Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Δρομολόγηση εργασιών πραγματικού χρόνου σε συστήματα υπολογιστικού νέφους

Διπλωματική εργασία

Κωνσταντίνου Μεσσή

Επιβλέπων καθηγητής: Νεκτάριος Κοζύρης

Αθήνα,
Ιούλιος 2017

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Δρομολόγηση εργασιών πραγματικού χρόνου σε συστήματα υπολογιστικού νέφους

Διπλωματική εργασία

Κωνσταντίνου Μεσσή

Επιβλέπων καθηγητής: Νεκτάριος Κοζύρης

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 5η Ιουλίου 2017.

_____	_____	_____
Νεκτάριος Κοζύρης	Γεώργιος Γκούμας	Δημήτριος Τσουμάκος
Καθηγητής Ε.Μ.Π.	Λέκτορας Ε.Μ.Π.	Αναπληρωτής Καθηγητής Ιονίου Πανεπιστημίου

Αθήνα,
Ιούλιος 2017

Κωνσταντίνος Μεσσής
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Copyright © Κωνσταντίνος Μεσσής, 2017.
Με επιφύλαξη παντός δικαιώματος. All rights reserved

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

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

Περίληψη

Στις μέρες μας, το cloud computing και η τεχνολογία virtualization έχουν επιφέρει επαναστατικές αλλαγές στις υπολογιστικές εφαρμογές γενικού σκοπού. Το cloud computing προσφέρει πλεονεκτήματα όπως η μείωση του λειτουργικού κόστους, η ενοποίηση servers και η ελαστική τροφοδοσία πόρων. Εντούτοις, οι υπάρχουσες τεχνολογίες cloud computing και virtualization αντιμετωπίζουν δυσκολίες στην υποστήριξη σύγχρονων χαλαρών εφαρμογών πραγματικού χρόνου, όπως online video streaming και cloud based gaming. Αυτό συμβαίνει επειδή οι σύγχρονοι hypervisors δεν διαθέτουν τους μηχανισμούς που μπορούν να εγγυηθούν επιδόσεις πραγματικού χρόνου, για εφαρμογές που εκτελούνται σε Virtual Machines. Πολλές λύσεις υπο-χρησιμοποιούν την επεξεργαστική χωρητικότητα του host συσστήματος. Το paravirtualization εμφανίζεται ως μια ταιριαστή λύση, μιας και δίνει τη δυνατότητα στα guest συστήματα να επικοινωνούν τις, πραγματικού χρόνου, απαιτήσεις τους στο host σύστημα. Για τεχνικούς λόγους, όμως, οι λύσεις βασισμένες στο paravirtualization που έχουν, έως τώρα, υλοποιηθεί αποτυγχάνουν να ικανοποιήσουν τις προκύπτουσες αναγκαιότητες πραγματικού χρόνου. Στην εργασία αυτή, δίνουμε τη δική μας απάντηση στο ερώτημα της υλοποίησης ενός hypervisor ικανού να εκπληρώσει τις ανάγκες εφαρμογών πραγματικού χρόνου. Βασίζουμε την προσέγγισή μας στο KairosVM, μια λύση που χρησιμοποιεί πλήρες virtualization, υλοποιημένη πάνω στο KVM module. Διαφοροποιείται από κοινούς hypervisors πλήρους virtualization, με τη χρήση μιας τεχνικής που ονομάζεται Virtual Machine Introspection (ενδοσκόπηση), έτσι ώστε τα guest συστήματα να μπορούν άμεσα να ενημερώσουν το host σύστημα για τις πραγματικού χρόνου ανάγκες τους. Υλοποιούμε διαφορετικές πολιτικές κατανομής του φόρτου εργασίας των guest συστημάτων στους φυσικούς πόρους του host. Παρουσιάζουμε πειραματικά αποτελέσματα που προσφέρουν σύγκριση τόσο μεταξύ των διάφορων πολιτικών κατανομής όσο και ανάμεσα στον KairosVM hypervisor και έναν κοινό γνωστό hypervisor, τον VanillaKVM.

Λέξεις Κλειδιά: Εφαρμογές πραγματικού χρόνου, Εικονοποίηση, Υπολογιστικό νέφος, Χρησιμοποίηση CPU

Abstract

During the last years, cloud computing and virtualization technology have brought revolutionary changes in general-purpose computing applications. Cloud computing offers benefits such as reduction of operation costs, server consolidation, flexible system configuration and elastic resource provisioning. However, existing cloud computing and virtualization technology face huge difficulties in supporting contemporary soft real-time applications such as online video streaming and cloud based gaming. This is due to the fact that modern hypervisors lack the mechanisms in place to guarantee real-time prerformance of applications running on virtual machines. Many solutions under-utilize the processing capacity of the host system. Paravirtualization appears as a possible adequate solution, since it gives the guest systems the ability to communicate their real-time needs to the host system. Still, due to technical reasons, implemented paravirtualized solutions fail to fulfil the emerging real-time necessities. In this thesis, we give our own answer to the problem of implementing a hypervisor able to satisfy the needs of real-time applications. We base our approach on KairosVM, a fully-virtualized solution, implemented on the KVM module, which differentiates itself from common full-virtualized hypervisrors, by using a technic called Virtual Machine Introspection so that the guest systems can immidiately make the host aware of their real-time needs. We implement different policies for the allocation of the guest systems' workload into the host system's physical resources. We present experimental results that offer comparison both between the different allocation policies, and between the KairosVM hypervisor and an existing well-known hypervisor called VanillaKVM.

Keywords:Real-time events, Virtualization, Cloud Computing, CPU Utilization

Ευχαριστίες

Αρχικά θα ήθελα να ευχαριστήσω τον κ. Νεκτάριο Κοζύρη που μου επέτρεψε να εκπονήσω διπλωματική εργασία στο εργαστήριο αυτό. Ακόμα, επειδή, όταν υπήρξαν λειτουργικές δυσκολίες κατά τη διάρκεια της εργασίας, με στήριξε και ανταποκρίθηκε άμεσα, ενθαρρύνοντάς με να συνεχίσω.

Πολλές ευχαριστίες, ακόμα, οφείλω στον κ. Γκούμα που για οτιδήποτε χρειάστηκα στα πλαίσια της εργασίας ήταν πάντα εκεί. Με πληθώρα συναντήσεων, οι οποίες οργανώνονταν πάντα άμεσα, με τις διευκρινιστικές και ξεκάθαρες υποδείξεις του αποτελούσε μια μεγάλη βοήθεια. Χάρις σ' αυτόν η εργασία ολοκληρώθηκε ομαλά, αποκτώντας τη μορφή που έχει σήμερα.

Ιδιαίτερες ευχαριστίες οφείλω στον διδακτορικό φοιτητή Κωνσταντίνο Παπαζαφειρόπουλο, τον βοηθό μου σε όλη αυτή την πορεία. Η συνεχής πρόθεσή του να βοηθήσει, η άμεση καθημερινή μας επικοινωνία, η καλή διάθεση την οποία εξωτερίκευε καθώς και οι γνώσεις που μου μετέδωσε στα πλαίσια της εργασίας, αποτέλεσαν για μένα μια τεράστια ώθηση για την περάτωση της. Με τον Κωστή κατάλαβα πόσο μεγάλη σημασία έχει να συνεργάζεσαι με κάποιον που σε υποστηρίζει, συνεχώς, με κάθε τρόπο.

Για την στήριξή τους στην μακρά αυτή προσπάθεια των σπουδών μου και την αγάπη τους ευχαριστώ τους γονείς μου και τον αδερφό μου. Η πρόοδος στη ζωή μου και κάθε επιτυχία μου οφείλεται κατά ένα μεγάλο κομμάτι σ' αυτούς. Επίσης, τους φίλους μου που πάντα με υποστήριζαν και με υποστηρίζουν ψυχολογικά. Ευχαριστώ, ξεχωριστά, τον Θοδωρή και τον Ηλία που βοήθησαν, όταν τους χρειάστηκα, και πρακτικά στην ολοκλήρωση της εργασίας.

Ευχαριστώ, τέλος, τη Jana, για την πολύ μεγάλη ψυχολογική και κάθε τύπου υποστήριξή της, αλλά και την αστείρευτη υπομονή της, δυο παράγοντες που καθημερινά διατηρούσαν και ενίσχυαν την θέληση μου να συνεχίσω την προσπάθεια αυτή παρ' όλες τις αντιξοότητες που προέκυπταν.

Κωνσταντίνος Μεσσής

Περιεχόμενα

Κατάλογος σχημάτων

Κατάλογος πινάκων

1

Πρόλογος

Η ευρεία δυνατότητα, σήμερα, για διαδικτιακές συνδέσεις υψηλής ταχύτητας σε ανεκτό κόστος, όπως το DSL και άλλες σύγχρονες τεχνολογίες, σε συνδιασμό με την άμεση και εύκολη συνδεσιμότητα με ενσύρματες και ασύρματες τεχνολογίες, δημιουργεί μια μαζική μετατώπιση προς κατανεμημένα υπολογιστικά μοντέλα. Οι εφαρμογές οι οποίες βασίζονται μόνο σε υλικό και δεδομένα που υπάρχουν σε έναν μόνο προσωπικό υπολογιστή σταδιακά μειώνονται δίνοντας τη θέση τους σε μια καινούρια εποχή κατανεμημένων συστημάτων. Ένα πολύ ισχυρό παράδειγμα της κίνησης αυτής είναι τα συστήματα υπολογιστικού νέφους, στα οποία οι υπολογιστικοί πόροι νοικιάζονται από παρόχους είτε on-demand είτε pay-per-use.

Σαν μια τεράστια ενιαία δομή, τα κέντρα δεδομένων υπολογιστικού νέφους προσφέρουν υπηρεσίες δομής, υλικού και λογισμικού στους πελάτες τους. Οι εφαρμογές νέφους μπορούν να εκτελούν και να διαμοιράζονται τους υπολογισμούς και τα δεδομένα τους σε όσους κόμβους χρειαστεί και έχουν πρόσβαση σε τεράστιους όγκους δεδομένων, άμεσα διαθέσιμους εντός των κτιριακών εγκαταστάσεων των κέντρων δεδομένων του νέφους. Έτσι, το υπολογιστικό νέφος επιτρέπει τη νεα γενιά υπηρεσιών, με προσανατολισμό στο υψηλά κατανεμημένο on-line computing και στην πραγματοποίηση ενός νέου μοντέλου on-demand computing υψηλής απόδοσης προσβάσιμο από οποιονδήποτε, οπουδήποτε, οποτεδήποτε χρειαστεί.

Οι ανάγκες των χρήστων από τη δομή του νέφους που γίνονται όλο και περισσότερες, αποτελούν πάντα πρόκληση. Οι χρήστες χρειάζονται πλέον πρόσβαση όχι μόνο σε συστήματα online αποθήκευσης αλλά και σε διαδραστικές εφαρμογές και υπηρεσίες άμεσης απόκρισης. Επιπλέον, σε ένα περιβάλλον νέφους υψηλής υπολογιστικής απόδοσης οι εφαρμογές έχουν ισχυρότερες χρονικές απαιτήσεις. Έτσι, γίνοται κρίσιμα χαρακτηριστικά της απόδοσης, συμπεριλαμβανομένων εγγυήσεων πόρων και έγκαιρης εξαγωγής αποτελεσμάτων. Γίνεται λοιπόν εμφανής η ανάγκη εξυπηρέτησης εντός του νέφους, εφαρμογών και διαδικασιών πραγματικού χρόνου.

Καθώς όλο και περισσότερες κατηγορίες εφαρμογών μπαίνουν στο νέφος, οι εφαρμογές πραγματικού χρόνου θα μπορούσαν κι αυτές να ακολουθήσουν αυτό το ρεύμα λόγω των τεράστιων δυνατοτήτων και της αυξημένης χρησιμότητας που μια τέτοια επιλογή θα μπορούσε να παρέχει. Παράδειγμα αποτελούν τόσο χαλαρές όσο και αυστηρές εφαρμογές πραγματικού χρόνου, όπως συστήματα χειρισμού μηχανών, μη επανδρομένα οχήματα κατευθυνόμενα από αισθητήρες του νέφους, online video streaming αλλά και gaming εφαρμογές που απαιτούν response delay πολλές φορές μέχρι και 100 ms.

Η εξυπηρέτηση εφαρμογών πραγματικού χρόνου σε συστήματα υπολογιστικού νέφους είναι ένα σύνθετο πρόβλημα που για να ανιτμετωπίσει κανείς χρειάζεται, μεταξύ άλλων, να εστιάσει στην αποδοτική πρόσβαση στο φυσικό υλικό. Υπάρχουν αρκετοί εμπορικοί hypervisors για συστήματα πραγματικού χρόνου, όπως οι WindRiver, LynuxWorks και Real Time Systems GmbH. Παρ΄ όλ΄ αυτά φαντάζει απίθανο να συμπεριληφθεί κάποιος εξ αυτών εντός των γνωστών εμπορικών παρόχων υπηρεσιών νέφους λόγω της έλλειψης συγκεκριμένων εγγυήσεων απόδοσης πραγματικού χρόνου. Αυτός είναι και ο λόγος για τον οποίο, ενώ πολλές εφαρμογές όπως data mining και εφαρμογές τεχνητής νοημοσύνης έχουν μεταναστεύσει σε συστήματα νέφους, οι περισσότερες εφαρμογές πραγματικού χρόνου εκτελούνται ακόμα σε πλατφόρμες υλικού.

Η εργασία, προς την κατεύθυνση αυτή, προσπαθεί να επιτύχει κάποιες τέτοιες εγγυήσεις απόδοσης. Συγκεκριμένα, στην εργασία αυτή μελετάμε διαφορετικές τεχνικές κατανομής Virtual Machines στο υλικό του host συστήματος. Ειδικού σκοπού συστήματα πραγματικού χρόνου είναι εγκατεστημένα τόσο στα guest Virtual Machines όσο και στον host. Ασχολούμαστε με την ανάδειξη των κριτηρίων εκείνων που αποτελούν τα κατάλληλα ώστε να αποτελέσουν μέτρο για την σκιαγράφηση της κατανομής. Εξετάζουμε διαφορετικές πολιτικές κατανομής. Υλοποιούμε τα παραπάνω με μια userspace εφαρμογή στο host σύστημα. Αξιολογούμε πειραματικά το κατα πόσον η κάθε πολιτική πετυχαίνει εγγυήσεις απόδοσης που αποζητούμε δίνοντας έμφαση στο κόστος από πλευράς επεξεργαστικού υλικού. Συγκρίνουμε μεταξύ τους τις προσεγγίσεις συγκρίνοντας τα αποτελέσματά τους, τόσο κατα την ορθότητα, όσο και κατα την βέλτιστη χρήση των υλικών πόρων.

1.1 Δομή Εργασίας

Τα υπόλοιπα κεφάλαια της εργασίας οργανώνονται ως εξής:

- Το Κεφάλαιο 2 παρέχει το θεωρητικό υπόβαθρο πάνω στις έννοιες των συστημάτων και εφαρμογών πραγματικού χρόνου, του Virtualization και του Cloud Computing καθώς και των συστημάτων στα οποία βασίστηκε η εργασία αυτή.
- Το Κεφαλαιο 3 αναφέρει τα μοντέλα στα οποία η εργασία αναφέρεται και τις τυχόν υποθέσεις που έγιναν.
- Το Κεφαλαιο 4 περιέχει την περιγραφή της προσέγγισης μας καθώς και στοιχεία από την υλοποίησή της
- Το Κεφάλαιο 5 προσφέρει αποτελέσματα από τις πειραματικές αξιολογήσεις της υλοποίησής μας.
- Το Κεφάλαιο 6 αναφέρει σχετικές υλοποιήσεις σε επίπεδο έρευνας, πάνω στην εξυπηρέτηση εφαρμογών πραγματικού χρόνου σε συστήματα νέφους
- Το Κεφάλαιο 7 αποτελεί μια σύνοψη του αντικειμένου με το οποίο ασχολήθηκε η εργασία αλλά και των αποτελεσμάτων της.

2

Θεωρητικό Υπόβαθρο

2.1 Συστήματα Πραγματικού Χρόνου

Τα συστήματα πραγματικού χρόνου [1] είναι συστήματα τα οποία χαρακτηρίζονται από τρεις παράγοντες και τις επιδράσεις τους. Πρώτον ο χρόνος είναι ο βασικότερος και πιο σημαντικός παράγοντας για την οργάνωση και την λειτουργία τους. Στα συστήματα πραγματικού χρόνου, εργασίες πρέπει να ολοκληρώνονται πριν από αυστηρά καθορισμένες προθεσμίες (deadlines). Η ορθότητα μιας μέτρησης έγκειται όχι μόνο στην ορθότητα του αποτελέσματος αλλά και στην χρονική στιγμή στην οποία αυτό παρήχθη. Δεύτερον, η αξιοπιστία είναι ένα πολύ κρίσιμο χαρακτηριστικό μιας και η αποτυχία ενός συστήματος πραγματικού χρόνου, ανάλογα τη χρήση του, θα μπορούσε να έχει καταστροφικές επιπτώσεις. Για παράδειγμα, σε ένα σύστημα φορητού πλήρους ελέγχου επανδρομένου οχήματος το παραμικρό λάθος θα μπορούσε να σημαίνει την απώλεια ανθρωπίνων ζωών. Τρίτον, το περιβάλλον στο οποίο λειτουργεί ο υπολογιστής είναι καίριας σημασίας.

