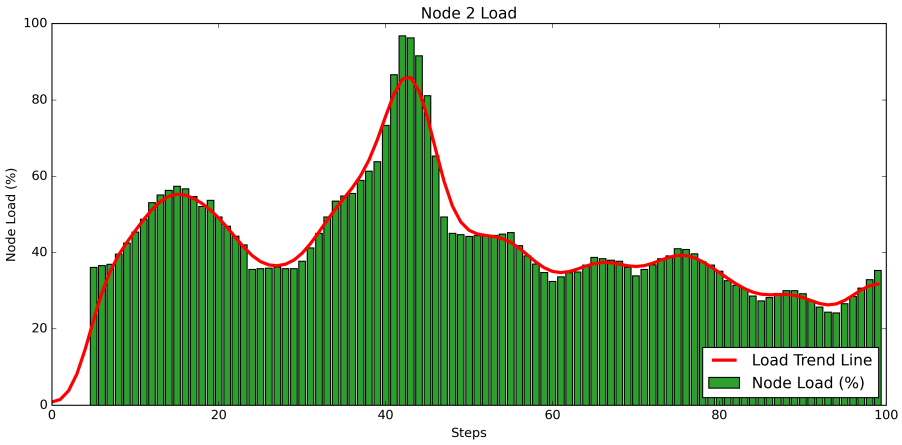
2. High Spike: Πιθανότητα μεταβολής ρυθμού αύξησης ίση με 50% ανά βήμα και ποσοστιαία μεταβολή ρυθμού άφιξης ίση με 150%.

**Medium Spike**

Στα σχήματα 6.7 και 6.8 παρατηρούμε τον φόρτο των δύο κόμβων του πρώτου σταδίου. Σε αντίθεση με το πείραμα της κανονικής λειτουργίας ο φόρτος του κάθε κόμβου έχει αρκετές αυξομειώσεις κατά την διάρκεια του πειράματος. Ειδικότερα, μεταξύ των βημάτων 38 - 47 όπου ο ρυθμός αφίξεων των κλειδιών αυξήθηκε πιο αισθητά ο φόρτος και γενικότερα το σύστημα είναι κόντα στο να υπεφορτωθεί. Ο κόμβος 1 συγκεκριμένα, παραμένει στο 100% λειτουργίας για μερικά βήματα. Όπως φαίνεται λοιπόν, η απότομη αύξηση των κλειδιών οδηγεί και σε αντίστοιχη αύξηση του συνολικού φόρτου όπως και αναμενόταν.



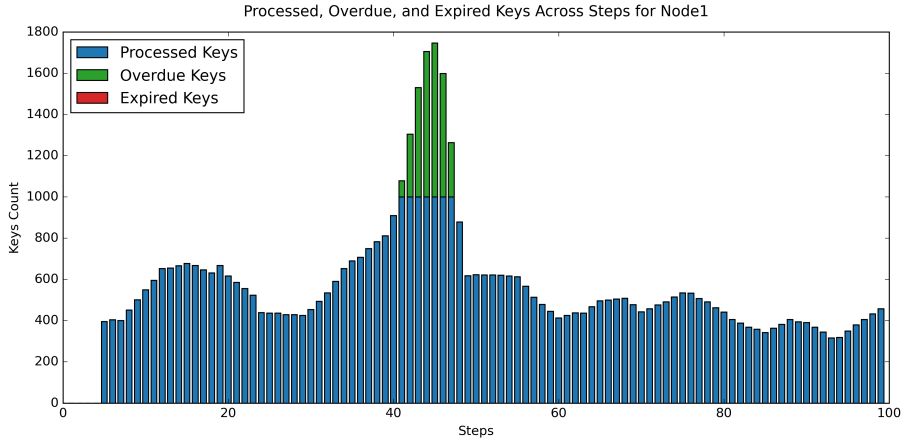

Σχήμα 6.7: *Μέσο Επίπεδο Αιχμής - Φόρτος Κόμβου 1*



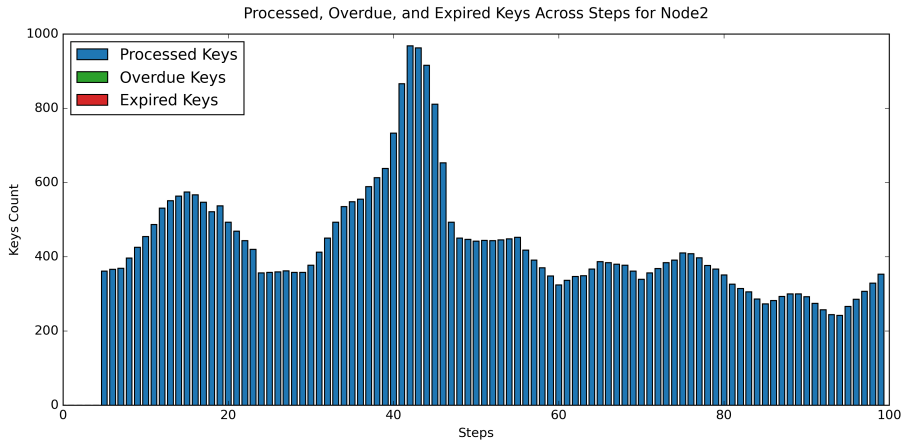

Σχήμα 6.8: *Μέσο Επίπεδο Αιχμής - Φόρτος Κόμβου 2*

Σημειώνουμε, ότι το πείραμα αυτό θέλει να εξετάσει και το κατά πόσο το σύστημα επιστρέφει στην κανονική λειτουργία μετά την πάροδο της αιχμής δεδομένων. Από τα προηγούμενα σχήματα σε συνδυασμό με τα σχήματα 6.9 και 6.10 που απεικονίζουν τον αριθμό των επεξεργασμένων, καθυστερημένων και ληγμένων (processed, overdue και expired) κλειδιών μπορούμε να δούμε το κατά πόσο το σύστημα επιστρέφει στην κανονική κατάσταση μετά την αιχμή. Ο κόμβος 1 κατά την διάρκεια που βρίσκεται στην μέγιστη χωρητικότητα (100% load) δεν μπορεί να επεξεργαστεί πλήρως τα παράθυρα στο πρώτο βήμα και του παραμένουν μη επεξεργασμένα κλειδιά στα παράθυρα, τα οποία τα επεξεργάζεται σε κάποιο επόμενο

βήμα. Τα υπολοιπόμενα κλειδιά ενός βήματος τα θεωρούμε καθυστερημένα (overdue) κλειδιά για ένα περιορισμένο χρονικό διάστημα μετά το πέρας του οποίου λήγουν (expired) . Σε ένα πραγματικό σύστημα η ύπαρξη των overdue κλειδιών θα μεταφραζόταν σε αύξηση του latency.




Σχήμα 6.9: *Μέσο Επίπεδο Αιχμής - Μετρικές Επεξεργασίας Κόμβου 1*



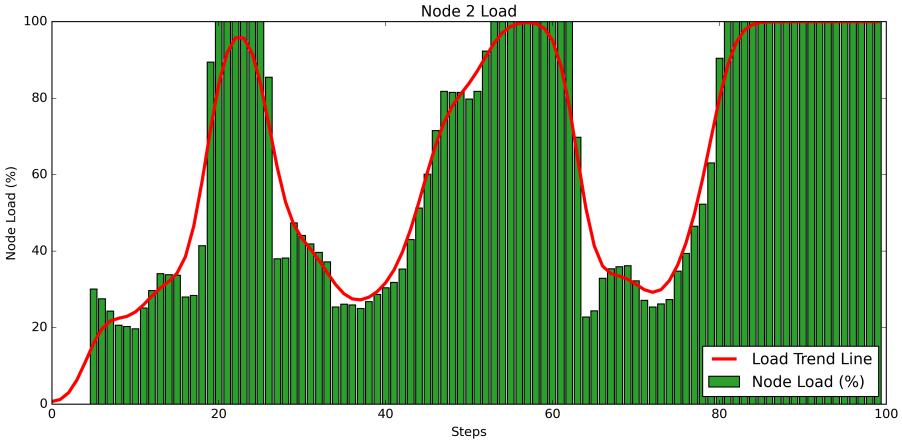

Σχήμα 6.10: *Μέσο Επίπεδο Αιχμής - Μετρικές Επεξεργασίας Κόμβου 2*

### High Spike

Στο δεύτερο πείραμα της συλλογής πειραμάτων με spiked δεδομένα εξετάζουμε ένα ισχυρότερο spike. Αντίστοιχα με το παραπάνω πείραμα, στα σχήματα 6.11 και 6.12 φαίνεται ο συνολικός φόρτος των κόμβων 1 και 2. Τα διαστήματα που οι κόμβοι βρίσκονται σε μέγιστη χωρητικότητα είναι εκτενέστερα, ως αποτέλεσμα των μεγαλύτερων spikes στα εισαγόμενα κλειδιά.

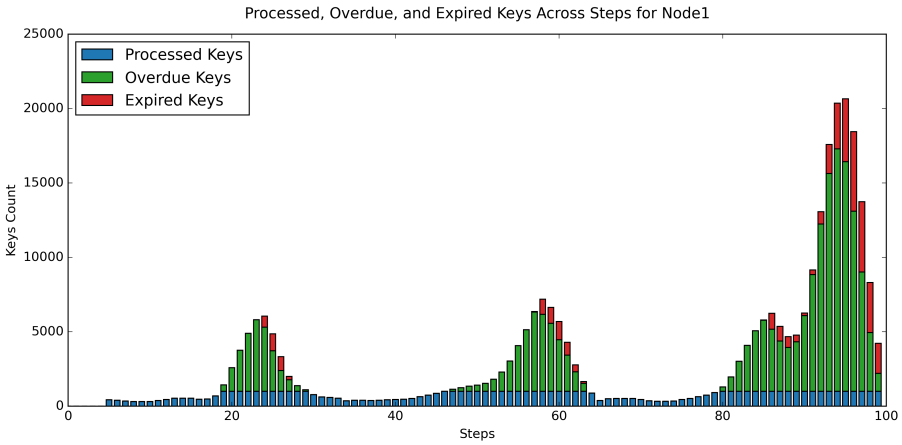
Παρατηρώντας και τα διαγράμματα με τα στατιστικά επεξεργασμένων κλειδιών ανά βήμα (6.13 και 6.14) παρατηρούμε πολύ μεγαλύτερο αριθμό κλειδιών που δεν μπορούν να επεξεργαστούν. Λόγω των αυξημένων καθυστερημένων (overdue) κλειδιών, για αρκετά βήματα, πολλά κλειδιά ξεπερνούν το όριο λήξης του παραθύρου και λήγουν. Αυτά τα κλειδιά δεν επεξεργάζονται και θεωρούνται ότι έληξαν από το παράθυρο επεξεργασίας. Πάλι, παρατηρούμε


Σχήμα 6.11: *Υψηλό Επίπεδο Αιχμής - Φόρτος Κόμβου 1*

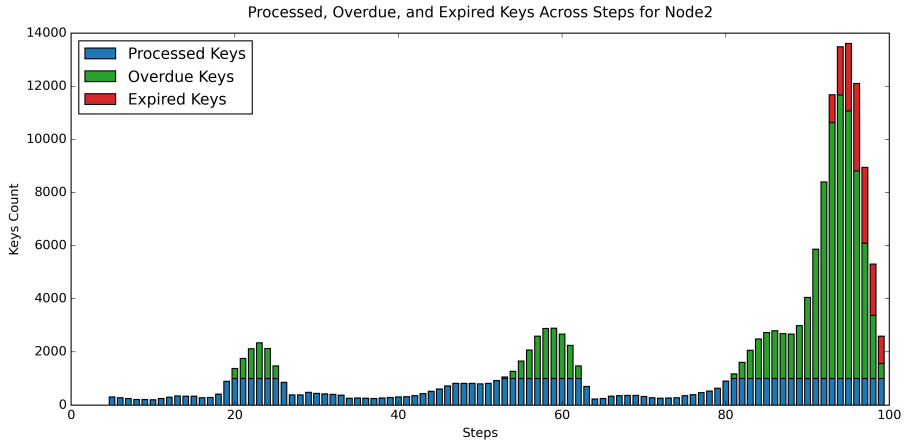



Σχήμα 6.12: *Υψηλό Επίπεδο Αιχμής - Φόρτος Κόμβου 2*

ότι το σύστημα επιστρέφει σε κανονική λειτουργία μετά την επεξεργασία των καθυστερημένων κλειδιών που δεν πρόλαβαν να λήξουν.




Σχήμα 6.13: *Υψηλό Επίπεδο Αιχμής - Μετρικές Επεξεργασίας Κλειδιών Κόμβου 1*




Σχήμα 6.14: *Υψηλό Επίπεδο Αιχμής - Μετρικές Επεξεργασίας Κλειδιών Κόμβου 2*

### 6.1.5 Συμπεράσματα

Σε αυτή τη σειρά πειραμάτων, διερευνήσαμε κάποια συνήθη σενάρια διαφορετικών χαρακτηριστικών συνθηκών στα κατανεμημένα συστήματα επεξεργασίας συνεχούς ροής δεδομένων. Εκτός από το σενάριο κανονικής λειτουργίας του προσομοιωτή, εξετάσαμε επίσης και τις καταστάσεις αυξημένης ροής δεδομένων, της άνισης κατανομής κλειδιών, καθώς και της απότομης (spiked) αύξησης ροής εδομένων. Όλες οι παραπάνω συνθήκες είναι κλασικές παράμετροι που χρησιμοποιούνται ερευνητικά για την αξιολόγηση της συνολικής επίδοσης των DSPSs. Τα αποτελέσματα των πειραμάτων ήταν σε πλήρη συνέπεια με τις αναμενόμενες συμπεριφορές των κατανεμημένων συστημάτων επεξεργασίας συνεχών ροών δεδομένων, επιβεβαιώνοντας την ορθότητα του προσομοιωτή μας.

## 6.2 Πειράματα Επίδοσης Συστημάτων

Αποδεικνύοντας, την εγγυρότητα του προσομοιωτή μας προχωράμε στην επόμενη κατηγορία πειραμάτων στην οποία χρησιμοποιούμε τον προσομοιωτή μας για σύγκριση της επίδοσης state-of-the-art αλγορίθμων διαμέρισης κλειδιών καθώς και την κλιμακωσιμότητα της τοπολογίας με αύξηση των κόμβων συγκεκριμένων σταδίων.

Τα πειράματα οργανώνονται σε δύο κύριες κατηγορίες:

- **Στρατηγικές Διαμερισμού Κλειδιών**: Αυτή η κατηγορία επικεντρώνεται στην αξιολόγηση διαφορετικών στρατηγικών διαμερισμού κλειδιών. Συγκεκριμένα, συγκρίνουμε την απόδοση των στρατηγικών hashing, power of two choices, και partial key grouping, με στόχο την κατανόηση του τρόπου με τον οποίο κάθε στρατηγική επηρεάζει την κατανομή φόρτου και την αποδοτικότητα του συστήματος.

- **Έλεγχος Κλιμακωσιμότητας**: Σε αυτή την κατηγορία, εξετάζεται η δυνατότητα κλιμάκωσης των DSPSs μέσω της προσθήκης επιπλέον κόμβων επεξεργασίας. Μεταβάλλουμε το πλήθος των stateful κόμβων στο πρώτο στάδιο της πρώτης τοπολογίας, και στο δεύτερο στάδιο της δεύτερης τοπολογίας, όπου παρατηρείται ο υψηλότερος φόρτος

εργασίας, από 2 σε 1, 4 ή/και 6, και από 3 σε 4 και 6 αντίστοιχα, και αναλύουμε την απόδοση των νέων τοπολογιών.

Για κάθε κατηγορία, τα πειράματα επαναλαμβάνονται υπό τις ίδιες συνθήκες που εξετάστηκαν προηγουμένως (υψηλός ρυθμός άφιξης, ανισοκατανομή κλειδιών, και περίοδοι αιχμής) με ακριβώς τα ίδια εισαγόμενα δεδομένα, ώστε να διευκολύνεται η άμεση σύγκριση των αποτελεσμάτων και να ενισχύεται η αξιοπιστία των αποτελεσμάτων. Τονίζουμε ότι επικεντρωνόμαστε κυρίως στη σύγκριση της επίδοσης του συστήματος για κάθε τοπολογία με σταθερή είσοδο ανά τύπο πειράματος και όχι στην επίδραση διαφορετικών συνθηκών στην ίδια τοπολογία.

## 6.3 Τοπολογία 1 - Απλή Τοπολογία

### 6.3.1 Στρατηγικές Διαμέρισης Κλειδιών

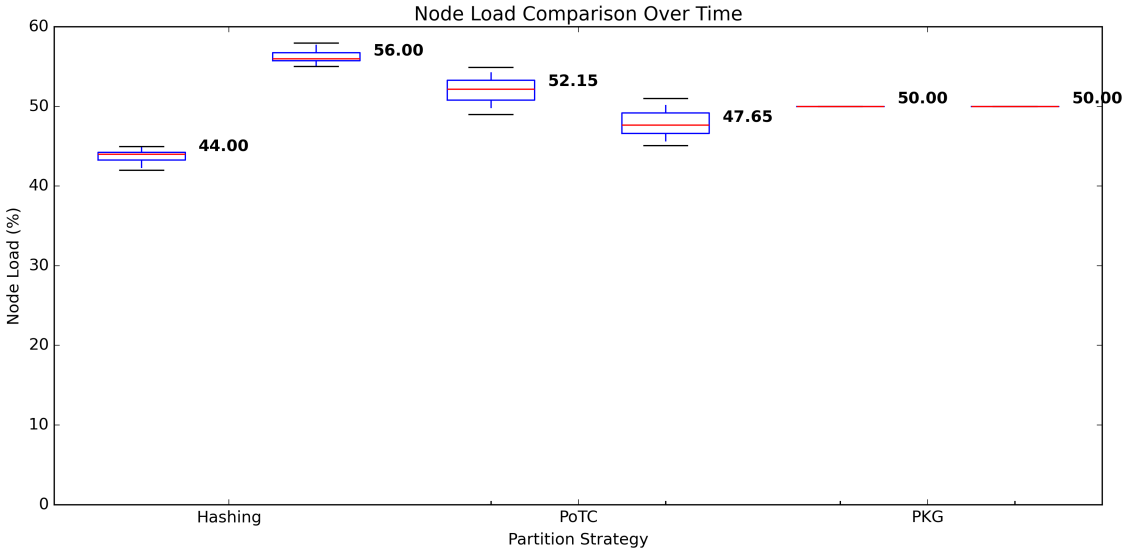
Η επιλογή κατάλληλης στρατηγικής διαμέρισης κλειδιών είναι ένα πολύ σημαντικό ζήτημα για την βελτιστοποίηση της απόδοσης των DSPSs. Επιλέξαμε να εξετάσουμε διαφορετικές στρατηγικές διαμέρισης από πολύ απλές έως πιο σύνθετες λόγω της εν γένει μεγάλης πολυπλοκότητας της αξιολόγησης διαφορετικών στρατηγικών διαμέρισης στα πραγματικά συστήματα συνεχούς ροής. Μέχρι την στιγμή συγγραφής αυτής της διπλωματικής εργασίας δεν υπάρχει κάποιο σύστημα συνεχούς ροής που να δίνει δυνατότητα παραμετροποίησης του τρόπου διαμέρισης των κλειδιών και για να εξετασθεί αυτό απαιτούνται αλλαγές στον εσωτερικό κώδικα του εκάστοτε συστήματος κάτι που είναι μία πολύ χρονοβόρα διαδικασία. Σημειώνουμε επίσης ότι από την φύση της υλοποίησης του προσομοιωτή η εξέταση της επίδοσης μίας νέας στρατηγικής είναι άμεση και απλώς απαιτείται η υλοποίηση μίας νέας κλάσης PartitionStrategy και η επιλογή της από το αρχείο παραμετροποίησης. Στο συγκεκριμένο πείραμα επιλέξαμε να υλοποίησουμε ορισμένες από τις στρατηγικές που παρουσιάστηκαν στην ενότητα 2.7.3 και πιο συγκεκριμένα, τις hashing, Power of Two Choices και Partial Key Grouping που υλοποιούν ένα εύρος στρατηγικών από πολύ απλές, ευρέως χρησιμοποιούμενες, έως και αρκετά πιο σύνθετες με υποσχόμενα θεωρητικά ερευνητικά αποτελέσματα.

Στη ενότητα αυτή, παρουσιάζεται η σύγκριση επίδοσης της κάθε στρατηγικής διαμερισμού κλειδιών στα διαφορετικά πειράματα που χρησιμοποιήσαμε νωρίτερα για να επαληθεύσουμε την ορθότητα του προσομοιωτή μας.

#### Κανονική Λειτουργία

Η κανονική λειτουργία είναι μία πρώτη μετρική που μπορεί να αξιολογήσει την επίδοση των διαφορετικών στρατηγικών διαμερισμού κλειδιών. Εξάλλου, τα συστήματα τις περισσότερες φορές βρίσκονται σε κατάσταση κανονικής λειτουργίας και μία καλύτερη επίδοση σε αυτό το πείραμα δείχνει μία γενική βελτίωση στην απόδοση του συστήματος. Στο σχήμα 6.15 φαίνεται ο φόρτος των κόμβων 1 και 2 (δηλαδή των κόμβων του πρώτου σταδίου) για κάθε στρατηγική διαμέρισης.

Συμβουλευόμενοι το διάγραμμα φόρτου παρατηρούμε ότι η στρατηγική κατακερματισμού παρουσιάζει την μεγαλύτερη απόκλιση φόρτου μεταξύ των κόμβων του ίδιου σταδίου

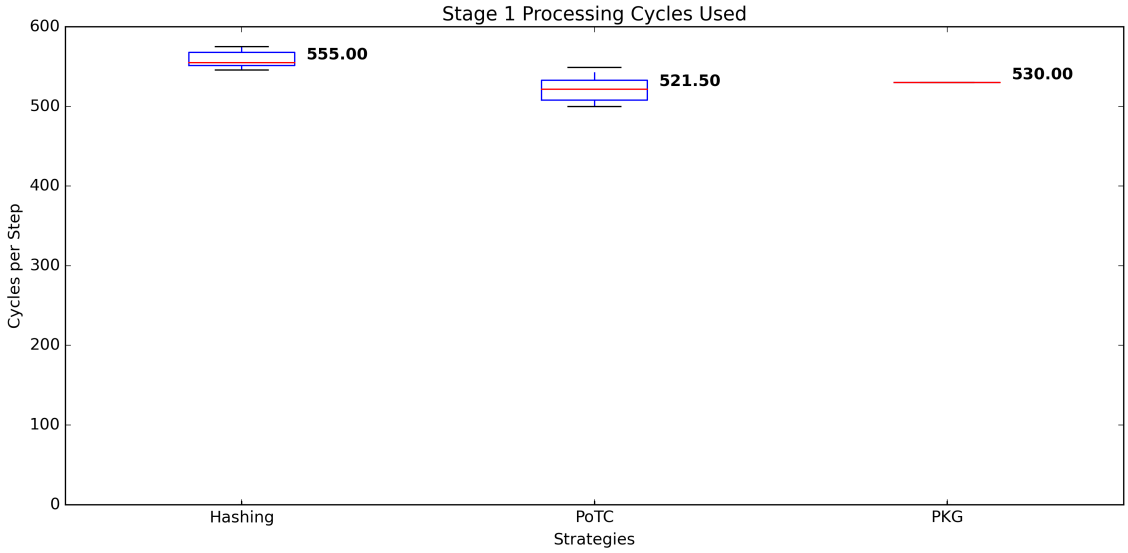



Σχήμα 6.15: *Κανονική Λειτουργία - Φόρτος Εργασίας Κόμβων ανά Στρατηγική Διαμέρισης*

(τιμές διαμέσου: 44% για τον κόμβο 1 και 56% για τον κόμβο 2), ακόμα και στο παράδειγμα με δεδομένα όσο το δυνατόν περισσότερο ισομοιρασμένα. Αντίθετα, ο στρατηγική Partial Key Grouping (PKG) παρουσιάζει τέλειο ισομοιρασμό 50% φόρτου μεταξύ των δύο κόμβων καθόλη την διάρκεια του πειράματος. Ο τέλειος διαμέρισμός οφείλεται στο ότι ο PKG πάντα επιλέγει δύο πιθανούς κόμβους διαμερισμού και διαλέγει αυτόν με τον τρέχον μικρότερο φόρτο. Άρα, εφόσον μπορεί να διαμερίσει κάθε κλειδί σε περισσότερους του ενός κόμβους, στην περίπτωση των δύο κόμβων λειτουργεί ανάλογα με μία στρατηγική Round Robin λόγω της ύπαρξης πάντα ενός εναλλακτικού κόμβου. Η στρατηγική Power of Two Choices (PoTC) παρουσιάζει μία ενδιάμεση επίδοση μεταξύ των άλλων δύο στρατηγικών. Σε άμεση σύγκριση με τον απλό κατακερματισμό, δεδομένου ότι και οι δύο αυτές στρατηγικές δεν μπορούν να διαχωρίσουν κλειδιά σε πολλαπλούς κόμβους, έχει καλύτερη επίδοση με τιμές διαμέσου φόρτου (47.65% και 52.15%).

Ένα πρώτο συμπέρασμα μας φανερώνει ότι η στρατηγική PKG επιτυγχάνει το καλύτερο ισοζύγισμα φόρτου εργασίας. Αυτό βέβαια, δεν μεταφράζεται απαραίτητα σε καλύτερη επίδοση συστήματος και συγκεκριμένα επίδοση του συγκεκριμένου σταδίου λόγω της ύπαρξης και του κόμβου συνάθροισης (Aggregator) στο αμέσως επόμενο επίπεδο από τους κόμβους επεξεργασίας. Ο κόμβος συνάθροισης, θα προσθέσει στον συνολικό αριθμό κύκλων επεξεργασίας του συγκεκριμένου σταδίου. Στο σχήμα 6.16 φαίνονται οι κύκλοι επεξεργασίας που χρειάστηκαν κατά μέσο όρο σε κάθε στρατηγική διαμερισμού.

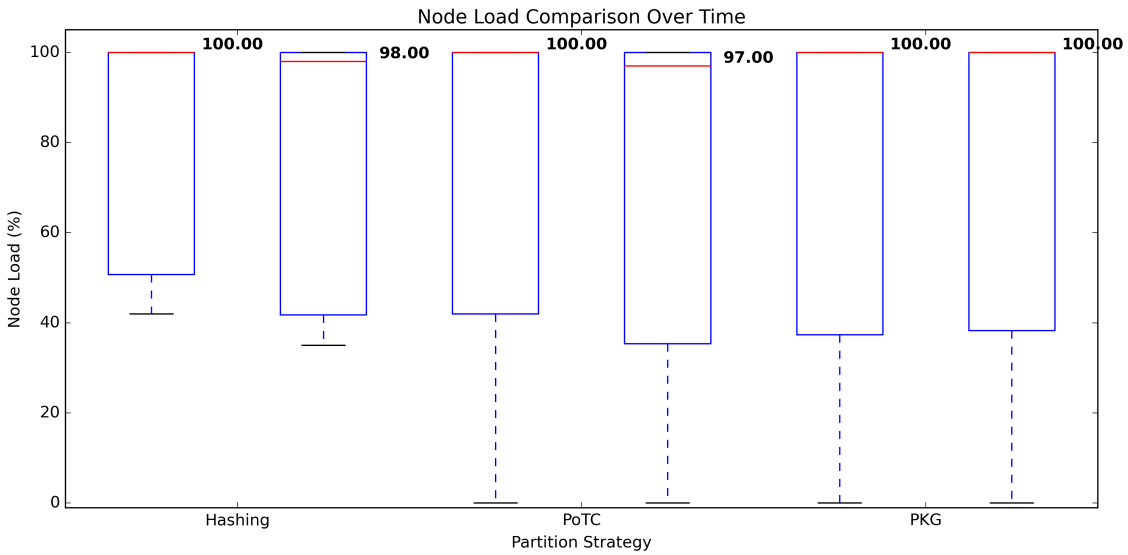
Όπως βλέπουμε, παρά το γεγονός ότι η PKG στρατηγική οδηγεί το σύστημα σε τέλειο ισομοιρασμό του φόρτου, ο κόμβος συνάθροισης που προστίθεται στην τοπολογία έχει ως αποτέλεσμα τον μεγαλύτερο μέσο αριθμό κύκλων επεξεργασίας ανά βήμα στο στάδιο. Στην πραγματικότητα αυτό θα παρατηρηθεί ως μία αύξηση του latency στα επεξεργασμένα κλειδιά.




Σχήμα 6.16: *Κανονική Λειτουργία - Μέσος Αριθμός Κύκλων Επεξεργασίας*

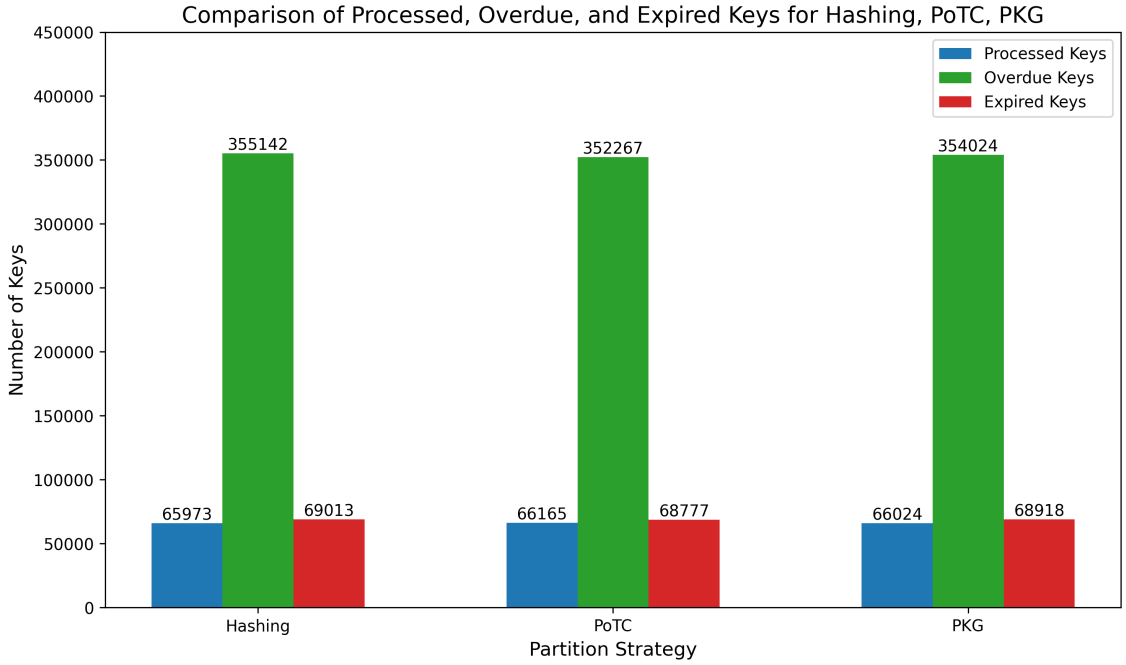
### Αυξανόμενος Ρυθμός Άφιξης Κλειδιών

Ως δεύτερη μετρική επίδοσης των διαφορετικών στρατηγικών διαμέρισης κλειδιών χρησιμοποιήσαμε την ανοχή και την συμπεριφορά του συστήματος σε σταδιακά αυξανόμενο ρυθμό άφιξης κλειδιών. Στο σχήμα 6.17 φαίνεται ο φόρτος των κόμβων του πρώτου σταδίου της τοπολογίας.