Μια εφαρμογή πραγματικού χρόνου συνήθως αποτελείται από μία ομάδα εργασιών που συνεργάζονται. Οι εργασίες αυτές κατά κύριο λόγο ενεργοποιούνται σε τακτικά διαστήματα και έχουν προθεσμίες εντός των οποίων πρέπει να ολοκληρώσουν την εκτέλεσή τους. Σε κάθε ενεργοποίησή της, μια εργασία εκτελεί μια υπολογιστική διαδικασία ενημερώνει το σύστημα και όσες άλλες εργασίες εξαρτόνται από το αποτέλεσμά της και εάν χρειάζεται στέλνει εντολές για την αλλαγή της κατάστασης του συστήματος.

Τέτοιου τύπου εργασίες, είναι γνωστές ως περιοδικές εργασίες. Ένα κοινό χαρακτηριστικό των περιοδικών εργασιών είναι ότι είναι χρονικά κρίσιμες (time-critical) με την έννοια ότι το σύστημα δεν μπορεί να λειτουργήσει αν δεν τις ολοκληρώσει εγκαίρως. Είναι, επομένως, πολύ σημαντικό το σύστημα πραγματικού χρόνου να εξασφαλίσει ότι οι προθεσμίες των εργασιών αυτών επιτυγχάνονται ανεξαρτήτως κάθε κατάστασης εντός του συστήματος.

Βεβαίως δεν καταφθάνουν όλες οι εργασίες ανά τακτά διαστήματα σε ένα σύστημα πραγματικού χρόνου. Κάποιες εργασίες (event-triggered) ενεργοποιούνται μόνο όταν συμβαίνουν συγκεκριμένα γεγονότα. Μιας και τα γεγονότα δεν καταφθάνουν σε συγκεκριμένα χρονικά σημεία, έτσι μη περιοδικά καταφθάνουν και οι εργασίες τις οποίες αυτά ενεργοποιούν. Εάν το γεγονός είναι χρονικά κρίσιμο τότε και η μη περιοδική εργασία που ενεργοποιείται έχει μια προθεσμία μέχρι την οποία πρέπει να ολοκληρώσει τη λειτουργία της. Αντίστοιχα αν το γεγονός δεν είναι χρονικά κρίσιμο, η προκύπτουσα διαδικασία δεν έχει κάποια προθεσμία, αλλά η εκτέλεσή της θα πρέπει να γίνει με τέτοιο τρόπο ώστε να μην τίθενται σε κίνδυνο οι προθεσμίες των άλλων χρονικά κρίσιμων εργασιών του συστήματος.

Ακολούθως, οι εφαρμογές πραγματικού χρόνου και επομένως και τα ίδια τα συστήματα πραγματικού χρόνου χωρίζονται σε αυστηρά (hard) και χαλαρά (soft). Στα αυστηρά συστήματα δεν επιτρέπονται υπολογισμοί εκτός των προθεσμιών μιας και αυτό θα αποτελέσει καταστροφικό σενάριο για το σύστημα. Με λίγα λόγια, είναι συστήματα στα οποία όλες οι εργασίες πρέπει να ολοκήρώνονται εγκαίρως. Στα χαλαρά συστήματα η χρησιμότητα των αποτελεμάτων που παρήχθησαν από μια εργασία μειώνεται όσο αυξάνεται ο χρόνος μετά τη λήξη της προθεσμίας. Το σύστημα διαθέτει μηχανισμούς για την αναφορά των εργασιών που δεν ολοκληρώθηκαν εντός της προθεσμίας καθώς και για την καταμέτρηση του χρονικού διαστήματος παραβίασης της.

2.1.1 Sched-Deadline

Ένα παράδειγμα συστήματος για χαλαρές εφαμογές πραγματικού χρόνου είναι το `Sched_Deadline` [2] μια Earliest Deadline First πολιτική δρομολόγησης πραγματικού χρόνου εντός του Linux Kernel. Ενσωματώθηκε στον Linux Kernel στην έκδοση 3.14.0. Δύναται να υποστηρίξει μόνο χαλαρές εφαρμογές πραγματικού χρόνου μιας και τα overheads του δρομολογητή των Linux είναι μη ντετερμινιστικά, ενώ αυστηρά συστήματα πραγματικού χρόνου απαιτούν αυστηρά περιορισμένα overheads.

Υλοποιείται ως μια κλάση δρομολόγησης εντός του δρομολογητή των Linux. Κάθε εφαρμογή πραγματικού χρόνου έχει τη δική της περίοδο, προθεσμία και χρόνο εκτέλεσης αποθηκευμένα σε ένα data structure. Περιέχει ακόμα τιμές για την απόλυτη προθεσμία και τον υπολοιπόμενο χρόνο εκτέλεσης της τρέχουσας εργασίας της εφαρμογής. Το `Sched_Deadline` αποθηκεύει την απόλυτη περίοδο κάθε εργασίας σε ένα red-black tree ταξινομημένο ως προς την προθεσμία από την κοντινότερη στην μεταγενέστερη ώστε η εργασία με την

νωρίτερη προθεσμία να βρίσκεται στον αριστερότερο κόμβο του δέντρου.

Αφού μια εργασία ολοκληρώσει την εκτέλεσή της αφαιρείται από την ουρά εκτέλεσης της CPU. Την ίδια στιγμή, η απόλυτη προθεσμία της αφαιρείται από το red-black tree .

2.1.2 ChronOS

Το ChronOS Linux [3] είναι ένα πραμγατικού χρόνου Linux-based λειτουργικό σύστημα το οποίο υλοποιεί διάφορους αλγορίθμους δρομολόγησης πραγματικού χρόνου καθώς και best-effort πολιτικές.

ChronOS Linux

```
int main() {
    struct timespec period, deadline, tmp;
    unsigned long prio = 90;
    period.tv_sec = 0;
    period.tv_nsec = 10000000;
    for (int i = 0; i < NUM_JOBS; i++) {
        gettimeofday(&tmp);
        deadline = timespec_add(&tmp, &period);

        /* Begin real-time segment; Pass in real-time
           parameters to scheduler */
        begin_rtseg_selfbasic(prio, &deadline, &period);

        /* ... Real-Time Code */

        /* End real-time segment; Let scheduler know it is
           free to schedule other tasks */
        end_rtseg_self(prio);

        gettimeofday(&tmp);
        tmp = timespec_sub(&deadline, &tmp);
        if (timespec_above_zero(&tmp))
            usleep(tmp.tv_nsec/1000);
    }
    return 0;
}
```

Σχήμα 2.1: Παράδειγμα εφαρμογής Πραγματικού χρόνου για ChronOS.

Σε αντίθεση με άλλες πολιτικές δρομολόγησης και με το Sched_Deadline ο δρομολογητής του ChronOS δεν υλοποιεί μια δική του επιπρόσθετη κλάση

δρομολόγησης. Συγκεκριμένα, το ChronOS επεκτείνει την κλάση δρομολόγησης πραγματικού χρόνου που υλοποιεί έναν δρομολογητή καθορισμένης προτεραιότητας. Η ουρά δρομολόγησης του ChronOS βρίσκεται σε μια καθορισμένη θέση προτεραιότητας n εντός της **rt-sched-class**. Διαφορετικές πολιτικές και αλγόριθμοι δρομολόγησης υλοποιούνται μέσω διάφορων kernel modules.

Οι εφαρμογές πραγματικού χρόνου που γράφονται για ChronOS Linux πρέπει να μαρκάρουν την αρχή και το τέλος του κώδικα ώστε κάθε εργασία πραγματικού χρόνου να αντιπροσωπεύεται με **begin-rtseg()** και **end-rtseg()** αντίστοιχα. Η **begin-rtseg()** system call χρειάζεται ως ορίσματα: μια σχετική περίοδο (που περιγράφεται ως ένα **struct timspec**), μια απόλυτη προθεσμία(deadline) (που περιγράφεται ως ένα timestamp με **struct timspec**) και μια προτεραιότητα (εκφραζόμενη με έναν ακέραιο αριθμό). Προεραιτικά ως όρισμα θα μπορούσε να δοθεί και ο χρόνος εκτέλεσης χειρότερης περίπτωσης σε μικρο-δευτερόπλεπτα (επίσης, εκφραζόμενος με έναν ακέραιο αριθμό). Η **end-rtseg()** system call χρησιμεύει στο να γνωρίζει ο δρομολογητής του ChronOS πότε μια εργασία ολοκλήρωσε την εκτέλεσή της, ώστε να δρομολογηθεί η επόμενη.

Το παραπάνω σχήμα δίνει ένα παράδειγμα κώδικα εφαρμογής πραγματικού χρόνου για ChronOS.

2.2 Cloud Computing & Virtualization

Το Cloud computing [4] είναι ένα μοντέλο που επιτρέπει την πανταχού παρούσα, βολική, κατ' απαίτηση (on demand) πρόσβαση σε δίκτυο σε πλήθος κοινόχρηστων υπολογιστικών πόρων (Π.χ. δίκτυα, διακομιστές, αποθηκευτικός χώρος, εφαρμογές και υπηρεσίες) που μπορεί να παρέχεται γρήγορα και να απελευθερώνεται με ελάχιστη προσπάθεια διαχείρισης ή αλληλεπίδραση παρόχου υπηρεσιών.

Το cloud computing και συγκεκριμένα η χρήση δημοσίων συστημάτων νέφους προσφέρει πλεονεκτήματα σε πολλούς τομείς, επιτρέποντας την συνένωση πολλών υπο-χρησιμοποιούμενων συστημάτων εντός ενός μικρότερου αριθμού server που τα φιλοξενούν (hosts). Ένας πάροχος νέφους μπορεί να οργανώσει φυσικούς πόρους με πολύ πιο πρακτικό τρόπο διαμοιράζοντάς τους σε μερικές εκατοντάδες ή χιλιάδες πελατών (ενοικιαστών) με δυναμική εναλλαγή των απαιτήσεων φόρτου εργασίας (workload) μέσω της βελτιστοποίησης ολόκληρης της δομής με έναν αυτοματοποιημένο (ή ημι-αυτοποιημένο) τρόπο οποτεδήποτε χρειαστεί, προσφέροντας έτσι, υψηλή διαθεσιμότητα και

αξιοπιστία.

Μια από τις σημαντικότερες λειτουργίες που επιτρέπουν τα παραπάνω είναι το virtualization [5] και πιο συγκεκριμένα το machine virtualization.

Το Machine virtualization [6] (λεγόμενο και processor virtualization) επιτρέπει μια μόνο μηχανή να προσομοιώσει την συμπεριφορά πολλών διαφορετικών μηχανών, έχοντας την δυνατότητα να φιλοξενήσει πολλά ετερογενή λειτουργικά συστήματα στο ίδιο υλικό. Τα λειτουργικά αυτά συστήματα ονομάζονται φιλοξενούμενα (guest) ενώ η μηχανή που τα εξυπηρετεί ονομάζεται οικοδεσπότης (host). Ονομάζουμε την δομή λογισμικού που εκτελείται (και έχει πλήρη έλεγχο) στον φυσικό αυτό οικοδεσπότη, η οποία είναι ικανή να εκτελέσει αυτή την προσομοίωση, Virtual Machine Monitor ή αλλιώς hypervisor.

Υπάρχουν διαφορετικές προσεγγίσεις στον τομέα της ανάπτυξης περιβαλλόντων εικονικής ετκέλεσης (virtual execution), ως προς την μέθοδο με την οποία αυτά υλοποιούν τις διάφορες διεπαφές που παρέχονται εντός της αρχιτεκτονικής ενός υπολογιστικού συστήματος, όπως την αρχιτεκτονική υλικού ή αρχιτεκτονική εντολών, την διεπαφή του λειτουργικού συστήματος, την διαδική διεπαφή εφαρμογών, ή την διεπαφή προγραμματισμού εφαρμογών. Η σχεδίαση των virtual environments προσεγγίζεται από δυο βασικές κατευθύνσεις τις hardware partitioning (διαχωρισμός υλικού) και hypervisor technology. Το hardware partitioning υποδιαιρεί την φυσική μηχανή σε ξεχωριστά τμήματα καθένα εκ των οποίων μπορεί να εκτελεί ένα διαφορετικό λειτουργικό σύστημα. Τέτοια τμήματα τυπικά δημιουργούνται με coarse μονάδες κατανομής, όπως ένας ολόκληρος επεξεργαστής.

2.2.1 Τύποι Hypervisors

Η σχεδίαση και υλοποίηση του hypervisor είναι πολύ σημαντική λόγω της άμεσης επιρροής της στις επιδόσεις των Virtual Machines, οι οποίες θα μπορούσαν να πλαισιώνουν αυτές του τοπικού υλικού. Οι Hypervisors ως τεχνολογία αποτελούν την πρώτη επιλογή για την για το system virtualization μιας και προσφέρουν υψηλή ελαστικότητα ως προς το πως ορίζονται και οργανώνονται οι εικονικοί πόροι. Όπως περιγράφεται στο [5], υπάρχουν δυο κύριοι τύποι Hypervisors: οι τύπου 1 (ή Bare Metal) και οι τύπου 2.

Οι Hypervisors τύπου 1 εκτελούνται απ᾽ ευθείας στο φυσικό υλικό. Εικονοποιούν τις κρίσιμες συσκευές υλικού προσφέροντας ορισμένα ανεξάρτητα και απομονωμένα τμήματα. Προσφέρουν επίσης υπηρεσίες διατμηματικού ελέγχου και επικοινωνίας. Αποτελούν μία πρακτική λύση για συστήματα πραγμα-

τικού χρόνου μιας και τα εικονικά περιβάλλοντα που τους χρησιμοποιούν είναι κοντά στο υλικό και μπορούν να χρησιμοποιήσουν απ' ευθείας πόρους υλικού.

Οι Hypervisros τύπου 2 εκτελούνται πάνω από ένα λειτουργικό σύστημα το οποίο συμπεριφέρεται ως host. Αποτελούν φιλοξενούμενους Hypervisors μιας και εκτελούνται μέσα σε ένα συμβατικό περιβάλλον λειτουργικού συστήματος. Το hypervisor στρώμα είναι τυπικά ένα διαφοροποιημένο επίπεδο λογισμικού πάνω από το host λειτουργικό σύστημα, το οποίο εκτελείται απ' ευθείας πάνω στο υλικό, και το guest λειτουργικό σύστημα εκτελείται σε ένα διαφορετικό ανώτερο επίπεδο.

2.2.2 Τεχνικές Virtualization

Οι διάφορες τεχνικές Virtualization χρησιμοποιούνται σε διαφορετικούς τομείς εργασίας κάθε μια προσφέροντας τα δικά της χαρακτηριστικά. Στην υποενότητα αυτή αναφερόμαστε περιληπτικά στις βασικές έννοιες της κάθε τεχνικής. Οι τεχνικές αυτές είναι οι Full Virtualization, Para-Virtualization, Hardware assisted Virtualization, Operating System level virtualization, Application level Virtualization και Network Virtualization.

Το Full Virtualization επιτρέπει την εκτέλεση μη τροποποιημένου guest λειτουργικού συστήματος με πλήρη προσομοίωση του υλικού πάνω από το οποίο έχει την εντύπωση ότι εκτελείται, όπως για παράδειγμα οι προσαρμογείς δικτύου και οι περιφεριακές τους συσκευές. Με τον τρόπο αυτό καθίσταται εύκολα δυνατή η εκτέλεση πολλαπλών λειτουργικών συστημάτων, ακόμα και ετερογενών, πάνω στο ίδιο υλικό. Ο hypervisor φροντίζει για την απαραίτητη προσωμοίωση δικτύου, ώστε τα Virtual Machines να μπορούν να επικοινωνήσουν με τον υπόλοιπο κόσμο και μεταξύ τους.

Το Kernel Based Virtual Machine [7] ή αλλιώς **KVM**, είναι ένα kernel module που μπορεί να ενσωματωθεί στο Linux σύστημα και μεσολαβεί στην πρόσβαση ανάμεσα στα Virtual Machines και το host σύστημα. Το KVM συνδιάζεται με το **QEMU** [8] ώστε να παρέχει προσωμοίωη την συσκευών υλικού. Ο συνδιασμός KVM/QEMU αποτελούν ένα παράδειγμα παροχής fully-virtualized περιβάλλοντος για τους guests.

Το Para-Virtualization είναι μια τεχνική κατά την οποία το guest λειτουργικό σύστημα τροποποιείται ώστε να γνωρίζει πως εκτελείται εντός ενός Virtual Machine. Αυτό βοηθά στην αποφυγή περιττής προσομοίωσης του virtualized υλικού. Αντ' αυτής ο τροποποιημένος πυρήνας και οι drivers του guest μπορούν να εκτελούν άμεσες κλήσεις στον hypervisor, γνωστές και ως hypercalls.

Ο ανοικτού κώδικα hypervisor **Xen** [9] είναι το ισχυρότερο παράδειγμα paravirtualized περιβάλλοντος.

Στο hardware assisted virtualization [10] το υλικό παρέχει επιπλέον λειτουργίες που επιταχύνουν την εκτέλεση των Virtual Machines. Για παράδειγμα, ένα επιπρόσθετο στρώμα μετάφρασης στο hardware από διευθύνσεις της virtualized μνήμης σε διευθύνσεις φυσικής μνήμης επιτρέπει στους μη τροποποιημένους guests να χειρίζονται τα page table τους χωρίς κάποιο trapping. Χάρις σ' αυτό τα Virtual Machines μπορούν ουσιαστικά να χειριστούν μόνο εικονικά page tables, ενώ τα πραγματικά page tables είναι υπό τον έλεγχο του hypervisor. Το στρώμα αυτό παρέχεται στην **VT-x** [11] τεχνολογία της Intel και στην **AMD-v** [12] της AMD. Αυτό ήταν μόνο ένα παράδειγμα επιπρόσθετης λειτουργίας. Στα πλαίσια της τεχνικής αυτής τα παραγείγματα τα οποία βρίσκουν ευρεία εμπορική εφαρμογή είναι πολυάριθμα.

Το Operating System Virtualization είναι μια τεχνική στην οποία ένα λειτουργικό σύστημα δίνει στο user-space λογισμικό την ψευδαίσθηση πολλαπλών λειτουργικών συστημάτων ή containers όπου το καθένα συμπεριφέρεται σαν ένα ανεξάρτητο σύστημα. Για παράδειγμα κάθε container λειτουργικού συστήματος έχει το δικό του χώρο process IDs, τη δική του μνήμη, εικονικές CPUs, σύστημα αρχείων και άλλα.

Τέλος, το network virtualization επιτρέπει την προσομοίωση ρυθμίσεων δικτύου σε λογισμικό. Έχει χρησιμότητα στο να επιτρέπει σε πολλά Virtual Machines να εκτελούνται στον ίδιο host, το κάθε ένα με τη δική του IP διεύθυνση συνδεδεμένα μαζί σε μια τοπολογία εικονικής γέφυρας (virtual bridge). Ένα ακόμα παράδειγμα είναι αυτό κατά το οποίο ένα σύνολο Virtual Machines αναπτυσσόμενο σε μια ομάδα πραγματικών hosts ανήκουν στο ίδιο υποδίκτυο ενώ οι hosts μπορούν να οργανώνονται με μια τελείως διαφορετική τοπολογία. Ενδεχομένως να μην ανήκουν καν στο ίδιο υποδίκτυο.

2.3 Virtual Machine Introspection - KairosVM

Η προσέγγιση στην οποία ένα Virtual Machine παρακολουθείται εξωτερικά με σκοπό την ανάλυση του λογισμικού που τρέχει σε αυτό ονομάζεται Virtual Machine Introspection (ενδοσκόπηση) και ξεκίνησε την εφαρμογή της εξυπηρετόντας σκοπούς ασφάλειας λειτουργικών συστημάτων, όπως στο Nitro [13], όπου κάθε syscall στους guests ελεγχόταν προκειμένου να εντοπιστούν κακόβουλες δραστηριότητες. Η ιδέα της χρήσης του Introspection σε virtualized συστήματα πραγματικού χρόνου αποτέλεσε αντικείμενο έρευνας, όμως οι λύ-

σεις που προτάθηκαν παρουσίαζαν ανεπίτρεπτα μεγάλο execution overhead, μιας και έλεγχαν όλα τα syscalls των guest συστημάτων και όχι μόνο αυτά που αφορούσαν εργασίες πραγματικού χρόνου.

Το **Kairos Introspection Engine** [14] δημιουργήθηκε το 2014 ως μια εναλλακτική μέθοδος έκθεσης των παραμέτρων πραγματικού χρόνου από τους guests στο λειτουργικό σύστημα του host. Υλοποιείται ως μια διαμόρφωση του KVM/QEMU και εκμεταλλεύεται ότι οι επεκτάσεις υλικού Intel-VT και AMD-V επιστρέφουν τον έλεγχο στον host όποτε μια εξαίρεση συμβαίνει στον guest.

Το Kairos Instrospection Engine λειτουργεί με την αναγραφή της μη ορισμένης x86 εντολής ud0 σε συγκεκριμένα σημεία στον χώρο διευθύνσεων του λειτουργικού συστήματος του guest. Η εισαγωγή της εντολής γίνεται με τη χρήση μιας διαμορφωμένης έκδοσης του QEMU. Εάν μια εντολή ud0 προστεθεί στις διευθύνσεις των συναρτήσεων πυρήνα του ChronOS, **begin-rtseg()** και **end-rtseg()** σε έναν ChronOS guest, οι επεκτάσεις υλικού θα δώσουν τον έλεγχο του επεξεργαστή στον host με την εκτέλεση των Virtual Machines να είναι σταματημένη στην αρχή των εντολών αυτών. Στο σημείο εκείνο, ο host εξάγει τα ορίσματα πραγματικού χρόνου από τους καταχωρητές του guest.

Γίνεται εμφανές πως η λύση αυτή δεδομένου ότι παρακολουθεί μόνο τα syscalls που σχετίζονται με τις εργασίες πραγματικού χρόνου παρουσιάζει πολύ μικρότερο overhead. Για το λόγο αυτό και αποτελεί μια πρακτική και αποτελεσματική υλοποίηση της χρήσης του Virtual Machine Introspection σε συστήματα πραγματικού χρόνου.

Όταν αντικαθίσταται το σύστημα διευθύνσεων του guest, το Kairos Introspection Engine σώζει την αρχική εντολή σε αυτό το σημείο ώστε ο guest να μπορεί ακόμα να εκτελέσει την εντολή μετά το "πιάσιμο" της εντολής ud0. Αυτό είναι σημαντικό για να εξασφαλισθεί πως οι guest συνεχίζουν να λειτουργούν χωρίς να έχουν υποστεί μεταβολές. Συμβάλλει, επίσης, στο να είναι αδύνατον οι επεκτάσεις υλικού να "πιάσουν" μη ορισμένες εντολές που δεν τοποθετήθηκαν από το Kairos αλλά από κάποιο άλλο κακόβουλο λογισμικό. Είναι κρίσιμο να μπορεί η διαφοροποιημένη έκδοση του KVM να ξεχωρίσει τη διαφορά μεταξύ εξαιρέσεων που προκάλεσε το Kairos και άλλων εξαιρέσεων. Αυτό εξασφαλίζεται με την αποθήκευση των διευθύνσεων που αντικαταστάθηκαν και εν συνεχεία με την σύγκριση του δείκτη εντολών (instruction pointer) του guest με αυτές τις αποθηκευμένες διευθύνσεις.

2.4 KairosVM- Full FLattening

Στο [15] επεκτείνεται η λειτουργία του KairosVM Introspection Engine. Η επέκταση είχε τόσο να κάνει με τον χειρισμό και την αναφορά στατιστικών όσο κυρίως με την χρήση του Introspection Engine για την βελτίωση δρομολόγησης από τον host των εργασιών που εκτελούνται στους guests.

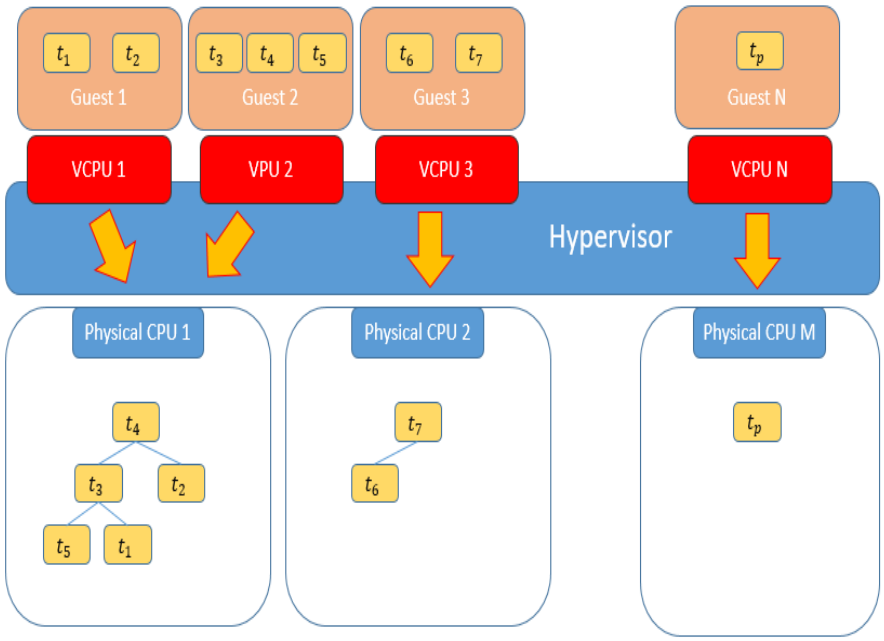
Αναφορικά με τον χειρισμό στατικών προστέθηκαν κάποιες συναρτήσεις που υπολογίζουν κάποιες ειδικού σκοπού μετρικές σχετιζόμενες άμεσα με συστήματα πραγματικού χρόνου. Μια βασική συνεισφορά ήταν πως η αναφορά και καταγραφή τους στον host γινότανε μέσω του directory /proc του host. Οι μετρικές σχετιζόμενες με τους χρόνους άφιξης, δρομολόγησης και διεκπαιρέωσης της κάθε εργασίας, γίνονται πλέον τόσο σε επίπεδο VM στο οποίο η εργασία εκτελείται, όσο και σε επίπεδο εφαρμογής στην οποία ανήκει η εργασία. Έτσι παρέχεται η δυνατότητα στον host να έχει μια πλήρη εικόνα για όλες τις εργασίες που εκτελούνται στους guests απ' όπου κι αν προέρχονται, όποια κι αν είναι τα χαρακτηριστικά τους.

Η κυριότερη συνεισφορά σχετίζεται με την αλλαγή μεθόδου δρομολόγησης από τον host των εργασιών όπως αυτές εκτελούνται στους διάφορους guests. Βασιζόμενος:

1. στην δυνατότητα του Kairos Introspection Engine σε συνδιασμό με τον σχεδιασμό των εφαρμογών πραγματικού χρόνου σε ChronOS, να δίνει τον χειρισμό στον host στην αρχή και στο τέλος κάθε νέας εργασίας πραγματικού χρόνου
2. στην γνώση ότι εργασίες πραγματικού χρόνου εξυπηρετούνται καλύτερα από τεχνικές δρομολόγησης νωρίτερης προθεσμίας (Earliest Deadline First), σε μη Virtualized συστήματα

ο συγγραφέας υλοποίησε ένα kernel module στο οποίο η πολιτική εξυπηρέτησης των εργασιών από τον host πλαισιώνει μια global Earliest Deadline First.

Υπό την έννοια αυτή ο host θα εξυπηρετεί πρώτα την εργασία της οποίας η προθεσμία λήγει νωρίτερα, από οποιοδήποτε Virtual Machine κι αν αυτή προέρχεται. Ουσιαστικά, ο δρομολογητής του host δε θα έχει καμία γνώση ότι οι εργασίες τις οποίες καλείται να δρομολογήσει προέρχονται από Virtual Machines σε Virtualized σύστημα, αλλά τις αντιμετωπίζει όπως θα αντιμετώπιζε τις ίδιες εργασίες αν αυτές εκτελούνταν στον ίδιο το host και η ορισμένη πολιτική δρομολόγησης τους θα ήταν Earliest Deadline First.



Σχήμα 2.2: KairosVM Full Flattening σε σύστημα πολλών πυρήνων.

Σχεδιαστικά, όταν μια νέα εργασία πραγματικού χρόνου καταφθάνει σε έναν guest τότε μέσω του interupt ο έλεγχος περνά στον host ο οποίος έχει αποκομίσει από τον guest τα χαρακτηριστικά πραγματικού χρόνου της εργασίας αυτής. Την στιγμή εκείνη ο host εισάγει την εργασία με τα χαρακτηριστικά της σε ένα red-black tree του οποίου ο αριστερότερος κόμβος αντιστοιχεί στην εργασία της οποίας η προθεσμία λήγει νωρίτερα, αντίστοιχα όπως και στο `Sched_Deadline`. Όταν ο δρομολογητής του host παύει να εκτελεί μια εργασία επιλέγει την επόμενη από τον αριστερότερο κόμβο του δέντρου αυτού. Η εργασία αυτή αφαιρείται από το δέντρο, το οποίο αναπροσαρμόζεται προκειμένου όσο πιο κοντά στην προθεσμία της είναι μια εργασία τόσο αριστερότερα να είναι ο αντίστοιχος κόμβος στο δέντρο.

Η δομή αυτή είναι αντίστοιχη για τον δρομολογητή κάθε CPU και έτσι έχουμε ένα red-black tree για κάθε physical CPU του host.

Το παραπάνω σχήμα απεικονίζει τη δομή αυτή. Οι VCPUs 1,2 εξυπηρετούνται από την physical CPU 1 ενώ η VCPU 3 από την physical CPU 2.

3

Μοντέλα και εικασίες

3.1 Μοντέλο εργασιών

Στη συγκεκριμένη εργασία βασιζόμαστε το μοντέλο εργασιών όπως αυτό ορίζεται στο [16]. Σύμφωνα με το μοντέλο αυτό μια εργασία πραγματικού χρόνου t_i χαρακτηρίζεται από συγκεκριμένη περίοδο T_i, προθεσμια (deadline) D_i, συγκεκριμένο χρόνο εκτέλεσης χειρότερης περίπτωσης C_i και χρόνο εκτέλεσης μέσης περίπτωσης A_i. Για την πειραματική αξιολόγηση, θεωρούμε την περίοδο κάθε εργασίας ίση με την προθεσμια της. Κάθε εργασία απελευθερώνει ένα κομμάτι της μια φορά κάθε περίοδο, το οποίο δεν θα υπερβεί ποτέ τον χρόνο εκτέλεσης χειρότερης περίπτωσης. Για μια εργασία t_i συμβολίζουμε το j-οστό της κομμάτι με t_i^j. Στο μοντέλο αυτό το σύνολο των εργασιών αποτελείται από ομογενείς εφαρμογές, ίδιων χρονικών χαρακτηριστικών. Εμείς, ξεφεύγουμε από τον περιορισμό αυτόν, επιτρέποντας, σε ορισμένες περιπτώσεις, την συνύπαρξη εφαρμογών με διαφορετικά χρονικά χαρακτηριστικά.

Name	Notation
Task	t_i
Job	t_i^j
Taskset	Γ
Period	T_i
Deadline	D_i
Worst Case Execution Time	C_i
WCET task Utilization	U_i^C
AvgET task Utilization	U_i^A
Overall Utilization	U
Guest Server	g_k
Number of Processors	m
Number of Tasks	n

Ένα σύνολο εργασιών συμβολίζεται με Γ. Η περίοδος ολόκληρου του συνόλου ορίζεται ως το ελάχιστο κοινό πολλαπλάσιο των περιόδων T_i για κάθε $t_i \in \Gamma$.

Ο παράγοντας χρησιμοποίησης (utilization) χειρότερης περίπτωσης U_i^C ορίζεται ως C_i/T_i και μέσης περίπτωσης U_i^A ως A_i/T_i. Ο γενικός παράγοντας για ένα σύνολο Γ ορίζεται ως $\sum t_i \in U_i$ τόσο για τη μέση όσο και για τη χειρότερη περίπτωση.

Ο παραπάνω πίνακας συγκεντρώνει τα χαρακτηριστικά του μοντέλου.

3.2 Μοντέλο Guests

Στην εργασία αυτή αναφερόμαστε σε μια ομάδα guest πραγματικού χρόνου με σύνολο εργασιών Γ, χωρισμένων σε μεμονωμένα σύνολα εργασιών g_k για κάθε guest. Οι guests είναι fully virtualized και λειτουργούν χωρίς κάποιου είδους προνομοιούχο πρόσβαση στο υλικό, αλλά την ίδια με κάθε άλλη user-space διεργασία σε Linux.

Κάθε guest εκτελεί το λειτουργικό σύστημα πραγματικού χρόνου ChronOS και εικάζεται ότι έχει έναν δρομολογητή τύπου Earliest Deadline First. Οι δρομολογητές των guest μένουν ανεπηρέαστοι τόσο από τον hypervisor όσο και από τους υπόλοιπους guests. Έτσι ένας guest θα δρομολογεί πάντα την ενεργό εργασία $t_i^j \in g_k$ η οποία έχει την ελάχιστη απόλυτη προθεσμία. Κάθε guest θεωρείται πως έχει μια μόνο vCPU.

3.3 Μοντέλο Υλικού

Όλα τα θεωρητικά και πειραματικά αποτελέσματα εικάζουν ότι ο host τρέχει σε ένα σύστημα ομογενούς x86-based αρχιτεκτονικής. Θεωρείται ότι ο host αποτελείται από έναν αριθμό m ίδιων ανεξάρτητων πυρήνων. Η εργασία δεν λαμβάνει υπόψιν τις επιδράσεις της cache στην απόδοση.