Σχήμα 6.17: *Αυξανόμενος Ρυθμός Άφιξης - Φόρτος Εργασίας Κόμβων ανά Στρατηγική Διαμέρισης*

Όπως φαίνεται, και οι 3 στρατηγικές διαμέρισης κλειδιών έχουν παρόμοια επίδοση στο πείραμα αυτό. Η διάμεσος του φόρτου είναι περίπου 100%, δηλαδή βρίσκεται σε μέγιστη χωρητικότητα για πάνω από το 50% των βημάτων προσομοίωσης. Στο σχήματα 6.18 φαίνονται αναλυτικά και τα συνολικά επεξεργασμένα, καθυστερημένα και ληγμένα κλειδιά κατά τη διάρκεια της προσομοίωσης.



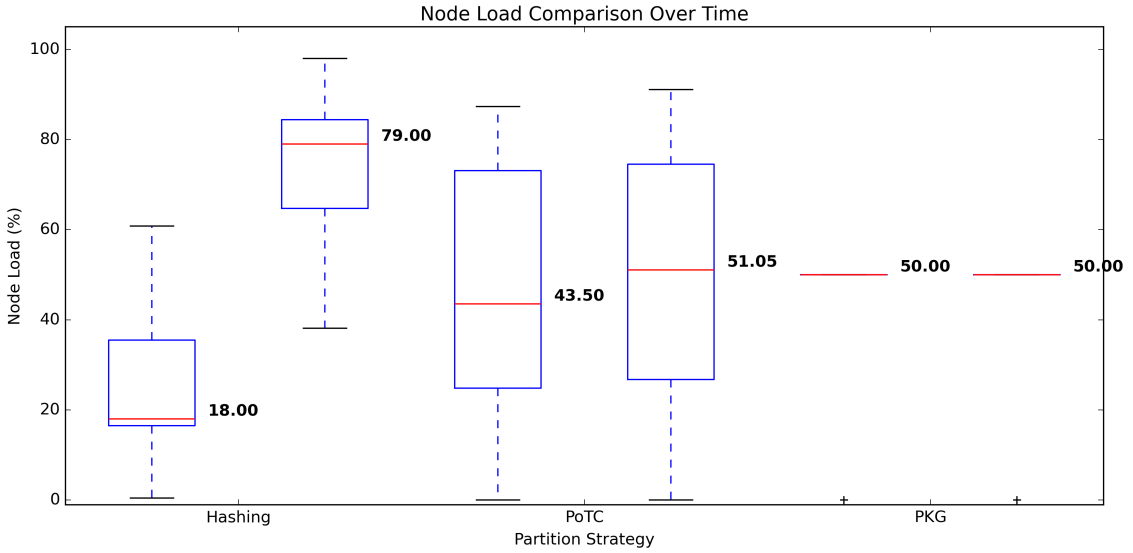

Σχήμα 6.18: *Αυξανόμενος Ρυθμός Άφιξης - Μετρικές Επεξεργασίας Κλειδιών*

Παρατηρούμε ότι οι μετρικές των κλειδιών είναι πανομοιότυπες για κάθε στρατηγική διαμερισμού. Καταλήγουμε λοιπόν στο συμπέρασμα ότι το bottleneck του συστήματος είναι ο αριθμός των κόμβων και όχι ο τρόπος διαμεριμού των κλειδιών. Όπως έχουμε αναφέρει, για το συγκεκριμένο πείραμα εισάγονται δεδομένα που ακολουθούν ομοιόμορφη κατανομή άρα δεν εμφανίζονται μεγάλες διακυμάνσεις κλειδιών που πιθανώς μία καλύτερη στρατηγική θα βελτίωνε τον διαμερισμό τους.

### Ανισοκατανομή Κλειδιών

Στο προηγούμενο πείραμα συμπεράναμε ότι η επιλογή καλύτερης στρατηγικής διαμέρισης κλειδιών δεν είχε ουσιώδη διαφορά στην επίδοση του συστήματος και ότι αυτό οφειλόταν κυριότερα στο γεγονός ότι τα δεδομένα ήταν ομοιόμορφα κατανεμημένα. Στο πείραμα αυτό εξετάσαμε την επίδοση των διαφορετικών στρατηγικών σε δεδομένα που είναι ανισοκατανεμημένα. Στο σχήμα 6.19 φαίνεται η επίδοση των κόμβων του πρώτου σταδίου για κάθε στρατηγική.

Τα αποτελέσματα αυτού του πειράματος είναι αρκετά πιο ενδιαφέροντα σε σχέση με το προηγούμενο πείραμα. Είναι εμφανές ότι η επιλογή κατάλληλης στρατηγικής είναι άμεσα συνδεδεμένη με τον μέσο φόρτο των κόμβων του σταδίου. Πιο συγκεκριμένα, με διαμέριση μέσω κατακερματισμού οι κόμβοι εμφανίζουν έντονη ανισοκατανομή φορτίου μεταξύ τους. Κατά μέσο όρο οι δύο κόμβοι έχουν απόκλιση φόρτου της τάξης του 4. Μία τέτοια ανισοκατανομή φορτίου οδηγεί σε υποχρησιμοποιήση των πόρων του συστήματος και σε πραγματικά συστήματα θα οδηγούσε σε άσκοπη αύξηση του latency. Με τον πιο έξυπνο καταμερισμό κλειδιών χωρίς διάσπαση ίδιων κλειδιών σε πολλαπλούς κόμβους, δηλαδή στο Power of Two Choice (PoTC), παρατηρούμε μία σημαντική βελτίωση της διαμέρισης των κλειδιών που οδηγεί σε καλύτερα ισομοιρασμένο φόρτο εργασιών μεταξύ των κόμβων. Από την απόκλιση της



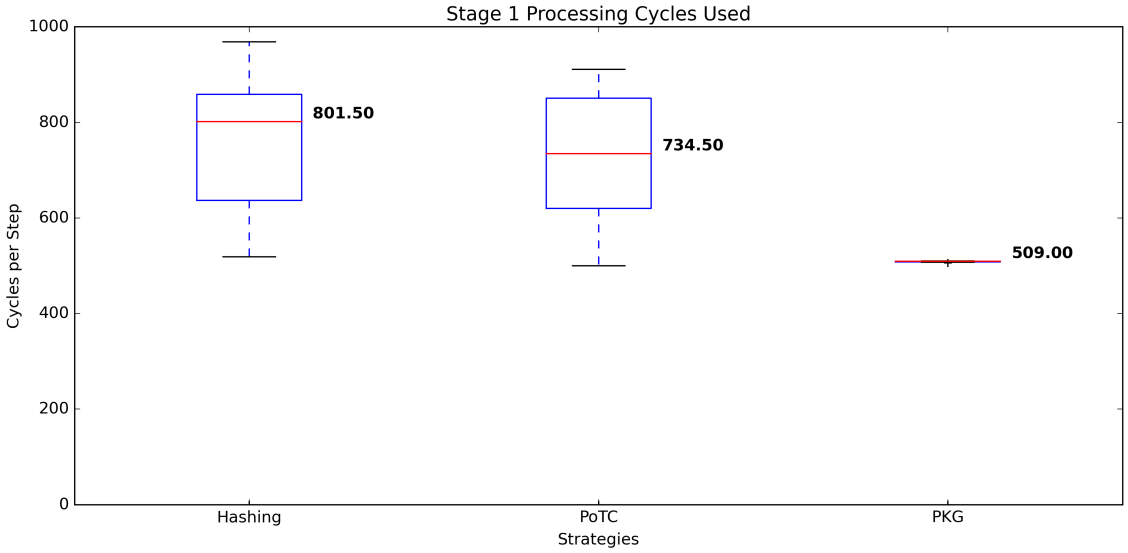

Σχήμα 6.19: *Ανισοκατανομή - Φόρτος Εργασίας Κόμβων ανά Στρατηγική Διαμέρισης*

τάξεως των 4 φορών της προηγούμενης στρατηγικής παρατηρούμε αρκετά ισομοιρασμένο φόρτο με τιμές διαμέσου 43.5% και 51.05% μεταξύ των δύο κόμβων. Τονίζουμε όμως ότι από το σχήμα η απόκλιση αυτών των τιμών δεν είναι αμελητέα άρα σε συγκεκριμένα βήματα παρατηρείται και πάλι ανισοκαταμερισμός του φορτίου. Αυτό μπορούμε να το εξηγήσουμε ότι στη μέση περίπτωση η στρατηγική PoTC λειτουργεί αρκετά πιο αποδοτικά αλλά στην χειρότερη περίπτωση είναι μόνο ελαφρώς καλύτερη από το hashing. Αυτό ερμηνεύεται σε περιπτώσεις με έντονη ανισοκατανομή της εισόδου όπου ο κύριος φόρτος αυτών των κλειδιών μπορεί να διαμεριστεί μόνο σε έναν συγκεκριμένο κόμβο και να μην μπορεί να αντισταθμηθεί από την κατανομή των υπόλοιπων κλειδιών στον άλλον κόμβο, δημιουργώντας έτσι μεγάλη απόκλση στον μεταξύ τους φόρτο. Τέλος, το Partial Key Grouping αντιμετωπίζει αυτή την περίπτωση εφόσον επιλέγει τον διαχωρισμό ίδιων κλειδιών ώστε να επιτύχει καλύτερο load balancing. Όπως αναφέραμε, στην περίπτωση των δύο κόμβων επιτυγχάνει πάντα 50% φόρτο σε σταθερό ρυθμό άφιξης. Πρέπει να εξετάσουμε όμως και την επιβάρυνση σε κύκλους επεξεργασίας που επιφέρει η συνάθροιση των κλειδιών στον Aggregator κόμβο ώστε να μπορούμε να αποφανθούμε αν η PKG είναι όντως η κατάλληλη στρατηγική για τα εν λόγω φορτία. Στο σχήμα 6.20 παρουσιάζονται συγκεκριμένα οι κύκλοι επεξεργασίας.

Εν αντιθέσει με το πείραμα κανονικής λειτουργίας βλέπουμε ότι παρά την ύπαρξη του επιπλέον επιπέδου συνάθροισης η στρατηγική PKG χρησιμοποιεί πολύ λιγότερους μέσους κύκλους επεξεργασίας σε σχέση με τις υπόλοιπες στρατηγικές. Μπορούμε να συμπεράνουμε λοιπόν, ότι η στρατηγική διάσπασης κλειδιών σε καταστάσεις ανισοκατανομής κλειδιών επιφέρει μία σημαντική βελτίωση επίδοσης. Όπως θα δούμε και στην συνέχεια, στην δεύτερη τοπολογία, εξίσου σημαντικό ρόλο παίζει και ο τύπος της προσομοιούμενης λειτουργίας του σταδίου και συνεπώς η πολυπλοκότητα της συνάρτησης συνάθροισης.

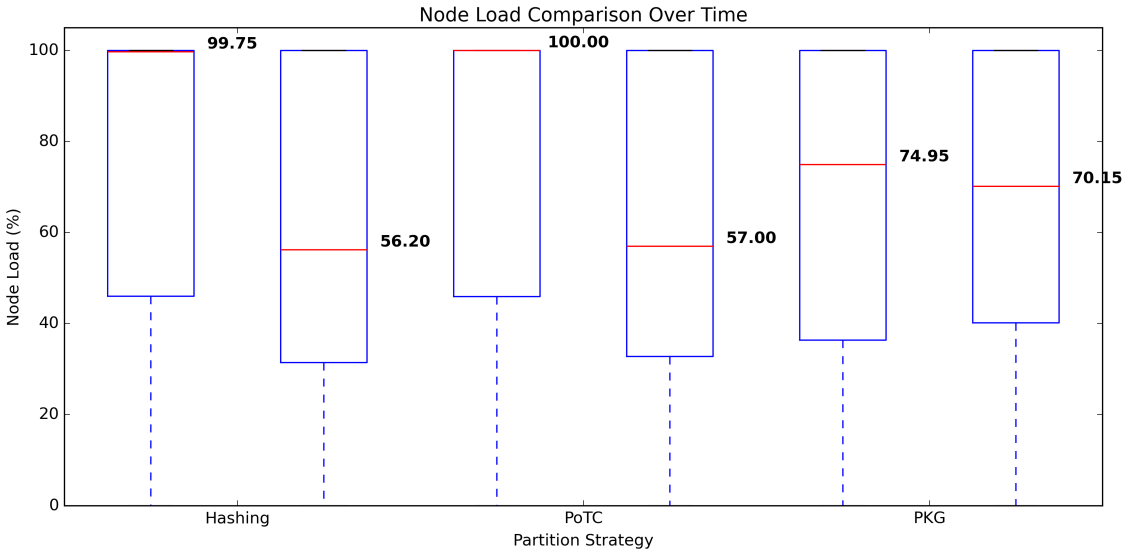
#### Μεταβαλλόμενο Spike

Το τελευταίο πείραμα για την ολοκληρωμένη αξιολόγηση των διαφορετικών στρατηγικών διαμερισμού στην πρώτη τοπολογία, είναι η εξέταση της συμπεριφοράς τους με είσοδο δε-




Σχήμα 6.20: *Ανισοκατανομή - Μέσος Αριθμός Κύκλων Επεξεργασίας*

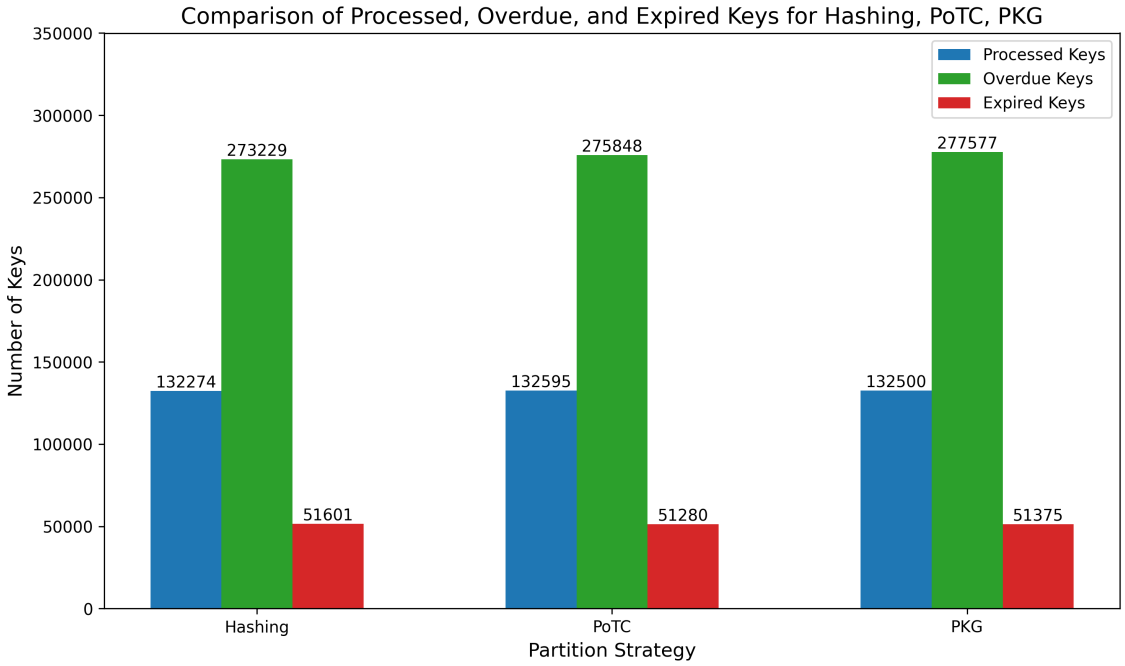
δομένα που εμφανίζουν spikes κατά την διάρκεια των βημάτων. Στο σχήμα 6.21 φαίνεται ο φόρτος των κόμβων κατά την εξέλιξη της προσομοίωσης.




Σχήμα 6.21: *Αιχμές - Φόρτος Εργασίας Κόμβων ανά Στρατηγική Διαμέρισης*

Και στο πείραμα αυτό βλέπουμε ότι το Partial Key Grouping (PKG) επιτυγχάνει καλύτερο μέσο φόρτο των κόμβων του σταδίου. Οι πρώτες δύο στρατηγικές, hashing και Power of Two Choices (PoTC) παρουσιάζουν παρόμοια συμπεριφορά με τον έναν κόμβο να βρίσκεται συνήθως στο 100% της χωρητικότητας ενώ ο άλλος πάνω από 55%. Αντίθετα, στο PKG ο μέσος φόρτος είναι 74.95% και 70.15% αντίστοιχα. Βέβαια, σε όλες τις στρατηγικές οι κόμβοι φτάνουν στο μέγιστο της χωρητικότητας αδυνατώντας να επεξεργαστούν έναν σημαντικό αριθμό κλειδιών όπως φαίνεται στο σχήμα 6.22. Η εξήγηση για το φαινόμενο αυτό πηγάζει από τα χαρακτηριστικά του ρυθμού άφιξης στο συγκεκριμένο πείραμα. Σε κάποια βήματα ο ρυθμός ξεπερνάει κατά πάρα πολύ την μέγιστη χωρητικότητα των κόμβων υπερφορτώνο-

ντάς τους. Σε αυτές τις περιπτώσεις, μικρή σημασία έχει η αλλαγή στρατηγικής διαμέρισης, εφόσον το bottleneck κείτεται στην συνολική μέγιστη ρυθμαπόδοση του σταδίου δεδομένου ότι παρατηρούμε στα συγκεκριμένα βήματα φόρτο 100% σε όλους τους κόμβους ανεξαρτήτου της χρησιμοποιούμενης στρατηγικής.




Σχήμα 6.22: *Αιχμές - Μετρικές Επεξεργασίας Κλειδιών*

### 6.3.2 Κλιμακωσιμότητα

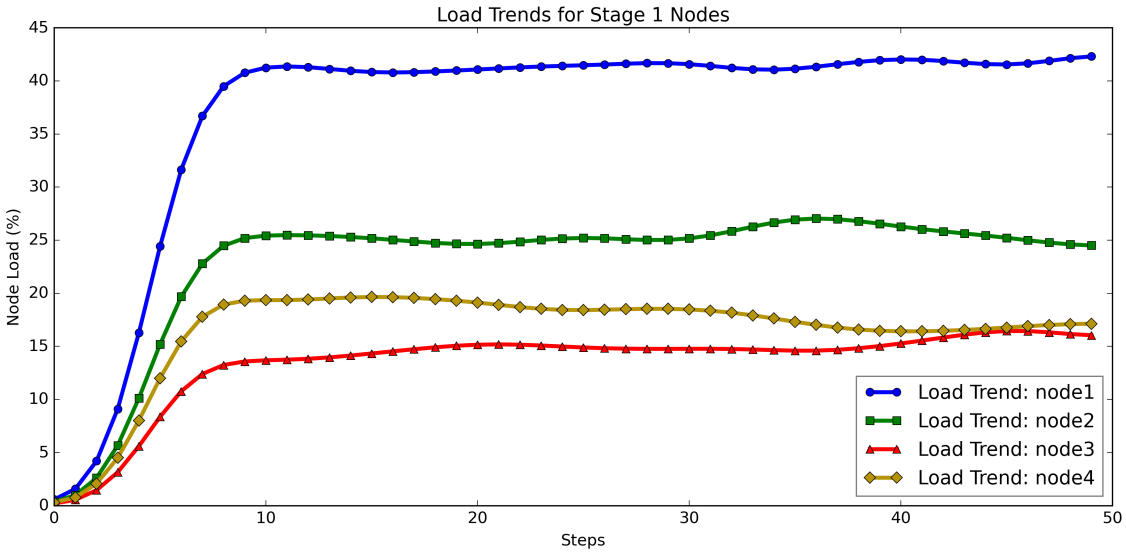
Σε αυτή τη σειρά πειραμάτων, μεταβάλλουμε το πλήθος των κόμβων στο πρώτο στάδιο (stateful) της τοπολογίας, καθώς είναι το στάδιο με τον μεγαλύτερο φόρτο εργασίας και το περισσότερο ενδιαφέρον στην συγκεκριμένη τοπολογία, προκειμένου να εξετάσουμε την κλιμακωσιμότητα του συστήματος.

#### Κανονική Λειτουργία

Στην αρχή, αυξάνουμε τους κόμβους του σταδίου από 2 σε 4 και εκτελούμε το πείραμα υπό κανονικές συνθήκες λειτουργίας.

Το Σχήμα 6.23 παρουσιάζει την κατανομή του φόρτου εργασίας για κάθε κόμβο. Συγκρίνοντας με το Σχήμα 6.2, όπου φαίνεται ο φόρτος του κάθε κόμβου στην περίπτωση τοπολογίας με δύο κόμβους στο πρώτο στάδιο, παρατηρείται σημαντική μείωση στον φόρτο. Συγκεκριμένα, η χειρότερη επίδοση κόμβου στην τοπολογία με 2 κόμβους ήταν κατά μέσο όρο στο 57% στον κόμβο 1 κατά την διάρκεια του πειράματος ενώ στην τοπολογία με 4 κόμβους έχουμε μέγιστο μέσο όρο φόρτου ίσο με 42% πάλι στον κόμβο 1. Όπως και αναμενόταν στις συνθήκες κανονικής λειτουργίας που τα κλειδιά είναι ισομοιρασμένα και με σταθερό ρυθμό αύξησης η αύξηση κόμβων επεξεργασίας μεταφράζεται σε μικρότερο συνολικό φόρτο του συστήματος. Αξίζει να σημειώσουμε εδώ, ότι η ποσοτική βελτίωση δεν είναι γραμμική εφόσον με διπλασιαμό των κόμβων έχουμε περίπου 25% βελτίωση επίδοσης. Αυτό

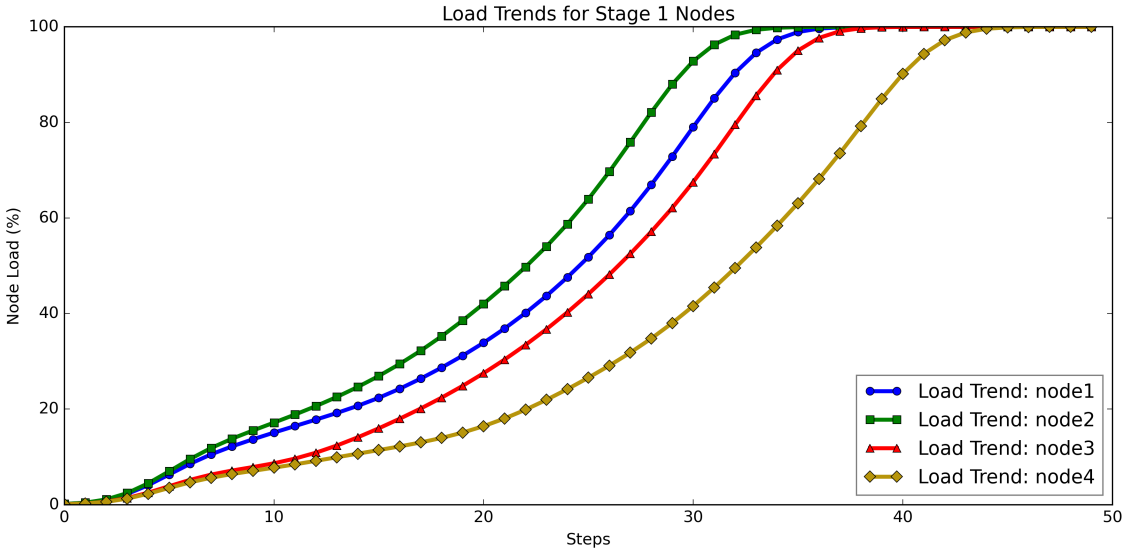
εξηγείται μέσω του σχήματος που μπορούμε να δούμε ξεκάθαρα σημαντική διακύμανση του φόρτου μεταξύ των κόμβων του ίδιου σταδίου και θα μπορούσε να βελτιωθεί με την επιλογή μίας καλύτερης στρατηγικής διαμέρισης.




Σχήμα 6.23: *Κανονική Λειτουργία - Κατανομή Φόρτου Εργασίας με 4 Κόμβους*

### Αυξανόμενος Ρυθμός Άφιξης Κλειδιών

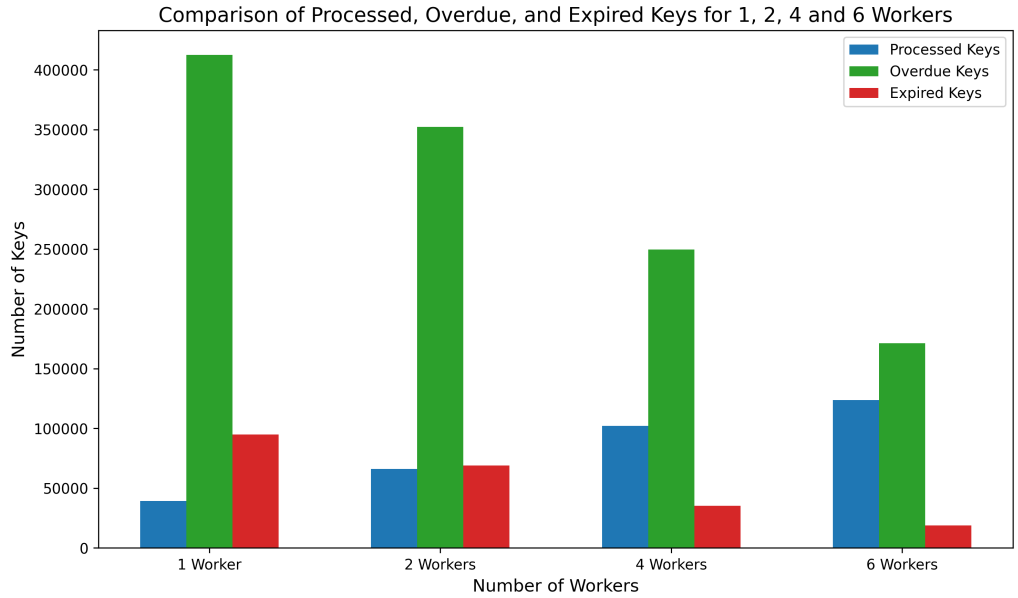
Στη συνέχεια, εξετάζουμε πώς συμπεριφέρεται το σύστημα όταν ο ρυθμός άφιξης κλειδιών αυξάνεται σταδιακά. Το Σχήμα 6.24 δείχνει ότι οι 4 κόμβοι υπερφορτώνονται με πιο αργό ρυθμό σε σχέση με την τοπολογία των 2 κόμβων, όπως φαίνεται και στο Σχήμα 6.3. Οι κόμβοι φτάνουν το peak φόρτου μετά το βήμα 35, σε αντίθεση με το βήμα 30 που παρατηρήθηκε στην προηγούμενη τοπολογία.




Σχήμα 6.24: *Αυξανόμενος Ρυθμός Άφιξης - Κατανομή Φόρτου Εργασίας με 4 Κόμβους*

Το γεγονός ότι το σύστημα φτάνει σε σημείο κορεσμού αργότερα υποδηλώνει ότι μπορεί

να επεξεργαστεί περισσότερα κλειδιά, με λιγότερα να λήγουν. Επαναλαμβάνοντας το πείραμα με 1, 2, 4 και 6 κόμβους, παρατηρούμε στο Σχήμα 6.25 ότι, καθώς αυξάνεται ο αριθμός των κόμβων και επομένως ο βαθμός παραλληλίας, τα έγκαιρα επεξεργασμένα κλειδιά αυξάνονται, ενώ τα εκπρόθεσμα και ληγμένα κλειδιά μειώνονται. Το αποτέλεσμα είναι λογικό και επιβεβαιώνει την υπόθεση κατά την εξέταση του ίδιου πειράματος με διαφορετικές στρατηγικές διαμέρισης. Περισσότεροι κόμβοι αυξάνουν την συνολική ρυθμαπόδοση κλειδιών του σταδίου και άρα η κλιμακωσιμότητα επιφέρει βελτίωση απόδοσης με την αύξηση του φόρτου δεδομένων.




Σχήμα 6.25: *Αυξανόμενος Ρυθμός Άφιξης - Μετρικές Επεξεργασίας Κλειδιών*

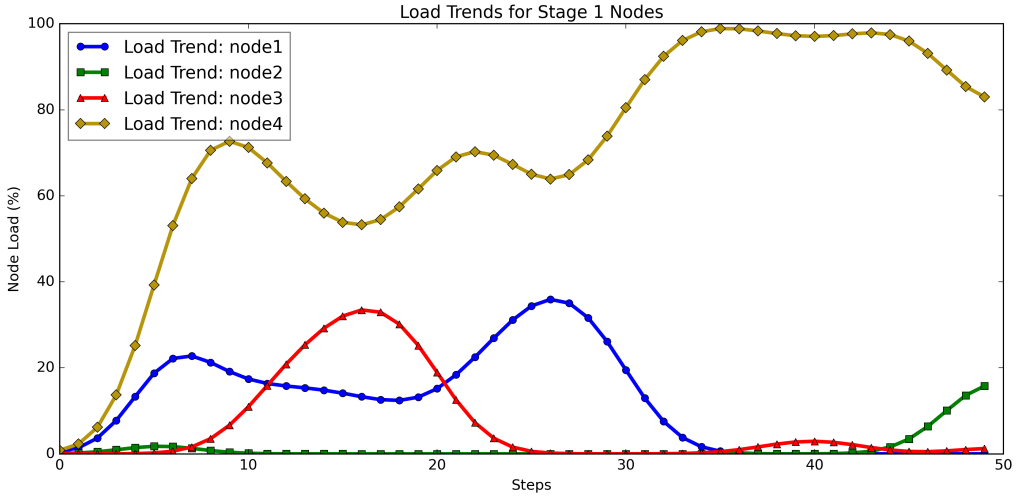
### Υψηλή Ανισοκατανομή Κλειδιών

Εκτελούμε το πείραμα με 4 κόμβους και σε συνθήκες υψηλής ανισοκατανομής κλειδιών. Στο Σχήμα 6.26 φαίνεται ότι η στρατηγική hashing, σε συνδυασμό με την ανισοκατανομή, οδηγεί σε ανομοιόμορφη κατανομή φόρτου, με έναν κόμβο να πλησιάζει το 100%. Παρά ταύτα, ο συνολικός φόρτος παραμένει μικρότερος σε σχέση με την τοπολογία των 2 κόμβων.

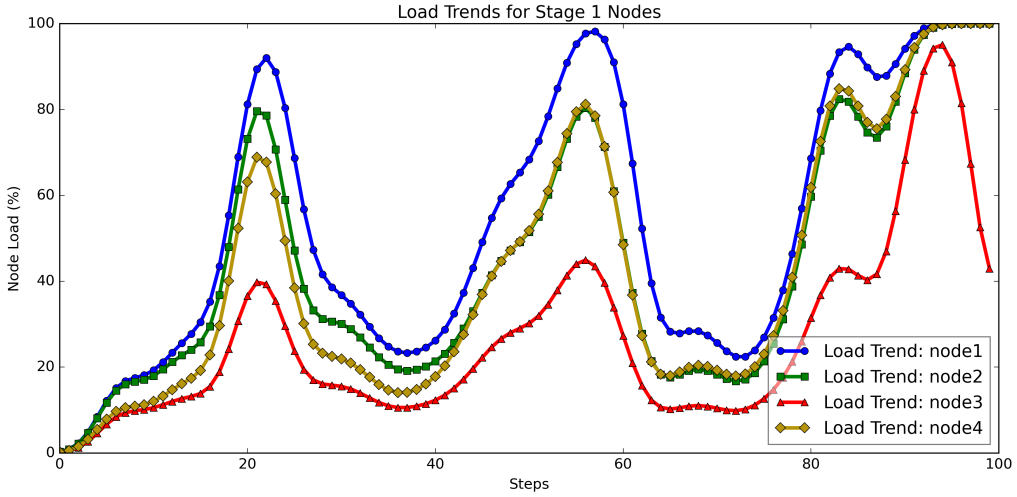
### Μεταβαλλόμενο (Spike)

Τέλος, εξετάζουμε την επίδραση των περιόδων αιχμής (spike) στο σύστημα. Στο Σχήμα 6.27, παρατηρούμε ότι όλοι οι κόμβοι εμφανίζουν παρόμοια συμπεριφορά, με ξαφνικές αυξήσεις και μειώσεις του φόρτου εργασίας, όπως αναμενόταν κατά τις περιόδους αιχμής. Τονίζουμε, όμως ότι σε συγκεκριμένες αιχμές παρατηρείται σημαντική ανισοκατανομή του φορτίου ανά κομβο υποδηλώνοντας ότι κάποια καλύτερη στρατηγικής διαμέρισης θα μπορούσε να βελτίωνε ακόμα περισσότερο την επίδοση του συστήματος.