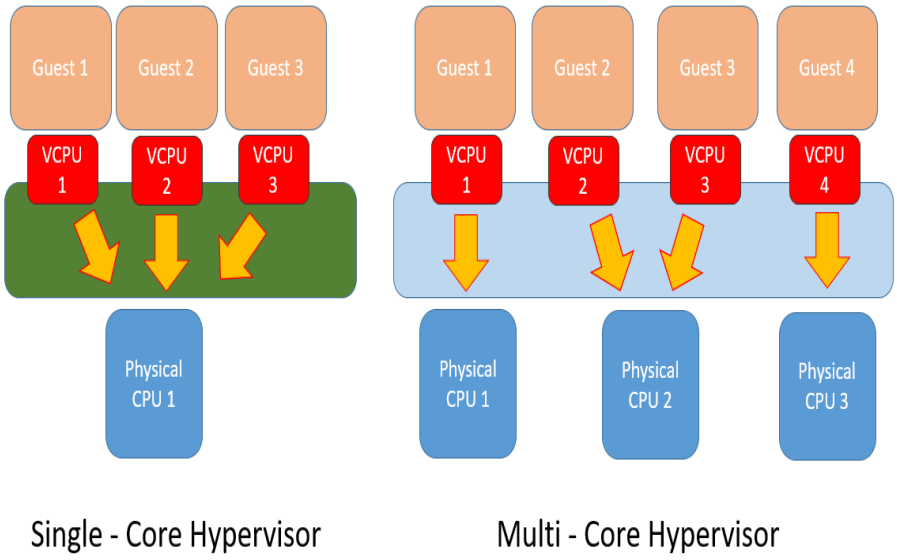
Στις μονοπυρηνικές μετρήσεις, ο host χρησιμοποιεί μόνο μια Physical CPU για την δρομολόγηση πολλών guests. Στις πολυεπεξεργασικές μετρήσεις, χρησιμοποιείται partitioned δρομολόγηση, κατά την οποία οι vCPUs δεν μπορούν να μεταναστεύσουν (migrate) από μια physical CPU σε μία άλλη, αφού ξεκινήσουν την εκτέλεση των εφαρμογών τους.

4

Σχεδίαση και υλοποίηση

4.1 Περιορισμοί παλαιότερων εργασιών

Τη βάση της εργασίας αυτή αποτέλεσε η [15]. Στην εργασία αυτή ο συγγραφέας χρησιμοποίησε την τεχνική της ενδοσκόπησης (introspection) που παρείχε το KairosVM module και το επέκτεινε με τέτοιο τρόπο ώστε να επιτυγχάνεται καλύτερη δρομολόγηση των εργασιών με έμφαση στην ολοκλήρωση των εργασιών εντός των προθεσμιών τους. Ολόκληρη η συνεισφορά της εργασίας περιγράφεται στο κεφάλαιο 2.



Σχήμα 4.1: Hypervisors μιας CPU και πολλών CPUs

Η εργασία περιορίστηκε σε θεωρητικό και πειραματικό επίπεδο στην χρήση του module σε host με ένα μόνο επεξεργαστή. Παρ' όλο που στο κομμάτι της θεωρίας υπήρχε αναφορά στην επέκταση σε πολυεπεξεργαστικά συστήματα,

τόσο η θεωρητική ενασχόληση ήταν ελάχιστη όσο και η πειραματαική αποτίμηση περιείχε μετρήσεις σε μια μόνο CPU.

Η δική μας εργασία, προκειμένου να πλησιάσει την ιδέα εκτέλεσης εργασιών πραγματικού χρόνου σε υπολογιστικό νέφος, έρχεται να επεκτήνει την αξιοποίηση του module σε σύστημα πολλών επεξεργαστών. Η προσπάθεια αυτή έχει ιδιαιτερότητες καθώς η έρευνα πλέον αποκτά μια ακόμα διάσταση πέραν αυτής της ορθής εκτέλεσης των εργασιών πραγματικού χρόνου, αυτή της βέλτιστης χρήσης υπολογιστικών πόρων.

Το παρακάτω σχήμα απεικονίζει χοντρικά την διαφορά στον τρόπο λειτουργίας ανάμεσα σε έναν hypervisor σε σύστημα ενός επεξεργαστή και σε έναν hypervisor σε σύστημα πολλών επεξεργαστών, που αποτέλεσε και την επέκταση της δικής μας εργασίας.

4.2 Περιγραφή του προβλήματος

Όπως προαναφέρθηκε η ορθή εκτέλεση εφαρμογών πραγματικού χρόνου αφορά την εύρεση ορθών αποτελεσμάτων και την ολοκλήρωσή τους πριν από αυστηρά καθορισμένες προθεσμίες. Σε ένα σύστημα όπου εκτελούνται ταυτόχρονα L αριθμός εργασιών πραγματικού χρόνου, η ορθή λειτουργία του συστήματος έγκειται στην εμπρόθεσμη ορθή ολοκλήρωση και των L εργασιών. Αυτό αφορά τόσο την διενέργεια ενός τέτοιου συστήματος τόσο σε πειραματικό επίπεδο, όπως στα πλαίσια της εργασίας, όσο και σε ένα πραγματικό σύστημα ενδεχομένως και σε ένα σύστημα νέφους. Για παράδειγμα σε εφαρμογές online streaming, εάν κάθε εργασία αντιστοιχεί σε ένα frame, τότε αν κάποιος αριθμός frame εκτελεστεί εκπρόθεσμα ο χρήστης ενδεχομένως να δει το frame καθυστερημένα ή και καθόλου σε περίπτωση που το video κολήσει. Προκύπτει λοιπόν η ανεύρεση μιας μετρικής η οποία θα μπορεί να εκφράσει την ορθή λειτουργία του όλου συστήματος. Στα πλαίσια της εργασίας χρησιμοποιούμε το Deadline Satisfaction Ratio (DSR) δηλαδή τον λόγο του αριθμού εργασιών που εκτελέστηκαν σωστά και εμπρόθεσμα προς το σύνολο των εργασιών αναφοράς. Οι συνολικές αυτές εργασίες θα μπορούσαν να είναι είτε όλες οι εκτελούμενς εργασίες, σε περίπτωση που θέλουμε να εξετάσουμε την ορθότητα ολόκληρου του συστήματος, είτε οι εργασίες μιας μόνο εφαρμογής, σε περίπτωση που θέλουμε να εξετάσουμε την ορθότητα εξυπηρέτησης της εφαρμογής αυτής από το σύστημα, είτε οποιεσδήποτε άλλες μπορούν να προκύψουν από τα κριτήρια που εμείς θέτουμε ως προς το τι θέλουμε να παρατηρήσουμε.

Σε ένα πολυεπεξεργαστικό σύστημα που προσπαθεί να εξυπηρετήσει αποδο-

τικά εφαρμογές πραγματικού χρόνου, και άρα να διατηρήσει ιδανικά το λόγο DSR στο 100%, προκύπτουν διαφορετικές μέθοδοι και πολιτικές κατανομής του όγκου εργασιών από τα Virtual machines στους επεξεργαστές του host ή στο σύνολο επεξεργαστών στις εγκαταστάσεις του νέφους, ανάλογα με το πως ο εκάστοτε διαχειριστής θέλει να αξιοποιήσει τους διαθέσιμους επεξεργαστικούς πόρους σύμφωνα με τις ανάγκες που θέλει να ικανοποιήσει. Οι δυο βασικές κατευθύνσεις είναι αυτές του partitioning δηλαδή της εξυπηρέτησης μιας ολόκληρης VCPU από μια και μόνο pCPU του host και του non-partitioning δηλαδή του διαμερισμού των εργασιών μιας VCPU σε πολλές διαφορετικές pCPUs. Σε κάθε μια από τις δυο προσεγγίσεις υπάρχει η δυνατότητα εφαρμογής μεταναστεύσεων εργασιών δηλαδή μετακίνησης κάποιων εργασιών από την ουρά εκτέλεσης μιας pCPU στην οποία είχαν αρχικά τοποθετηθεί στην ουρά μιας άλλης, ανάλογα με κάποια κριτήρια, τα οποία πάλι θέτει ο διαχειριστής του συστήματος. Εμείς ακολουθούμε την partitioning μέθοδο κατανομής VCPUs χωρίς μεταναστεύσεις.

Δεδομένου ότι στο μοντέλο μας τα Virtual Machines αποτελούνται από μόνο μια VCPU το πρόβλημα της ανάθεσης των VCPUs σε pCPUs ανάγεται σε πρόβλημα των Virtual Machines σε pCPUs. Βασισμένοι στην υπόθεση ότι ένα σύστημα νέφους θα προτιμούσε να εκτελεί όσο το δυνατόν περισσότερες εφαρμογές σπαταλώντας όσο το δυνατόν λιγότερη επεξεργαστική ισχύ, εστιάζουμε την έρευνα της εργασίας στην ορθή και εμπρόθεσμη εκτέλεση όσο το δυνατόν περισσότερων εργασιών χρησιμοποιώντας όσο το δυνατόν λιγότερες pCPUs στον host. Υπό την έννοια αυτή, αν L εργασιές πραγματικού χρόνου μπορούν να εκτελεστούν ορθά και εμπρόθεσμα από τουλάχιστον M pCPUs τότε το σύστημα θα πρέπει να χρησιμοποιεί M και όχι παραπάνω

Η επίλυση του προβλήματος αυτού εμπεριέχει την όσο το δυνατόν βέλτιστη επίλυση δυο υποπροβλημάτων,

- Της εύρεσης της κατάλληλης μετρικής η οποία να εκφράζει την δυνατότητα του επεξεργαστή να εξυπηρετήσει μια ομάδα εργασιών, η οποία θα αποτελέσει βάσει πρόβλεψης το κριτήριο επιλογής για το αν ένα νέο σύνολο εργασιών μπορεί να εκτελεστεί χωρίς απώλειες από έναν επεξεργαστή που είτε εκτελεί ήδη κάποιες εργασίες είτε δεν χρησιμοποιείται
- Της αναζήτησης και εφαμογής αλγορίθμων/πολιτικών κατανομής των VCPUs σε pCPUS έτσι ώστε να πλαισιώνεται όσο το δυνατόν περισσότερο ο ελάχιστος δυνατός αριθμός χρησιμοποιούμενων pCPUs.

Οι παρακάτω υποεννότητες αναλύουν ακριβώς ένα προς ένα τα υποπροβλήματα αυτά και την δική μας προσέγγιση για την επίλυσή τους.

pCPU	VCPU(s)	number of tasks
1	1 4	4n/3
2	2 5	n/3
3	3 6	n/3

Πίνακας 4.1: Round Robin Κατανομή.

pCPU	VCPU(s)	number of tasks
1	1	n
2	2 3 4 5 6	n
3		0

Πίνακας 4.2: Βέλτιστη Allocation.

4.3 Το utilization factor ως μετρική δυνατότητας εξυπηρέτησης

Διακρίνουμε το εξής σενάριο.

Ας υποθέσουμε πως δεν διαθέτουμε μια μονάδα μέτρησης που να προσδιορίζει κατά πόσον ένα σύνολο εργασιών μπορεί να εξυπηρετηθεί ορθά και εμπρόθεσμα από έναν επεξεργαστή. Έχουμε στη διάθεσή μας ένα μόνο είδος εργασιών με συγκεκριμένες τιμές στα χρονικά χαρακτηριστικά του, όπως αυτά ορίζονται στο Κεφάλαιο 3. Έστω ότι πειραματικά έχουμε εντοπίσει ότι μια CPU μπορεί να εκτελέσει μέχρι n επανεκτελέσεις της εργασίας αυτής χωρίς απώλειες (το DSR διατηρείται στο 100%) και πως αν ο αριθμός αυτός αυξηθεί παρατηρείται ανεπιθύμητη πτώση του ποσοστού επιτυχίας.

Έστω ότι στη διάθεσή μας έχουμε τρεις physical CPUs και επιλέγουμε το ποιά VCPU θα εξυπηρετηθεί από ποια pCPU, με μια Round Robin πολιτική, τέτοια ώστε η m-οστή VCPU να εξυπηρετηθεί από την m modulo n pCPU . Αν στο σύστημα μας θελήσουμε να εκτελέσουμε παράλληλα 6 VCPUs όπου το σύνολο εργασιών της κάθε μιας περιέχει [n,n/6,n/6,n/3,n/6,n/6] εργασίες, τότε σύμφωνα με τη Round Robin πολιτική στην CPU 1 θα εξυπηρετούσαμε τις VCPUs 1 και 3 με σύνολο 4n/3 εργασιών, στην CPU 2 τις VCPUs 2 και 5 με σύνολο n/3 εργασιών και στην CPU 3 τις VCPUs 3 και 6 με σύνολο n/3 εργασιών. Η παραπάνω διανομή συνόλων εργασίας θα είχε σαν αποτέλεσμα να χρησιμοποιήσουμε και τις 3 physical CPUs και παράλληλα κατά την εκτέλεση των εργασιών οι CPUs 2 και 3 δεν θα παρουσίαζαν απώλειες ενώ η CPU 1 θα παρουσίαζε απώλειες ενδεχομένως μεγάλες, αφού ο αριθμός εργασιών ανέρχεται στο 133% των εξυπηρετίσημων.

Αντ' αυτού αν κατά την διανομή η CPU 1 εξυπηρετούσε μόνο την VCPU 1 και η CPU 2 τις 2,3,4,5,6 τότε δεν θα είχαμε απώλειες σε καμία επεξεργαστική μο-

νάδα και ταυτόχρονα η CPU 3 θα έμενε αχρησιμοποίητη, δίνοντας την δυνατότητα να εξυπηρετηθεί μεγαλύτερος όγκος εργασιών κι από άλλες VCPUs που ενδεχομένως να χρειαστεί να εξυπηρετηθούν μεταγενέστερα. Οι παραπάνω πίνακες συνοψίζουν τις δυο προσεγγίσεις.

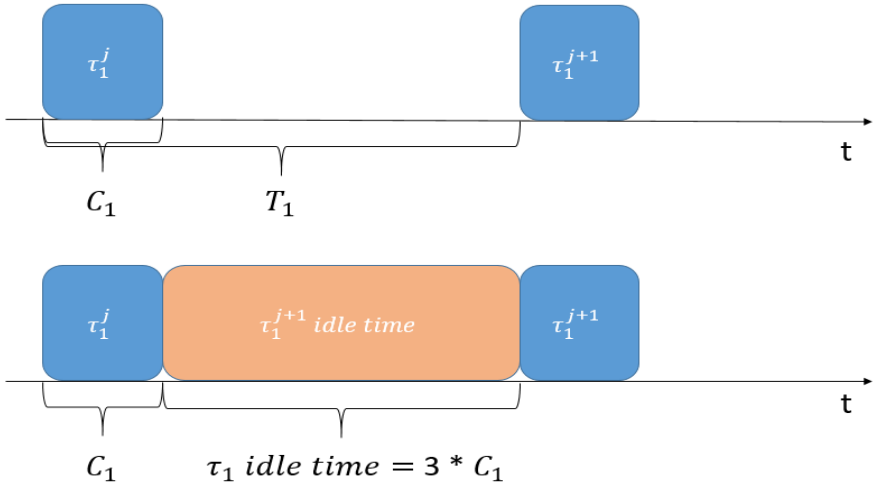
Μέσω αυτού του απλού και συγκεκριμένου παραδείγματος γίνεται εμφανής η εύρεση μιας πολιτικής τέτοιας που επιτρέπει βέλτιστες διανομές ώστε να ελαχιστοποιείται η χρησιμοποιούμενη επεξεργαστική ισχύς αλλά και να μην υπάρχουν απώλειες στην ορθή εκτέλεση των εργασιών.

Προκειμένου να μπορέσουμε να διαμορφώσουμε μια τέτοια πολιτική θα πρέπει να προσδιορίσουμε το κριτήριο με το οποίο αυτή θα μπορεί να διακρίνει κατά πόσον μια CPU δεδομένου συνόλου εργασιών που ήδη εκτελούνται σ αυτή δύναται να εξυπηρετήσει άλλη μια VCPU που δεν έχει συνδεθεί ακόμα με κάποια CPU. Στο παράδειγμά μας ίσως ήταν εμφανές το κατά πόσον ποιά CPU μπορεί να εξυπηρετήσει ικανοποιητικά ποια VCPU καθώς και η βέλτιστη διανομή, μιας και το μέγεθος δεδομένων ήταν πολύ μικρό και οι αριθμητικοί υπολογισμοί τετριμένοι. Σε ένα σύστημα υπολογιστικού νέφους όπου ο όγκος τόσο physical όσο και virtual CPUs είναι πολλές τάξεις μεγέθους μεγαλύτερος αλλά και τα χαρακτηριστικά των εκτελούμενων εργασιών θα είναι περισσότερο ανομοιογενή η διαδικασία διανομής πρέπει να εφαρμόζεται από αυτοματοποιημένους μηχανισμούς. Ακόμα πιο σημαντικό είναι το γεγονός ότι πρέπει να υπάρχει η δυνατότητα να αποφασίζεται η εξυπηρεσιμότητα ενός συνόλου εργασιών ανεξάρτητα από την εφαρμογή στην οποία αυτές ανήκουν. Το κριτήριο του αριθμού των εκτελούμενων εργασιών εξυπηρέτησε τις ανάγκες του παραδείγματος αλλά δεν μπορεί να αποτελέσει σε καμία περίπτωση τη μετρική μας σε συστήματα πραγματικού κόσμου. Την μετρική την οποία χρειαζόμαστε θα αποτελέσει το CPU Utilization.

Ορισμός: CPU utilization [17] είναι το ποσοστό χρόνου κατά τον οποίο μια εκτελούμενη διαδικασία αφήνει τον επεξεργαστή μη αδρανή.

Με λίγα λόγια, μια εργασία πού από τη στιγμή έναρξης της ως το πέρας της χρειάστηκε να εκτελείται εντός του επεξεργαστή το 1/4 του συνολικού αυτού χρονικού διαστήματος, ενώ κατά τα υπόλοιπα 3/4 ο επεξεργαστής ήταν αδρανής, έχει CPU utilization 1/4. Μια ακριβώς τέτοια εργασία απεικονίζεται στο παρακάτω σχήμα.

Η χρήση αυτής της μετρικής είναι αρκετά κατάλληλη και βολική τόσο για συστήματα εργασιών πραγματικού χρόνου που δρομολογούνται με πολιτική νωρίτερης προθεμσίας (earliest deadline first) όσο και για το δικό μας μοντέλο εργασιών όπως περιγράφεται στο Κεφάλαιο 3. Αναφορικά, αν μια εργασία έχει



Σχήμα 4.2: Παράδειγμα Utilization εργασίας

χρόνο εκτέλεσης χειρότερης περίπτωσης C_i και περίοδο $T_i >= C_i$ τότε στην χειρότερη περίπτωση όπου όλες οι επανεκτελέσεις της που καταφθάνουν εκτελούνται με χρόνο C_i, με μια ιδανική δρομολόγηση, T_i/C_i εργασίες θα εκτελούνταν χωρίς απώλειες. Ενδεχομένως στην περίπτωση που ο χρόνος εκτέλεσης χειρότερης περίπτωσης εμφανίζεται μόνο σε λίγες, ακραίες περιπτώσεις επανεκτελέσεων της εργασίας και είναι χρονικά πολύ μεγαλύτερος από αυτόν στη μέση περίπτωση, ο αριθμός των εργασιών που μπορύν να εκτελεστούν ορθά να πλαισιώνεται καλύτερα από τον λόγο T_i/A_i όπου A_i ο χρόνος εκτέλεσης μέσης περίπτωσης.

Σε κάθε περίπτωση οι παράγοντες U_i^C και U_i^A του μοντέλου εργασιών μας προσφέρουν την απαιτούμενη σαφήνεια και θα χρησιμοποιηθούν ως μετρικές εξυπηρεσιμότητας. Η καταλληλότητα των δυο παραγόντων εξετάζεται πειραματικά στην εργασία, τα αποτελέσματα υποδεικνύονται στο Κεφάλαιο 5.

4.4 Πολιτικές κατανομής

Με το παραπάνω παράδειγμα έγινε εμφανές ότι μια τυχαιοκρατική πολιτική κατανομής θα αποφέρει κακά αποτελέσματα τόσο από άποψης ορθής εκτέλεσης των εργασιών όσο και από άποψης καλής χρήσης των επεξεργαστικών πόρων. Στην υποεννότητα αυτή συσχετίζουμε το πρόβλημα της κατανομής VCPUs σε pCPUs με ένα αλγοριθμικά γνωστό πρόβλημα για πρακτικούς λόγους και εν συνεχεία παρουσιάζουμε τις τρεις πολιτικές κατανομής που αναπτύχθη-

καν και χρησιμοποιήθηκαν στα πλαίσια της εργασίας και εκμεταλεύονται τους παράγοντες U_i^C ή/και U_i^A προκειμένου να πάρουν τις κατάλληλες αποφάσεις.

4.4.1 Το Bin-Packing πρόβλημα

Το **Bin Packing** είναι το πρόβλημα κατά το οποίο [18] : Δοθέντος ενός συνόλου I από n αντικείμενα όπου κάθε αντικείμενο $i \in I$ έχει μέγεθος $s_i \in (0, 1]$ και ενός συνόλου m κουβάδων B, χωρητικότητας ένα ο κάθε ένας, ζητείται να βρεθεί η αντιστοίχιση $a : I \rightarrow B$ τέτοια ώστε ο αριθμός των μη κενών κουβάδων να είναι ελάχιστος. Το πρόβλημα έχει την ίδια δομή με το δικό μας υπό την έννοια ότι αν μια physical CPU θεωρηθεί ένας κουβάς χωρητικότητας ένα και κάθε vCPU ως ένα αντικείμενο μεγέθους ίσο με το άθροισμα των utilization factors των εργασιών που τρέχουν σ' αυτή τότε το I ταυτίζεται με το σύνολο των vCPUs και το B με αυτό των pCPUs. Η διαφορά είναι ότι το πρόβλημα με το οποίο ασχολούμαστε στην εργασία είναι το συμπληρωματικό του μιας και η δική μας αντιστοίχιση πρέπει να μεγιστοποιεί τον αριθμό των άδειων κουβάδων.

Το Bin Packing είναι ένα γωνστό NP-πλήρες πρόβλημα και άρα το συμπληρωματικό του είναι CoNP-πλήρες. Αυτό σημαίνει πως δεν υπάρχει βέλτιστος αλγόριθμος ως προς τον τρόπο αντιστοίχισης. Στην πραγματικότητα στο [19] αποδικνύεται ότι κανένας αλγόριθμος δεν εγγυάται ότι θα απέχει από τη βέλτιστη λύση λιγότερο από 22%.

Εντούτοις μιας και το πρόβλημα το οποίο μελετάμε εμφανίζεται συνεχώς και ιδίως σε συστήματα πραγματικού χρόνου έχουν υπάρξει πολλές διαφορετικές προσεγγίσεις ως προς το ποια πολιτική, ανάμεσα στις υπάρχουσες, δίνει τα καλύτερα αποτελέσματα. Στα πλαίσια της εργασίας εφαρμόσαμε τρεις εξ αυτών.

4.4.2 First-Fit

Η First-Fit πολιτική είναι αυτή κατα την οποία μια VCPU θα επιλεγεί να εξυπηρετηθεί από την πρώτη pCPU που μπορεί να την εξυπηρετήσει χωρίς απώλειες, ξεκινώντας από την πρώτη. Έτσι όταν καταφθάνει μια προς εξυπηρέτηση VCPU, εφόσον το άθροισμα του συνολικού της utilization και του συνολικού utilization των εργασιών που ήδη εξυπηρετούνται από την pCPU 1 είναι μικρότερο ίσο του 1 τότε θα εξυπηρετηθεί από αυτήν. Αν είναι μεγαλύτερο τότε ο έλεγχος θα συνεχιστεί με τη pCPU 2 κοκ.

Στο [20] αποδικνύεται ποιό είναι το άνω φράγμα utilization χειρότερης περίπτωσης για την πολιτική αυτή.

4.4.3 Best-Fit

Κατά την Best-Fit πολιτική μια VCPU θα επιλεγεί να εξυπηρετηθεί από την pCPU με το μεγαλύτερο έως τότε συνολικό utilization που μπορεί να την εξυπηρετήσει. Ουσιαστικά η VCPU προσπαθεί θα "τοποθετηθεί" στο μικρότερο κενό, στο οποίο "χωράει".

4.4.4 Worst-fit-rt

Σε αντίθεση με την Best-Fit στην Worst-Fit-rt πολιτική, η οποία αναπτύχθηκε για τα πλαίσια της εργασίας και δεν ενδείκνυται γενικά ως λύση, μια VCPU θα επιλεγεί να εξυπηρετηθεί από την pCPU με το μικρότερο έως τότε συνολικό utilization που μπορεί να την εξυπηρετήσει, με μια μικρή παραλλαγή. Μιας και σύμφωνα με αυτή την πολιτική μετά από n VCPUs θα είχαν χρησιμοποιηθεί n pCPUs πράγμα ακριβώς αντίθετο με τον στόχο της εργασίας, η VCPU θα επιλέξει με την παραπάνω πολιτική ανάμεσα στις pCPUs που ήδη χρησιμοποιούνται. Αν καμία από αυτές δεν μπορεί να την εξυπηρετήσει τότε θα επιλέξει να εξυπηρετηθεί από μια έως τότε μη χρησιποιούμενη pCPU, υπό την έννοια ότι δεν εκτελούνται σε αυτήν έως τότε εργασίες.

4.5 Υλοποίηση

Στην υποεννότητα αυτή επεξηγούμε ορισμένες τεχνικές λεπτομέρειες του πειράματος. Δεδομένου ότι η υλοποίησή μας χρησιμοποιεί μια τροποποιημένη έκδοση του KVM/QEMU οι guests αποτελούν Virtual Machines που εκτελούνται ως userspace εφαρμογές στον host. Καθώς εκκινούμε τα Virtual Machines το κάθε ένα από αυτά επιλέγεται να εκτελέσει κάποιου είδους φόρτο εργασίας. Προτού ξεκινήσει η εκτέλεση προσμετρούνται στον host οι παράγοντες utilization του κάθε guest.

Σύμφωνα με τους παράγοντες αυτούς και την πολιτική που έχει κάθε φορά επιλεχθεί γίνεται κατάλληλα η κατανομή των VCPUs σε pCPUs του host. Για την επίτευξη της κατανομής εφαρμόζουμε **cpusets** που εξασφαλίζουν ότι κάθε VCPU συνδεδεμένη με μια pCPU θα εκτελέσει όλες τις εργασίες της μόνο σε

αυτή, αλλά και ότι κανένα άλλου τύπου φόρτο εργασίας του συστήματος δεν θα εκτελεστεί στην pCPU, επηρεάζοντας τις επιδόσεις εξυπηρέτησης της συνδεδεμένης VCPU. Η κατανομή λαμβάνει χώρα ως μια userspace διαδικασία. Η επιλογή της πολιτικής κατανομής, ο αριθμός των guest προς εξυπηρέτηση καθώς και ποιές pCPUs του host καλούνται να εξυπηρετήσουν το φόρτο εργασίας των guest δηλώνονται ως παράμετροι πριν την εκκίνηση του πειράματος.

Κατά την υλοποίηση, δόθηκε έμφαση στο να είναι οι χρονικές πολυπλοκότητες χειρότερης περίπτωσης, για την κατανομή μιας VCPU σε μία pCPU, όλων των πολιτικών κατανομής ίσες. Αυτό προκειμένου να εξασφαλίσουμε όσο το δυνατόν περισσότερο, ότι η διαδικασία της κατανομής, ανεξαρτήτως της εκάστοτε επιλεγμένης πολιτικής, δεν θα επηρεάσει την ορθότητα των αποτελεσμάτων, η οποία και είναι άμεσα συνδεδεμένη με το χρόνο.

Έστω ότι N ο αριθμός των pCPUs που θα χρησιμοποιηθούν και M ο αριθμός των VCPUs προς εξυπηρέτηση. Θεωρούμε μια λίστα από VCPUs προς ευπηρέτηση και τα utilization τους και μια αντίστοιχη λίστα από pCPUs και τα utilization του συνολικού φόρτου εργασίας που προς το παρόν κάθε μια καλείται να εξυπηρετήσει (με αρχικές τιμές 0). Στην First-Fit πολιτική για κάθε VCPU ελέγχουμε αν μπορεί να εξυπηρετηθεί από την πρώτη pCPU και αν όχι συνεχίζουμε τον έλεγχο για όλες τις N pCPUs. Βάσει αυτού, προκύπτει μια πολυπλοκότητα O(N) στη χειρότερη περίπτωση, για την εξυπηρέτηση μιας VCPU. Αυτή η πολυπλοκότητα είναι αναπόφευκτη μιας και από ορισμού ο First-Fit αλγόριθμος, στα πλαίσια του bin-packing προβλήματος, υπονοεί την απόπειρα ταιριάσματος κάθε αντικειμένου με κάθε κουβά εφόσον αυτή απέτυχε με τους προηγούμενους κουβάδες.

Για την υλοποίηση της Best-Fit πολιτικής διατηρούμε μια λίστα των pCPUs που έχουν μη μηδενικό φόρτο εργασίας, ταξινομημένη με φθίνουσα σειρά ως προς τα utilizations. Η λίστα αυτή είναι αρχικά κενή. Όταν κατανεμηθεί η πρώτη κατά σειρά VCPU η αντίστοιχη pCPU εισάγεται στη λίστα σε O(1). Για κάθε επόμενη VCPU ελέγχεται αν αυτή μπορεί να εξυπηρετηθεί από την pCPU που βρίσκεται στην αρχή της λίστας, και αν όχι από την εκάστοτε επόμενη. Ο έλεγχος αυτός χρειάζεται O(N) στη χειρότερη περίπτωση. Αν καμία από τις pCPUs της λίστας δεν μπορεί να την εξυπηρετήσει, η VCPU εξυπηρετείται από μια αχρησιμοποίητη, εφόσον υπάρχει, η οποία εισάγεται στη λίστα με Insertion Sort, ώστε να διατηρηθεί η ταξινόμηση, σε O(N). Αν κάποια από τις pCPUs της λίστας μπορεί να την εξυπηρετήσει τότε αυξάνουμε το utilization της και διατηρούμε την ταξινόμηση της λίστας αλλάζοντας με διαδοχικές συγκρίσεις την θέση του κόμβου που αντιστοιχεί στην pCPU σε O(N). Τελικά, για την εξυπηρέτηση μιας VCPU χρειαζόμαστε στη χειρότερη περίπτωση O(N) για την εύρεση της pCPU και O(N) για την διατήρηση της ταξινόμησης στη λίστα. Συ-

νολικά O(N).

Η υλοποίηση για την Worst-Fit-rt είναι ακριβώς αντίστοιχη με την Best-fit με τη διαφορά ότι η λίστα είναι ταξινομημένη σε αύξουσα σειρά και πως ο έλεγχος για το αν η VCPU μπορεί να εξυπηρετηθεί από τις pCPUs τις λίστας χρειάζεται να γίνει μόνο με τον αρχικό της κόμβο. Η συνολική χρονική πολυπλοκότητα χειρότερης περίπτωσης προκύπτει αντίστοιχα O(N).

Τονίζεται, πως το utilization threshold που μπορεί ορθά να εξυπηρετήσει μια pCPU ορίζεται από εμάς ως παράμετρος και πως σε περίπτωση που μια VCPU δεν μπορεί να εξυπηρετηθεί από καμία pCPU του συστήματος βάσει αυτού του threshold, τότε αυτό αυξάνεται κατά μια μικρή τιμή και η διαδικασία κατανομής επαναλαμβάνεται επαναληπτικά μέχρι κάθε VCPU να εξυπηρετείται από μια pCPU. Σε περίπτωση που το φόρτο εργασίας σε μια pCPU είναι μη εξυπηρετήσιμο αυτό αποτυπώνεται στα αποτελέσματα.

Η υλοποίηση αποτελεί ένα σύνολο python και bash scripts.

5

Πειρματική αξιολόγηση

Για την εκπόνηση των πειραμάτων χρησιμοποιήθηκε ένας Intel server με 4GB RAM και έναν τετραπήρυνο Intel Core2 στα 2.67 GHz. Για την εφαρμογή partitioned δρομολόγησης ώστε μια vCPU να εκτελείται μόνο σε μια physical CPU χωρίς μεταναστεύσεις χρησιμοποιήσαμε cpusets.

5.1 Επεξήγηση της πειραματικής υλοποίησης

Στα πειράματα ως εφαρμογές προς εκτέλεση στους guests χρησιμοποιήσαμε πολλαπλές εκτελέσεις πέντε εφαρμογών.

1. x264: Μια εφαρμογή συμπίεσης video. [21]
2. mplayer: Η γνωστή εφαρμογή αναπαραγογής αρχείων video και ταινιών.
3. disparity: Μια εφαρμογή εντοπισμού κίνησης.
4. blackscholes: Εφαρμογή για υπολογισμό, αναλυτικά, τιμών σύμφωνα με την διαφορική εξίσωση Black-Scholes, δοθέντων των κατάλληλων δεδομένων εισόδου. Κομμάτι του Parsec Benchmark Suite [22].
5. bitcnts: Εφαρμογή για τον υπολογισμό των bit δοσμένου δεκαδικού αριθμου με διάφορες μεθόδους. Κομμάτι των MiBench [23].

Όλες οι εφαρμογές είναι γραμμένες σε C κώδικα. Προκειμένου να ταιριάζουν στο μοντέλο όπως αυτό περιγράφεται στο Κεφάλαιο 3, οι εφαρμογές έχουν τροποποιηθεί κατάλληλα ώστε η εκτέλεση κάθε μιας να διαχωρίζεται σε μικρότερα μέρη, ταυτόσημα των εργασιών t_i. Στην αρχή της και στο τέλος της η κάθε εργασία μαρκάρεται με τις εντολές **begin-rt**() και **end-rt**(), του ChronOS, ώστε να λαμβάνει χώρα η ενδοσκόπηση (introspection).

Στους guest μετριούνται οι χρόνοι αρχής και τέλους εκτέλεσης της κάθε εργασίας με τη χρήση της **gettimeofday**(), οι οποίοι αποθηκεύονται σε **timespec**

structs προκειμένου κατά την ενδοσκόπηση να γνωστοποιηθούν στον host. Έτσι υπολογίζονται οι χρόνοι εκτέλεσης μέσης και χειρότερης περίπτωσης καθώς και γίνεται η διαπίστωση αν η εργασία εκτελέστηκε πριν την προθεσμία της επιτυγχάνοντας στο πείραμα. Οι χρόνοι μέσης και χειρότερης περίπτωσης υπολογίστηκαν αναλυτικά για όλες τις εφαρμογές στον Intel server που χρησιμοποιήθηκε για τα πειράματα.

Η τιμή της περιόδου με την οποία επναλαμβάνεται μια εργασία t_i καθώς και της προθεσμίας της, που δίνονται ως ορίσματα της begin-rtseg() και για το δικό μας πείραμα θεωρούμε ίσες, προκαθορίζονται από εμάς.