Επαναλαμβάνοντας τα πειράματα με 1, 2, 4 και 6 κόμβους, βλέπουμε στο Σχήμα 6.28 ότι τα αποτελέσματα είναι συνεπή με τα προηγούμενα πειράματα: καθώς αυξάνεται το πλήθος




Σχήμα 6.26: *Ανισοκατανομή - Φόρτος Εργασίας σε Συνθήκες με 4 κόμβους*



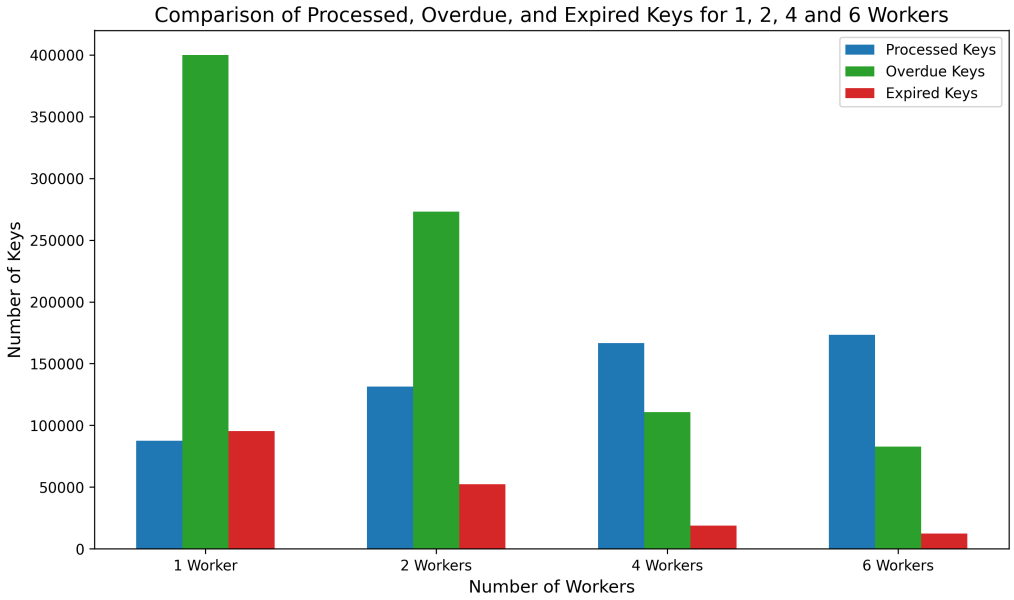

Σχήμα 6.27: *Μεταβαλλόμενο Spike - Κατανομή Φόρτου Εργασίας με 4 Κόμβους*

των κόμβων, ο αριθμός των επεξεργασμένων κλειδιών αυξάνεται, ενώ ο αριθμός των εκπρόθεσμων και ληγμένων κλειδιών μειώνεται σταθερά.

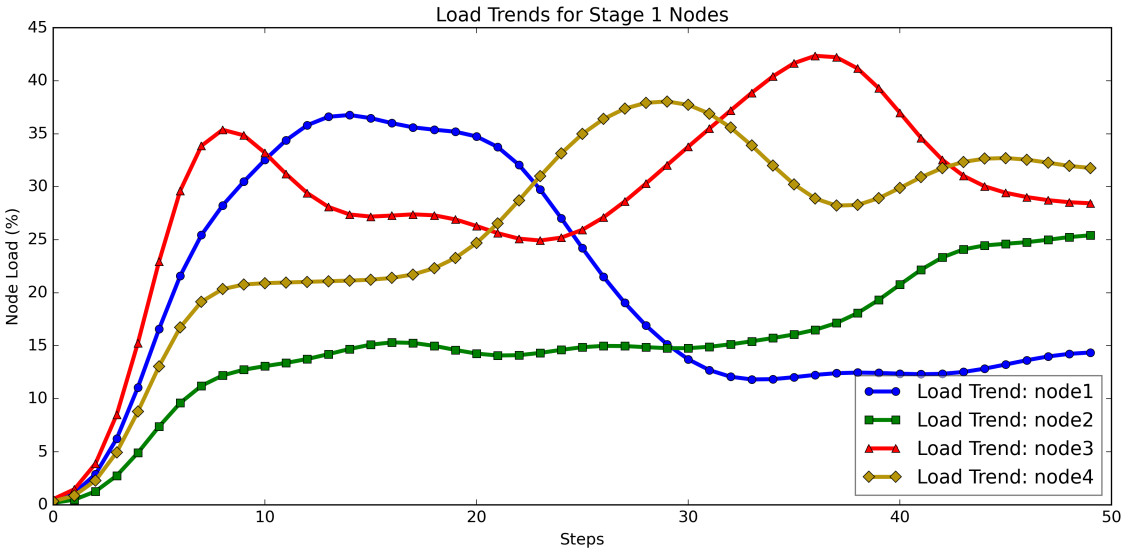
### 6.3.3 Μεταβαλλόμενο Πλήθος Κόμβων και Στρατηγικές Διαμερισμού Κλειδιών

Παρατηρώντας το Σχήμα 6.26, είδαμε ότι η στρατηγική hashing αποτελεί φραγμό σε συνθήκες ανισοκατανομής των κλειδιών, καθώς κάποια hot keys ανατίθενται σε ορισμένους μόνο κόμβους με αποτέλεσμα αυτοί να έχουν αρκετά υψηλότερο φόρτο εργασίας. Αξίζει λοιπόν, να εξετάσουμε πώς λειτουργούν τα συστήματα με περισσότερους κόμβους, όταν χρησιμοποιείται μία πιο έξυπνη στρατηγική διαμέρισης των κλειδιών. Για το σκοπό αυτό επαναλαμβάνουμε το πείραμα με υψηλή ανισοκατανομή και 4 κόμβους επεξεργασίας στο στάδιο 1, εφαρμόζοντας στρατηγική PoTC και PKG (σχήματα 6.29 και 6.30 αντίστοιχα).

Όπως βλέπουμε, η PoTC βελτιώνει τον διαμοιρασμό του φόρτου μεταξύ των κόμβων, ωστόσο και πάλι τα αποτελέσματα δεν είναι ικανοποιητικά, καθώς υπάρχουν σημαντικές

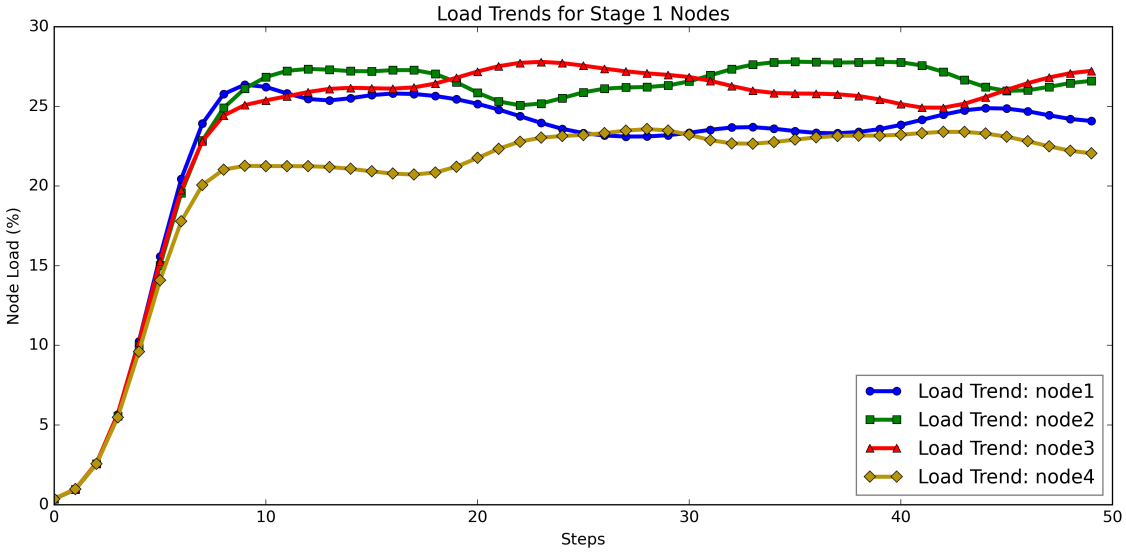



Σχήμα 6.28: *Μεταβαλλόμενο Spike - Μετρικές Επεξεργασίας Κλειδιών*




Σχήμα 6.29: *PoTC Στρατηγική με 4 Κόμβους και Υψηλή Ανισοκατανομή*

αυξομοιώσεις και αποκλίσεις μεταξύ των κόμβων. Εφαρμόζοντας όμως τη στρατηγική PKG βλέπουμε ότι το φόρτος μεταξύ των κόμβων είναι αρκετά κοντά και δείχνει να συγκλίνει. Αργότερα θα δούμε ότι με χρήση περισσότερων βημάτων προσομοίωσης και πιο σύνθετης τοπολογίας, τα αποτελέσματα είναι ακόμη περισσότερο ικανοποιητικά.




Σχήμα 6.30: *PKG Στρατηγική με 4 Κόμβους και Υψηλή Ανισοκατανομή*

## 6.4 Τοπολογία 2 - Σύνθετη Τοπολογία

### 6.4.1 Στρατηγικές Διαμέρισης Κλειδιών

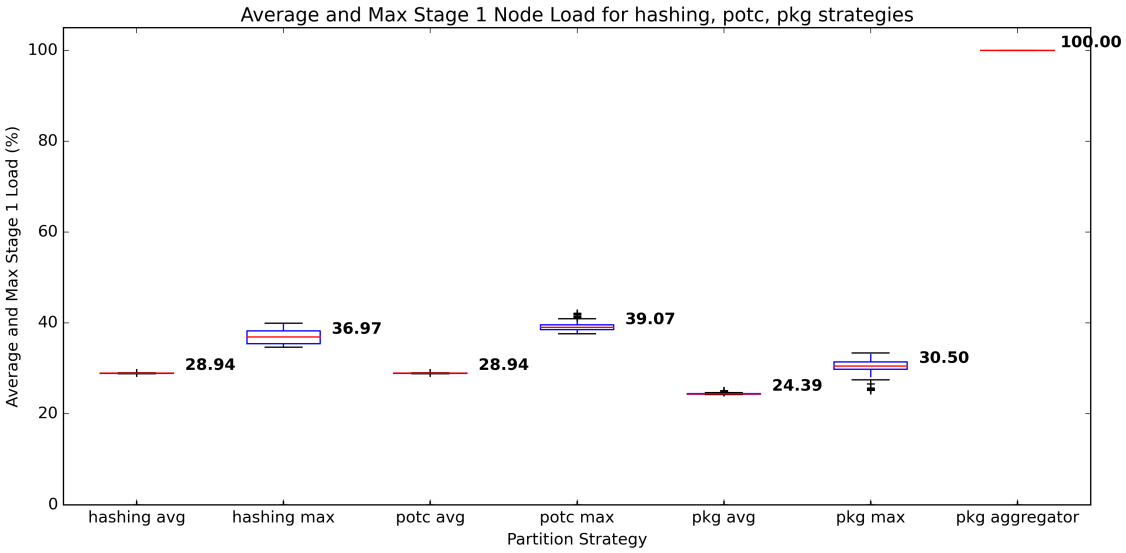
#### Κανονική Λειτουργία

Στα σχήματα 6.31 και 6.32 φαίνεται ο μέσος φόρτος του πρώτου και δεύτερου σταδίου της τοπολογίας καθώς και ο μέσος μέγιστος φόρτος κόμβου (peak load) σε κάθε στάδιο.

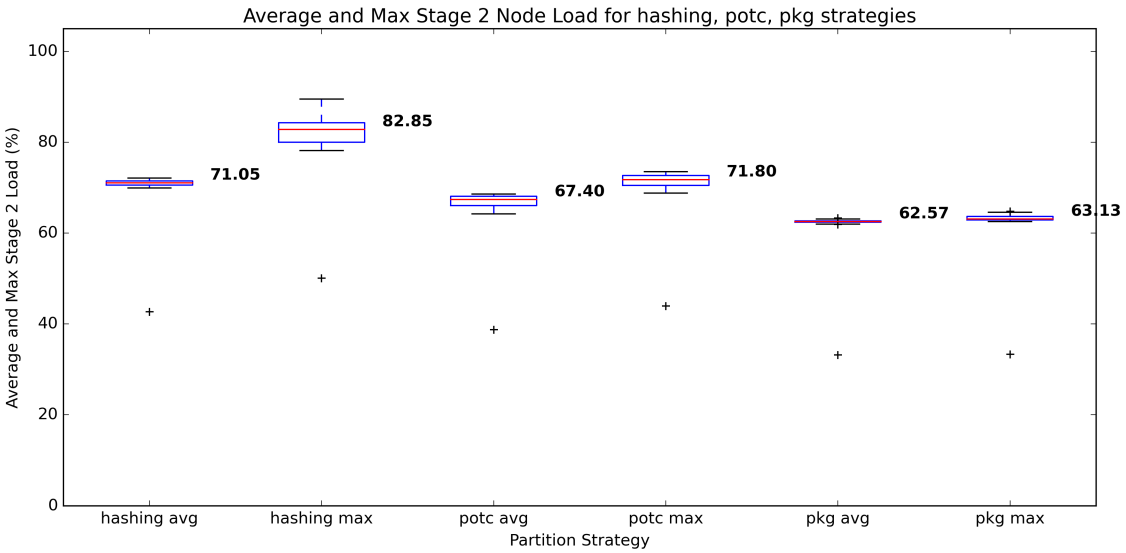
Παρατηρούμε αντίστοιχα αποτελέσματα με το ίδιο πείραμα στην πρώτη τοπολογία. Πιο συγκεκριμένα, η Partial Key Grouping (PKG) έχει καλύτερο ισομοιρασμό φόρτου, με την Power of Two Choices (PoTC) και το hashing δεύτερο και τρίτο καλύτερο ισομοιρασμό αντίστοιχα.

Ενδιαφέρον, εμφανίζει η επίδοση του κόμβου συνάθροισης στην στρατηγική PKG στο πρώτο στάδιο. Όπως φαίνεται από το σχήμα 6.31 ο Aggregator βρίσκεται ήδη από την κατάσταση κανονικής λειτουργίας σε κατάσταση μέγιστης χωρητικότητας. Υπενθυμίζουμε ότι στην δεύτερη τοπολογία το πρώτο στάδιο κάνει μία λειτουργία ταξινόμησης. Άρα, όλα οι εγγραφές που εισάγονται στο στάδιο 1 πρέπει μετά τον υπολογισμό να προωθηθούν στο στάδιο 2. Στην περίπτωση του PKG αυτές οι εγγραφές πρέπει πρώτα να περάσουν από τον συναθροιστή ώστε να υπολογιστεί η συνολική ταξινόμηση για το κάθε κλειδί. Εμφανές είναι ότι η διαδικασία συνάθροισης είναι αρκετά πιο πολύπλοκη στην τοπολογία 2 έναντι της αντίστοιχης συνάθροισης της τοπολογίας 1. Αυτό έχει ως αποτέλεσμα, ο κόμβος να υπερφορτωθεί. Αυτή η υπερφόρτωση φαίνεται καλύτερα στα σχήματα 6.33 και 6.34.

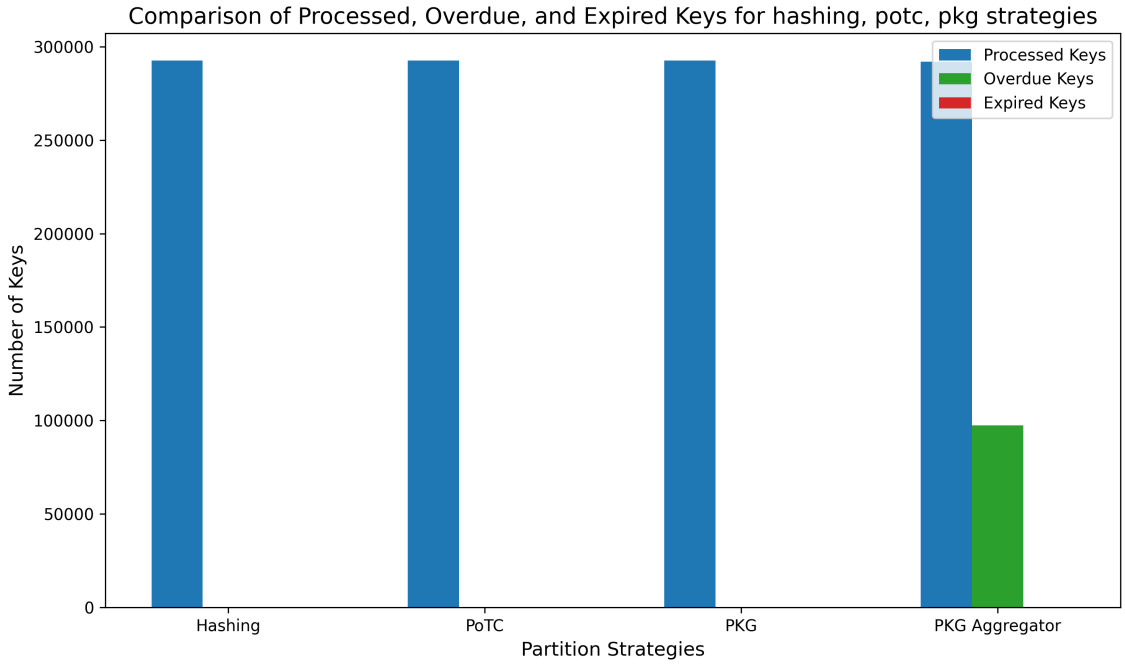
Στο στάδιο 1 βλέπουμε ότι ο κόμβος συνάθροισης παρουσιάζει εκπρόθεσμα κλειδιά τα οποία μεταφράζονται σε λιγότερα επεξεργασμένα κλειδιά στο δεύτερο στάδιο. Μπορούμε να αποφανθούμε ότι παρά την καλύτερη διαμέριση κλειδιών εσωτερικά του σταδίου που προσφέρει η PKG στρατηγική η στενωπός είναι ο κόμβος συνάθροισης και μεγάλης σημασίας είναι ο τύπος της συνάρτησης συνάθροισης.



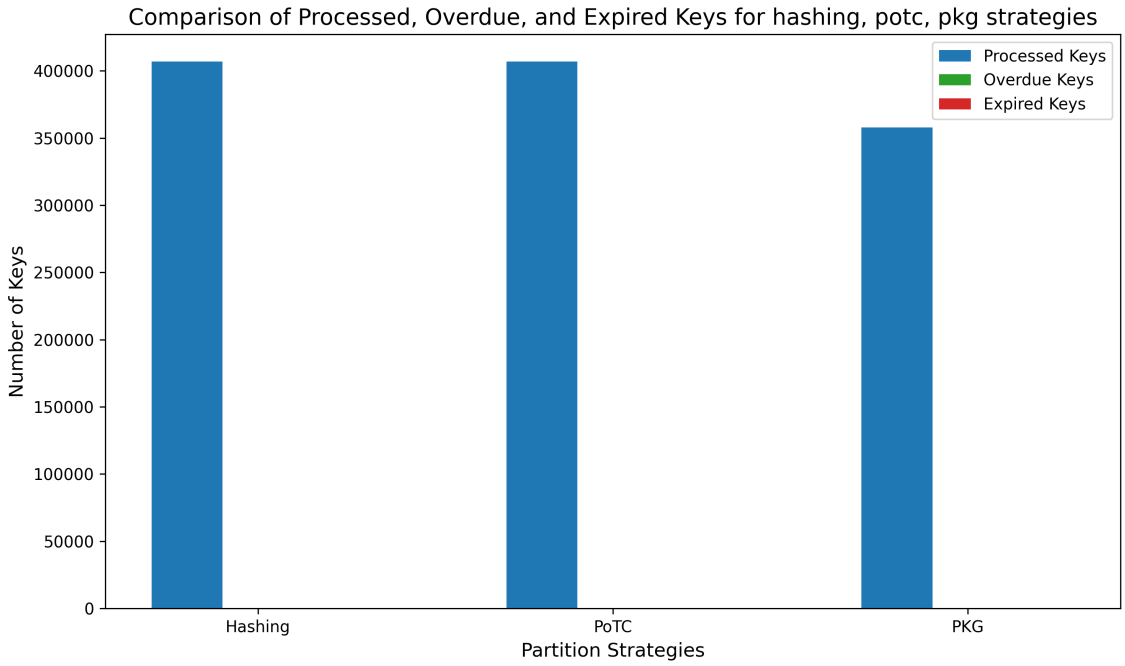

Σχήμα 6.31: *Κανονική Λειτουργία - Μέσος & Μέγιστος Φόρτος Σταδίου 1*




Σχήμα 6.32: *Κανονική Λειτουργία - Μέσος & Μέγιστος Φόρτος Σταδίου 2*



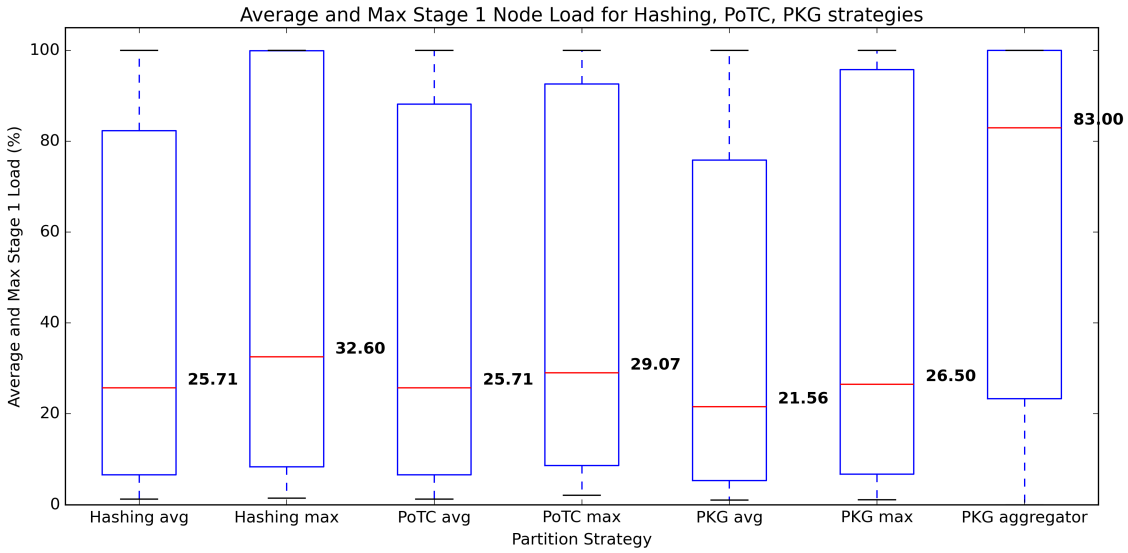

Σχήμα 6.33: *Κανονική Λειτουργία - Μετρικές Επεξεργασίας Κλειδιών ανά Στρατηγική Διαμέρισης για το Στάδιο 1*



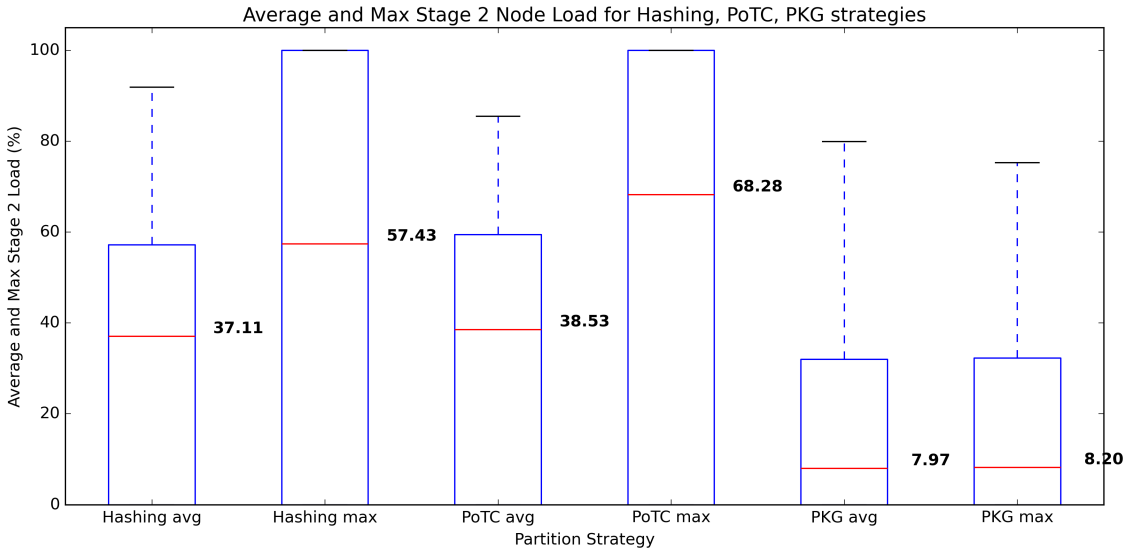

Σχήμα 6.34: *Κανονική Λειτουργία - Μετρικές Επεξεργασίας Κλειδιών ανά Στρατηγική Διαμέρισης για το Στάδιο 2*

### Αυξανόμενος Ρυθμός Άφιξης Κλειδιών

Στα σχήματα 6.35 και 6.36 φαίνεται ο μέσος φόρτος του πρώτου και δεύτερου σταδίου της τοπολογίας καθώς και ο μέσος μέγιστος φόρτος κόμβου (peak load) σε κάθε στάδιο.




Σχήμα 6.35: *Αυξανόμενος Ρυθμός Άφιξης - Μέσος & Μέγιστος Φόρτος Σταδίου 1*

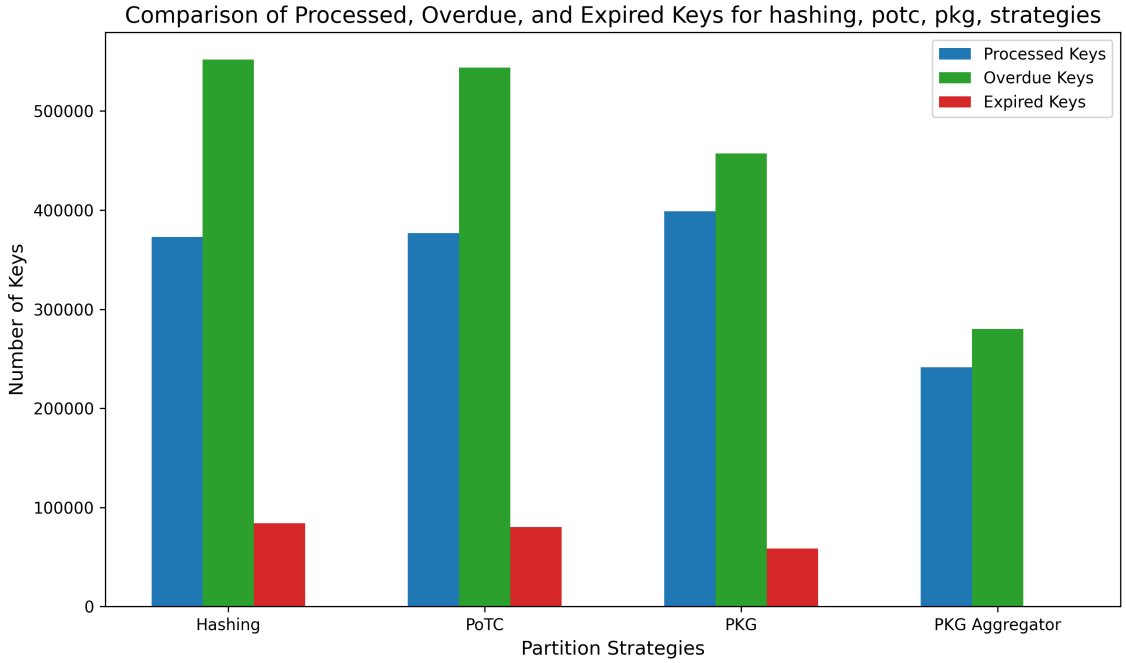



Σχήμα 6.36: *Αυξανόμενος Ρυθμός Άφιξης - Μέσος & Μέγιστος Φόρτος Σταδίου 2*

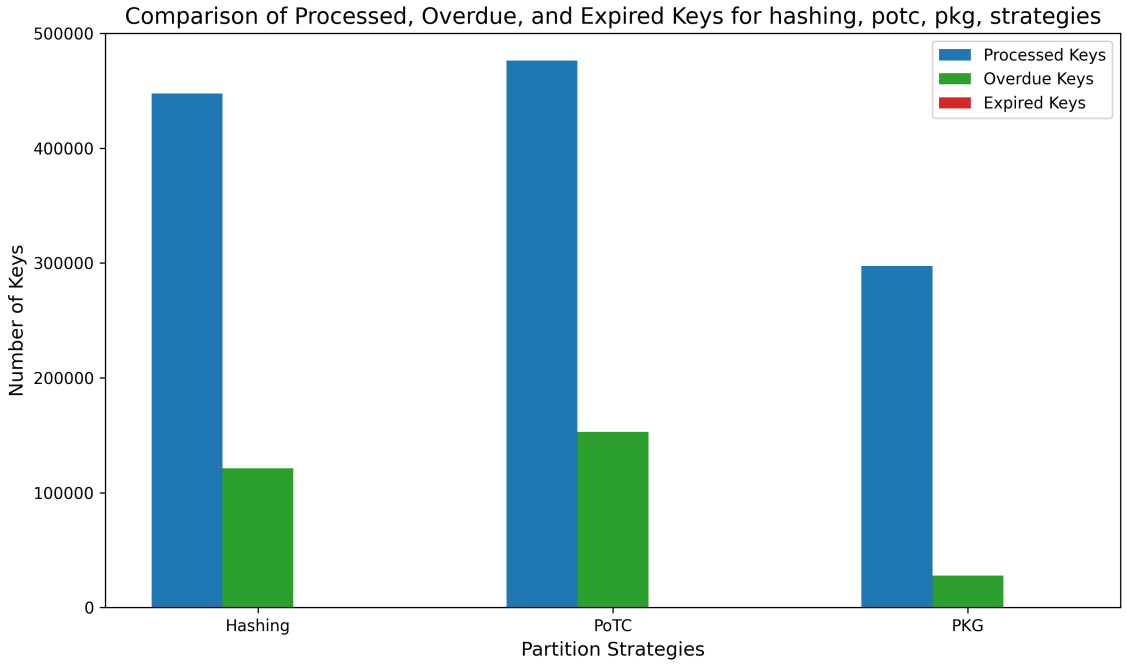
Παρατηρούμε ότι και οι τρεις στρατηγικές έχουν παρόμοια συμπεριφορά στο σενάριο αυξανόμενου ρυθμού άφιξης στο πρώτο στάδιο επεξεργασίας. Μία ελαφρώς καλύτερη στρατηγική εκ των τριών φαίνεται να είναι η Power of Two Choices (PoTC) η οποία φαίνεται να έχει ελαφρώς πιο ισοζυγισμένο φόρτο μεταξύ των κόμβων και μικρότερο peak load. Στην περίπτωση της στρατηγικής Partial Key Grouping (PKG) ο κόμβος συνάθροισης (Aggregator) αποτελεί το κύριο bottleneck του πρώτου σταδίου με πάνω απο 80% φόρτο καθ'όλη την διάρκεια της προσομοίωσης.