Κατά την εκτέλεση του πειράματος τα στατιστικά του, τα οποία αφορούν

1. τον αριθμό των εργασιών που εκτελούνται ως σύνολο αλλά και ανά guest ή ανά εφαρμογή
2. τις τιμές των χαρακτηριστικών της κάθε εργασίας όπως αυτά περιγράφονται στο Κεφάλαιο 3 και
3. τις επιτυχείς εκτελέσεις εργασιών εντός των προθεσμιών,

αναγράφονται στην διεύθυνση /proc του host.

Η υλοποίηση της αναφοράς των στατιστικών αυτών υιοθετήθηκε απ' το [15].

Η επικοινωνία με τους guests για την ανάθεση φόρτου εργασίας έγινε μέσω secure shell. Για το λόγο αυτό έγινε χρήση της πλατφόρμας εικονικού δικτύου virsh.

5.2 Μονοεπεξεργαστικές Αξιολογήσεις

Στις παρακάτω ομάδες αξιολογήσεων το σύνολο εργασιών αποτελείται από επανεκτελέσεις των παραπάνω εφαρμογών, που καταφθάνουν προς εξυπηρέτηση, από διαφορετικό αριθμό guest Virual Machines. Οι εργασίες ισοκατανέμονται στα VMs τα οποία όλα εξυπηρετούνται από μια και μόνο physical CPU. Το πείραμα εφαρμόζεται τόσο με το KairosVM-Full Flattening module όσο και με το πρότυπο VanillaKVM module.

Σκοπός των πειραμάτων είναι τόσο να καταλήξουμε σε συμπεράσματα σχετικά με την γενική συμπεριφορά του KairosVM όσο και να να συγκρίνουμε τους δυο αλγορίθμους δρομολόγησης ως προς την δυνατότητα τους να εξυπηρετήσουν εμπρόθεσμα τις εργασίες.

MPlayer	
A_i	0.05 ms
C_i	0.6 ms
T_i	2.5 ms , 5 ms

Πίνακας 5.1: Χαρακτηριστικά MPlayer

x264	
A_i	10.5 ms
C_i	52 ms
T_i	120 ms, 180 ms, 240 ms, 360 ms

Πίνακας 5.2: Χαρακτηριστικά x264

Disparity	
A_i	132 ms
C_i	150 ms
T_i	2000 ms

Πίνακας 5.3: Χαρακτηριστικά Disparity

Blackscholes	
A_i	13 ms
C_i	16 ms
T_i	50 ms

Πίνακας 5.4: Χαρακτηριστικά blackscholes

Προκειμένου να αποκτήσουμε μια πλήρη εικόνα σχετικά με την δυνατότητα των hypervisors να εξυπηρετούν ορθά εφαρμογές πραγματικού χρόνου, χρησιμοποιούμε εφαρμογές με πολύ μικρή περίοδο (της τάξης των 5ms) αλλά και εφαρμογές με μεγαλύτερη περίοδο (από 50 ms έως και 2 sec). Οι εφαρμογές αυτές παρουσιάζουν ακραίες τιμές χρόνου εκτέλεσης, κατά την εκκίνησή τους, πολύ μεγαλύτερες του μετρημένου μέσου χρόνου εκτέλεσης. Για το λόγο αυτό επιλέγουμε τον παράγοντα μέσης χρησιμοποίησης αντί του παράγοντα χρησιμοποίησης χειρότερης περίπτωσης για την μέτρηση του όγκου του φόρτου εργασίας.

Στις γραφικές παραστάσεις που ακολουθούν ο x-άξονας αντιπροσωπεύει τον παράγοντα μέσης χρησιμοποίησης και ο y-άξονας το ποσοστό των εργασιών που δρομολογήθηκαν και εκτελέστηκαν εντός της προθεσμίας τους και άρα το ποσοστό επιτυχίας του πειράματος.

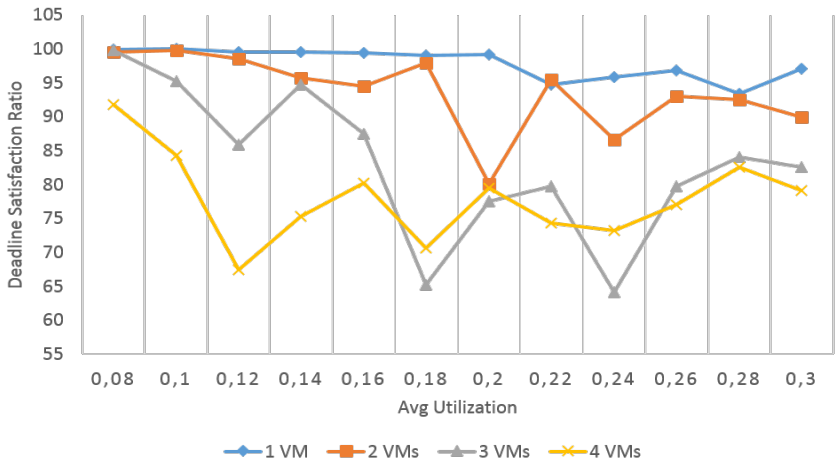
Οι εφαρμογές που χρησιμοποιήθηκαν μαζί με τα χαρακτηριστικά τους, δίνονται στους παραπάνω πίνακες.

Η εφαρμογή bitcnts δεν μαρκάρεται με τις εντολές εργασίας πραγματικού χρόνου του ChronOS ώστε να μην αντιμετωπιστεί από το σύστημα ως εργασία πραγματικού χρόνου. Η υλοποίηση αυτή συμβαίνει προκειμένου να προσομοιωθεί καλύτερα ένα σενάριο πραγματικού κόσμου στο οποίο εργασίες πραγματικού και μη πραγματικού χρόνου λαμβάνουν χώρα ταυτόχρονα.

5.2.1 Εφαρμογές με πολύ μικρές περιόδους

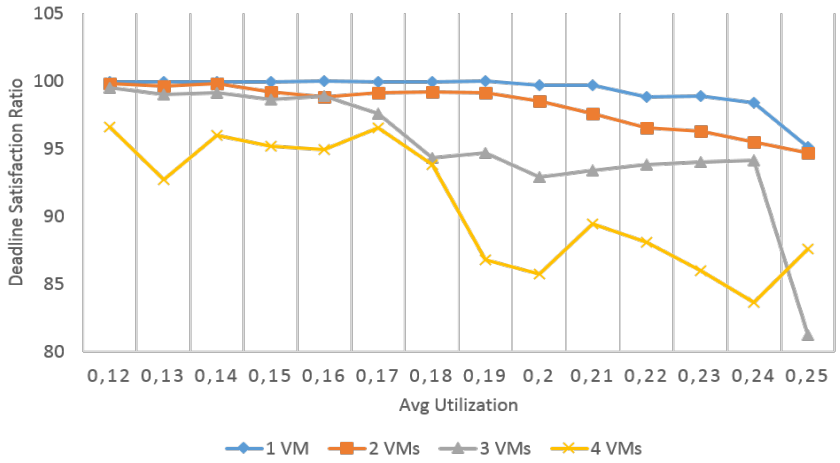
Στη υποεννότητα αυτή παρουσιάζουμε πειραματικές αποτιμήσεις από την εκτέλεση της, ειδικά διαμορφωμένης, παραλλαγής της εφαρμογής MPlayer ώστε, να αντιμετωπίζεται από το σύστημα ως εφαρμογή πραγματικού χρόνου. Για την εφαρμογή αυτή το κάθε t_i^j, με A_i 0.05 ms αντιστοιχεί στην επεξεργασία ενός frame του video χωρίς την προβολή του. Λόγω του πάρα πολύ χαμηλού χρόνου εκτέλεσής της, η εφαρμογή επιλέχθηκε να έχει αντίστοιχα πολύ μικρή περίοδο, προκειμένου να μπορούμε να επιτύχουμε λογικές τιμές παράγοντα χρησιμοποίησης με όχι πολύ μεγάλο αριθμό επανεκτελέσεων.

Στην παρακάτω γραφική παράσταση απεικονίζουμε την εκτέλεση της εφαρμογής με περίδο 2.5 ms. Στην εκάστοτε διαφορετική υποπερίπτωση ο όγκος εργασιών καταφθάνει από διαφορετικό αριθμό guests (1 έως 4).



Σχήμα 5.1: MPlayer με περίοδο 2.5 ms

Από την γραφική παράσταση γίνεται εμφανές πως σε περίπτωση που το φόρτο εργασίας καταφθάνει από περισσότερο από έναν guest, το DSR πέφτει σε τιμές κάτω του 95% σε τιμές παράγοντα χρησιμοποίησης της τάξης του 0,25. Ένα ακόμα χρήσιμο συμπέρασμα είναι πως τα αποτελέσματα δεν παρουσιάζουν ομοιομορφία στην συμπεριφορά ως προς τον αριθμό των guests ή τον όγκο φόρτου εργασίας, αλλά παρατηρούνται ασταθείς τιμές.



Σχήμα 5.2: MPlayer με περίοδο 5 ms

Αντίστοιχη είναι η εικόνα όταν η εφαρμογή εκτελείται με περίοδο 5 ms. Παρ' όλο που, στην περίπτωση αυτή, τα αποτελέσματα επηρεάζονται εμφανώς από των αριθμό των guests και της μέσης χρησιμοποίησης, για πάνω από 2 guests το DSR πηγαίνει σε χαμηλές τιμές, για όχι και τόσο μεγάλες αυτής.

Αυτές οι μη αποδεκτές συμπεριφορές παρατηρούνται ακριβώς λόγω της πολύ χαμηλής τιμής της περιόδου ενός t_i^j. Ο αλγόριθμος δρομολόγησης του KairosVM καθώς και τα πολλά context switches λόγω του Introspection, παρουσιάζουν overheads που επιφέρουν χρονικές επιβαρύνσεις ίδιας τάξης μεγέθους τιμών με αυτές της περιόδου και του χρόνου εκτέλεσης. Για χαμηλότερες τιμές της περιόδου, μάλιστα, η εκτέλεση του πειράματος ήταν αδύνατη.

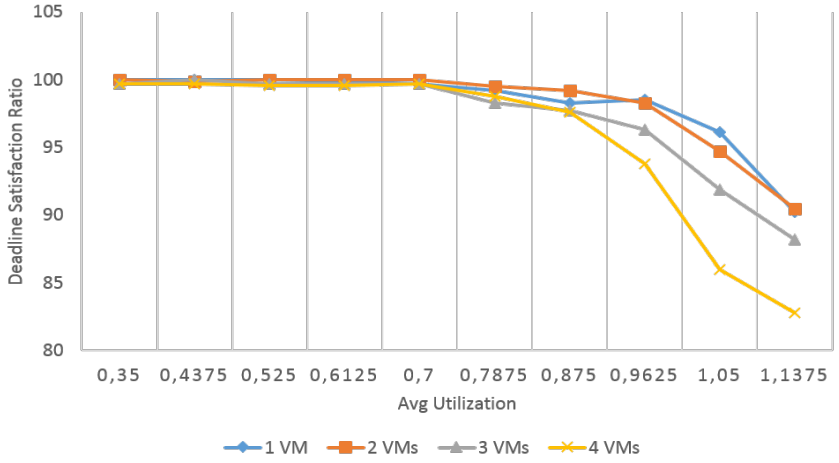
5.2.2 Εφαρμογές με μεγαλύτερες περιόδους

Τα χαρακτηριστικά των υπολοίπων εφαρμογών επιτρέπουν την ρύθμιση της περιόδου σε μεγαλύτερες τιμές.

Στην παρακάτω γραφική παράσταση απεικονίζεται το ίδιο σενάριο, εξυπηρέτησης φόρτου εργασίας από έναν έως τέσσερεις guests.

Η συμπεριφορά του συστήματος είναι ξεκάθαρα επηρεασμένη από τον αριθμό των guests. Για μεγαλύτερο αριθμό guests, μειώνεται πιο εύκολα το DSR για αντίστοιχο όγκου εργασιών. Επίπλέον, οι επιδόσεις ως προς την χρησιμοποί-

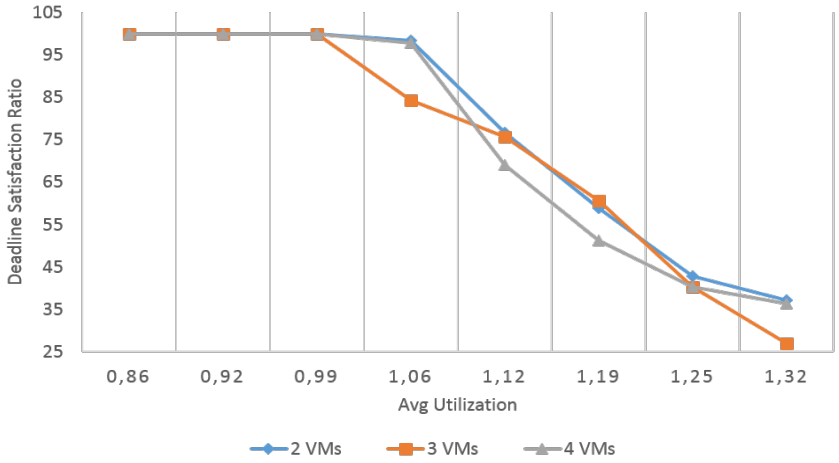
ηση είναι φανερά βελτιωμένες με το DSR να πέφτει σε τιμές κάτω του 95% σε όλες τις περιπτώσεις για τιμές χρησιμοποίησης άνω του 0,8.



Σχήμα 5.3: x264 με περίοδο 120 ms

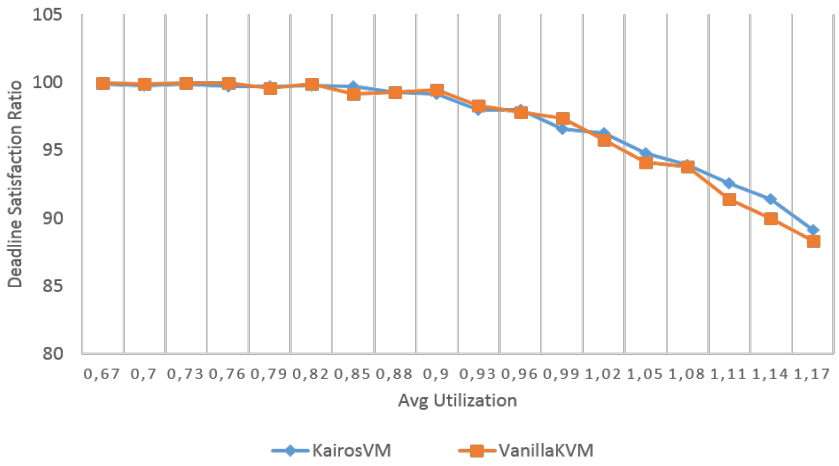
Η δυνατότητα του KairosVM hypervisor να εξυπηρετεί αποδοτικά συνολικό όγκο φόρτου εργασίας με παράγοντα χρησιμοποίησης κοντά στο 1 βελτιώνεται ακόμα όσο η περίοος της εφαρμογής μεγαλώνει. Αυτό φαίνεται από την παρακάτω γραφική παράσταση που απεικονίζει την εξυπηρέτηση συνόλου εργασιών από επανεκτελέσεις της disparity με περίοδο 2000 ms, που καταφθάνει από δυο έως τέσσερεις guests.

Οι τιμές του DSR μένουν σε πολύ ικανοποιητικά επίπεδα ακόμα και για τιμές χρησιμοποίησης ελάχιστα μεγαλύτερες του 1. Στην γραφική παράσταση αυτή γίνεται ακόμα εμφανές πως η απόδοση του συστήματος καταρρέει για τιμές χρησιμοποίησης κοντά στο 1,3.



Σχήμα 5.4: Disparity με περίοδο 2000 ms

Τέλος προκειμένου να συγκρίνουμε την απόδοση του KairosVM hypervisor με αυτή του VanillaKVM στην παρακάτω γραφική παράσταση δίνονται τα αποτελέσματα απόδοσης και των δυο, σε σύνολο εργασιών που καταφθάνει από τέσσερεις guests και αποτελείται από επανεκτελέσεις της x264 με περίοδο 360 ms, που είναι αντιπροσωπευτική της περιόδου μιας εργασίας σε ένα αληθινό σύστημα πραγματικού χρόνου.

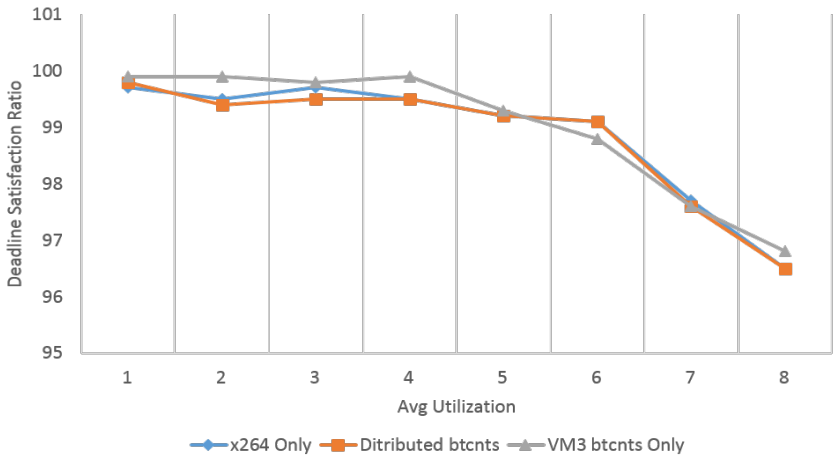


Σχήμα 5.5: Σύγκριση KairosVM, Vanilla KVM

Οι δυο hypervisors δείχνουν να δίνουν παρόμοια αποτελέσματα, με το DSR του KairosVM να υποχωρεί με λίγο μικρότερους ρυθμούς για τιμές χρησιμοποίησης μεγαλύτερες του 1. Σε τιμές χρησιμοποίησης μικρότερες του 1 τα αποτελέσματα είναι πολύ κοντά και δεν οδηγούν σε κάποιο βέβαιο συμπέρασμα.