Στο δεύτερο στάδιο παρατηρούμε μία εντελώς διαφορετική εικόνα. Το hashing και το

PoTC έχουν παρόμοιο φόρτο με το PoTC να έχει κατά μέσο όρο 20% παραπάνω μέγιστο φόρτο έναντι της hashing στρατηγικής. Αντίθετα, το στάδιο με στρατηγική PKG έχει κατά πολύ λιγότερο φόρτο σε σχέση με τις υπόλοιπες στρατηγικές. Για καλύτερη κατανόηση των αποτελεσμάτων παραθέτουμε και τα συνολικά κλειδιά του κάθε σταδίου στα σχήματα 6.37 και 6.38.




Σχήμα 6.37: *Αυξανόμενος Ρυθμός Άφιξης - Μετρικές Επεξεργασίας Κλειδιών ανά Στρατηγική Διαμέρισης για το Στάδιο 1*



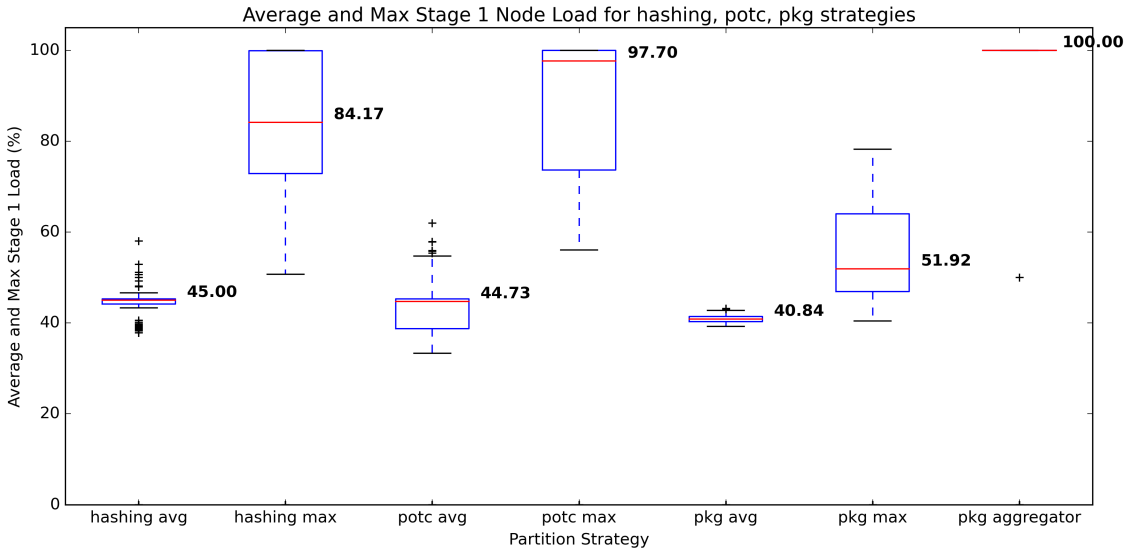

Σχήμα 6.38: *Αυξανόμενος Ρυθμός Άφιξης - Μετρικές Επεξεργασίας Κλειδιών ανά Στρατηγική Διαμέρισης για το Στάδιο 2*

Αρχικά, όπως αναφέραμε και προηγουμένως στην στρατηγική PKG η στενωπός είναι ο κόμβος συνάθροισης. Εμφανώς επεξεργάζεται πολύ λιγότερα από τα κλειδιά που του καταφθάνουν από τους κόμβους επεξεργασίας και χρειάζονται aggregation. Η κακή απόδοση του συγκεκριμένου κόμβου είναι αυτή που οδηγεί στην υποχρησιμοποιήση των κόμβων του επόμενου σταδίου και τον γενικό μικρό φόρτο επεξεργασίας του δεύτερου σταδίου στην PKG στρατηγική. Συγκεκριμένα, μπορεί να επιβεβαιωθεί στο σχήμα 6.38 όπου η στρατηγική PKG καταλήγει να έχει τα λιγότερα επεξεργασμένα κλειδιά εκ των τριών στρατηγικών στο δεύτερο στάδιο.

Συμπερασματικά, η Power of Two Choices στρατηγική επιφέρει μία βελτίωση σε σχέση με την κλασική στρατηγική κατακερματισμού και αποτυπώνεται μέσω των περισσότερων επεξεργασμένων κλειδιών και των λιγότερων expired κλειδιών και στα δύο στάδια. Αντίθετα, η Partial Key Grouping πάσχει από κακή απόδοση του Aggregator κόμβου. Σημαντικός παράγοντας είναι και ότι η προσομοιούμενη λειτουργία στην δεύτερη τοπολογία είναι μία που απαιτεί μεγαλύτερη επεξεργασία και συνάθροιση σε σχέση με την προσομοιούμενη λειτουργία της πρώτης τοπολογίας. Ο κόμβος συνάθροισης σε αυτή την τοπολογία χρειάζεται να συναθροίσει όλα τα κλειδιά του προηγούμενου σταδίου οδηγώντας τον σε κατάσταση υπερφόρτωσης.

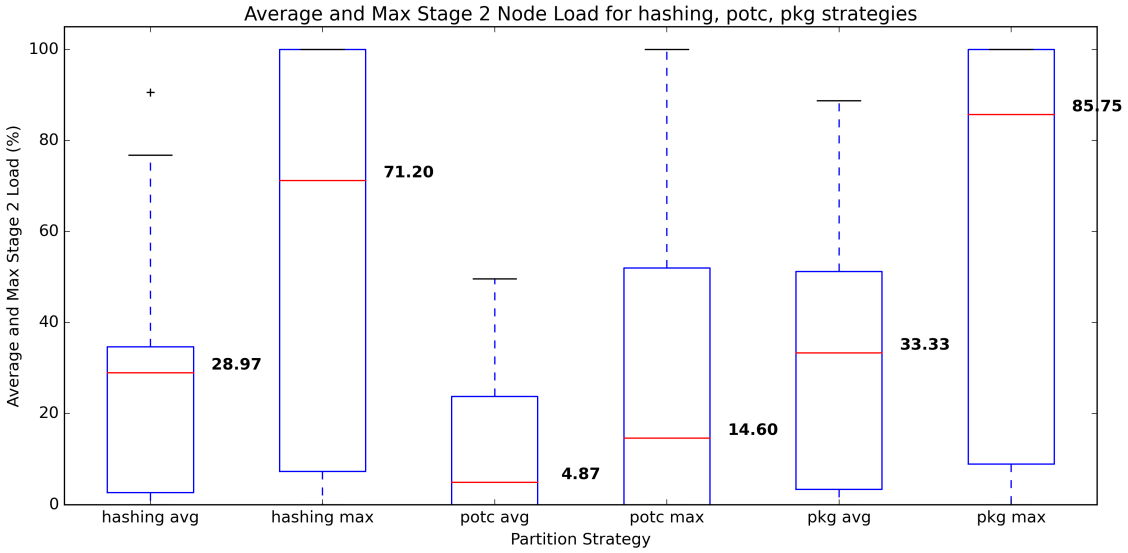
### Ανισοκατανομή Κλειδιών

Σε αυτή την ενότητα παρουσιάζονται οι επιδόσεις του συστημάτος σε δεδομένα εισόδου που παρουσιάζουν ανισοκατανομή. Στα σχήματα 6.39 και 6.40 βλέπουμε τον μέσο και τον μέγιστο φόρτο του σταδίου 1 και 2 της τοπολογίας αντίστοιχα.



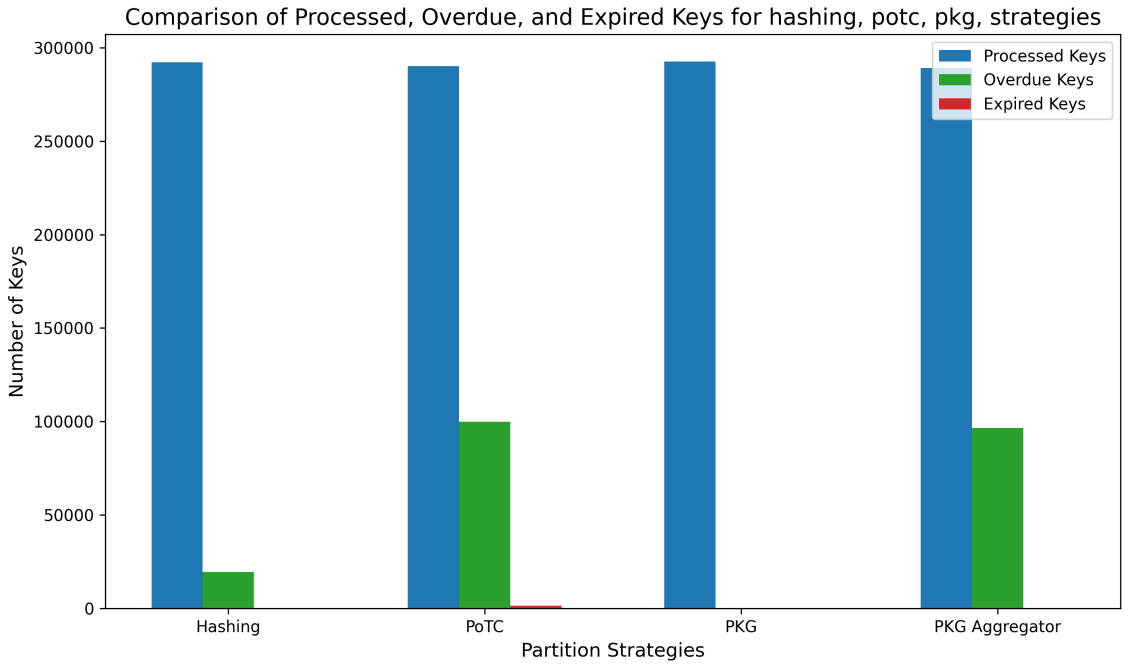

Σχήμα 6.39: *Ανισοκατανομή - Μέσος & Μέγιστος Φόρτος Σταδίου 1*

Στο στάδιο 1 φαίνεται ότι το μέσο load είναι παρόμοιο και στις 3 στρατηγικές, hashing, Power of Two Choices (PoTC) και Partial Key Grouping (PKG). Βέβαια παρατηρούμε ότι στην περίπτωση του PKG ο μέσος φόρτος εμφανίζει σημαντικά λιγότερη διασπορά έναντι των άλλων δύο στρατηγικών. Στον μέγιστο φόρτο τώρα του πρώτου σταδίου παρατηρούμε ότι η επίδοση του PKG είναι σημαντικά καλύτερη σε σχέση με τις άλλες δύο στρατηγικές (51.92%




Σχήμα 6.40: *Ανισοκατανομή - Μέσος & Μέγιστος Φόρτος Σταδίου 2*

έναντι 84.17% και 97.70% για hashing και PoTC αντίστοιχα). Αναφορικά με τον κόμβο συνάθροισης επαναλαμβάνεται το μοτίβο που συναντήσαμε και στο πείραμα αυξανόμενου ρυθμού αύξηση άφιξης, έχοντας δηλαδή μέγιστο φόρτο και αποτελώντας το bottleneck της στρατηγικής αυτής. Στο σχήμα 6.41 επιβεβαιώνονται καλύτερα όσο είδαμε από τα διαγράμματα φόρτου για το πρώτο στάδιο. Η PKG στρατηγική είναι η καλύτερη και επεξεργάζεται όλα τα κλειδιά χωρίς να εισάγει περαιτέρω καθυστέρηση αλλά πάσχει από έναν αργό συναθροιστή. Η PoTC είναι σαφώς χειρότερη από τις άλλες δύο επιλογές.

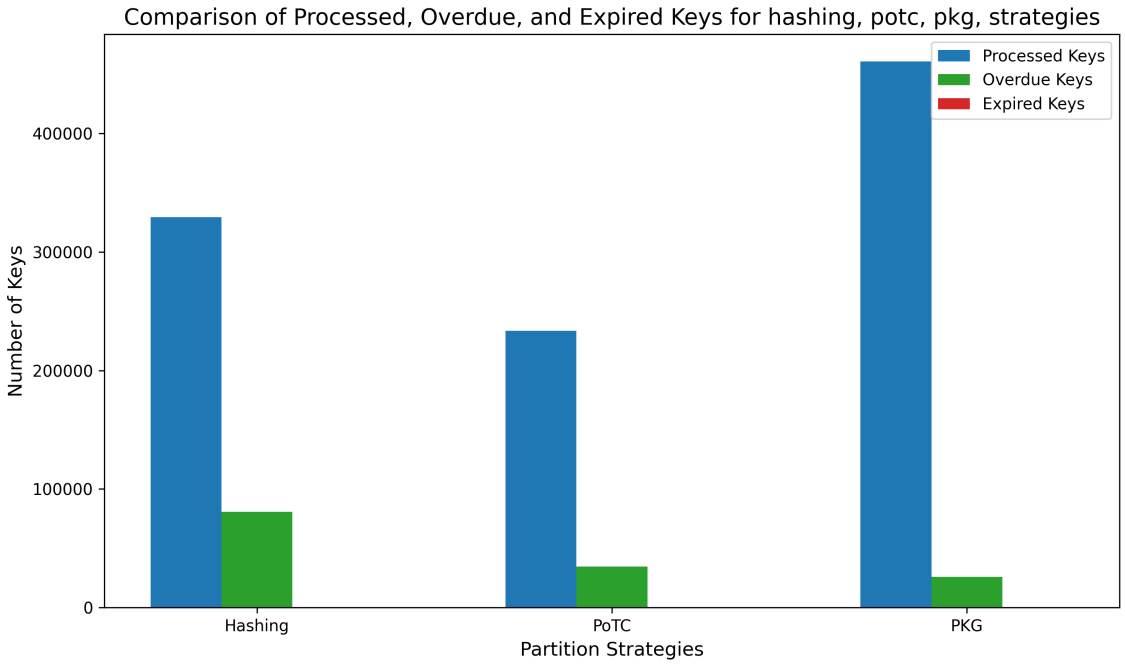



Σχήμα 6.41: *Ανισοκατανομή - Μετρικές Επεξεργασίας Κλειδιών ανά Στρατηγική Διαμέρισης για το Στάδιο 1*

Ενδιαφέρον παρουσιάζει ότι η PoTC συμπεριφέρεται χειρότερα έναντι του hashing και

έρχεται σε αντιπαραβολή με το ίδιο πείραμα στην πρώτη τοπολογία. Η χειρότερη απόδοση αποδίδεται στην έλλειψη προσαρμοστικότητας και δυναμικότητας της PoTC στρατηγικής. Η επιλογή μεταξύ των πιθανών κόμβων γίνεται μόνο μία φορά χωρίς να λαμβάνει υπόψη τις μετέπειτα αλλαγές στον φόρτο. Βέβαια, και το hashing δεν παρουσιάζει κάποια δυναμικότητα αλλά ο αυθαίρετος διαχωρισμός που επιλέχθει αποδείχθηκε καταλληλότερος.

Στα στάδιο 2 (σχήμα 6.40) βλέπουμε τον φόρτο του δεύτερου σταδίου της τοπολογίας. Ο φόρτος είναι μεγαλύτερος στην PKG με ακολοθούμενες το hashing και PoTC. Παρατηρώντας και το σχήμα 6.42 είναι εμφανές ότι η PKG στρατηγική είναι αυτή που επεξεργάζεται μεγαλύτερο αριθμό κλειδιών σε σχέση με τις υπόλοιπες. Ο αυξημένος φόρτος λοιπόν οφείλεται στην καλύτερη απόδοση του πρώτου σταδίου που καταφέρνει και επεξεργάζεται μεγαλύτερο αριθμό κλειδιών τα οποία προωθούνται στο δεύτερο στάδιο.

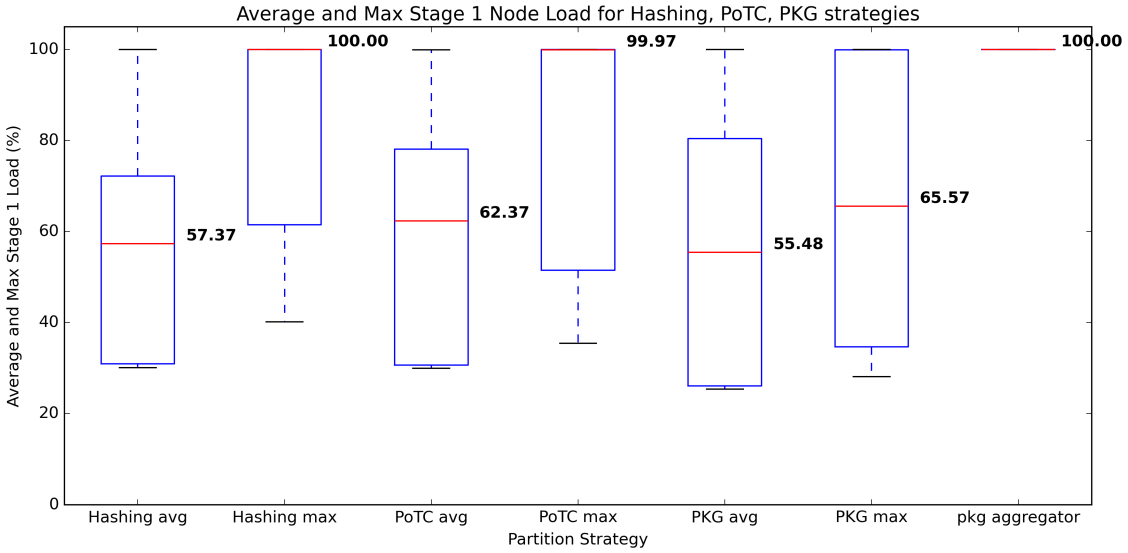



Σχήμα 6.42: *Ανισοκατανομή - Μετρικές Επεξεργασίας Κλειδιών ανά Στρατηγική Διαμέρισης για το Στάδιο 2*

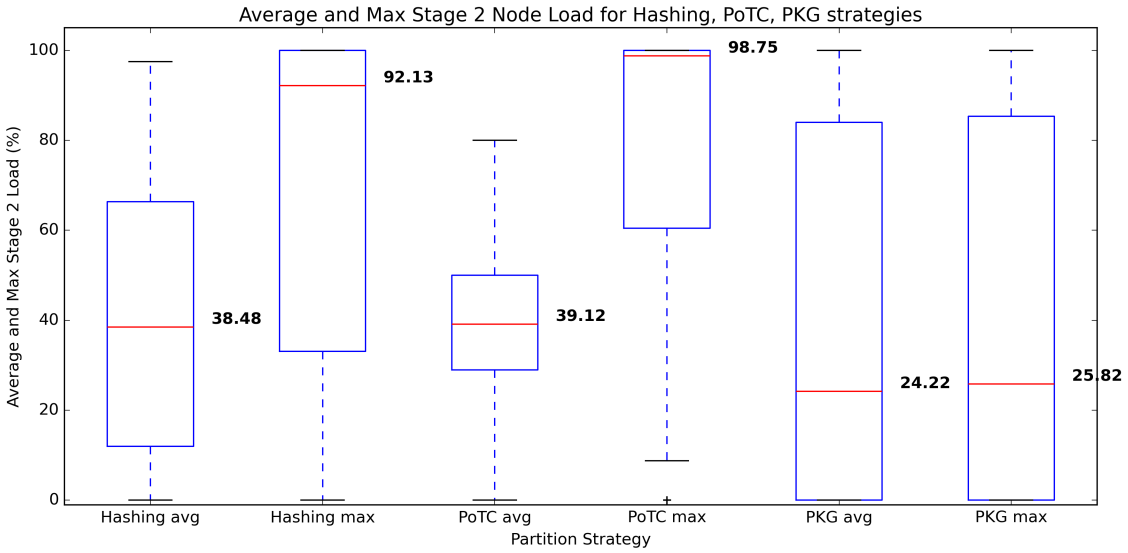
#### Μεταβαλλόμενο Spike

Τέλος, εξετάσαμε τις διαφορετικές στρατηγικές διαμέρισης στην τοπολογία με ύπαρξη αιχμών (spikes) στα δεδομένα. Στα σχήματα 6.43 και 6.44 φαίνεται το load του κάθε σταδίου της τοπολογίας.

Παρατηρούμε ότι στον μέσο φόρτο και οι τρεις στρατηγικές έχουν παρόμοια επίδοση με την PoTC να είναι ελαφρώς χειρότερη. Στον μέγιστο φόρτο σταδίο (peak load) τα αποτελέσματα είναι εντελώς διαφορετικά. Η PKG στρατηγική δεν βρίσκεται τόσο συχνά σε μέγιστη χωρητικότητα εν αντιθέσει με τις άλλες δύο στρατηγικές που τουλάχιστον στο 50% των βημάτων βρίσκονται σε 100% φόρτο. Αποδίδουμε αυτή την χειρότερη επίδοση στην έλλειψη δυναμικής απόφασης των πρώτων δύο στρατηγικών σε κάθε διαμέριση. Υπενθυμίζουμε ότι η στρατηγική PKG επιλέγει σε κάθε λειτουργία διαμέρισης τους κόμβους με τον λιγότερο



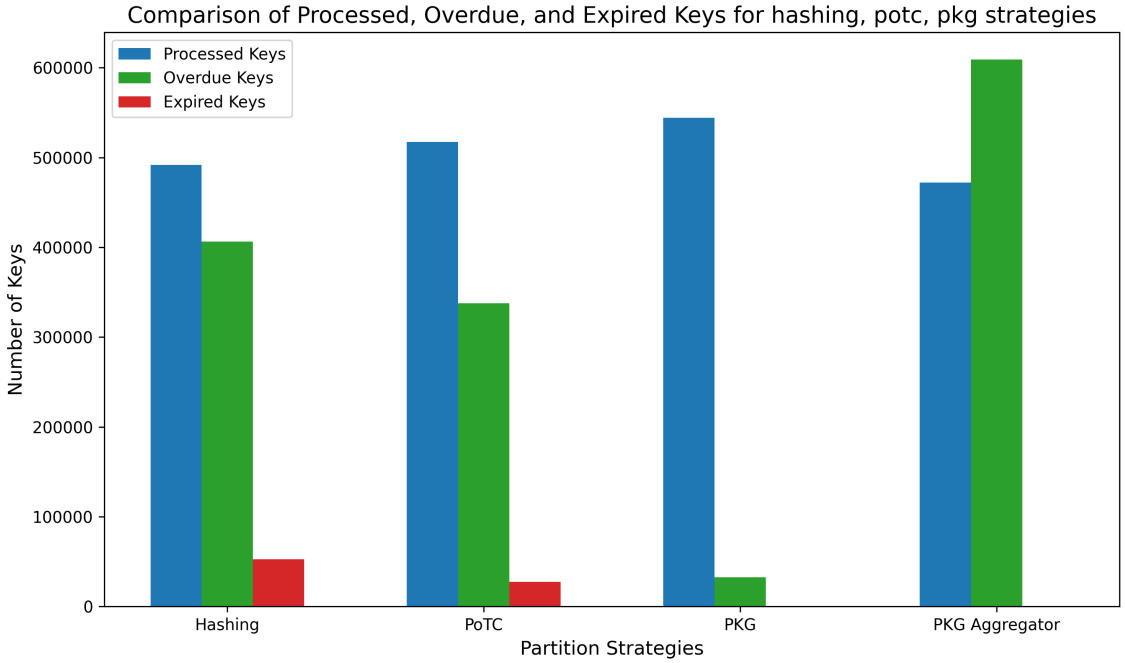

Σχήμα 6.43: *Μεταβαλλόμενο Spike - Μέσος & Μέγιστος Φόρτος Σταδίου 1*



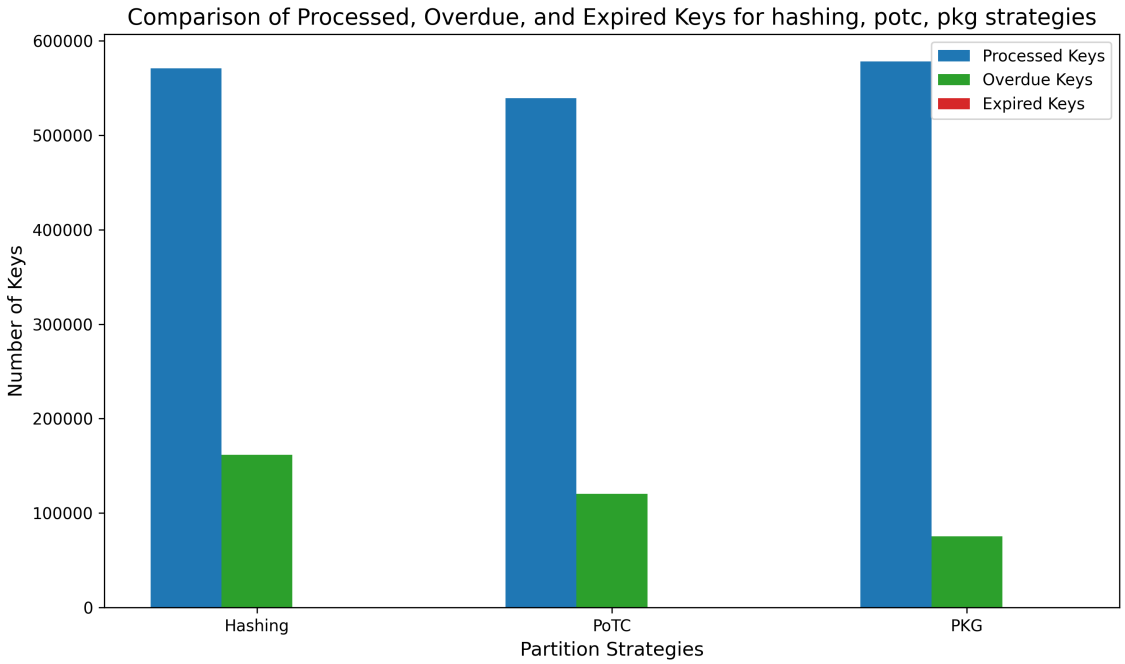

Σχήμα 6.44: *Μεταβαλλόμενο Spike - Μέσος & Μέγιστος Φόρτος Σταδίου 2*

φόρτο έχοντας και την δυνατότητα διάσπασης εγγραφών ίδιου κλειδιού σε περισσότερους κόμβους. Αντίστοιχα όμως με τα υπόλοιπα πειράματα αυτής της τοπολογίας παρατηρούμε υπεφόρτωση του κόμβου συνάθροισης για τους ίδιους λόγους που έχουμε αναφέρει κατά την διάρκεια των πειραμάτων στην τοπολογία 2.

Στα σχήματα 6.45 και 6.46 μπορούμε να δούμε αναλυτικά τα στατιστικά των κλειδιών. Στο στάδιο 1 παρατηρούμε ότι η PKG στρατηγική καταφέρνει και αντιμετωπίζει επιτυχώς τις καταστάσεις αιχμής με λιγοστά εκπρόθεσμα κλειδιά. Αντίθετα, ο κατακερματισμός και η PoTC παρουσιάζουν πολλά περισσότερα καθυστερημένα κλειδιά και ακόμα και ληγμένα κλειδιά. Σημειώνουμε, ότι η PoTC στρατηγική εμφανίζει λίγο καλύτερη απόδοση από το hashing παρά το μεγαλύτερο φόρτο που παρατηρήσαμε στο στάδιο 1 στην συγκεκριμένη στρατηγική. Επιπλέον, επιβεβαιώνονται και οι παρατηρήσεις για τον κόμβο συνάθροισης στην PKG όπου πολλά κλειδιά υπολογίζονται καθυστερημένα.

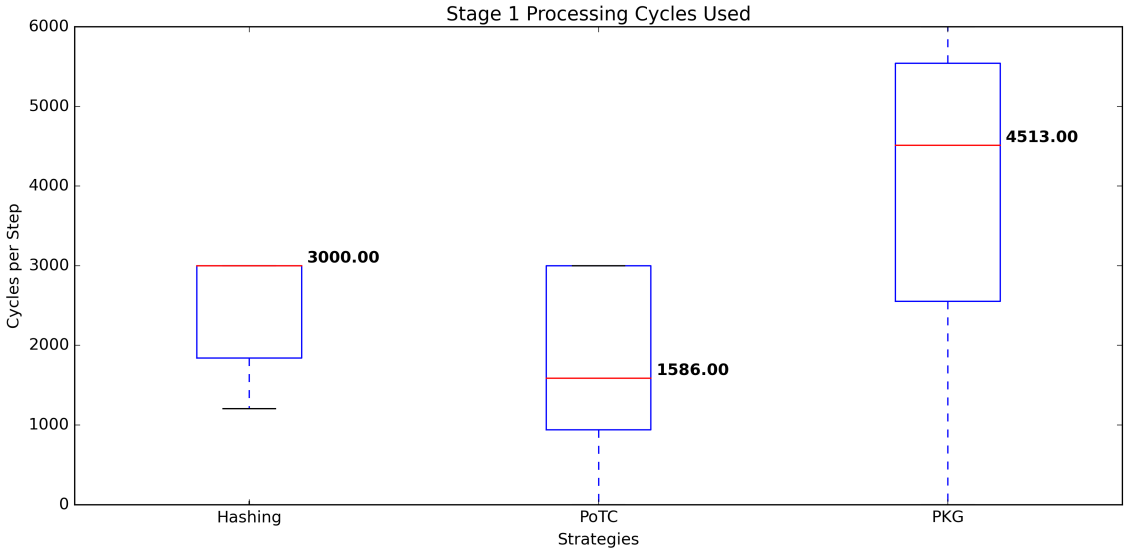



Σχήμα 6.45: *Μεταβαλλόμενο Spike - Μετρικές Επεξεργασίας Κλειδιών ανά Στρατηγική Διαμέρισης για το Στάδιο 1*




Σχήμα 6.46: *Μεταβαλλόμενο Spike - Μετρικές Επεξεργασίας Κλειδιών ανά Στρατηγική Διαμέρισης για το Στάδιο 2*

Στο στάδιο 2 το ίδιο μοτίβο συνεχίζεται, η PKG αποδίδει καλύτερα με τις άλλες δύο στρατηγικές να την ακολουθούν με λιγότερα επεξεργασμένα κλειδιά και περισσότερα κλειδιά που επεξεργάζονται αργοπορημένα. Για να ποσοτικοποίησουμε καλύτερα την επίδοση της PKG παραθέτουμε και το διάγραμμα με τους απαιτούμενους κύκλους επεξεργασίας στο πρώτο στάδιο για κάθε βήμα της προσομοίωσης.




Σχήμα 6.47: *Μεταβαλλόμενο Spike - Μέσος Αριθμός Κύκλων Επεξεργασίας για το 1ο Στάδιο*

Το διεπίπεδο στάδιο της στρατηγικής PKG έχει σημαντική επίδραση στον αριθμό κύκλων επεξεργασίας. Κατά μέσο όρο είναι 3 φορές περισσότεροι κύκλοι επεξεργασίας έναντι της PoTC που είναι η στρατηγική με τους λιγότερους κύκλους επεξεργασίας. Οι παραπάνω κύκλοι επεξεργασίας οδηγούν σε αύξηση του latency στην επεξεργασία των κλειδιών. Σημειώνουμε ότι εδώ είναι στην κρίση του αρχιτέκτονα ή του χρήστη η απάντηση στο αντίστοιχο δίλημμα δηλαδή στην απόφαση μεταξύ περισσότερων επεξεργασμένων κλειδιών έναντι της αύξησης του χρόνου επεξεργασίας ή λιγότερα επεξεργασμένα κλειδιά με μικρότερο χρόνο επεξεργασίας.

### 6.4.2 Κλιμακωσιμότητα

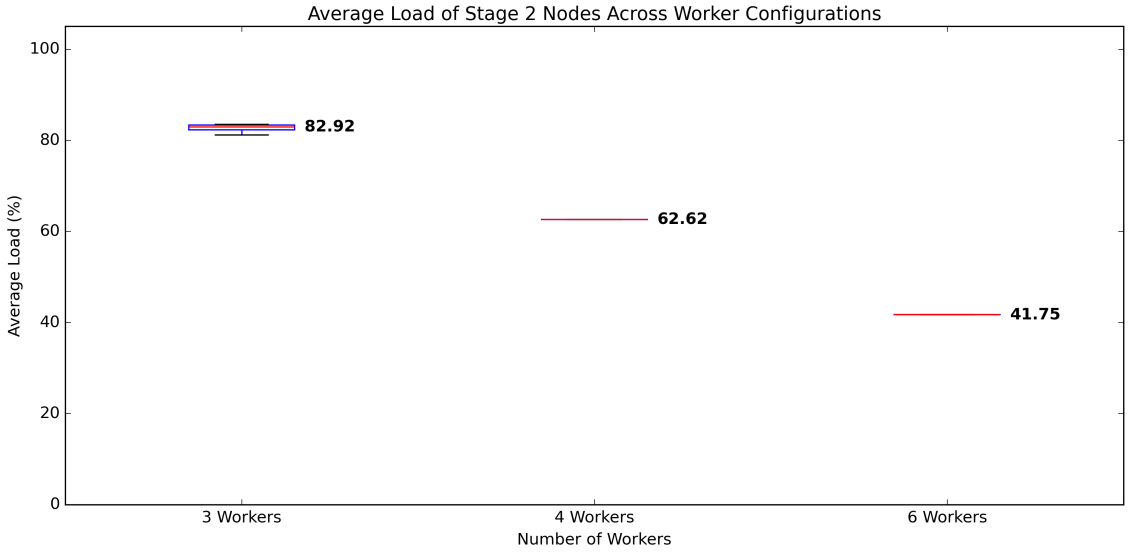
Σε αυτή τη σειρά πειραμάτων, επικεντρωνόμαστε στο στάδιο 2 της τοπολογίας, που όπως είδαμε νωρίτερα, αποτελεί στενωπό (bottleneck) του συστήματος. Για το σκοπό αυτό, επαναλαμβάνουμε τα πειράματα για 3, 4 και 6 κόμβους.

#### Κανονική Λειτουργία

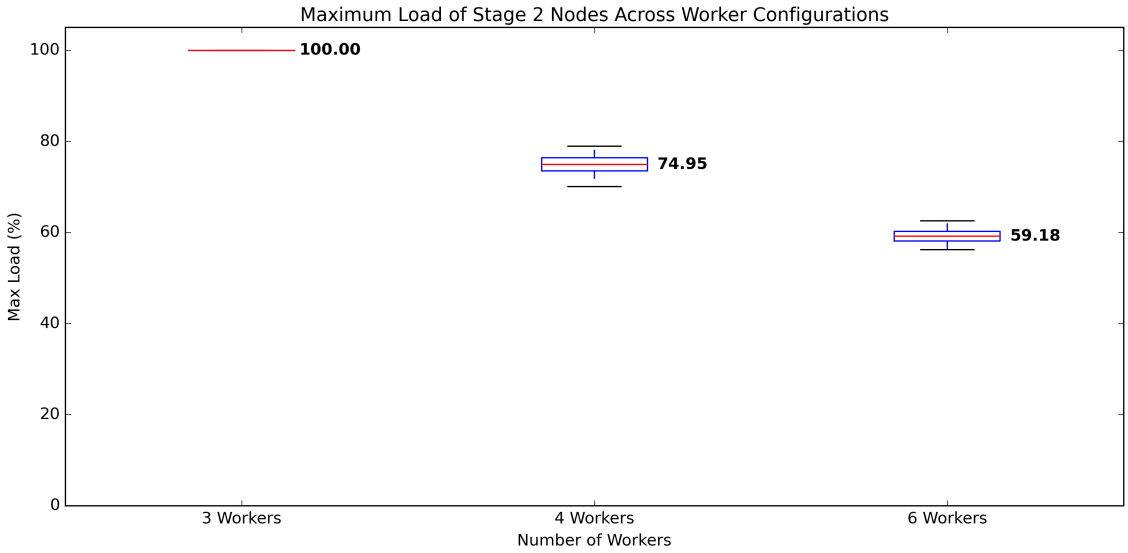
Στα σχήματα 6.48 και 6.49 παρουσιάζονται ο μέσος και ο μέγιστος φόρτος όλων των κόμβων του σταδίου 2 συνολικά για την τοπολογία, κατά τη διάρκεια προσωμοίωσης με 3, 4 και 6 κόμβους υπό συνθήκες κανονικής λειτουργίας.

Και στα δύο διαγράμματα, οι μέσες και μέγιστες τιμές του φόρτου στο στάδιο 2, σε κάθε βήμα, βρίσκονται σχεδόν πάντα κοντά στις αντίστοιχες τιμές της διαμέσου. Όσον αφορά τις μέσες τιμές, παρατηρείται σχεδόν άριστη κλιμάκωση: η διάμεσος αρχικά βρίσκεται στο 82.92% με 3 κόμβους και μειώνεται στο 62.62% με 4 κόμβους (μείωση περίπου 25%) και στο 41.75% με 6 κόμβους (μείωση 50% σε σχέση με την αρχική). Αυτό δείχνει ότι η προσθήκη κόμβων οδηγεί σε σχεδόν γραμμική βελτίωση.

Παρόμοια τάση παρατηρείται και στις μέγιστες τιμές, όπου η διάμεσος μειώνεται από το 100% στο 74.95% και στο 59.18%. Η μείωση δεν είναι απόλυτα γραμμική και αυτό




Σχήμα 6.48: *Κανονική Λειτουγία - Μέσος Φόρτος Κόμβων Σταδίου 2 Συνολικά για 3,4 και 6 Κόμβους.*



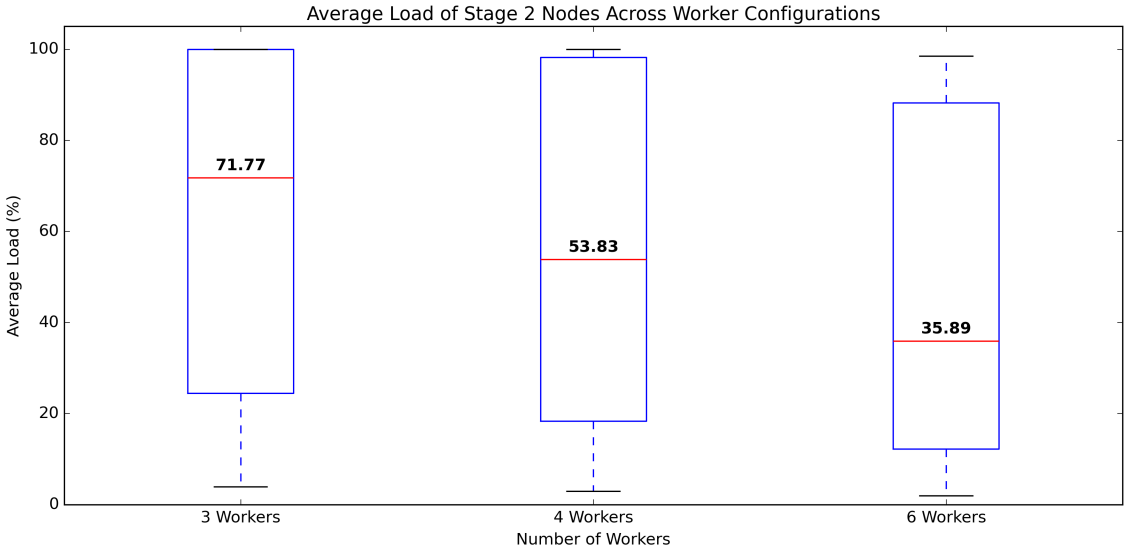

Σχήμα 6.49: *Κανονική Λειτουγία - Μέγιστος Φόρτος Κόμβων Σταδίου 2 Συνολικά για 3,4 και 6 Κόμβους.*

οφείλεται στο γεγονός ότι η μέγιστη τιμή, ως δείκτης, δεν είναι πάντα τόσο αντιπροσωπευτική για την αποτύπωση μιας τέλειας γραμμικότητας.

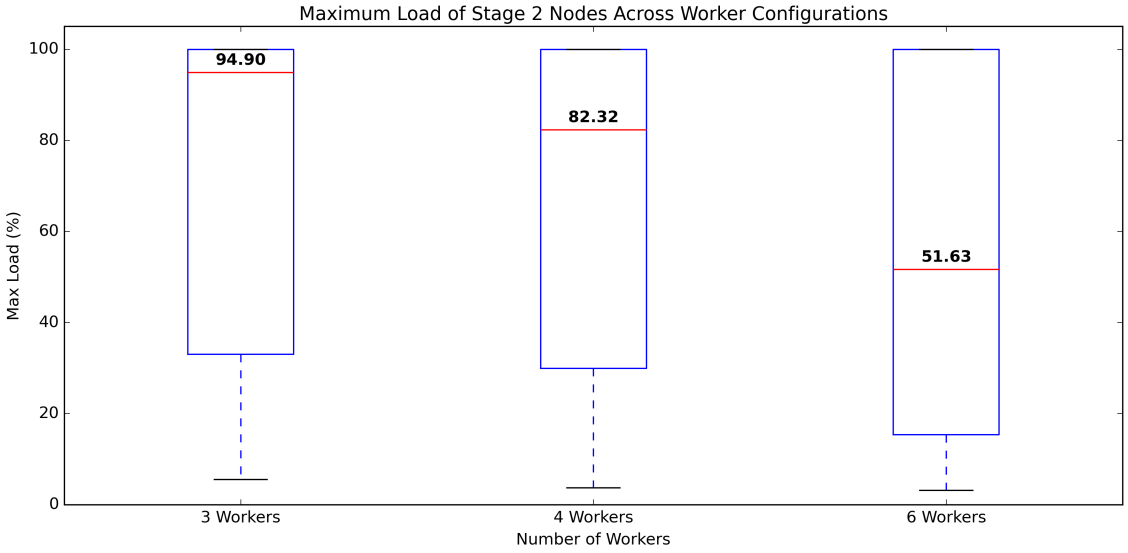
Αυτά τα αποτελέσματα είναι αναμενόμενα, καθώς έχουμε επιλέξει τη στρατηγική κατανομής κλειδιών hashing, και ο μεγάλος αριθμός βημάτων και διαφορετικών κλειδιών, σε συνδυασμό με την ομοιόμορφη κατανομή τους, έχει ως αποτέλεσμα ο κατανεμητής να κατανέμει τα κλειδιά σχεδόν τέλεια ισομερώς.

### Αυξανόμενος Ρυθμός Άφιξης Κλειδιών

Στα σχήματα 6.50 και 6.51 φαίνεται ο μέσος και ο μέγιστος, αντίστοιχα, φόρτος όλων των κόμβων του σταδίου 2 συνολικά, για τις διαφορετικές παραμετροποιήσεις.




Σχήμα 6.50: *Αυξανόμενος Ρυθμός Άφιξης - Μέσος Φόρτος Κόμβων Σταδίου 2 Συνολικά για 3,4 και 6 Κόμβους.*



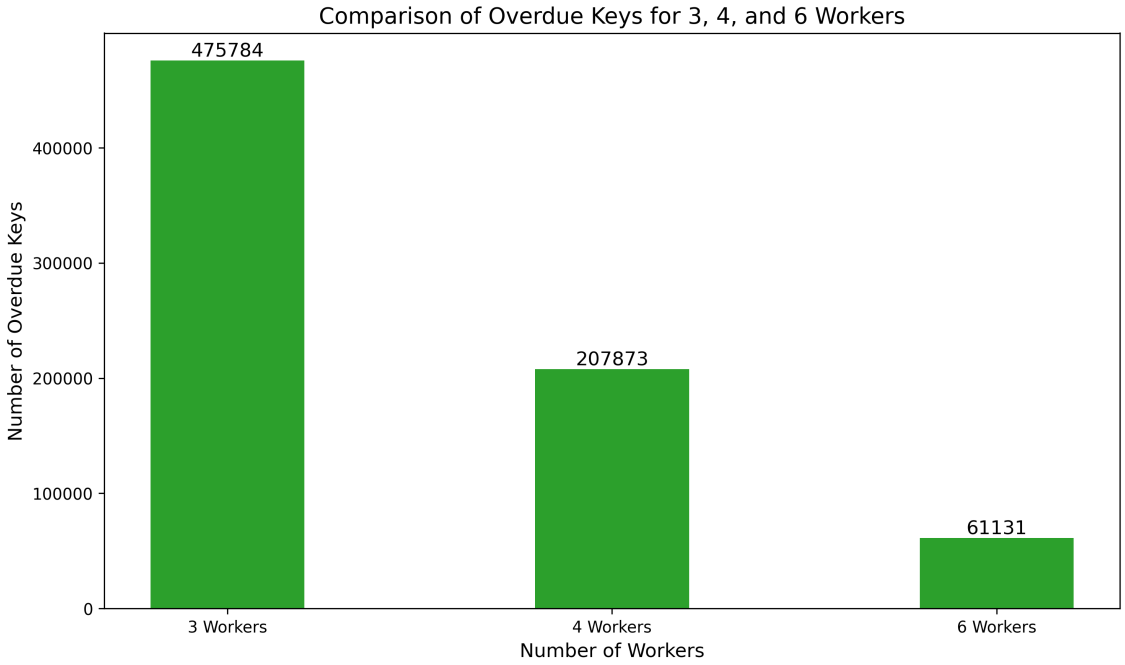

Σχήμα 6.51: *Αυξανόμενος Ρυθμός Άφιξης - Μέγιστος Φόρτος Κόμβων Σταδίου 2 Συνολικά για 3,4 και 6 Κόμβους.*

Στην τοπολογία με 3 κόμβους, ο μέσος φόρτος του σταδίου παίρνει τιμές από 25% έως και 100% με τη διάμεσο να βρίσκεται κοντά στο 72%. Με την προσθήκη ενός μόλις κόμβου, το εύρος βελτιώνεται σε πολύ μικρό βαθμό με τιμές 20% - 98%, αλλά η διάμεσος μειώνεται σημαντικά και πηγαίνει κοντά στο 54% . Με την προσθήκη δύο ακόμη κόμβων, το εύρος τιμών μειώνεται σε 15% - 87% και διάμεσο αρκετά χαμηλά, στο 35.89%.

Αντίστοιχες παρατηρήσεις κάνουμε και για τον μέγιστο φόρτο, όπου το εύρος μεταβάλλεται από 35% - 100%, σε 30% - 100% και 18% - 100%, και η διάμεσος από 94.9% , σε 82.32% και 51.63% για 3, 4 και 6 κόμβους αντίστοιχα. Ακόμη, βλέπουμε ότι σε κάθε περίπτωση υπάρχουν στιγμές που ο φόρτος φτάνει το 100%.

Οι παραπάνω παρατηρήσεις, αποτυπώνονται και στο Σχήμα 6.52, όπου όπως βλέπουμε,

το συνολικό πλήθος εκπρόθεσμων κλειδιών μειώνεται με την είσοδο επιπλέον κόμβων.




Σχήμα 6.52: *Αυξανόμενος Ρυθμός Άφιξης - Συνολικά Εκπρόθεσμα Κλειδιά στο Στάδιο 2 για 3, 4 και 6 Κόμβους.*

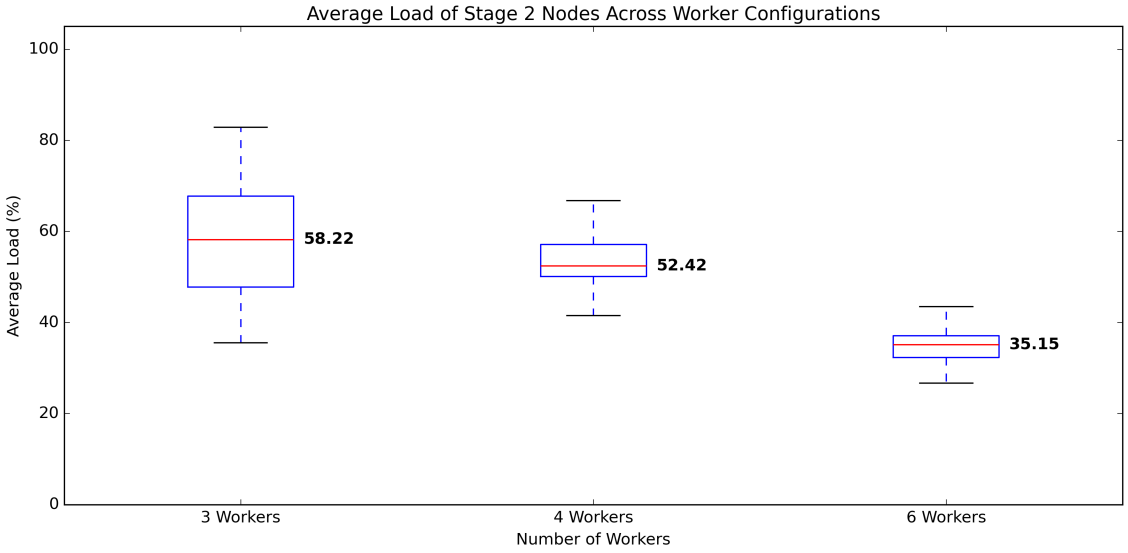
Συμπεραίνουμε λοιπόν, πως η συγκεκριμένη τοπολογία, κλιμακώνεται αρκετά καλά, σε συνθήκες αυξανόμενου ρυθμού άφιξης κλειδιών, με την εισαγωγή λίγων μόνο κόμβων.

**Ανισοκατανομή Κλειδιών**

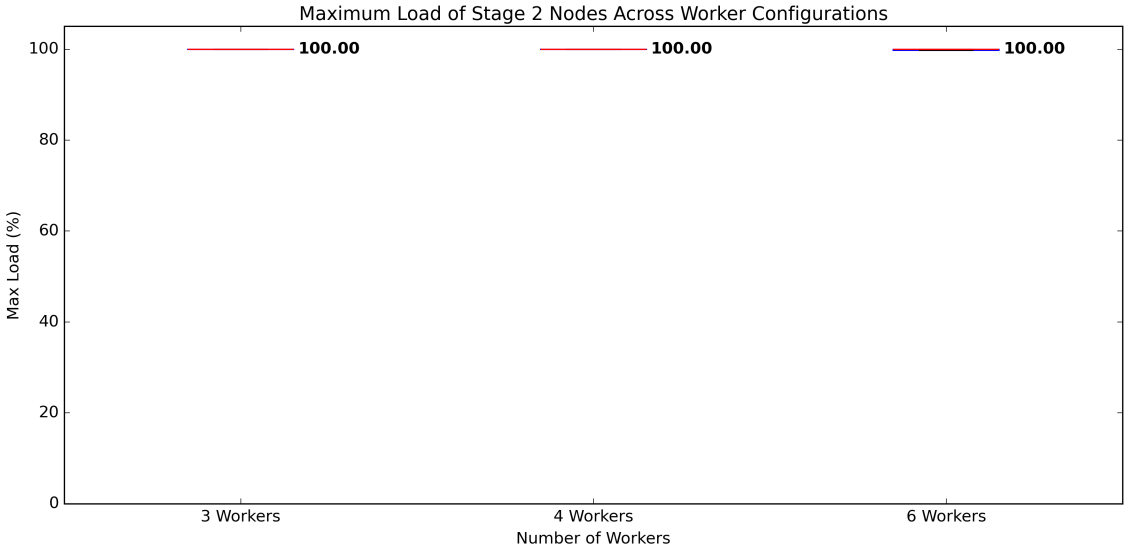
Στα σχήματα 6.53 και 6.54 φαίνονται τα ίδια διαγράμματα όταν το σύστημα βρίσκεται σε συνθήκες υψηλής ανισοκατανομής. Σύμφωνα με αυτά, παρατηρούμε ότι πάλι υπάρχει μία μικρή βελτίωση τόσο στο εύρος του μέσου φόρτου, όσο και στη διάμεσο , που από 58.22% μειώνεται σε 52.42% και 35.15%. Ωστόσο, αφενός οι βελτιώσεις αυτές είναι μικρότερου βαθμού σε σχέση με την προηγούμενη περίπτωση και αφετέρου, αν προσέξουμε το γράφημα του μέγιστου φόρτου, βλέπουμε ότι παντού είναι 100%, πράγμα που υποδηλώνει ότι συνεχώς τουλάχιστον ένας κόμβος είναι υπερφορτωμένος, ενώ άλλοι κόμβοι του σταδίου υπολειτουργούν.

Αυτό, αποτυπώνεται και στο Σχήμα 6.55 όπου βλέπουμε μία σημαντική πτώση, κατά 2.4 φορές, στα συνολικά εκπρόθεσμα κλειδιά, όταν μεταβαίνουμε από 3 σε 4 κόμβους, αλλά πολύ μικρή πτώση 19.5% όταν μεταβαίνουμε σε 6 κόμβους.

Συμπεραίνουμε, πως το σύστημα εξακολουθεί να κλιμακώνεται, αλλά σε μικρότερο βαθμό. Αυτό οφείλεται στη στρατηγική διαμερισμού κλειδιών που έχουμε επιλέξει, δηλαδή στη στρατηγική hashing, καθώς τα κλειδιά δεν μπορούν να κατανεμηθούν ομοιόμορφα στους κόμβους, υπερφορτώνοντας συνεχώς τουλάχιστον έναν και υποβαθμίζοντας τη συνολική απόδοση του συστήματος.




Σχήμα 6.53: *Ανισοκατανομή - Μέσος Φόρτος Κόμβων Σταδίου 2 Συνολικά για 3,4 και 6 Κόμβους.*

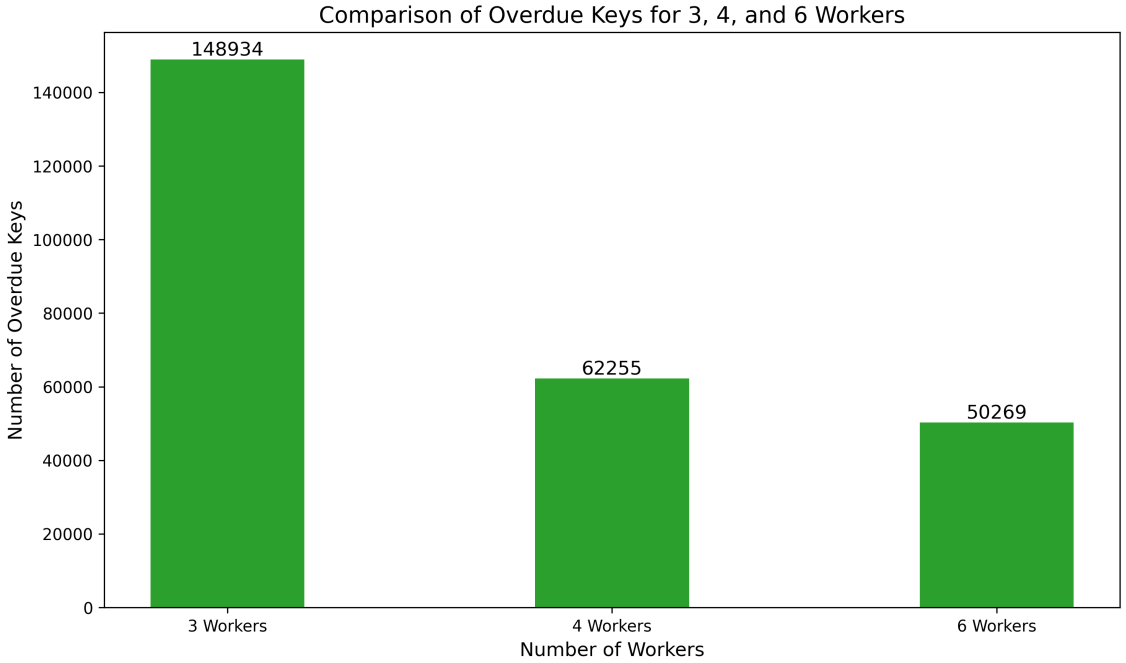



Σχήμα 6.54: *Ανισοκατανομή - Μέγιστος Φόρτος Κόμβων Σταδίου 2 Συνολικά για 3,4 και 6 Κόμβους.*

### Μεταβαλλόμενο Spike

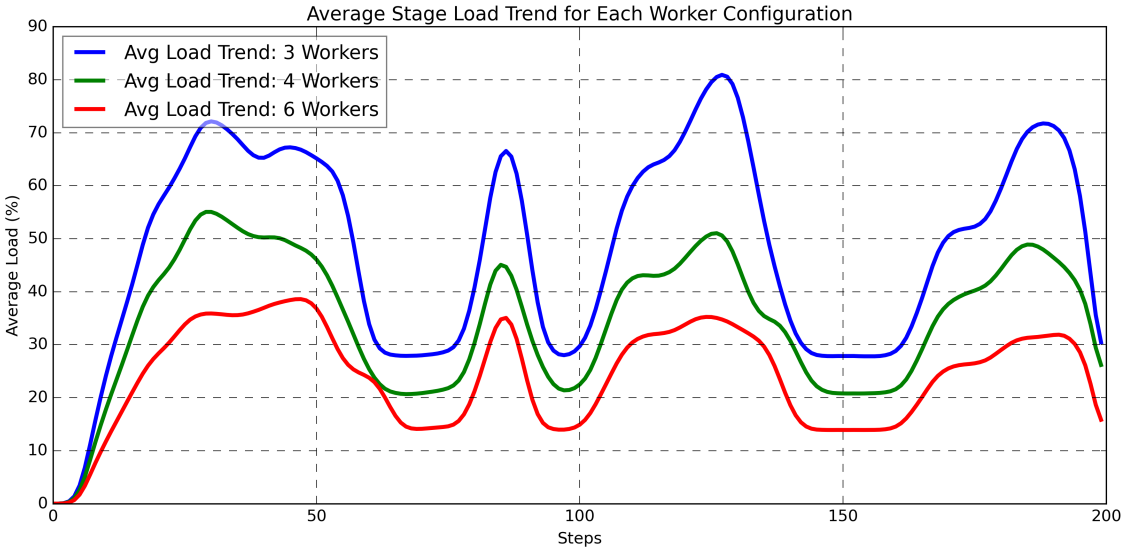
Παρά το γεγονός ότι οι προηγούμενες γραφικές παραστάσεις παρείχαν πολύτιμα αποτελέσματα, δεν είναι κατάλληλες για την απεικόνιση του συστήματος υπό συνθήκες μεταβαλλόμενου spike, όπου ο ρυθμός άφιξης των δεδομένων παρουσιάζει συνεχείς αυξομειώσεις. Για να αντιμετωπίσουμε αυτό το ζήτημα, χρησιμοποιούμε δύο διαφορετικά διαγράμματα.

Στο Σχήμα 6.56 απεικονίζεται η μέση εξέλιξη του φόρτου του σταδίου 2 καθ' όλη τη διάρκεια της προσομοίωσης. Όπως φαίνεται, ο μικρότερος φόρτος επιτυγχάνεται όταν χρησιμοποιούνται 6 κόμβοι. Ωστόσο, αυτό που μας ενδιαφέρει κυρίως είναι οι απότομες αυξομειώσεις του φόρτου, δηλαδή ο ρυθμός με τον οποίο αλλάζει κατά την εμφάνιση αιχμών. Παρατηρώντας προσεκτικά το γράφημα, διαπιστώνουμε ότι στην περίπτωση των 3




Σχήμα 6.55: *Ανισοκατανομή - Συνολικά Εκπρόθεσμα Κλειδιά στο Στάδιο 2 για 3, 4 και 6 Κόμβους.*

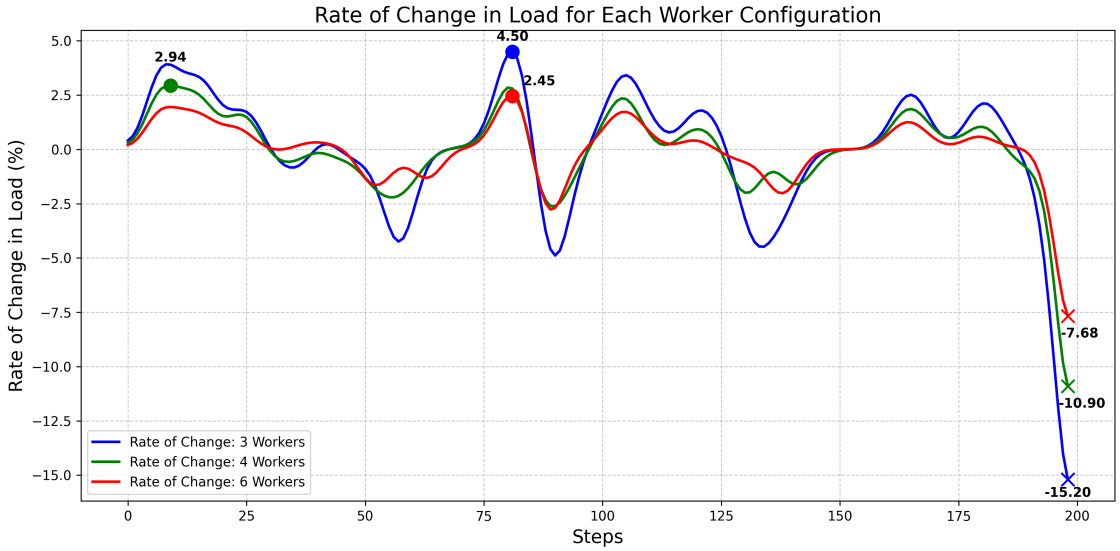
κόμβων, οι μεταβολές του φόρτου είναι πολύ πιο έντονες. Για παράδειγμα, ο μέσος φόρτος μεταβάλλεται, μέσα σε λίγα βήματα, από περίπου 30% σε 80%, ενώ στην περίπτωση των 6 κόμβων, η αντίστοιχη μεταβολή κυμαίνεται από 15% έως 35%.




Σχήμα 6.56: *Μεταβαλλόμενο Spike - Μέσος Φόρτος Σταδίου 2 για 3, 4 και 6 Κόμβους.*

Για να αναδείξουμε καλύτερα αυτές τις διακυμάνσεις, χρησιμοποιούμε το Σχήμα 6.57, το οποίο παρουσιάζει τον ρυθμό μεταβολής του μέσου φόρτου για το στάδιο 2, καθώς και τις μέγιστες και ελάχιστες τιμές για τις τρεις τοπολογίες. Όπως παρατηρούμε, στην περίπτωση των 3 κόμβων (μπλε γραμμή), σημειώνονται οι μεγαλύτερες αποκλίσεις από το 0, με μέγιστη θετική τιμή 4.5% και μέγιστη αρνητική απόκλιση -15.2%. Αντίθετα, στην περίπτωση

των 6 κόμβων (κόκκινη γραμμή), οι αποκλίσεις είναι πολύ πιο περιορισμένες, με μέγιστη θετική τιμή 2.45% και μέγιστη αρνητική -7.68%. Τέλος, για τους 4 κόμβους, οι αποκλίσεις βρίσκονται κάπου στο ενδιάμεσο, με μέγιστη τιμή 2.95% και ελάχιστη -10.9%.




Σχήμα 6.57: *Μεταβαλλόμενο Spike - Ρυθμός Μεταβολής Μέσου Φόρτου Σταδίου 2 για 3, 4 και 6 Κόμβους.*

Από τα παραπάνω διαγράμματα, παρατηρούμε ότι με την αύξηση του αριθμού των κόμβων, το σύστημα είναι σε θέση να διαχειριστεί καλύτερα τις συνθήκες υψηλής αιχμής, περιορίζοντας τις απότομες μεταβολές του φόρτου.