5.2.3 Συνδιασμός με εφαρμογές μη πραγματικού χρόνου

Είναι σημαντικό να αξιολογήσουμε την συμπεριφορά του KairosVM hypervisor κατά την εξυπηρέτηση ενός συνόλου εργασιών που συνδιάζει εφαρμογές πραγματικού και μη πραγματικου χρόνου. Η παρακάτω γραφική παράσταση απεικονίζει την εξυπηρέτηση πολλαπλών επανεκτελέσεων της x264 με περίοδο 180 ms παράλληλα με επανεκτελέσεις της btcnts.



Σχήμα 5.6: Παράλληλη Εκτέλεση με εργασίες μη Πραγματικού χρόνου

Στην αξιολόγηση αυτή το φόρτο εργασίας καταφθάνει από 4 guest Virtual Machines. Η μπλε γραμμή αντιστοιχεί σε επανεκτελέσεις μόνο της x264 για τις διάφορες τιμές του Average Utilization. Η κόκκινη γραμμή αφορά την εξυπηρέτηση ακριβώς του ίδιου όγκου εργασιών πραγματικού χρόνου σε συνδιασμό με 4 επανεκτελέσεις της btcnts οι οποίες είναι ομοιόμορφα κατανεμημένες στους guests. Στην περίπτωση της γκρι γραμμής στο VM 3 εκτελούνται μόνο 4 επανεκτελέσεις της btcnts ενώ οι υπόλοιποι 3 guests εκτελούν μόνο εργασίες πραγματικού χρόνου, αντίστοιχου Utilization με τις προηγούμενες περιπτώσεις.

Όπως γίνεται εμφανές, οι διαφορές στο DSR είναι μηδαμινές. Ειδικά στην τρίτη περίπτωση οι επιδόσεις φαίνονται στοιχειωδώς καλύτερες, πλησιάζοντας αυτές που πετυχαίνει το σύστημα κατά την εξυπηρέτηση τριών μόνο guests. Συνεπώς, η ορθή και εμπρόθεσμη εκτέλεση των εργασιών πραγματικού χρόνου μένει ανεπηρέαστη από το επιπλέον φόρτο εργασιών μη πραγματικού χρόνου. Χάρις στο Instrospection Engine, μόνο οι εφαρμογές πραγματικού χρόνου εισάγονται στο δέντρο εξυπηρέτησης εργασιών της pCPU, απ' όπου κι αν καταφθάνουν. Μετά την εκτέλεση όλων των t_i^j εξυπηρετούνται και οι εφαρμογές μη πραγματικού χρόνου. Ουσιαστικά, η εκτέλεση δεν συνέβη παράλληλα, αλλά σειριακά κι έτσι το αποτέλεσμα είναι ίδιο με το να υπήρχαν, μόνο, εφαρμογές πραγματικού χρόνου.

Σημειώνουμε ότι το επιπρόσθετο φόρτο εργασίας επανεκτελέσεων της btcnts δεν διαφοροποιεί την τιμή του utilization το οποίο αφορά, μόνο, τις επανεκτελέσεις της x264.

5.3 Πολυεπεξεργαστικές Αξιολογήσεις

Στην εννότητα αυτή επεκτείνουμε το πείραμα, ώστε η εξυπηρέτηση των VCPUs να γίνεται από, έως και 3 pCPUs του συστήματος μας. Για όλες τις μετρήσεις που ακολουθούν, το σύνολο εργασιών αποτελείται, από επανεκτελέσεις συνδιασμού των εφαρμογών x264, blackscholes, disparity για τις διάφορες τιμές των περιόδων τους. Οι εφαρμογές αυτές επιλέχθησαν λόγω της ομοιογένειας που παρουσιάζουν ως προς τη δυνατότητα του συστήματος να τις εξυπηρετεί, με παρόμοιες τιμές DSR, για αντίστοιχο όγκο εργασιών εκφραζόμενο σε Average Utilization.

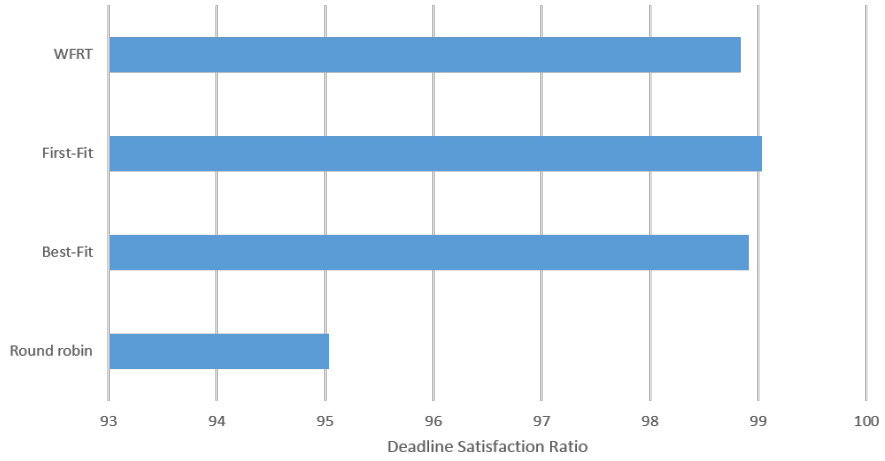
Στην πρώτη υποεννότητα συγκρίνουμε τις επιδόσεις της κάθε πολιτικής κατανομής όπως καταγράφονται τόσο ως προς το DSR και όσο και ως προς τον αριθμό των pCPUs που η κάθε μια επιλέγει να χρησιμοποιήσει. Στην δεύτερη υποεννότητα συγκρίνουμε τις επιδόσεις του KairosVM hypervisor με αυτές του VanillaKVM πλέον σε πολυεπεξεργαστικό σύστημα.

5.3.1 Σύγκριση Πολιτικών Κατανομής

Για την διενέργεια των πειραμάτων οι εφαρμογές εκτελούνται σε 6 guest Virtual Machines. Για την καταγραφή των μετρήσεων δημιουργήσαμε 50 σενάρια μέσης χρησιμοποίησης από 1,4 έως 3,4. Για το κάθε σενάριο η κατανομή VC-

PUs σε pCPUs έγινε και με τις τρεις σχεδιασμένες πολιτικές κατανομής καθώς και με Round-Robin πολιτική.

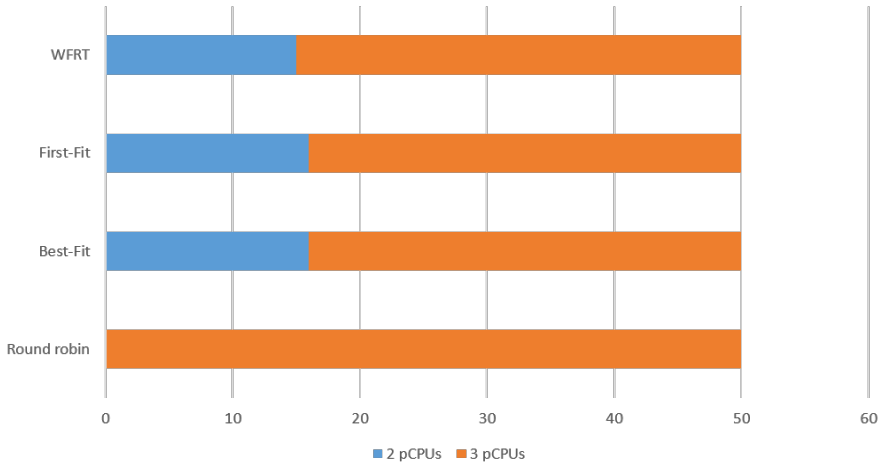
Στην πρώτη γραφική παράσταση απεικονίζεται το μέσο DSR που επιτεύχθηκε από κάθε πολιτική κατανομής για το σύνολο των σεναρίων.



Σχήμα 5.7: Μέσο Deadline Satisfaction Ratio για κάθε πολιτική

Η δεύτερη γραφική παράσταση εκφράζει το ποσοστό επί του συνόλου των σεναρίων κατά το οποίο η κάθε κατανομή χρησιμοποίησε 2 ή 3 pCPUs. Αυτό, προκειμένου να συγκρίνουμε τις πολιτικές ως προς την δυνατότητά τους να χρησιμοποιούν τον μικρότερο δυνατό αριθμό pCPUs με τον οποίο είναι εφικτό οι τιμές του DSR να παραμείνουν ικανοποιητικές.

Από τα αποτελέσματα καταλήγουμε πως και οι τρεις πολιτικές εξυπηρετούν εμπρόθεσμα με μεγαλύτερο ποσοστό επιτυχίας, από την Round-Robin τις εφαρμογές πραγματικού χρόνου, ακόμα και αν χρησιμοποιούν μικρότερο αριθμό pCPUs σε αρκετά σενάρια.

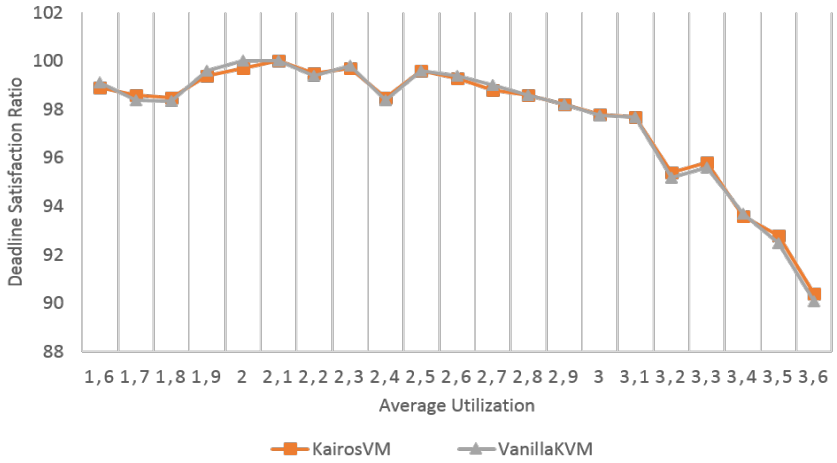


Σχήμα 5.8: Αριθμός pCPUs που χρησιμοποίησε κάθε πολιτική

Παρ' όλο που είναι, μέσω των αποτελεσμάτων, εμφανής η χρησιμότητα των τριών σχεδιασμένων πολιτικών έναντι της Round-Robin, μεταξύ τους οι πολιτικές παρουσιάζουν ελάχιστες αποστάσεις και στις δυο γραφικές παραστάσεις. Έτσι δεν μπορούμε με σιγουριά, να υποστηρίξουμε πως κάποια ξεχωρίζει ως προς τις επιδόσεις της από κάποια άλλη. Αυτό συμβαίνει επειδή ο αριθμός των pCPUs του συστήματος αλλά και των guests είναι αρκετά μικρός, με αποτέλεσμα στην πληθώρα των σεναρίων οι διάφορες πολιτικές, που κατά βάση τους έχουν "fit" κριτήρια επιλογής, να δίνουν την ίδια κατανομή. Ο περιορισμός αυτός, προκύπτει λόγω των δυνατοτήτων υλικού του συστήματος, που χρησιμοποιήθηκε για τις μετρήσεις. Σε ένα αληθινό σύστημα υπολογιστικού νέφους όπου ο αριθμός των επεξεργαστικών μονάδων αλλά και ο αριθμός των guests θα ήταν πολύ μεγαλύτερος, οι εναλλακτικές επιλογές, για την κάθε κατανομή, θα ήταν πολύ περισσότερες και η διαφορά τους θα γινόταν έμπρακτα εμφανέστερη.

5.3.2 Σύγκριση KairosVM με VanillaKVM

Στο πείραμα αυτό εφαρμόζοντας αποκλειστικά First-Fit πολιτική κατανομής, επιχειρούμε να εξυπηρετήσουμε ένα σύνολο εφαρμογών πραγματικού χρόνου και με τους δυο hypervisors. Το φόρτο εργασίας καταφθάνει από 6 guest Virtual Machines. Συγκρίνουμε τους δυο hypervisors ως προς τις επιδόσεις DSR που προκύπτουν.



Σχήμα 5.9: Σύγκριση KairosVM με VanillaKVM σε σύστημα τριών επεξεργαστών

Για την ερμηνεία της γραφικής παράστασης είναι σημαντικό να αναφέρουμε πως το αποδεκτό threshold χρησιμοποίησης για κάθε επεξεργαστή είναι ρυθμισμένο στο 0,9. Έτσι κατα κύριο λόγο, τα σύνολα φόρτου εργασίας με χρησιμοποίηση κάτω του 1,8 εξυπηρετούνται από 2 επεξεργαστές και εκείνα με χρησιμοποίηση άνω του 1,8 με τρεις.

Παρατηρούμε πως η εικόνα είναι αρκετά αντίστοιχη με την μονοεπεξεργαστική σύγκριση των δυο hypervisors στο σχήμα 5.5. Για χρησιμοποίηση κάτω, αλλά κοντά στο 1,8 όπου η μέση χρησιμοποίηση ανά επεξεργαστή είναι κοντά στο 0,9 οι τιμές του DSR απομακρίνονται από το 100 με τον KairosVM hypervisor να εμφανίζει ελάχιστα καλύτερες τιμές. Σε τιμές χρησιμοποίησης, δε, που το σύστημα διαμοιράζει το φόρτο εργασίας και στον τρίτο επεξεργαστή με χαμηλότερη μέση τιμή χρησιμοποίησης ανά επεξεργαστική μονάδα, οι τιμές DSR πλησιάζουν πάλι το 100 και ανάμεσα στα αποτελέσματα των 2 hypervisors δείχνει να μην παρατηρείται ιδιαίτερη διαφορά. Τέλος, για τιμές χρησιμοποίησης κοντά και μεγαλύτερες του 2,7 όπου πλέον η μέση χρησιμοποίηση ανά επεξεργαστή φτάνει σε τιμές 1,2, το DSR μειώνεται σημαντικά με τα αποτελέσματα του KairosVM hypervisor να δείχνουν λίγο καλύτερα από εκείνα του VanillaKVM.

6

Σχετικές Υλοποιήσεις

Όπως αναλυτικά περιγράφηκε στα προηγούμενα κεφάλαια στην εργασία αυτή προσπαθήσαμε να δώσουμε την δική μας λύση στο πρόβλημα της δρομολόγησης εργασιών πραγματικού χρόνου βασισμένοι σε hypervisor που χρησιμοποιεί τις ιδιότητες που παρέχει το KairosVM module, σε συνδιασμό με μια τύπου global EDF πολιτική δρομολόγησης και υλοποιεί First-Fit, Best-Fit πολιτικές, καθώς και μια ακόμη μη ενδεδειγμένη πολιτική φόρτου εργασίας από τα guest Virtual Machines στις CPUs του host μηχανήματος. Στο κεφάλαιο αυτό αναφερόμαστε σε άλλες γνωστές προσεγγίσεις για την επίλυση του προβλήματος αυτού που διαφέρουν από τη δική μας είτε ως προς τη χρησιμοποιούμενη τεχνολογία στο host μηχάνημα, είτε ως προς την τεχνική virtualization, είτε ως προς τον τρόπο κατανομής του φόρτου εργασίας στους υπολογιστικούς πόρους του host μηχανήματος.

6.1 IRMOS real-time scheduler

Προκειμένου να μπορέσουν να ικανοποιηθούν ορισμένες εγγυήσεις δρομολόγησης σε μεμονομένα Virtual Machines που δρομολογούνται εντός του ίδιου συστήματος, στα πλαίσια του IRMOS[24] project, δημιουργήθηκε ένας deadline-based δρομολογητής πραγματικού χρόνου. Παρέχει προσωρινή απομόνωση μεταξύ πολλαπλών πιθανώς πολύπλοκων κομματιών λογισμικού, όπως ολόκληρα Virtual Machines (χρησιμοποιώντας το KVM ως hypervisor, ένα VM τρέχει ως μια Linux διεργασία). Χρησιμοποιεί μια διαφοροποιημένη έκδοση του Constant Bandwidth Server αλγόριθμου [25], βασισμένη σε πολιτική νωρίτερης προθεσμίας, για να εξασφαλίσει ότι κάθε ομάδα διεργασιών/νημάτων δρομολογείται στις διαθέσιμες CPUs για ένα συγκεκριμένο χρονικό διάστημα κάθε μια προκαθορισμένη περίοδο, ειδική για κάθε Virtual Machine. Επιπλέον ο δρομολογητής του IRMOS μπορεί να απομονώσει προσωρινά ένα Virtual Machine από επιπρόσθετο φόρτο εργασίας που προκύπτει από την οργάνωση του νέφους, όπως για παράδειγμα φόρτο εργασίας για την παρακολούθηση και

τον έλεγχο του συστήματος νέφους, για την αρχικοποίηση των Virtual Machines ή για την μετανάστευση των Virtual Machines από ένα host σε έναν άλλο. Η προσέγγιση αυτή έχει χρησιμοποιθεί και αξιολογηθεί σε multimedia υπηρεσίες νέφους, συμπεριλαμβανομένων e-learning και επεξεργασίας βίντεο υψηλής απόδοσης με χρήση στην παραγωγή ταινιών.