## 6.5 Συμπεράσματα

Εκτελώντας τα πειράματα στις δύο διαφορετικές τοπολογίες, εξάγαμε σταθερά μοτίβα που αναδεικνύουν την επίδοση του συστήματος υπό διάφορες συνθήκες, όπως κανονική λειτουργία, σταδιακή αύξηση ρυθμού άφιξης κλειδιών, ανισοκατανομή των κλειδιών, καθώς και περιόδους αιχμής.

Τα αποτελέσματα δείχνουν ότι η αύξηση του αριθμού των κόμβων ενισχύει τη δυνατότητα του συστήματος να ανταπεξέλθει σε καταστάσεις μη κανονικής λειτουργίας, ιδιαίτερα όταν αντιμετωπίζει αυξημένο ρυθμό άφιξης κλειδιών. Αυτό υποδεικνύει ότι για εφαρμογές που χαρακτηρίζονται από μεταβλητό ή ασυνεπή ρυθμό άφιξης, η επιλογή μιας τοπολογίας με περισσότερους κόμβους είναι πιο αποτελεσματική για τη διατήρηση της απόδοσης και της αξιοπιστίας του συστήματος.

Επιπλέον, παρατηρήθηκε ότι η επιλογή της κατάλληλης στρατηγικής διαμερισμού κλειδιών μπορεί να βελτιώσει σημαντικά την ανοχή του συστήματος σε περιπτώσεις όπου οι εισερχόμενες πληροφορίες είναι συγκεντρωμένες σε λίγα μόνο κλειδιά, μειώνοντας τον κίνδυνο συμφόρησης σε συγκεκριμένους κόμβους. Ιδιαίτερα, η στρατηγική Partial Key Grouping (PKG) εμφάνισε πολύ ικανοποιητικά δεδομένα στα πειράματα με ανισοκατανομή των δεδομένων εισόδου.

Βέβαια, η επιτυχία και η απόδοση του PKG εξαρτάται άμεσα με την συνάρτηση συνάθροι-

σης αλλά και από την γενική λειτουργία που επιτελεί το στάδιο στο οποίο απαρτίζεται. Σε ορισμένα πειράματα φάνηκε ότι η συγκεκριμένη στρατηγική παρά την πολύ καλή συμπεριφορά υστερούσε σε σχέση με τις άλλες στρατηγικές λόγω του επιπλέον επιπέδου συνάθροισης και των περιορισμένων δυνατοτήτων του συγκεκριμένου κόμβου.

Θεωρούμε χρήσιμο να μελετηθεί περαιτέρω και να εξετασθεί η περίπτωση ενός κατανεμημένου σταδίου συνάθροισης των διασπασμένων κλειδιών της στρατηγικής PKG ώστε να μπορεί σε αξιοποιηθεί σε περισσότερες και διαφορετικές συνθήκες έχοντας παρόμοια επίδοση. Εναλλακτικά, θα μπορούσε να δημιουργηθεί μία δυναμική επιλογή της στρατηγικής ανάλογα με τον εκτιμόμενο φόρτο και κατανομή των κλειδιών, ώστε να επιλέγεται η συγκεκριμένη στρατηγική μόνο σε συγκεκριμένες καταστάσεις εισόδου.

# Μέρος III

# Επίλογος

Κεφάλαιο 7

# Επίλογος

## 7.1 Συμπεράσματα

Τα κατανεμημένα συστήματα επεξεργασίας συνεχών ροών δεδομένων αναπτύχθηκαν για να καλύψουν την αυξανόμενη ανάγκη για άμεση ανάλυση και επεξεργασία δεδομένων που ρέουν συνεχώς. Τα DSPSs είναι ιδανικά για αυτές τις λειτουργίες, καθώς προσφέρουν χαμηλή καθυστέρηση, ανθεκτικότητα σε σφάλματα, και οριζόντια κλιμάκωση (scale-out), ενώ μπορούν να διατηρούν εσωτερική κατάσταση για την εκτέλεση σύνθετων υπολογισμών (stateful processing).

Ωστόσο, προκλήσεις όπως η υπερβολικά υψηλή ροή δεδομένων, η άνιση κατανομή των κλειδιών (data skewness), και οι αιφνίδιες αυξήσεις στον ρυθμό άφιξης των δεδομένων (spikes), μπορούν να επηρεάσουν σημαντικά την απόδοση αυτών των συστημάτων. Για να αντιμετωπιστούν αυτές οι προκλήσεις, τα DSPSs εφαρμόζουν πρακτικές όπως η αύξηση των κόμβων για καλύτερη κλιμάκωση και η χρήση αποδοτικότερων στρατηγικών διανομής κλειδιών.

Η πρόβλεψη της συμπεριφοράς ενός DSPS σε τέτοιες συνθήκες, καθώς και η εκτίμηση της αποτελεσματικότητας των παραπάνω λύσεων, αποτελεί μια δύσκολη και συχνά κοστοβόρα διαδικασία, απαιτώντας συνήθως την εγκατάσταση φυσικών υποδομών και σύνθετων τοπολογιών.

Η συνεισφορά αυτής της διπλωματικής εργασίας είναι η ανάπτυξη ενός παραμετροποιήσιμου εργαλείου προσομοίωσης κατανεμημένων συστημάτων επεξεργασίας συνεχών ροών δεδομένων. Ο προτεινόμενος προσομοιωτής, με την προσαρμογή ενός μόνο αρχείου παραμετροποίησης, μπορεί να αναπαραστήσει τη λειτουργία ενός DSPS υπό ποικίλες συνθήκες, παρέχοντας άμεσα και χωρίς επιπλέον κόστος σημαντικές μετρικές απόδοσης.

Μέσα από μια σειρά πειραμάτων, επαληθεύτηκε η ορθότητα του προσομοιωτή, με τα αποτελέσματα να είναι συμβατά με τις θεωρητικές προσδοκίες. Επίσης, παρουσιάστηκε μια μεθοδολογία για τη χρήση του προσομοιωτή, με στόχο την αξιολόγηση της απόδοσης και τη βελτιστοποίηση ενός DSPS.

Συνοψίζοντας, ο προσομοιωτής που αναπτύχθηκε στο πλαίσιο αυτής της διπλωματικής αποτελεί ένα ολοκληρωμένο εργαλείο, ικανό να προσομοιώνει διαφορετικές λειτουργίες και συνθήκες ενός DSPS άμεσα και χωρίς επιπρόσθετο κόστος, καλύπτοντας ένα σημαντικό κενό στον τομέα της προσομοίωσης κατανεμημένων συστημάτων επεξεργασίας συνεχών ροών δεδομένων.

## 7.2 Μελλοντικές επεκτάσεις

Καθώς τα DSPSs μελλετώνται συνεχώς ολοένα και περισσότερο, προκύπτουν διαρκώς νέες προκλήσεις και τα συστήματα εξελίσσονται συνεχώς για να τις καλύψουν. Ο προσομοιωτής που αναπτύχθηκε στα πλαίσια αυτής της διπλωματικής εργασίας, θα μπορούσε να βελτιωθεί και να επεκταθεί περαταίρω, ενσωματώνοντας τις ακόλουθες λειτουργίες:

1. **Κυκλοφορία Δικτύου (Network Traffic)**: Ο φόρτος και η ταχύτητα του δικτύου, καθώς και η συχνότητα που επικοινωνούν οι κόμβοι ενός κατανεμημένου συστήματος μεταξύ τους, μπορεί να επηρεάσει σημαντικά την απόδοση και να παίξει καθοριστικό ρόλο. Η ενσωμάτωση αυτού του παράγοντα, θα προσέδιδε μεγαλύτερη ευελιξία και ακρίβεια στον προσομοιωτή μας.

2. **Κατανάλωση Μνήμης (Memory Footprint)**: Εκτός από τη χρονική πολυπλοκότητα και το χρόνο καθυστέρησης, σημαντική μετρική για τα DSPSs αποτελεί και η κατανάλωση της μνήμης. Τα DSPSs, για την εκτέλεση σύνθετων υπολογισμών, τις περισσότερες φορές, απαιτούν και την αποθήκευση μίας εσωτερικής κατάστασης (stateful processing). Καθώς προχωράει η επεξεργασία των ροών δεδομένων, η κατάσταση αυτή μπορεί να μεγαλώνει σε σημαντικό βαθμό και σε οριακές καταστάσεις, να υπερφορτώνει έναν κόμβο, οδηγώντας σε υποβάθμιση του συστήματος. Η συγκεκριμένη λειτουργία θα μπορούσε να ενσωματωθεί στον προσομοιωτή μας, προκειμένου να λαμβάνει υπόψη και την κατανάλωση της μνήμης, προσφέροντας ακόμα περισσότερη ακρίβεια και τη δυνατότητα για περισσότερες προσομοιώσεις.

3. **Παράθυρα Συνένωσης (Windowed Join)**: Τα παράθυρα συνένωσης στα DSPSs, ενώνουν στοιχεία ενός παραθύρου από δύο διαφορετικές ροές, με βάση ένα κοινό κλειδί. Τέτοιες λειτουργίες είναι αρκετά απαιτητικές και χρησιμοποιούνται για διάφορες εφαρμογές όπως η ανίχνευση ανωμαλιών, τα συστήματα προτάσεων και ο έλεγχος χρηματοικονομικών συναλλαγών [54]. Η ενσωμάτωση αυτής της λειτουργίας, θα έδινε τη δυνατότητα για προσομοίωση ακόμη περισσότερων εργασιών.

4. **Βελτιστοποιητής (Optimizer)**: Επέκταση του προσομοιωτή ώστε να μπορεί να μεταφράζει ορισμένες εργασίες (high level jobs) σε συγκεκριμένες τοπολογίες για καλύτερη εξέταση συγκεκριμένων συνθηκών. Πρόταση συγκεκριμένων βελτιώσεων τοπολογίας σε συγκεκριμένες εργασίες με διαφορετικές στρατηγικές διαμέρισης και συγκεκριμένο αριθμό κόμβων.

# Παραρτήματα

Παράρτημα Α΄

# Υλοποίηση

Στο παράρτημα αυτό περιγράφεται η υλοποίηση του προσομοιωτή, με βάση τη μελέτη που παρουσιάστηκε στο Κεφάλαιο 4. Αρχικά γίνεται αναφορά στα προγραμματιστικά εργαλεία που χρησιμοποιήθηκαν και στη συνέχεια δίνονται λεπτομέρειες υλοποίησης και η δομή του κώδικα.

## Α΄.1 Προγραμματιστικά εργαλεία

Ο προσομοιωτής είναι εξ ολοκλήρου υλοποιημένος στη γλώσσα προγραμματισμού Python.

## Α΄.2 Περιγραφή κλάσεων

Στην ενότητα αυτή δίνεται μία περιγραφή των κλάσεων, των πεδίων και των μεθόδων που τις απαρτίζουν.

### Α΄.2.1 KeyGenerator

**Περιγραφή:** Η κλάση αυτή, όπως προσδίδει και το όνομά της, παράγει τα κλειδιά για κάθε βήμα του προσομοιωτή με συγκεκριμένα χαρακτηριστικά.

**Πεδία (Attributes):** Τα πεδία της κλάσης είναι τα εξής:

- `config (dict)`: Το λεξικό παραμετροποίησης της γεννήτριας κλειδιών.
- `streams (int)`: Το πλήθος των διαφορετικών ροών (streams) που θα παράξει η γεννήτρια.
- `steps (int)`: Ο αριθμός των βημάτων προσομοίωσης.
- `num_keys (int)`: Το πλήθος των διαφορετικών κλειδιών.
- `arrival_rate (int)`: Ο αριθμός των κλειδιών που θα παραχθούν ανά βήμα.
- `arrival_rate_ot (int)`: Ο ρυθμός αύξησης / μείωσης των παραγόμενων κλειδιών ανά βήμα.
- `spike_probability (int)`: Ορίζει την πιθανότητα (ραγδαίας) μεταβολής του ρυθμού άφιξης των κλειδιών σε κάθε βήμα.

- `spike_magnitude (int)`: Καθορίζει το ποσοστιαίο μέγεθος του `spiked` ρυθμού μεταβολής παραγωγής κλειδιών.
- `dist_type (class)`: Η συμβολοσειρά του τύπου της κατανομής κλειδιών.
- `distribution (class)`: Η κλάση που υλοποιεί τον κάθε τύπο κατανομής κλειδιών.
- `extra_dir (str)`: Ορίζει τον επιθυμητό φάκελο στον οποίο θα αποθηκεύονται τα αρχεία καταγραφών (logs).
- `key_logger`: Ο logger που κάνει τις απαραίτητες καταγραφές στατιστικών.
- `distribution (object)`: Η κλάση κατανομής που παράγει αριθμούς με βάση την κατανομή. Περισσότερα για αυτή την κλάση στην Ενότητα Α΄.2.1.

**Constructor:** `KeyGenerator(config)`

Αρχικοποιεί την γεννήτρια κλειδιών (KeyGenerator) με την δοσμένη παραμετροποίηση (config). Το configuration αποτελεί κομμάτι του γενικού αρχείου παραμετροποίησης που περιγράφηκε στην Ενότητα 4.1.2. Ένα παράδειγμα της config παραμέτρου δίνεται παρακάτω:

```
config = {
    "streams": 1,
    "steps": 50,
    "number_of_keys": 10,
    "arrival_rate": 50,
    "arrival_rate_ot": 10,
    "spike_probability": 0,
    "spike_magnitude": 0,
    "distribution": {
        "type": "normal"
        "mean": 5,
        "stddev": 2
    }
}
```

Συγκεκριμένα, αναφορικά με τις κατανομές, οι διαθέσιμες υλοποιημένες κλάσεις κατανομών κλειδιών είναι οι UniformDistribution, NormalDistribution, PoissonDistribution, ZipfDistribution και κατασκευάζονται αναλόγως με το ποια κατανομή έχει οριστεί στο αρχείο παραμετροποίησης. Οι κλάσεις αυτές κληρονομούν από την αφηρημένη κλάση Distribution και υλοποιούν την μέθοδο generate η οποία παράγει μία σειρά αριθμών. Περισσότερα σχετικά με τις κατανομές στο Α΄.2.1. Τέλος, θέτει το πεδίο extra_dir ίσο με το extra_dir που έχει οριστεί στο GlobalConfiguration (θα αναφερθούμε σε αυτό στον Simulator), καθώς και αρχικοποιεί τον key_logger.

**Μέθοδοι:**

Οι μέθοδοι που υλοποιούνται στην κλάση KeyGenerator είναι οι εξής:

- `_init_distribution`: Αρχικοποιεί την κατανομή κλειδιών βάσει του τύπου της κατανομής.

**Επιστρέφει**:

- Ένα στιγμιότυπο της καθορισμένης κατανομής, δηλαδή μία εκ των UniformDistribution, NormalDistribution, PoissonDistribution, ZipfDistribution, ανάλογα με τον τύπο κατανομής που έχει οριστεί.

**Εξαιρέσεις**:

- `ValueError`: Εάν ο τύπος της κατανομής δεν υποστηρίζεται.

- `create_key_array(length, key)`: Κατασκευάζει έναν πίνακα κλειδιών ή ακεραίων βάσει του δοθέντος μήκους και του flag key. Εάν το key είναι ορισμένο ως True, η συνάρτηση δημιουργεί κλειδιά στη μορφή key0, key1 κ.ο.κ. Εάν είναι ορισμένο ως False, ο πίνακας θα περιέχει ακέραιους (π.χ. '0', '1' κ.ο.κ.).

  **Παράμετροι**:

  - `length` (int): Το μήκος του πίνακα που θα δημιουργηθεί.
  - `key` (bool): Εάν είναι `True`, θα δημιουργηθούν κλειδιά, αλλιώς χαρακτήρες που αναπαριστούν ακέραιους αριθμούς.

  **Επιστρέφει**:

  - `list`: Ένας πίνακας που περιέχει τα δημιουργημένα κλειδιά ή τους χαρακτήρες ακεραίων.

  **Παραδείγματα**:

```
>>> create_key_array(3, key=True)
['key0', 'key1', 'key2']

>>> create_key_array(3, key=False)
['0', '1', '2']

```

- `adjust_or_create_key_dist(key_array, swap)`: Δημιουργεί ή προσαρμόζει την κατανομή των κλειδιών βάσει του flag `swap`. Χρησιμοποιείται για να αλλάζει την ιεραρχία των κλειδιών από βήμα σε βήμα. Με `swap = False` η συνάρτηση ορίζει μία τυχαία σειρά συχνότητας εμφάνισης των κλειδιών. Παραπέμποντας στο Παράδειγμα 2 η έξοδος προσδιορίζει ότι το key2 θα είναι το συχνότερο, το key0 το δεύτερο συχνότερο κ.ο.κ. Αυτή θα είναι και η ιεραρχία συχνοτήτων. Οι επόμενες κλήσεις της συνάρτησης θα είναι με τη σημαία swap ορισμένη ως True. Κατά αυτή την λειτουργία η συνάρτηση απλώς προσαρμόζει την ιεραρχία των κλειδιών θεωρώντας ότι το κάθε κλειδί μπορεί να ανέβει ή να κατέβει το πολύ μία θέση στην ιεραρχία.

  **Παράμετροι**:

  - `key_array` (list): Μια λίστα με την ιεραρχία κλειδιών που πρέπει να προσαρμοστεί ή να ανακατευτεί.

- `swap` (bool): Ένδειξη για το αν θα προσαρμοστεί (`True`) ή θα δημιουργηθεί (`False`) νέα κατανομή των κλειδιών.

**Επιστρέφει**:

- `list`: Η προσαρμοσμένη ή η νέα ιεραρχία συχνοτήτων των κλειδιών.

**Παραδείγματα**:

```
>>> adjust_or_create_key_dist(['key0', 'key1', 'key2', 'key3'], True)}
['key0', 'key2', 'key1', 'key3'] # Example output, actual output may vary

>>> adjust_or_create_key_dist(['key0', 'key1', 'key2', 'key3'], False)}
['key2', 'key0', 'key3', 'key1'] # Example output, actual output may vary

```

- `replace_step_with_keys(step, keys)`: Αντικαθιστά κάθε στοιχείο στη λίστα `step` με το αντίστοιχο κλειδί, βάσει της συχνότητας εμφάνισης του στοιχείου.

  Η συνάρτηση αρχικά υπολογίζει τη συχνότητα εμφάνισης κάθε στοιχείου στη λίστα `step`. Έπειτα ταξινομεί τα στοιχεία κατά φθίνουσα σειρά συχνότητας. Κάθε στοιχείο αντιστοιχίζεται με ένα κλειδί από τη λίστα `keys`, σύμφωνα με αυτήν την κατάταξη συχνοτήτων.

  Η αρχική λίστα (`step`) είναι η παραγόμενη λίστα από την κλάση Κατανομής (μέσω της μεθόδου `generate`) και είναι μία λίστα ακεραίων (το πλήθος των διαφορετικών ακεραίων είναι ίσο με το πλήθος των διαφορετικών κλειδιών). Περισσότερα στο Α΄.2.1. Μετά το πέρασμα από αυτή την συνάρτηση οι ακέραιοι αριθμοί αντικαθίστανται με κλειδιά ακολουθούντας την συχνότητα εμφάνισης των κλειδιών που παράγει η μέθοδος `adjust_or_create_key_dist`.

  **Παράμετροι**:

  - `step` (list): Λίστα στοιχείων, όπου κάθε στοιχείο θα αντικατασταθεί με βάση τη συχνότητά του.
  - `keys` (list): Λίστα κλειδιών που θα αντικαταστήσουν τα στοιχεία της λίστας `step`. Το μήκος της λίστας `keys` θα πρέπει να είναι τουλάχιστον ίσο με τον αριθμό των μοναδικών στοιχείων της `step`.

  **Επιστρέφει**:

  - `list`: Μία νέα λίστα, όπου κάθε στοιχείο στη `step` έχει αντικατασταθεί με το αντίστοιχο κλειδί από την `keys`.

  **Παραδείγματα**:

```
>>> replace_step_with_keys(['a', 'b', 'a', 'c', 'a', 'b'], ['key0', 'key1', 'key2'])
['key0', 'key1', 'key0', 'key2', 'key0', 'key1']

```

- `generate_step`: Η συνάρτηση παράγει ένα βήμα (δηλαδή μία λίστα από κλειδιά) βάσει της κατανομής των κλειδιών για το τρέχον στάδιο της προσομοίωσης. Προσαρμόζει επίσης το arrival rate των κλειδιών, λαμβάνοντας υπόψη πιθανές αιχμές (spikes) και την ποσοσιστιαία αύξηση ανά βήμα (over time), αν υπάρχει, αυτού του ρυθμού.

  Χρησιμοποιεί την μέθοδο `generate` της αφηρημένης κλάσης Κατανομής (Distribution) για να παράξει τυχαίους ακέραιους του οποίους αντικαθιστά με κλειδιά μέσω της μεθόδου `replace_step_with_keys`.

  **Παράμετροι**:

  - `key_dist` (list): Η λίστα που αντιπροσωπεύει την ιεραρχία συχνοτήτων των κλειδιών στο συγκεκριμένο βήμα της προσομοίωσης.

  **Επιστρέφει**:

  - `list`: Μία λίστα με τα κλειδιά που δημιουργήθηκαν σε αυτό το βήμα της προσομοίωσης.

- `generate_stream(output_file)`: Δημιουργεί μια ροή κλειδιών για την προσομοίωση. Χρησιμοποιεί τη βοηθητική συνάρτηση `generate_step` για κάθε βήμα δημιουργίας.

  **Παράμετροι**:

  - `output_file` (str): Το αρχείο όπου θα αποθηκευτεί η ροή δεδομένων (stream).

  **Λειτουργία**: Δημιουργεί μια ροή κλειδιών σε διάφορα βήματα της προσομοίωσης και γράφει το αποτέλεσμα στο αρχείο `output_file`. Η συνάρτηση:

  - Αρχικοποιεί το `key_dist` χρησιμοποιώντας την `create_key_array` για να ορίσει τα χρησιμοποιούμενα κλειδιά σε αυτή την ροή κλειδιών.
  - Για κάθε βήμα, προσαρμόζει την ιεραρχία συχνοτήτων κλειδιών χρησιμοποιώντας την `adjust_or_create_key_dist`.
  - Δημιουργεί τα κλειδιά του βήματος μέσω της `generate_step` και καταγράφει στατιστικά μέσω της `log_key_statistics`.
  - Τέλος, γράφει τη ροή στο `output_file` χρησιμοποιώντας τη `write_output`.

- `generate_input(output_file)`: Δημιουργεί τα δεδομένα εισόδου που χρησιμοποιούνται στον κώδικα της προσομοίωσης. Δημιουργεί πολλαπλές ροές, καθεμία αποθηκευμένη σε διαφορετικό αρχείο για κάθε ξεχωριστή ροή.

  **Παράμετροι**:

  - `output_file` (str): Το βασικό όνομα του αρχείου όπου θα αποθηκευτεί η ροή δεδομένων (stream). Σε περίπτωση πολλαπλών ροών τα τελικά ονόματα αρχείων θα περιέχουν και έναν αύξοντα αριθμό που συμβολίζει την κάθε ροή.

  **Λειτουργία**: Δημιουργεί αρχεία με ονομασίες όπως "stream_output0.txt", "stream_output1.txt", κ.λπ., το καθένα περιέχει μια ροή που δημιουργήθηκε βάσει της κατανομής.

**Distributions**

**Περιγραφή:** Είναι η αφηρημένη κλάση Κατανομών.
**Πεδία (Attributes):**

- `keys (list)`: Ορίζει τα κλειδιά που θα χρησιμοποιηθούν από τις κατανομές.

**Αφηρημένες Μέθοδοι:**

- `generate(arrival_rate)`: Παράγει αριθμό κλειδιών ίσο με `arrval_rate` σύμφωνα με την υλοποιημένη κατανομή.

**Uniform**

**Περιγραφή:** Η κλάση ομοιόμορφης Κατανομής που παράγει κλειδιά.
**Μέθοδοι:**

- `generate(arrival_rate)`: Παράγει συγκεκριμένο αριθμό κλειδιών με ομοιόμορφη κατανομή.

**Normal**

**Περιγραφή:** Η κλάση κανονικής Κατανομής που παράγει κλειδιά.
**Πεδία (Attributes):**

- `mean (float)`: Ο μέσος όρος της Κατανομής.
- `stddev (float)`: Η τυπική απόκληση της Κατανομής.

**Μέθοδοι:**

- `generate(arrival_rate)`: Παράγει συγκεκριμένο αριθμό κλειδιών με κανονική κατανομή.

**Poisson**

**Περιγραφή:** Η κλάση Poisson κατανομής που παράγει κλειδιά.
**Πεδία (Attributes):**

- `lam (float)`: Η `lambda` $\lambda$ παράμετρος της Poisson κατανομής. Ορίζει τον μέσο όρο εμφανίσεων ενός γεγονότος σε συγκεκριμένο χρονικό διάστημα

**Μέθοδοι:**

- `generate(arrival_rate)`: Παράγει συγκεκριμένο αριθμό κλειδιών με Poisson κατανομή.

#### Zipf

**Περιγραφή:** Η κλάση Zipf κατανομής που παράγει κλειδιά. Η κατανομή προσομοιάζει την συχνότητα λέξεων στην φυσική γλώσσα.

**Πεδία (Attributes):**

- `alpha (float)`: Παράμετρος της κατανομής. Ορίζει το πόσο απότομα πέφτει η συχνότητα εμφάνισης των κλειδιών.

**Μέθοδοι:**

- `generate(arrival_rate)`: Παράγει συγκεκριμένο αριθμό κλειδιών με Zipf κατανομή.

### Α΄.2.2 Simulator

**Περιγραφή:** Αφορά την βασική κλάση του προσομοιωτή μας και ενώνει την γεννήτρια κλειδιών με την τοπολογία του συστήματος. Ορίζει τα βήματα της προσομοίωσης και εκτυπώνει μία τελική κατάσταση του συστήματος κατά την ολοκλήρωσή της.

**Πεδία (Attributes):**

- `topology (Topology)`: Η κλάση που περιέχει την πλήρη τοπολογία του προς προσομοίωση συστήματος.
- `input_partitioner (KeyPartitioner)`: Η KeyPartitioner κλάση (περισσότερα για την κλάση στην ενότητα Α΄.2.5) που κατανέμει τα κλειδιά στο πρώτο στάδιο.

**Μέθοδοι:**

- `_find_stage0_partitioner()`: Αρχικοποιεί το πεδίο `input_partitioner`.

  **Επιστρέφει**:

  - Ένα στιγμιότυπο της κλάσης `KeyPartitioner`.

  **Εξαιρέσεις**:

  - `ValueError`: Εάν δεν υπάρχει το στάδιο 0 (`stage.id = 0`) στην τοπολογία.

- `sim(steps_data)`: Εκτελεί την προσομοίωση με εισαγόμενα κλειδια `steps_data` στέλνοντας τα κλειδιά στον αρχικό διαμεριστή.

  **Παράμετροι:**

  - `steps_data (list)`: Η λίστα των παραγόμενων κλειδιών.

- `report()`: Εκτυπώνει την τελική κατάσταση του συστήματος. Καλείται μετά το τελευταίο βήμα της προσωμοίωσης.

### Α΄.2.3 Topology

**Περιγραφή:** Η κλάση περιέχει την πλήρη τοπολογία του συστήματος. Η τοπολογία διατηρεί την αρχιτεκτονική του συστήματος μέσω μίας λίστας που περιέχει το κάθε στάδιο (`Stage`) που με την σειρά του διατηρεί όλους του κόμβους (`Nodes`) σε μία νέα λίστα. Μέσω του `Constructor` γίνονται οι κατάλληλες αρχικοποιήσεις των σταδιών - κόμβων.

**Πεδία (Attributes):**

- `stages (list)`: Η λίστα των σταδιών (`stages`) της τοπολογίας.

**Constructor:** `Topology(topology_config)`
Αρχικοποιεί το πεδίο `stages` μέσω της κλήσης της μεθόδου `_create_stages`. Αναλυτικά ένα παράδειγμα της μορφής του `topology_config` φαίνεται στο 4.1.2.

**Μέθοδοι:**

- `_create_stages(stages_data)`: Δημιουργεί τις απαραίτητες `Stage` κλάσεις για κάθε στάδιο της τοπολογίας και τις επιστρέφει σε μορφή λίστας.

  **Παράμετροι**:

  - `stages_data` (list): Μια λίστα από λεξικά που αντιπροσωπεύουν τις ρυθμίσεις των σταδίων.

  **Επιστρέφει**:

  - `list`: Μια λίστα με στιγμιότυπα της κλάσης `Stage`.

  **Λειτουργία**:

  - Δημιουργεί στιγμιότυπα της κλάσης `Stage` για κάθε καταχώριση στη λίστα `stages_data`.
  - Καθορίζει την αναφορά στο επόμενο στάδιο για κάθε στιγμιότυπο `Stage` μέσω της μεθόδου `_set_next_stage`.

- `__repr__()`: Επιστρέφει την κλάση σε μορφή εκτύπωσης (`printable object`).

### Α΄.2.4 Stage

**Περιγραφή:** Η κλάση αυτή υλοποιεί το κάθε διακριτό στάδιο της τοπολογίας του προς προσομοίωση συστήματος. Κάθε στάδιο περιέχει έναν καθορισμένο αριθμό από ίδιου τύπου κόμβους (`Stateless, Stateful`) .

**Πεδία (Attributes):**

- `id (int)`: Το μοναδικό αναγνωριστικό του σταδίου.
- `stage_type (str)`: Ο τύπος του σταδίου. Ουσιαστικά αναφέρεται στον τύπο των κόμβων που περιέχει το συγκεκριμένο στάδιο.
- `key_splitting (bool)`: Ορίζει αν στο τρέχον στάδιο έχει επιτραπεί η διαμέριση ίδιων κλειδιών σε περισσότερους τους ενός κόμβους.

- `next_stage (Stage)`: Αναφορά στο στάδιο του επόμενου επιπέδου, αν υπάρχει.
- `next_stage_len (int)`: Το μήκος του επόμενου επιπέδου, αν υπάρχει, αλλιώς 0.
- `terminal_stage (bool)`: Ορίζει αν το στάδιο είναι το τελευταίο στάδιο της τοπολογίας.
- `hash_seed (int)`: Seed ώστε να εξασφαλίζεται η ομοιόμορφη συμπεριφορά των `KeyPartitioners` με στρατηγική `Hashing`.
- `key_node_map Dict[str, int]`: Λεξικό που χρησιμοποιείται από την στρατηγική διαμέρισης Power of Two Choices και αποθηκεύει για κάθε κλειδί τον επιλεγμένο κόμβο διαμέρισης.
- `key_candidates Dict[str, Tuple[int, int]]`: Λεξικό που χρησιμοποιείται από την στρατηγική διαμέρισης Partial Key Grouping και αποθηκεύει για κάθε κλειδί τους δύο πιθανούς κόμβους διαμέρισης.
- `nodes (list)`: Η λίστα των κόμβων που περιέχονται στο στάδιο.
- `aggregator (AggregatorNode)`: Η κλάση του κόμβου AggregatorNode που συλλέγει και ομαδοποιεί τα ίδια κλειδιά από διαφορετικούς κόμβους. Υλοποιείται στο στάδιο μόνο στην περίπτωση που η διαμέριση ίδιων κλειδιών σε πάνω από έναν κόμβο είναι ενεργή.

**Constructor:** `Stage(stage_data, next_stage_len)`

Αρχικοποιεί τους κόμβους του σταδίου μέσω της μεθόδου `_create_nodes` και τα υπόλοιπα πεδία της κλάσης. Σε περίπτωση που η διαμέριση ίδιων κλειδιών σε πολλαπλούς κόμβους `key_splitting` είναι ενεργή κατασκευάζει και τον AggregatorNode για την συλλογή των κλειδιών. Ένα παράδειγμα της μορφής του `stage_data` φαίνεται στο 4.1.2.

**Μέθοδοι:**

- `_set_next_stage(stage)`: Ορίζει το επόμενο στάδιο στην τοπολογία.

  **Παράμετροι**:

  - `stage` (Stage): Το αντικείμενο του επόμενου σταδίου.

  **Λειτουργία**: Θέτει την αναφορά στο επόμενο στάδιο για το τρέχον στάδιο.

- `_create_nodes(nodes_data)`: Δημιουργεί στιγμιότυπα κόμβων βάσει του τύπου τους.

  **Παράμετροι**:

  - `nodes_data` (list): Μια λίστα από λεξικά που αναπαριστούν τις ρυθμίσεις των κόμβων.

  **Επιστρέφει**:

  - `list`: Μια λίστα με στιγμιότυπα των κόμβων (WorkerNode, StatelessNode ή KeyPartitioner).

  **Λειτουργία**:

  - Δημιουργεί στιγμιότυπα κόμβων βάσει του τύπου τους (stateful, stateless ή key_partitioner).
  - Αν ο τύπος κόμβου είναι key_partitioner και η στρατηγική διαμερισμού είναι ο κατακερματισμός (Hashing), προστίθεται ένα hash_seed για να διασφαλιστεί η ομοιόμορφη συμπεριφορά κατακερματισμού σε κάθε στάδιο.

- `__repr__()`: Επιστρέφει την κλάση σε μορφή εκτύπωσης (`printable object`).

### Α΄.2.5 Node

**Περιγραφή:** Η αφηρημένη κλάση κόμβων.
**Πεδία (Attributes):**

- `uid (int)`: Το μοναδικό αναγνωριστικό του κόμβου στην τοπολογία.
- `stage_node_id (int)`: Το αναγνωριστικό του κόμβου εσωτερικά του σταδίου που βρίσκεται.
- `type (str)`: Η συμβολοσειρά του τύπου του κόμβου.
- `throughput (int)`: Η μέγιστη ρυθμαπόδοση του κόμβου.
- `stage (Stage)`: Το αντικείμενο της κλάσης του σταδίου `Stage` που βρίσκεται ο κόμβος

**Constructor:** `node(uid, stage_node_id, type, throughput, stage)`
Αρχικοποιεί τα πεδία του κόμβου.
**Μέθοδοι:**

- `receive_and_process(keys, step)`: Η αφηρημένη μέθοδος που λαμβάνει από κάποιον άλλον κόμβο και επεξεργάζεται τα κλειδιά (`keys`) ενός βήματος (`step`) σύμφωνα με τις προδιαγραφές του κάθε κόμβου.

#### StatelessNode

**Περιγραφή:** Η αφηρημένη κλάση των κόμβων χωρίς εσωτερική κατάσταση (`Stateless`) που κληρονομεί από την αφηρημένη κλάση `Nodes`.
**Constructor:** Καλεί τον constructor της κληρονομούμενης κλάσης.
**Μέθοδοι:**

- `__repr__()`: Επιστρέφει την κλάση σε μορφή εκτύπωσης (`printable object`

#### StatefulNode

**Περιγραφή:** Η αφηρημένη κλάση των κόμβων με εσωτερική κατάσταση (`Stateful`) που κληρονομεί από την αφηρημένη κλάση `Nodes`.
**Πεδία (Attributes):**

- `uid (int)`: Το μοναδικό αναγνωριστικό του κόμβου στην τοπολογία.

- `stage_node_id (int)`: Το αναγνωριστικό του κόμβου εσωτερικά του σταδίου που βρίσκεται.
- `type (str)`: Η συμβολοσειρά του τύπου του κόμβου.
- `throughput (int)`: Η μέγιστη ρυθμαπόδοση του κόμβου.
- `stage (Stage)`: Το αντικείμενο της κλάσης του σταδίου `Stage` που βρίσκεται ο κόμβος.
- `terminal (bool)`: Ορίζει αν το ο κόμβος είναι τερματικός κόμβος.
- `extra_dir (str)`: Ορίζει τον επιθυμητό φάκελο στον οποίο θα αποθηκεύονται τα αρχεία καταγραφών (logs).
- `default_logger` : Ο logger που κάνει τις απαραίτητες καταγραφές στατιστικών για τον προσομοιωτή.
- `node_logger` : Ο logger που κάνει τις απαραίτητες καταγραφές στατιστικών για τον κόμβο.

**Constructor:** `StatefulNode(uid, stage_node_id, type, throughput, stage, terminal)`
Καλεί τον constructor της κληρονομούμενης κλάσης και αρχικοποεί τους loggers.
**Μέθοδοι:**

- `receive_and_process(keys, step)`: Η αφηρημένη μέθοδος που λαμβάνει από κάποιον άλλον κόμβο και επεξεργάζεται τα κλειδιά (`keys`) ενός βήματος (`step`) σύμφωνα με τις προδιαγραφές του κάθε κόμβου.
- `emit_keys(keys, step)`: Η αφηρημένη μέθοδος που στέλνει τα επεξεργασμένα κλειδιά στον επόμενο κόμβο.
- `__repr__()`: Επιστρέφει την κλάση σε μορφή εκτύπωσης (`printable object`).

**Σημείωση!** Επιλέξαμε τη δημιουργία μίας επιπλέον βαθμίδας αφαίρεσης στους (`Stateless`) και (`Stateful`) κόμβους ώστε να μπορούν να δημιουργηθούν κόμβοι με ή χωρίς εσωτερική κατάσταση με διαφορετικές λειτουργίες. Εμείς υλοποιήσαμε τον κατανεμήτη (`KeyPartitioner`) ως έναν κόμβο χωρίς εσωτερική κατάσταση και τους κόμβους επεξεργασίας (`WorkerNode`) και κόμβους συνάθροισης (`AggregatorNode`) ως κόμβους με εσωτερική κατάσταση.

#### KeyPartitioner

**Περιγραφή:** Η κλάση κατανομής κλειδιών. Κληρονομεί από τους κόμβους χωρίς εσωτερική κατάσταση.
**Πεδία (Attributes):**

- `uid (int)`: Το μοναδικό αναγνωριστικό του κόμβου στην τοπολογία.
- `stage_node_id (int)`: Το αναγνωστικό του κόμβου εσωτερικά του σταδίου που βρίσκεται.
- `type (str)`: Η συμβολοσειρά του τύπου του κόμβου.

- `throughput (int)`: Η μέγιστη ρυθμαπόδοση του κόμβου.
- `stage (Stage)`: Το αντικείμενο της κλάσης του σταδίου `Stage` που βρίσκεται ο κόμβος
- `strategy (PartitionStrategy)`: Το αντικείμενο κλάσης που υλοποιεί μία συγκεκριμένη στρατηγική διαμέρισης κλειδιών.
- `buffers (dict)`: Υλοποιεί μία λίστα για όλους τους κόμβους του επόμενου σταδίου. Σε κάθε λίστα αποθηκεύονται τα κλειδιά που θα λάβουν οι κόμβοι στο τρέχον βήμα.
- `extra_dir (str)`: Ορίζει τον επιθυμητό φάκελο στον οποίο θα αποθηκεύονται τα αρχεία καταγραφών (logs)
- `default_logger` : Ο logger που κάνει τις απαραίτητες καταγραφές στατιστικών για τον προσομοιωτή.
- `node_logger` : Ο logger που κάνει τις απαραίτητες καταγραφές στατιστικών για τον κόμβο.

**Constructor:** `KeyPartitioner(uid, stage_node_id, throughput, stage, partitioning_strategy, strategy_params)`:

Καλεί τον `Constructor` της κληρονομούμενης κλάσης ώστε να κατασκευαστεί μία κλάση τύπου `Node`. Αρχικοποιεί τους `buffers` με βάση τον αριθμό των κόμβων του επόμενου σταδίου. Κατασκευάζει μία κλάση τύπου `PartitionStrategy` μέσω της μεθόδου `_init_strategy` η οποία θα ορίζει την στρατηγική διαμέρισης των κλειδιών.

**Μέθοδοι:**

- `_init_strategy(strategy_name, strategy_params)`: Αρχικοποιεί τη στρατηγική κατανομής κλειδιών βάσει του ονόματος και των παραμέτρων που δίνονται.

  **Παράμετροι**:

  - `strategy_name` (str): Το όνομα της στρατηγικής κατανομής κλειδιών.
  - `strategy_params` (dict): Παράμετροι για τη στρατηγική κατανομής κλειδιών.

  **Επιστρέφει**:

  - `PartitionStrategy`: Ένα στιγμιότυπο της συγκεκριμένης στρατηγικής κατανομής.

  **Λειτουργία**: Κατασκευάζει και επιστρέφει τη στρατηγική κατανομής κλειδιών βάσει του `strategy_name`.

  **Εξαιρέσεις**:

  - `ValueError`: Εάν ο τύπος της στρατηγικής δεν υποστηρίζεται.

- `receive_and_process(keys, step)`: Λαμβάνει μια λίστα από κλειδιά και τα κατανέμει στο επόμενο στάδιο.

  **Παράμετροι**:

  - `keys` (list): Λίστα κλειδιών προς επεξεργασία.

- `step` (int): Το τρέχον βήμα της προσομοίωσης.

**Λειτουργία**:

- Καταγράφει τα κλειδιά που έλαβε ο κόμβος
- Αν το στάδιο δεν είναι το τελικό, κατανέμει τα κλειδιά στο επόμενο στάδιο μέσω της επιλεγμένης στρατηγικής κατανομής κλειδιών.
- Στέλνει τα κλειδιά στους επόμενους κόμβους μέσω της μεθόδου `send_buffered_keys`.

- `send_buffered_keys(step_count)`: Στέλνει τα κλειδιά από τον προσωρινό χώρο αποθήκευσης στους επόμενους κόμβους και καθαρίζει τους buffers.

  **Παράμετροι**:

  - `step_count` (int): Ο αριθμός του τρέχοντος βήματος στην προσομοίωση.

  **Λειτουργία**: Για κάθε κόμβο στο επόμενο στάδιο, στέλνει τα προσωρινά αποθηκευμένα (`buffered`) κλειδιά, προσθέτει έναν δείκτη ενημέρωσης βήματος και αδειάζει τους `buffers` για το επόμενο βήμα.

- `__repr__()`: Επιστρέφει την κλάση σε μορφή εκτύπωσης (`printable object`).

**WorkerNode**

**Περιγραφή:** Η κλάση κόμβων επεξεργασίας με εσωτερική κατάσταση. Κληρονομεί από τους κόμβους με εσωτερική κατάσταση (`StatefulNode`).

**Πεδία (Attributes):**

- `uid (int)`: Μοναδικό αναγνωριστικό του κόμβου στην τοπολογία.
- `stage_node_id`: Το αναγνωστικό του κόμβου εσωτερικά του σταδίου που βρίσκεται.
- `type (str)`: Ο τύπος του κόμβου ("Worker").
- `throughput (int)`: Η ρυθμαπόδοση, δηλαδή ο μέγιστος αριθμός υπολογιστικών κύκλων που μπορεί να εκτελέσει ο κόμβος ανά βήμα.
- `stage (Stage)`: Το αντικείμενο της κλάσης του σταδίου `Stage` που βρίσκεται ο κόμβος.
- `terminal (bool)`: Καθορίζει αν ο κόμβος είναι τερματικός.
- `key_splitting (bool)`: Ορίζει αν στο τρέχον στάδιο έχει επιτραπεί η διαμέριση ίδιων κλειδιών σε περισσότερους τους ενός κόμβους.
- `state (State)`: Η εσωτερική κατάσταση του κόμβου.

**Constructor:** `WorkerNode(uid, stage_node_id, throughput, operation_type, stage, window_size, slide, terminal, key_splitting)`

Καλεί τον `Constructor` της κληρονομούμενης κλάσης ώστε να κατασκευαστεί μία κλάση τύπου `StatefulNode`. Αρχικοποεί την αρχική κατάσταση του κόμβου μέσω της κατασκευής

μιας κλάσης `WorkerState` και ορίζει αναλόγως αν επιτρέπεται η διαμέριση του ίδιου κλειδιού σε πολλαπλούς κόμβους.

**Μέθοδοι:**

- `receive_and_process(keys, step)`: Λαμβάνει και επεξεργάζεται μια λίστα κλειδιών ενημερώνοντας παράλληλα την εσωτερική κατάσταση του κόμβου.

  **Παράμετροι**:

  - `keys` (list): Λίστα κλειδιών προς επεξεργασία.
  - `step` (int): Το τρέχον βήμα της προσομοίωσης.

  **Λειτουργίες**:

  - Ανανεώνει την κατάσταση του κόμβου με βάση τα ληφθέντα κλειδιά.
  - Καταγράφει στατιστικά σχετικά με την επεξεργασία και την επίδοση του κόμβου.
  - Σε περίπτωση που δεν είναι κόμβος τερματικού σταδίου στέλνει τα επεξεργασμένα κλειδιά στο επόμενο στάδιο μέσω της μεθόδου `emit_keys`.

- `emit_keys(keys, step)`: Εκπέμπει τα υπολογισμένα κλειδιά στο επόμενο στάδιο. Σε περίπτωση που ο διαχωρισμός ομοίων κλειδιών (key_splitting) είναι ενεργός τα κλειδιά στέλονται στον AggregatorNode του σταδίου, αλλιώς στον KeyPartitioner που του αντιστοιχεί στο επόμενο στάδιο.

  **Παράμετροι**:

  - `keys` (list): Λίστα κλειδιών που εκπέμπονται στο επόμενο στάδιο.
  - `step` (int): Το τρέχον βήμα της προσομοίωσης.

  **Λειτουργίες**:

  - Εκπέμπει τα επεξεργασμένα κλειδιά στο επόμενο στάδιο της τοπολογίας ή στον AggregatorNode του σταδίου .
  - Καταγράφει στατιστικά σχετικά με την επεξεργασία και την επίδοση του κόμβου.

- `__repr__()`: Επιστρέφει την κλάση σε μορφή εκτύπωσης (`printable object`).

### AggregatorNode

**Περιγραφή:** Η κλάση κόμβων για συνάθροιση κλειδιών (`AggregatorNode`) στον προσομοιωτή μας.

**Πεδία (Attributes):**

- `uid (str)`: Μοναδικό αναγνωριστικό του κόμβου στη μορφή "stage_node_id_aggr".
- `stage_node_id (int)`: Το αναγνωριστικό του κόμβου εσωτερικά του σταδίου.
- `type (str)`: Ο τύπος του κόμβου ("Aggregator").

- `throughput`: Η ρυθμαπόδοση, δηλαδή ο μέγιστος αριθμός υπολογιστικών κύκλων που μπορεί να εκτελέσει ο κόμβος ανά βήμα.
- `stage (Stage)`: Το αντικείμενο της κλάσης του σταδίου `Stage` που βρίσκεται ο κόμβος.
- `operation_type (str)`: Ο τύπος της προσομοιούμενης λειτουργίας (`Operation`) από το τρέχον στάδιο.
- `terminal (bool)`: Καθορίζει αν ο κόμβος είναι τερματικός.
- `state (dict)`: Η εσωτερική κατάσταση του κόμβου που αποθηκεύει τα δεδομένα συνάθροισης.

**Constructor:** `AggregatorNode(stage_node_id, operation_type, stage, window_size, slide, stage_operation, terminal)`

Ο κατασκευαστής αρχικοποιεί τον κόμβο συνάθροισης με ένα μοναδικό `uid` και τις παραμέτρους που καθορίζουν τη λειτουργία του. Επίσης, καλεί τον κατασκευαστή της κλάσης `StatefulNode` και αρχικοποιεί την εσωτερική κατάσταση του κόμβου (`AggregatorState`).

**Μέθοδοι:**

- `receive_and_process(keys, step, sender_stage_node_id)`: Λαμβάνει και επεξεργάζεται τα κλειδιά που απαιτούν συνάθροιση και αποστέλλονται από άλλους κόμβους.

  **Παράμετροι**:

  - `keys` (dict): Λεξικό με τα δεδομένα κλειδιών προς επεξεργασία.
  - `step` (int): Το τρέχον βήμα της προσομοίωσης.
  - `sender_stage_node_id`: Το αναγνωριστικό του κόμβου που αποστέλλει τα δεδομένα.

  **Λειτουργίες**:

  - Ενημερώνει την εσωτερική κατάσταση του κόμβου με βάση τα ληφθέντα δεδομένα.
  - Αν ο κόμβος δεν είναι τερματικός, εκπέμπει τα επεξεργασμένα κλειδιά στο επόμενο στάδιο μέσω της μεθόδου `emit_keys`.

- `emit_keys(keys, step)`: Εκπέμπει τα υπολογισμένα κλειδιά στο επόμενο στάδιο.

  **Παράμετροι**:

  - `keys` (list): Λίστα κλειδιών που εκπέμπονται στο επόμενο στάδιο.
  - `step` (int): Το τρέχον βήμα της προσομοίωσης.

  **Λειτουργίες**:

  - Εκπέμπει τα επεξεργασμένα κλειδιά στο επόμενο στάδιο.
  - Καταγράφει πληροφορίες σχετικά με την εκπομπή κλειδιών.

### Α΄.2.6 Partition Strategies

**Περιγραφή:** Η αφηρημένη κλάση των στρατηγικών διαμέρισης κλειδιών.
**Μέθοδοι:**

- `partition(keys, nodes, buffers)` Η αφηρημένη μέθοδος που διαμερίζει τα κλειδιά (`keys`) στους κόμβους (`nodes`) αποθηκεύοντάς τα προσωρινά στους `buffers`.

#### Hashing

**Περιγραφή:** Η κλάση που υλοποιεί την στρατηγική κατακερματισμού (`hashing`) διαμέρισης κλειδιών.
**Πεδία (Attributes):**

- `hash_seed`: Η τιμή `seed` που χρησιμοποιείται για την κατανομή των κλειδιών στους κόμβους διαμέρισης ενός σταδίου, εξασφαλίζοντας συγχρονισμό μεταξύ των κόμβων στη διαδικασία κατακερματισμού.

**Constructor:** `Hashing(hash_seed)` Αρχικοποιεί το `seed` του κόμβου.
**Μέθοδοι:**

- `partition(keys, nodes, buffers)`: Κατανέμει κλειδιά στους κόμβους βάσει των τιμών κατακερματισμού τους.

  **Παράμετροι**:

  - `keys List[str]`): Τα κλειδιά που θα κατανεμηθούν.
  - `nodes List[Node]`): Οι κόμβοι στους οποίους θα κατανεμηθούν τα κλειδιά.
  - `buffers (dict)`: Ένα λεξικό όπου το κλειδί αντιστοιχεί σε έναν δείκτη κόμβου και η τιμή είναι μια λίστα κλειδιών προς αποθήκευση προσωρινά.

  **Λειτουργία**: Για κάθε κλειδί, υπολογίζεται η τιμή κατακερματισμού, η οποία στη συνέχεια υποβάλλεται σε μια πράξη XOR με το `hash_seed` για τη διασφάλιση συνέπειας στον κατακερματισμό με τους υπόλοιπους καταμεριστές κλειδιών του ίδιου σταδίου. Τα κλειδία προστίθεται στους (buffers) του κόμβου.

#### KeyGrouping

**Περιγραφή:** Η κλάση που υλοποιεί την διαμέριση κλειδιών βάσει ενός δοθέντος μήκους προθέματος κλειδιών (`prefix_length`).
**Πεδία (Attributes):**

- `prefix_length` (int): Το μήκος του προθέματος που χρησιμοποιείται για την ομαδοποίηση των κλειδιών.

- `group_map` (dict): Ένας χάρτης που αντιστοιχεί ομάδες κλειδιών σε δείκτες κόμβων.

**Constructor:** `__init__(prefix_length=1)` Αρχικοποιεί τη στρατηγική `KeyGrouping` με το καθορισμένο μήκος προθέματος.
**Μέθοδοι:**

- `partition(keys, nodes, buffers)`: Κατανέμει τα κλειδιά στους κόμβους βάσει του προθέματός τους και τα αποθηκεύει προσωρινά.

  **Παράμετροι**:

  - `keys (List[str])`: Η λίστα των κλειδιών που θα κατανεμηθούν.
  - `nodes (List[Node])`: Οι κόμβοι στους οποίους θα κατανεμηθούν τα κλειδιά.
  - `buffers (dict)`: Ένα λεξικό όπου κάθε κλειδί αντιστοιχεί σε έναν κόμβο, και η τιμή είναι μια λίστα με κλειδιά που αποθηκεύονται προσωρινά για αυτόν τον κόμβο.

  **Λειτουργία**: Για κάθε κλειδί, το πρόθεμα καθορίζεται από το μήκος `prefix_length`. Το κλειδί κατακερματίζεται βάσει του προθέματος. Στη συνέχεια προστίθεται στην κατάλληλη θέση στον `buffer` ανάλογα με τον κόμβο στον οποίο θα κατανεμηθεί το κλειδί.

**ShuffleGrouping**

**Περιγραφή:** Υλοποιεί την κυκλική (`round robin`) διαμέριση κλειδιών.
**Πεδία (Attributes):**

- `current_index (int)`: Ο δείκτης του κόμβου στον οποίο θα διαμεριστεί το κλειδί στο τρέχον βήμα.

**Constructor:** `ShuffleGrouping()` Αρχικοποιεί τη στρατηγική `ShuffleGrouping` με αρχικό δείκτη 0.
**Μέθοδοι:**

- `partition(keys, nodes, buffers)`: Κατανέμει τα κλειδιά στους κόμβους με κυκλικό τρόπο.

  **Παράμετροι**:

  - `keys` (List[str]): Η λίστα των κλειδιών που θα κατανεμηθούν.
  - `nodes` (List[Node]): Οι κόμβοι στους οποίους θα κατανεμηθούν τα κλειδιά.
  - `buffers` (dict): Ένα λεξικό όπου κάθε κλειδί αντιστοιχεί σε έναν δείκτη κόμβου, και η τιμή είναι μια λίστα με κλειδιά που αποθηκεύονται προσωρινά για αυτόν τον κόμβο.

  **Λειτουργία**: Για κάθε κλειδί, η στρατηγική το αποθηκεύει στο `buffer` του κόμβου που αντιστοιχεί στον τρέχοντα δείκτη. Ο δείκτης `current_index` αυξάνεται κυκλικά με κάθε κατανομή για να εξασφαλιστεί ότι τα κλειδιά κατανέμονται εξίσου στους κόμβους.

**PowerOfTwoChoices**

**Περιγραφή:** Διαμερίζει το κάθε κλειδί μέσω δύο κλήσεων της συνάρτησης κατακερματισμού. Στην δεύτερη κλήση εισάγεται στο κλειδί και ένα "salt" ώστε να παραχθούν δύο διαφορετικοί πιθανοί κόμβοι διαμέρισης. Επιλέγεται ο κόμβος που έχει το μικρότερο φόρτο

(load) εκ των δύο. Η επιλογή γίνεται στην πρώτη κλήση της μεθόδου partition της κλάσης. Περισσότερα για την συνάρτηση διαμερισμού στο Κεφάλαιο 2.7.3.

**Πεδία (Attributes):**

- `key_node_map (Dict[str, int])`: Λεξικό που χαρτογραφεί τον επιλεγμένο κόμβο διαμέρισης για το κάθε διαφορετικό κλειδί. Το λεξικό διατηρείται κοινό για κάθε στάδιο (stage).

**Constructor:** `PowerOfTwoChoices(key_node_map)` Αρχικοποιεί τη στρατηγική `PowerOfTwoChoices`.
**Μέθοδοι:**

- `partition(keys, nodes, buffers)`: Κατανέμει τα κλειδιά στους κόμβους.

  **Παράμετροι**:

  - `keys` (List[str]): Η λίστα των κλειδιών που θα κατανεμηθούν.
  - `nodes` (List[Node]): Οι κόμβοι στους οποίους θα κατανεμηθούν τα κλειδιά.
  - `buffers` (dict): Ένα λεξικό όπου κάθε κλειδί αντιστοιχεί σε έναν δείκτη κόμβου, και η τιμή είναι μια λίστα με κλειδιά που αποθηκεύονται προσωρινά για αυτόν τον κόμβο.

  **Λειτουργία**: Για κάθε κλειδί, κατά την πρώτη εμφάνισή του, υπολογίζονται δύο υποψήφιοι κόμβοι μέσω δύο συναρτήσεων κατακερματισμού. Στη συνέχεια, υπολογίζεται ο φόρτος εργασίας των δύο κόμβων και επιλέγεται αυτός με το μικρότερο φόρτο. Ο κόμβος αυτός αποθηκεύεται στο πεδίο key_node_map για μελλοντική χρήση. Από το σημείο αυτό και έπειτα, το κλειδί αυτό κατευθύνεται πάντα σε αυτόν τον κόμβο.

### PartialKeyGrouping

**Περιγραφή:** Ορίζονται δύο πιθανοί κόμβοι διαμέρισης μέσω δύο κλήσεων της συνάρτησης κατακερματισμού στην αρχική κλήση της μεθόδου. Σε κάθε μετέπειτα κλήση της μεθόδου, για κάθε κλειδί, επιλέγεται δυναμικά ο κόμβος με τον μικρότερο φόρτο εργασίας εκ των δύο πιθανών (candidate) κόμβων. Περισσότερα για την συνάρτηση διαμερισμού στο Κεφάλαιο 2.7.3.

**Πεδία (Attributes):**

- `key_candidates (Dict[str, Tuple[int, int])`: Λεξικό που χαρτογραφεί τους πιθανούς κόμβους διαμέρισης για το κάθε διαφορετικό κλειδί. Το λεξικό διατηρείται κοινό για κάθε στάδιο (stage).

**Constructor:** `PartialKeyGrouping(key_candidates)` Αρχικοποιεί τη στρατηγική `PowerOfTwoChoices`.
**Μέθοδοι:**

- `partition(keys, nodes, buffers)`: Κατανέμει τα κλειδιά στους κόμβους.

  **Παράμετροι**:

  - `keys` (List[str]): Η λίστα των κλειδιών που θα κατανεμηθούν.
  - `nodes` (List[Node]): Οι κόμβοι στους οποίους θα κατανεμηθούν τα κλειδιά.

- `buffers` (dict): Ένα λεξικό όπου κάθε κλειδί αντιστοιχεί σε έναν δείκτη κόμβου, και η τιμή είναι μια λίστα με κλειδιά που αποθηκεύονται προσωρινά για αυτόν τον κόμβο.

**Λειτουργία**: Για κάθε κλειδί, κατά την πρώτη εμφάνισή του, υπολογίζονται δύο υποψήφιοι κόμβοι μέσω δύο συναρτήσεων κατακερματισμού και επιλέγονται και οι δύο, δηλαδή αποθυκεύονται στο πεδίο key_candidates. Κάθε φορά που εμφανίζεται το κλειδί, υπολογίζεται ο φόρτος των δύο υποψήφιων κόμβων και το κλειδί κατευθύνεται σε αυτόν με το μικρότερο φόρτο εργασίας τη δεδομένη στιγμή.

### Α΄.2.7 State

#### BaseState

**Περιγραφή:** Η αφηρημένη κλάση της κατάστασης των κόμβων με εσωτερική κατάσταση (StatefulNodes).

**Σημείωση!** Η κατάσταση των κόμβων υλοποιήθηκε ως αφηρημένη κλάση, προκειμένου να δοθεί η δυνατότητα υλοποίησης εξειδικευμένων καταστάσεων για κάθε τύπο κόμβου (π.χ. WorkerNode, AggregatorNode). Αυτό μας επιτρέπει να καλύψουμε τις ιδιαίτερες ανάγκες και τη συμπεριφορά που απαιτείται από κάθε τύπο κόμβου ξεχωριστά.

**Πεδία (Attributes):**

- `node_id (int)`: Μοναδικό αναγνωριστικό του κόμβου στην τοπολογία.
- `throughput (int)`: Η ρυθμαπόδοση, δηλαδή οι μέγιστοι υπολογιστικοί κύκλοι που μπορεί να εκτελέσει ο κόμβος ανά βήμα.
- `operation_type (Operation)`: Η συμβολοσειρά του τύπου της προσομοιούμενης λειτουργίας (`Operation`) από το τρέχον στάδιο.
- `operation (Operation)`: Ο τύπος της προσομοιούμενης λειτουργίας (`Operation`) από το τρέχον στάδιο.
- `window_size (int)`: Το μέγεθος του παραθύρου επεξεργασίας.
- `slide (int)`: Το διάστημα μετατόπισης (κύλισης) του παραθύρου επεξεργασίας.
- `extra_dir (str)`: Ορίζει τον επιθυμητό φάκελο στον οποίο θα αποθηκεύονται τα αρχεία καταγραφών (logs).
- `default_logger` : Ο logger που κάνει τις απαραίτητες καταγραφές στατιστικών για τον προσομοιωτή.
- `node_logger` : Ο logger που κάνει τις απαραίτητες καταγραφές στατιστικών για τον κόμβο.

**Constructor** `BaseState(node_id, throughput, operation_type, window_size, slide)`: Αρχικοποιεί την βασική εσωτερική κατάσταση. Ορίζει μέσω της βοηθητικής συνάρτησης `create_operation` τον τύπο λειτουργίας.

**Μέθοδοι:**

- `update(keys, step, terminal)`: Η αφηρημένη μέθοδος που λαμβάνει τα κλειδιά keys που έλαβε ο κόμβος στο τρέχον βήμα (step) και ενημερώνει την κατάστασή του.
- `process_full_windows(terminal)`: Η αφηρημένη μέθοδος που αναλαμβάνει την επεξεργασία των γεμάτων παραθύρων.
- `remove_expired_windows()`: Η αφηρημένη μέθοδος που είναι υπεύθυνση για τη διαγραφή των ληγμένων παραθύρων.

**WorkerState**

**Περιγραφή:** Αντιπροσωπεύει την εσωτερική κατάσταση ενός κόμβου επεξεργασίας στην προσομοίωση.

**Πεδία (Attributes):**

- `node_id (int)`: Μοναδικό αναγνωριστικό του κόμβου στην τοπολογία.
- `throughput (int)`: Η ρυθμαπόδοση, δηλαδή οι μέγιστοι υπολογιστικοί κύκλοι που μπορεί να εκτελέσει ο κόμβος ανά βήμα.
- `operation_type (Operation)`: Η συμβολοσειρά του τύπου της προσομοιούμενης λειτουργίας (`Operation`) από το τρέχον στάδιο.
- `operation (Operation)`: Ο τύπος της προσομοιούμενης λειτουργίας (`Operation`) από το τρέχον στάδιο.
- `window_size (int)`: Το μέγεθος του παραθύρου επεξεργασίας.
- `slide (int)`: Το διάστημα μετατόπισης (κύλισης) του παραθύρου επεξεργασίας.
- `received_keys (List[tuple[str, int, int]])`: Λίστα με τα κλειδιά που ελήφθησαν, μαζί με το βήμα άφιξης και το `max_step` τους.
- `windows (Dict[int, Window])`: Λεξικό που διαχειρίζεται τα παράθυρα (`Windows`) κλειδιών.
- `current_step (int)`: Το τρέχον βήμα της προσομοίωσης.
- `minimum_step (int)`: Το ελάχιστο βήμα που λαμβάνεται υπόψη για την επεξεργασία κλειδιών.
- `step_cycles (int)`: Ο τρέχον αριθμός υπολογιστικών κύκλων που χρησιμοποιήθηκαν σε αυτό το βήμα προσομοίωσης.
- `total_keys (int)`: Συνολικός αριθμός κλειδιών που λήφθηκαν.
- `total_processed (int)`: Συνολικός αριθμός επεξεργασμένων κλειδιών.
- `total_expired (int)`: Συνολικός αριθμός κλειδιών που έληξαν.
- `total_cycles (int)`: Συνολικός αριθμός υπολογιστικών κύκλων που χρησιμοποιήθηκαν.

- `extra_dir (str)`: Ορίζει τον επιθυμητό φάκελο στον οποίο θα αποθηκεύονται τα αρχεία καταγραφών (logs).
- `default_logger` : Ο logger που κάνει τις απαραίτητες καταγραφές στατιστικών για τον προσομοιωτή.
- `node_logger` : Ο logger που κάνει τις απαραίτητες καταγραφές στατιστικών για τον κόμβο.

**Constructor:** `WorkerState(node_id, throughput, operation_type, window_size, slide)` Αρχικοποιεί τα πεδία της κλάσης και κατασκευάζει μία κλάση τύπου `Operation` για τη λειτουργία επεξεργασίας κλειδιών μέσω της μεθόδου `create_operation(operation_type)`.

**Μέθοδοι:**

- `update(keys, step, terminal)`: Ενημερώνει την κατάσταση του κόμβου με νέα κλειδιά και το τρέχον βήμα της προσομοίωσης.

  **Παράμετροι**:

  - `keys list[str])`: Λίστα με τα κλειδιά που ελήφθησαν.
  - `step (int)`: Το τρέχον βήμα της προσομοίωσης.
  - `terminal (bool)`: Αν ο κόμβος είναι τερματικός.

  **Λειτουργίες**:

  - Επεξεργάζεται τα γεμάτα παράθυρα μέσω της μεθόδου `process_full_windows()` πριν ανανεώσει την τρέχουσα κατάσταση με τα κλειδιά του τωρινού βήματος προσομοίωσης.
  - Υπολογίζει το τρέχον καθώς και το ελάχιστο βήμα προσομοιώσης για το οποίο μπορεί να δεχθεί κλειδιά.
  - Ανανεώνει την εσωτερική κατάσταση και τα παράθυρα του κόμβου κάνοντας χρήσης της μεθόδου `update_windows`.
  - Αφαιρεί τα ληγμένα (`expired`) παράθυρα και κλειδιά καλώντας την `remove_expired_windows()` και `remove_expired_keys()` αντίστοιχα.
  - Διατηρεί στατιστικά σχετικά με την επεξεργασία και ανανέωση της κατάστασης του κόμβου.

  **Επιστρέφει**:

  - `list[list]`: Λίστα με τα επεξεργασμενα κλειδιά που θα μεταβιβαστούν στο επόμενο στάδιο, αν ο κόμβος δεν είναι τερματικός.

- `update_windows(key, step)`: Ενημερώνει τα παράθυρα δημιουργώντας νέα με βάση το step και προσθέτοντας το κλειδί key σε ήδη υπάρχοντα παράθυρα.
- `process_full_windows(terminal)`: Επεξεργάζεται τα παράθυρα που έχουν γεμίσει.

- `remove_expired_windows()`: Αφαιρεί παράθυρα που έχουν λήξει βάσει του τρέχοντος βήματος.
- `remove_expired_keys()`: Αφαιρεί κλειδιά που έχουν λήξει βάσει του τρέχοντος βήματος τους.
- `load()`: Υπολογίζει το φόρτο του κόμβου με βάση το συνολικό πλήθος κλειδιών σε όλα τα ενεργά παράθυρα. Χρησιμοποιείται στις στρατηγικές διαμερισμού PoTC και PKG.
- `process_window(window, terminal)`: Επεξεργάζεται ένα πλήρες παράθυρο και ενημερώνει την κατάσταση του κόμβου.

  **Παράμετροι:**

  - `window (Window)`: Το προς επεξεργασία παράθυρο.
  - `terminal (bool)`: Αν ο κόμβος είναι τερματικός.

  **Λειτουργίες**:

  - Επεξεργάζεται το παράθυρο μέσω της μεθόδου `process` της κλάσης `Window`. Περισσότερα για την κλάση και την μέθοδο στην ενότητα Α΄.2.8.
  - Με βάση τον αριθμό των κύκλων επεξεργασίας υπολογίζει τον φόρτο (load) του κόμβου.
  - Ορίζει τα εναπομείναντα μη επεξεργασμένα κλειδιά του παραθύρου ως εκπρόθεσμα (overdue) και δίνει έναν επιπλέον χρόνο για την επεξεργασία τους ίσο με το τριπλάσιο της ολίσθησης. Αν περάσει και αυτό το διάστημα δίχως να επεξεργαστούν, τότε αυτά θεωρούνται ληγμένα και διαγράφονται.
  - Καταγράφει στατιστικά όπως τον φόρτο του κόμβου, τα επεξεργασμένα και τα εκπρόθεσμα κλειδιά.

  **Επιστρέφει**:

  - `list`: Τα κλειδιά που θα μεταβιβαστούν από το παράθυρο.
- `__repr__()`: Επιστρέφει την κλάση σε μορφή εκτύπωσης (`printable object`).

  **Παράδειγμα Χρήσης**:

```
>>> print(state)
Node ID: 1
Received Keys: [('key1', 1, 10), ('key2', 2, 11)]
Key Counts: {'key1': 1, 'key2': 1}
Current Step: 2
Total Keys Received: 2
Total Processed: 1

```

**AggregatorState**

**Περιγραφή:** Αντιπροσωπεύει την εσωτερική κατάσταση ενός κόμβου συνάθροισης στην προσομοίωση.

**Πεδία (Attributes):**

- `node_id (int)`: Μοναδικό αναγνωριστικό του κόμβου στην τοπολογία.
- `throughput (int)`: Η ρυθμαπόδοση, δηλαδή οι μέγιστοι υπολογιστικοί κύκλοι που μπορεί να εκτελέσει ο κόμβος ανά βήμα.
- `operation_type (Operation)`: Η συμβολοσειρά του τύπου της προσομοιούμενης λειτουργίας (`Operation`) από τον κόμβο.
- `window_size (int)`: Το μέγεθος του παραθύρου επεξεργασίας.
- `slide (int)`: Το διάστημα μετατόπισης (κύλισης) του παραθύρου επεξεργασίας.
- `stage_operation (Operation)`: Ο τύπος της προσομοιούμενης λειτουργίας (`Operation`) από το τρέχον στάδιο.
- `stage_node_count (int)`: Το πλήθος των κόμβων του σταδίου στο οποίο βρίσκεται.
- `windows (Dict[int, Window])`: Λεξικό που διαχειρίζεται τα παράθυρα (`Windows`) κλειδιών.
- `current_step (int)`: Το τρέχον βήμα της προσομοίωσης.
- `minimum_step (int)`: Το ελάχιστο βήμα που λαμβάνεται υπόψη για την επεξεργασία κλειδιών.
- `total_processed (int)`: Συνολικός αριθμός επεξεργασμένων κλειδιών.
- `total_expired (int)`: Συνολικός αριθμός κλειδιών που έληξαν.
- `total_cycles (int)`: Συνολικός αριθμός υπολογιστικών κύκλων που χρησιμοποιήθηκαν.
- `extra_dir (str)`: Ορίζει τον επιθυμητό φάκελο στον οποίο θα αποθηκεύονται τα αρχεία καταγραφών (logs).
- `default_logger` : Ο logger που κάνει τις απαραίτητες καταγραφές στατιστικών για τον προσομοιωτή.
- `node_logger` : Ο logger που κάνει τις απαραίτητες καταγραφές στατιστικών για τον κόμβο.

**Constructor**: `AggregatorState(node_id, throughput, operation_type, window_size, slide, stage_operation, stage_nodes_count)`

Αρχικοποιεί τα πεδία της κλάσης και κατασκευάζει μία κλάση τύπου `Operation` για τη λειτουργία επεξεργασίας κλειδιών μέσω της μεθόδου `create_operation(operation_type)`.

**Μέθοδοι:**

- `update(keys, step, terminal, sender_stage_id)`: Ενημερώνει την κατάσταση του κόμβου με νέα κλειδιά και το τρέχον βήμα της προσομοίωσης.

  **Παράμετροι**:

  - `keys list[str])`: Λίστα με τα κλειδιά που ελήφθησαν.
  - `step (int)`: Το τρέχον βήμα της προσομοίωσης.
  - `terminal (bool)`: Αν ο κόμβος είναι τερματικός.
  - `sender_stage_id (int)`: Το id του κόμβου αποστολέα, μέσα στο στάδιο.

  **Λειτουργίες**:

  - Υπολογίζει το τρέχον καθώς και το ελάχιστο βήμα προσομοιώσης για το οποίο μπορεί να δεχθεί κλειδιά.
  - Για κάθε κλειδί που έρχεται, ενημερώνει την εσωτερική κατάσταση και τα παράθυρα μέσω της μεθόδου `update_windows`.
  - Σε περίπτωση που έρθει το κλειδί "finished" αυτό σημαίνει ότι ο αποστολέας τελείωσε την επεξεργασία του παραθύρου, δεν έχει σκοπό να στείλει ξανά για αυτό το παράθυρο και συνεπώς η κατάσταση ενημερώνεται για αυτή την πληροφορία, μέσω της μεθόδου `update_boolean_for_window`.
  - Επεξεργάζεται τα γεμάτα παράθυρα μέσω της μεθόδου `process_full_windows()`.
  - Αφαιρεί τα ληγμένα (`expired`) παράθυρα καλώντας την `remove_expired_windows()`.
  - Διατηρεί στατιστικά σχετικά με την επεξεργασία και ανανέωση της κατάστασης του κόμβου.

  **Επιστρέφει**:

  - `list[list]`: Λίστα με τα επεξεργασμενα κλειδιά που θα μεταβιβαστούν στο επόμενο στάδιο, αν ο κόμβος δεν είναι τερματικός.

- `update_windows(key, step)`: Ενημερώνει τα παράθυρα δημιουργώντας νέα με βάση το step και προσθέτοντας το κλειδί key σε ήδη υπάρχοντα παράθυρα.

- `process_full_windows(terminal)`: Επεξεργάζεται τα παράθυρα που έχουν γεμίσει και για τα οποία έχει λάβει πληροφορία από όλους τους κόμβους του σταδίου, ότι τελείωσαν την επεξεργασία αυτών των παραθύρων.

- `remove_expired_windows()`: Αφαιρεί παράθυρα που έχουν λήξει βάσει του τρέχοντος βήματος.

- `process_window(window, terminal)`: Επεξεργάζεται ένα πλήρες παράθυρο και ενημερώνει την κατάσταση του κόμβου.
  **Παράμετροι**:

  - `window (Window)`: Το προς επεξεργασία παράθυρο.
  - `terminal (bool)`: Αν ο κόμβος είναι τερματικός.

**Λειτουργίες**:

- Επεξεργάζεται το παράθυρο μέσω της μεθόδου της κλάσης `Window`, `process`.
- Με βάση τον αριθμό των κύκλων επεξεργασίας υπολογίζει τον φόρτο (load) του κόμβου.
- Ορίζει τα εναπομείναντα μη επεξεργασμένα κλειδιά του παραθύρου ως εκπρόθεσμα (overdue) και δίνει έναν επιπλέον χρόνο για την επεξεργασία τους ίσο με το τριπλάσιο της ολίσθησης. Αν περάσει και αυτό το διάστημα δίχως να επεξεργαστούν, τότε αυτά θεωρούνται ληγμένα και διαγράφονται.
- Καταγράφει στατιστικά όπως τον φόρτο του κόμβου, τα επεξεργασμένα και τα εκπρόθεσμα κλειδιά.

**Επιστρέφει**:

- `list`: Τα κλειδιά που θα μεταβιβαστούν από το παράθυρο.

- `update_boolean_for_window(window_start_step, sender_id)`: Ενημερώνει ότι ο κόμβος με id σταδίου ίσο με sender_id τελείωσε την επεξεργασία του παραθύρου που ξεκίνησε στο βήμα window_start_step.

  **Παράμετροι**:

  - `window_start_step (int)`: Το βήμα έναρξης του παραθύρου.
  - `sender_id (int)`: Το id του αποστολέα, μέσα στο στάδιο.

- `__repr__()`: Επιστρέφει την κλάση σε μορφή εκτύπωσης (`printable object`).

### Α΄.2.8 Window

**Περιγραφή:** Υλοποιεί ένα χρονικό παράθυρο (με βάση τις συμβάσεις χρόνου που έχουμε ορίσει στο κεφάλαιο 4.3).

**Πεδία (Attributes):**

- `start_step (int)`: Το αρχικό βήμα του παραθύρου.
- `size (int)`: Το μέγεθος του παραθύρου σε βήματα.
- `slide (int)`: Η ολίσθηση του παραθύρου σε βήματα.
- `keys (list)`: Λίστα με τα κλειδιά που λήφθηκαν σε αυτό το παράθυρο.

**Constructor:** `window(start_step, window_size, slide)` Αρχικοποιεί ένα παράθυρο με βήμα έναρξης ίσο με `start_step`.

**Μέθοδοι:**

- `add_key(key)`: Προσθέτει ένα κλειδί στη λίστα του παραθύρου.

  **Παράμετροι**:

- `key` (str): Το κλειδί που θα προστεθεί.

- `process(throughput, operation, step_cycles)`: Επεξεργάζεται τα κλειδιά του παραθύρου βάσει του `throughput`, της πολυπλοκότητας της λειτουργίας (operation) και του πλήθους επεξεργαστικών κύκλων που έχουν δαπανηθεί μέχρι τώρα στο τρέχον βήμα.

  **Παράμετροι**:

  - `throughput` (int): Ο μέγιστος αριθμός υπολογιστικών κύκλων ανά βήμα.
  - `operation` (Operation): Αντικείμενο λειτουργίας για τον υπολογισμό των κύκλων.
  - `step_cycles` (int): Το πλήθος των υπολογιστικών κόμβων που έχουν χρησιμοποιηθεί στο τρέχον βήμα. Χρησιμοποείται στην περίπτωση που επεξεργάζονται περισσότερα του ενός παραθύρων, στο ίδιο βήμα, ώστε να μη ξεπεραστεί η ρυθμαπόδοση του κόμβου.

  **Επιστρέφει**:

  - `tuple[int, int, dict[str, int]`: Ο αριθμός των επεξεργασμένων κλειδιών, οι συνολικοί κύκλοι που χρησιμοποιήθηκαν, και η καταμέτρηση των κλειδιών που επεξεργάστηκαν.

- `compute_cost(processed_key_count, operation)`: Υπολογίζει τους συνολικούς κύκλους που απαιτούνται για τα κλειδιά που επεξεργάστηκαν.

  **Παράμετροι**:

  - `processed_key_count` (dict): Λεξικό με τα κλειδιά και τις εμφανίσεις τους.
  - `operation` (Operation): Αντικείμενο λειτουργίας για τον υπολογισμό των κύκλων.

  **Επιστρέφει**:

  - `int`: Οι συνολικοί κύκλοι που απαιτούνται για την επεξεργασία των τρεχόντων κλειδιών.

- `is_processable(current_step)`: Ελέγχει αν το παράθυρο βρίσκεται υπό επεξεργασία, δηλαδή αν έχουν περάσει λιγότερα από $window\_size + 3 \times slide$ βήματα από τη στιγμή έναρξης του παραθύρου (start_step).

- `is_expired(current_step)`: Ελέγχει αν το παράθυρο έχει λήξει βάσει του τρέχοντος βήματος.

- `__repr__()`: Επιστρέφει την κλάση σε μορφή εκτύπωσης (`printable object`).

### Α΄.2.9 Operations

#### Operation

**Περιγραφή:** Η αφηρημένη κλάση των διαφορετικών λειτουργίων (Operations) ενός σταδίου.

**Μέθοδοι**:

- `calculate_cycles(n)`: Αφηρημένη μέθοδος που υπολογίζει τον αριθμό των κύκλων επεξεργασίας που απαιτούνται για την επεξεργασία `n` κλειδιών με βάση την ορισμένη λειτουργία (τελεστή επεξεργασίας).

- `to_str()`: Αφηρημένη μέθοδος που επιστρέφει τον τύπο λειτουργία ως συμβολοσειρά.

#### StatelessOperation

**Περιγραφή:** Αυτή η κλάση αντιπροσωπεύει μια stateless λειτουργία.
**Μέθοδοι**:

- `calculate_cycles(n)`: Επιστρέφει 1, προσομοιώνοντας σταθερή πολυπλοκότητα $O(1)$.

- `to_str()`: Επιστρέφει τη συμβολοσειρά "StatelessOperation".

#### BinaryOperation

**Περιγραφή:** Αυτή η κλάση αντιπροσωπεύει δυαδικές λειτουργίες, όπως η δυαδική αναζήτηση.
**Μέθοδοι**:

- `calculate_cycles(n)`: Λαμβάνει ως είσοδο τον αριθμό n των κλειδιών και επιστρέφει $\log n$, προσομοιώνοντας πολυπλοκότητα $O(\log n)$.

- `to_str()`: Επιστρέφει τη συμβολοσειρά "BinaryOperation".

#### Aggregation

**Περιγραφή:** Αυτή η κλάση αντιπροσωπεύει λειτουργίες συνάθροισης, όπως τον υπολογισμό αθροίσματος ή την εύρεση μέγιστου στοιχείου.
**Μέθοδοι**:

- `calculate_cycles(n)`: Λαμβάνει ως είσοδο τον αριθμό n των κλειδιών και επιστρέφει n, προσομοιώνοντας γραμμική πολυπλοκότητα $O(n)$.

- `to_str()`: Επιστρέφει τη συμβολοσειρά "Aggregation".

#### Sorting

**Περιγραφή:** Αυτή η κλάση αντιπροσωπεύει λειτουργίες ταξινόμησης.
**Μέθοδοι**:

- `calculate_cycles(n)`: Λαμβάνει ως είσοδο τον αριθμό n των κλειδιών και επιστρέφει $n \log n$, προσομοιώνοντας πολυπλοκότητα $O(n \log n)$.

- `to_str()`: Επιστρέφει τη συμβολοσειρά "Sorting".

**NestedLoop**

**Περιγραφή:** Αυτή η κλάση αντιπροσωπεύει λειτουργίες συνένωσης και πολλαπλασιασμού.

**Μέθοδοι**:

- `calculate_cycles(n)`: Λαμβάνει ως είσοδο τον αριθμό n των κλειδιών και επιστρέφει $n^2$, προσομοιώνοντας πολυπλοκότητα $O(n^2)$.
- `to_str()`: Επιστρέφει τη συμβολοσειρά "NestedLoop".

# Βιβλιογραφία


[1] Haruna Isah, Tariq Abughofa, Sazia Mahfuz, Dharmitha Ajerla, Farhana Zulkernine και Shahzad Khan. *A Survey of Distributed Data Stream Processing Frameworks*. *IEEE Access*, 7:154300–154316, 2019.

[2] GeeksforGeeks. *Difference Between Batch Processing and Stream Processing*, 2024. Ημερομηνία πρόσβασης: 29-04-2024.

[3] Atlan. *Batch Processing vs Stream Processing: 7 Key Differences*, 2023. Ημερομηνία πρόσβασης: 29-04-2024.

[4] P. Frossard, O. Verscheure και C. Venkatramani. *Signal Processing Challenges in Distributed Stream Processing Systems*. *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, τόμος 5, σελίδες "–", 2006.

[5] Nataliia Manakova και Anna Vergeles. *Calibration of Low-Cost IoT Sensors in Streams*. *2020 IEEE Third International Conference on Data Stream Mining  Processing (DSMP)*, σελίδες 449–454, 2020.

[6] Ravi Kiran Mallidi, Manmohan Sharma και Sreenivas Rao Vangala. *Streaming Platform Implementation in Banking and Financial Systems*. *2022 2nd Asian Conference on Innovation in Technology (ASIANCON)*, σελίδες 1–6, 2022.

[7] Prashant Chauhan και Madhu Shukla. *A review on outlier detection techniques on data stream by using different approaches of K-Means algorithm*. *2015 International Conference on Advances in Computer Engineering and Applications*, σελίδες 580–585, 2015.

[8] Rajeshwari U και B Sathish Babu. *Real-time credit card fraud detection using Streaming Analytics*. *2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*, σελίδες 439–444, 2016.

[9] Anatoliy Batyuk και Volodymyr Voityshyn. *Apache storm based on topology for real-time processing of streaming data from social networks*. *2016 IEEE First International Conference on Data Stream Mining  Processing (DSMP)*, σελίδες 345–349, 2016.

[10] Aleksandra I. Stojnev και Dragan H. Stojanović. *Software systems for processing and analysis of big data and event streams*. *2017 13th International Conference on Advanced Technologies, Systems and Services in Telecommunications (TELSIKS)*, σελίδες 128–131, 2017.

[11] Guenter Hesse και Martin Lorenz. *Conceptual Survey on Data Stream Processing Systems*. *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*, σελίδες 797–802, 2015.

[12] Dremio. *Data Skew*, 2024. Ημερομηνία πρόσβασης: 30-04-2024.

[13] H. C. M. Andrade, B. Gedik και D. S. Turaga. *Fundamentals of Stream Processing: Application Design, Systems, and Analytics*. Cambridge University Press, United Kingdom, 2014.

[14] F. Hueske και V. Kalavri. *Stream Processing with Apache Flink: Fundamentals, Implementation, and Operation of Streaming Applications*. O'Reilly Media, United States, 1η έκδοση, 2019.

[15] Giselle Van Dongen και Dirk Van Den Poel. *Influencing Factors in the Scalability of Distributed Stream Processing Jobs*. *IEEE Access*, 9:109413–109431, 2021.

[16] Ritika Pandey, Akanksha Singh, Angel Kashyap και Abhineet Anand. *Comparative Study on Realtime Data Processing System*. *2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU)*, σελίδες 1–7, 2019.

[17] Hamid Nasiri, Saeed Nasehi και Maziar Goudarzi. *Evaluation of distributed stream processing frameworks for IoT applications in Smart Cities*. *Journal of Big Data*, 6(1):52, 2019. Ημερομηνία πρόσβασης: 20-05-2024.

[18] Eran Levy. *7 Popular Stream Processing Frameworks Compared*, 2019. Ημερομηνία πρόσβασης: 16-04-2024.

[19] Jeffrey Richman. *9 Best Stream Processing Frameworks: Comparison 2024*, 2024. Ημερομηνία πρόσβασης: 20-04-2024.

[20] Wikipedia contributors. *BackType — Wikipedia, The Free Encyclopedia*, 2024. Ημερομηνία πρόσβασης: 17-07-2024.

[21] Sharad Toshniwal, Atul Taneja και Amit Shukla. *Storm at Twitter: Distributed Realtime Computation System*. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, σελίδες 147–156, 2014. Ημερομηνία πρόσβασης: 16-03-2024.

[22] Lennart Neumeyer, Bruce Robbins, Anand Nair και Anand Kesari. *S4: Distributed Stream Computing Platform*. *Proceedings of the 2010 IEEE International Conference on Data Mining Workshops*, σελίδες 170–177, 2010. Ημερομηνία πρόσβασης: 16-03-2024.

[23] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi και Kostas Tzoumas. *Apache Flink: Stream and Batch Processing in a Single Engine*. *IEEE Data Engineering Bulletin*, 38(4):28–38, 2015. Ημερομηνία πρόσβασης: 15-03-2024.

[24] Asterios Katsifodimos, Paris Carbone, Stephan Ewen, Volker Markl, Seif Haridi και Kostas Tzoumas. *Distributed Stream Processing with Apache Flink*. *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, σελίδες 179–190, 2015. Ημερομηνία πρόσβασης: 20-03-2024.

[25] Jay Kreps, Neha Narkhede και Jun Rao. *Kafka: A Distributed Messaging System for Log Processing*. *Proceedings of the 6th International Workshop on Networking Meets Databases (NetDB)*, σελίδες 1–7, 2011. Ημερομηνία πρόσβασης: 17-03-2024.

[26] N. Narkhede, G. Shapira και T. Palino. *Kafka: The Definitive Guide: Real-Time Data and Stream Processing at Scale*. O'Reilly Media, United States, 2017.

[27] Shadi A. Noghabi, Kartik Paramasivam, Yi Pan, Navina Ramesh, Jon Bringhurst, Indranil Gupta και Roy H. Campbell. *Samza: stateful scalable stream processing at LinkedIn*. *Proc. VLDB Endow.*, 10(12):1634–1645, 2017.

[28] Carlos Segarra, Enric Muntané, Mathieu Lemay, Valerio Schiavoni και Ricard Delgado-Gonzalo. *Secure Stream Processing for Medical Data*. *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, σελίδες 3450–3453, 2019.

[29] Apache Kafka Streams Documentation Team. *Core Concepts*, 2024. Ημερομηνία πρόσβασης: 05-05-2024.

[30] T. Akidau, S. Chernyak και R. Lax. *Streaming Systems: The What, Where, When, and How of Large-Scale Data Processing*. O'Reilly Media, United States, 2018.

[31] Matei Zaharia, Tathagata Das, Haoyuan Li, Scott Shenker και Ion Stoica. *Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters*. *Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Ccomputing*, HotCloud'12, σελίδα 10, USA, 2012. USENIX Association.

[32] Tim Sawicki. *Navigating stateful stream processing*, 2024. Ημερομηνία πρόσβασης: 20-09-2024.

[33] Jeffrey Richman. *Stateful Stream Processing: Concepts, Tools, Challenges*, 2023. Ημερομηνία πρόσβασης: 20-09-2024.

[34] Micah Wylde. *What is stateful stream processing?*, 2024. Ημερομηνία πρόσβασης: 20-09-2024.

[35] Ofili Lewis. *Windowing in Stream Processing*, 2023. Ημερομηνία πρόσβασης: 23-04-2024.

[36] Macrometa. *What is Windowing?* Ημερομηνία πρόσβασης: 23-04-2024.

[37] Daniil Gusev. *A guide to windowing in stream processing*, 2024. Ημερομηνία πρόσβασης: 19-09-2024.

[38] Juliane Verwiebe, Philipp M. Grulich, Jonas Traub και Volker Markl. *Survey of window types for aggregation in stream processing systems*. *The VLDB Journal*, 32:985–1011, 2023.

[39] M. Kleppmann. *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. O'Reilly Media, United States, 2017.

[40] Wikipedia contributors. *Word count — Wikipedia, The Free Encyclopedia*, 2024. Ημερομηνία πρόσβασης: 17-09-2024.

[41] Wikipedia contributors. *Round-robin scheduling — Wikipedia, The Free Encyclopedia*, 2024. Ημερομηνία πρόσβασης: 05-07-2024.

[42] M. Mitzenmacher. *The power of two choices in randomized load balancing*. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1094–1104, 2001.

[43] Muhammad Anis Uddin Nasir, Gianmarco De Francisci Morales, David García-Soriano, Nicolas Kourtellis και Marco Serafini. *The power of both choices: Practical load balancing for distributed stream processing engines*. *2015 IEEE 31st International Conference on Data Engineering*, σελίδες 137–148, 2015.

[44] Muhammad Anis Uddin Nasir, Gianmarco De Francisci Morales, Nicolas Kourtellis και Marco Serafini. *When two choices are not enough: Balancing at scale in Distributed Stream Processing*. *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, σελίδες 589–600, 2016.

[45] Hanhua Chen, Fan Zhang και Hai Jin. *Popularity-aware differentiated distributed stream processing on skewed streams*. *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, σελίδες 1–10, 2017.

[46] Eleni Zapridou, Ioannis Mytilinis και Anastasia Ailamaki. *Dalton: Learned Partitioning for Distributed Data Streams*. *Proc. VLDB Endow.*, 16(3):491–504, 2022.

[47] Alfred J. Park, Cheng Hong Li, Ravi Nair, Nobuyuki Ohba, Uzi Shvadron, Ayal Zaks και Eugen Schenfeld. *Flow: A Stream Processing System Simulator*. *2010 IEEE Workshop on Principles of Advanced and Distributed Simulation*, σελίδες 1–9, 2010.

[48] Gayashan Amarasinghe, MarcosDias de Assuncao, Aaron Harwood και Shanika Karunasekera. *ECSNeT++: A simulator for distributed stream processing on edge and cloud environments*. *Future Generation Computer Systems*, 111:401–418, 2020.

[49] Jeyhun Karimov, Tilmann Rabl, Asterios Katsifodimos, Roman Samarev, Henri Heiskanen και Volker Markl. *Benchmarking Distributed Stream Data Processing Systems*. *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, σελίδες 1507–1518, 2018.

[50] John D. Sutter. *Fans asked to tweet from Olympics only if it's 'urgent'*, 2012. Ημερομηνία πρόσβασης: 25-05-2024.

[51] Anatoliy Batyuk και Volodymyr Voityshyn. *Apache storm based on topology for real-time processing of streaming data from social networks*. *2016 IEEE First International Conference on Data Stream Mining  Processing (DSMP)*, σελίδες 345–349, 2016.

[52] Wikipedia contributors. *Gaussian Filter — Wikipedia, The Free Encyclopedia*, 2024. Ημερομηνία πρόσβασης: 01-10-2024.

[53] Wikipedia contributors. *Zipf's law — Wikipedia, The Free Encyclopedia*, 2024. Ημερομηνία πρόσβασης: 20-08-2024.

[54] OpenAI. *Understanding Streaming Joins in RisingWave*, 2024. Ημερομηνία πρόσβασης: 15-09-2024.

# Απόδοση ξενόγλωσσων όρων

| Απόδοση | Ξενόγλωσσος όρος |
|---|---|
| ανοχή σε σφάλματα | fault tolerance |
| αποδοτικότητα | efficiency |
| αποδοτικότητα συστήματος | system efficiency |
| αποθήκευση | storage |
| αυξανόμενος ρυθμός άφιξης | increasing arrival rate |
| αιχμές δεδομένων | data spikes |
| ανισοκατανομή δεδομένων | data skew |
| αρχιτεκτονική συστημάτων συνεχούς ροής | stream processing system architecture |
| διαχείριση παραθύρων | windowing techniques |
| διασύνδεση | interface |
| καθυστέρηση | latency |
| κανονική λειτουργία | normal operation |
| καταβόθρες | sinks |
| κατευθυνόμενος άκυκλος γράφος | directed acyclic graph (DAG) |
| κατανεμημένα συστήματα | distributed systems |
| κατανομή δεδομένων | data distribution |
| κατανομή κλειδιών | key partitioning |

| | |
|---|---|
| κατανομή φόρτου εργασίας | load distribution |
| κλίμακωση | scalability |
| κόμβοι επεξεργασίας | processing nodes |
| κόμβος | node |
| με εσωτερική κατάσταση | stateful |
| μεταβλητό φορτίο | dynamic load |
| μεταβλητό φορτίο εργασίας | variable workload |
| μετρικές | metrics |
| μοντέλο συνεχούς ροής | continuous stream model |
| μοτίβα επεξεργασίας | processing patterns |
| μικρο-ομαδοποίηση | micro-batching |
| παράλληλη επεξεργασία | parallel processing |
| παράμετροι προσομοίωσης | simulation parameters |
| παράθυρα | windows |
| παρτίδες | batches |
| πηγές | sources |
| προσομοίωση | simulation |
| προσομοιωτές συνεχούς ροής | stream processing simulators |
| προσομοίωση φορτίου | load simulation |
| πρότυπο επεξεργασίας κλειδιών | key processing pattern |
| ροές | streams |
| ροές δεδομένων | data streams |

| | |
|---|---|
| ρυθμαπόδοση | throughput |
| στρατηγικές διαμερισμού κλειδιών | key partitioning strategies |
| σενάρια λειτουργίας | operation scenarios |
| τελεστές | operators |
| τελεστές χωρίς κατάσταση | stateless operators |
| τελεστές με κατάσταση | stateful operators |
| τεχνική κατανομής δύο επιλογών | power of two choices load balancing |
| τεχνικές κλιμάκωσης | scaling techniques |
| υπολογιστικός φόρτος | computational load |