6.2 RT-Xen

Το RT-Xen project ανέπτυξε ένα πλαίσιο δρομολόγησης Virtual Machines πραγματικού χρόνου στον Xen hypervisor. Ο RT-Xen δρομολογητής[26], γεφυρώνει το κενό μεταξύ της θεωρίας δρομολόγησης πραγματικού χρόνου και της Xen τεχνολογίας μέσω της δρομολόγησης Virtual Machines με τη χρήση αλγορίθμων προκαθορισμένης προτεραιότητας σχεδιαμσένων βάσει αυτής της θεωρίας. Ο δρομολογητής των Virtual Machines στον hypervisor και οι δρομολογητές των guest λειτουργικών συστημάτων διαμορφώνουν μια ιεραρχία δρομολόγησης.

Ο αρχικός δρομολογητής RT-Xen 1.0 έχει υποστεί 2 επαυξήσεις. Ο RT-Xen 1.1 υποστηρίζει συνθετική δρομολόγηση πραγματικού χρόνου όπου η ζήτηση πόρων από τις εργασίες των Virtual Machines παρουσιάζεται στην διεπαφή υπολογιστικών πόρων των Virtual Machines. Εάν η διεπαφή αυτή μένει ικανοποιημένη από τις παροχές του δρομολογητή του hypervisor το λειτουργικό σύστημα του guest επιτρέπει την δρομολόγηση των εργασιών. Η έκδοση RT-Xen 2.0 [27] αποτελεί έναν πολυπύρηνο δρομολογητή πραγματικού χρόνου, με μια μεγάλη ποικιλία ρυθμίσιμων χαρακτηριστικών όπως global και partitioned δρομολογητές, σχήματα στατικής και δυναμικής προτεραιότητας και διαφορετικούς αλγορίθμους κατανομής φόρτου εργασίας στους servers.

Ένα χαρακτηριστικό στοιχείο της αρχιτεκτονικής Xen είναι ότι βασίζεται στο στρώμα διαχείρισης προκειμένου να επεξεργαστεί πακέτα ανάμεσα στους guests. Σαν αποτέλεσμα τόσο ο guest δρομολογητής όσο και το στρώμα διαχείρισης μπορούν να επηρεάσουν το latency επικοινωνίας και να επιφέρουν αντιστροφή προτεραιοτήτων. Ένας δρομολογητής πραγματικού χρόνου ενός guest δεν μπορεί να αποτρέψει τέτοιου τύπου εναλλαγή προτεραιότητας στην διαστρωματική επικοινωνία. Για να ανταποκριθεί στο πρόβλημα αυτό, το RT-Xen παρέχει μια αρχιτεκτονική επικοινωνίας πραγματικού χρόνου (RTCA) για να υποστηρίξει τέτοιου τύπου επικοινωνία ανάμεσα στα διάφορα στρώματα (Virtual Machines) που συνυπάρχουν στον ίδιο φυσικό host. Η RTCA έχει πλέον επεκταθεί ώστε να υποστηρίζει επικοινωνία δικτύου πραγματικού χρόνου μεταξύ guests σε διαφορετικούς φυσικούς host.

Συνοπτικά, ο RT-Xen παρέχει ένα νέο σχήμα δρομολόγησης που είναι ξεχωριστό από τους υπάρχοντες δρομολογητές και είναι σχεδιασμένο ώστε να παρέχει αποδόσεις πραγματικού χρόνου βασισμένο στη θεωρία συνθετικής δρομολόγησης πραγματικού χρόνου.

6.3 *Poris* real-time Scheduler

Αρκετά porject επέκτειναν τον προκαθοριμένο δρομολογητή του Xen προκειμένου να παρέχουν καλύτερη υποστήριξη σε χαλαρές εφαρμογές πραγματικού χρόνου. Ένα τέτοιο παράδειγμα αποτελεί ο *Poris* real-time scheduler [28].

Στα πλαίσια της έρευνας αυτής οι συγγραφείς διαχώρισαν τις εφαρμογές πραγματικου χρόνου σε time-driven και event-driven. Προκειμένουν να αντιμετωπίσουν και τις δυο κατηγορίες με τέτοιο τρόπο ώστε να ικανοποιούν απαιτήσεις πραγματικού χρόνου, για κάθε κατηγορία εισήγαγαν μια αλλαγή στο δρομολογητή του Xen ως προς τον τρόπο χειρισμού των Virtual Machines. Συγκεκριμένα, για τον κατάλληλο χειρισμό event-driven εφαρμογών εισήγαγαν μια ακόμα τάξη προτεραιότητας πέρα από τις προκαθορισμένες 3 του δρομολογητή. Αυτή η τάξη αφορά μόνο τις VCPU με εφαρμογές πραγματικού χρόνου οι οποίες και αρχικοποιούνται με αυτή την τάξη προτεραιότητας.

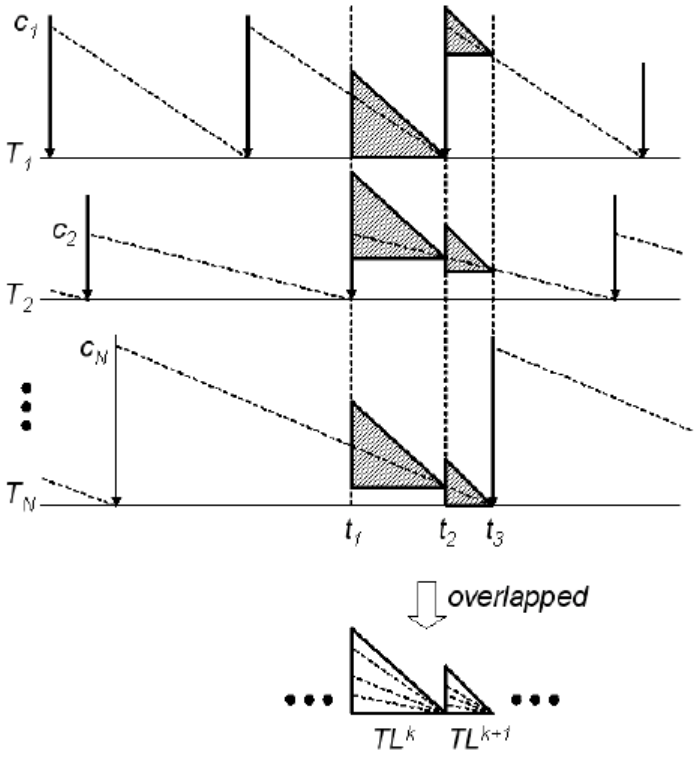
Αντίστοιχα, για την αντιμετώπιση time-driven εφαρμογών προστέθηκε η δυνατότητα δυναμικής αναπροσαρμογής του χρονικού διαστήματος εξυπηρέτησης της κάθε VCPU. Συγκεκριμένα ο δρομολογητής του Xen δρομολογεί μια VCPU κάθε συγκεκριμένο χρονικό διάστημα που αντιστοιχεί σε 30ms. Στα πλαίσια του *Poris* το χρονικό αυτό διάστημα μεταβάλλεται, αν χρειαστεί, ως συνάρτηση των υπαρχόντων Virtual Machines πραγματικού χρόνου και του λόγου εξυπηρέτησης VCPU ανά pCPU ώστε να προσαρμοστεί σε μικρότερες τιμές ικανές να πλαισιώσουν καλύτερα τις απαιτήσεις πραγματικού χρόνου αλλά και όχι τόσο μικρές ώστε να επηρεάζουν αρνητικά την απόδοση CPU-intensive ή memory-intensive εφαρμογών.

Τέλος, αναγνωρίζοντας κάποια κομμάτια της υλοποίησης του Xen δρομολογητή που εισήγαγαν δυσκολίες στον αρμονικό τρόπο λειτουργίας του αλγορίθμου, οι συγγραφείς προχώρησαν και σε κάποιες τροποποιήσεις για την αποφυγή τους. Κυριότερη εξ αυτών αποτελεί πως οι VCPUs ενός guest που έχει μαρκαριστεί ως guest πραγματικού χρόνου κατανέμονται σε διαφορετική pCPU η κάθε μια αν αυτό είναι δυνατόν, μιας και κάτι τέτοιο θα περιόριζε τις δυνατότητες του συστήματος να εξυπηρετήσει κατάλληλα όλες τις VCPU του guest. Ακόμα, πως όταν η πρώτη VCPU ενός τέτοιου guest δρομολογηθεί για πρώτη

φορά, τότε όλες οι άλλες ορίζονται να έχουν την τάξη προτεραιότητας πραγματικού χρόνου. Αυτό υλοποιήθηκε προκειμένου να αποφευχθεί η μετανάστευση κάποιας VCPU σε άλλη pCPU όπου εξυπηρετείται κάποια VCPU του ίδιου guest, μιας και υπο προυποθέσεις ο προκαθορισμένος δρομολογητής του Xen θα επέβαλλε μια τέτοια μετανάστευση.

6.4 A T-L plane based approach: the LLREF algorithm

Σε μια προσπάθεια να διαφοροποιηθούν από τις θεωρίες κατανομής σταθερού κβάντου χρόνου σε κάθε guest στο [29] οι συγγρφείς εισάγουν το T-L Plane ως μια καινούρια έννοια γύρω από την οποία μπορεί να γίνει η κατανομή των υπολογιστικών πόρων ενός πολυεπεξεργαστικού συστήματος στους διάφορους guests. Συγκεκριμένα ο διαμερισμός πόρων εξυπηρέτησης γίνεται όχι ανά guest αλλά globaly ανά εργασία.



Σχήμα 6.1: Απεικόνιση του T-L Plane

Για να ορίσουν το T-L plane οι συγγραφείς χρησιμοποιούν το fluid μοντέλο δρομολόγησης κατά το οποίο κάθε επανεκτέλεση μιας εργασίας απεικονίζεται ως ένα ισοσκελές τρίγωνο εντός ενός συστήματος αξόνων χρόνου εκτέλεσης (y-άξονας) και χρόνου (x-άξονας). Δεδομένων N εργασιών και των fluid διαγραμμάτων τους, ένα ισοσκελές τρίγωνο προκύπτει ανάμεσα σε κάθε 2 συνεχόμενες επανεκτελέσεις μιας εργασίας και έτσι προκύπτουν N επικαλυπτόμενα τρίγωνα μεταξύ διαδοχικών επανεκτελέσεων, για κάθε χρονική στιγμή k. Το τρίγωνο που προκύπτει από την επικάλυψη αυτή ονομάζουν TL^k. Η βάση του είναι ο εναπομείνων χρόνος του και η αριστερή του πλευρά χαρακτηρίζει τον χρόνο εκτέλεσης που πρέπει να καταναλωθεί πριν το τέλος του TL^k τον οποίον και ονομάζουν *τοπικό υπολοιπόμενο χρόνο εκτέλεσης*.

Βάσει της προσέγγισης αυτής εισάγουν τον LLREF αλγόριθμο[30] σύμφωνα με τον οποίο, με αρχική χρονική στιγμή την στιγμή 0, κάθε φορά που η μια επανάληψη εκτέλεσης μιας εργασίας τελειώνει, υπολογίζονται τα νέα TL^k για το χρονικό διάστημα από την αρχική στιγμή ως την στιγμή αυτή. Ο αλγόριθμος θα επιλέξει να δρομολογήσει τις εργασίες με τους M μεγαλύτερους τοπικούς υπολοιπόμενους χρόνους εκτέλεσης, όπου M ο αριθμός των επεξεργαστών στο host μηχάνημα. Όταν μια εξ αυτών εξαντλήσει το χρόνο εκτελεσής της ο υπολογισμός επαναλαμβάνεται με νέα αρχική χρονική στιγμή την παλιά τελική.

Οι συγγραφείς αποδεικνύουν θεωρητικά ότι το σχήμα αυτό παρέχει εγγυήσεις πραγματικού χρόνου τόσο αν $N <= M$ όσο και αν $N > M$ (N ο αριθμός των εργασιών). Εντούτοις δεν παρέχουν πειραματική αξιολόγηση της προσέγγισής τους σε πραγματικό σύστημα.

7

Σύνοψη

Οι εφαρμογές πραγματικού χρόνου, δηλαδή η κατηγορία εφαρμογών των οποίων η ορθή λειτουγία είναι αναπόσπαστα συνδεδεμένη με το χρόνο στον οποίο ολοκληρώνονται τα μέρη τους, γίνονται μέρα με τη μέρα όλο και πιο μεγάλο κομμάτι των υπολογιστικών εφαρμογών τόσο σε εμπορικό επίπεδο όσο και σε επίπεδο χρηστών. Την ίδια στιγμή, το cloud computing και οι δυνατότητες που προσφέρει στην κατεύθυνση της βέλτιστης οργάνωσης και χρήσης υπολογιστικών πόρων συντελεί στην μετανάστευση πολλών κατηγοριών εφαρμογών εντός συστημάτων νέφους. Ωστόσο, μια από αυτές τις κατηγορίες δεν είναι αυτή των εφαρμογών πραγματικού χρόνου, αν και το όφελός που ένα σύστημα υπολογιστικού νέφους θα προσέφερε θα ήταν πολύ μεγάλο. Αυτό επειδή κανείς πάροχος υπολογιστικού νέφους δεν παρουσιάζεται ως εγγυητής επίτευξης ικανοποιητικών επιδόσεων που θα καθιστούσαν την λειτουργία των εφαρμογών ορθή. Παρά την υλοποίηση, σε ερευνητικό επίπεδο, hypervisors που παρέχουν μερικώς τέτοιες εγγυήσεις, δεν υπάρχει, κάποιος εμπορικός πάροχος υπηρεσιών νέφους που να αναδικνείεται ως αξιόπιστη λύση.

Τα παραπάνω, μας παρακίνησαν ώστε να προσπαθήσουμε να δώσουμε μια δική μας προσέγγιση στην κατεύθυνση της δημιουργίας ενός hypervisor ικανού να εξυπηρετεί αποδοτικά όγκο εφαρμογών πραγματικού χρόνου, σε ένα πολυεπξεργαστικό σύστημα ή και ένα σύστημα πολλών servers.

Προκειμένου να αναδείξουμε τις ιδιαιτερότητες που ανακύπτουν από αυτό το πρόβλημα, αναλύσαμε τις τεχνολογίες στις οποίες βασιστήκαμε. Παρουσιάσαμε διεξοδικά τα ειδικά χαρακτηριστικά και τις αρχές που διέπουν μια εφαρμογή πραγματικού χρόνου. Επιμείναμε στον προσδιορισμό των στοιχείων αυτών που συντελούν στην αποδοτική οργάνωση και εκτέλεση εργασιών εντός ενός συστήματος πραγματικού χρόνου και σχολιάσαμε κατηγορίες εφαρμογών που συνδέονται άμεσα με το σύστημα που αποτέλεσε τη βάση της εργασίας αυτής. Περιγράψαμε με περισσότερη σαφήνεια την έννοια του cloud computing και φανερώσαμε τα στοιχεία αυτά που το συνδέουν άμεσα με την έννοια του virtualization. Στα πλαίσια αυτού, αναφερθήκαμε στις επιμέρους τε-

χνολογίες virtualization που είτε αποτελούν λύσεις σε πληθώρα προσεγγίσεων δρομολόγησης εργασιών πραγματικού χρόνου σε virtualized συστήματα, είτε έχουν στοιχεία που χρησιμοποιήθηκαν στα πλαίσια της δικής μας προσέγγισης. Τέλος, σταθήκαμε λεπτομερειακά στις βασικές αρχές του KairosVM hypervisor που αποτέλεσε την τεχνολογία στην οποία στηρίχθηκε η εργασία και επεκτάθηκε προκειμένου να δημιουργηθεί ένας hypervisor πολυ-επεξεργαστικού συστήματος.

Προτού προχωρήσουμε στην ανάλυση της προσέγγισής μας διατυπώσαμε σύντομα τις εικασίες και τα μοντέλα σε επίπεδο υλικού, guest συστημάτων και εργασιών πραγματικού χρόνου που χρησιμοποιήθηκαν στην εργασία.

Ακολούθως, προσδιορίσαμε τα ερωτήματα που πρέπει να απαντηθούν στα πλαίσια του σχεδιασμού ενός hypervisor πολλών πυρήνων. Στο ερώτημα της ανάδειξης μιας μονάδας μέτρησης όγκου φόρτου εργασίας, απαντήσαμε με την έννοια της χρησιμοποίησης δηλαδή του ποσοστού χρόνου κατά τον οποίο μια εργασία αφήνει τον επεξεργαστή μη αδρανή. Δεδομένης της απάντησής μας αυτής, προκειμένου να απαντήσουμε στο ερώτημα της εύρεσης μια βέλτιστης πολιτικής κατανομής όγκου εργασιών στις επεξεργαστικές μονάδες, αντιστοιχίσαμε το πρόβλημα αυτό σε ένα γνωστό NP-πλήρες πρόβλημα το Bin-packing. Κατανοώντας έτσι πως δεν υπάρχει μοναδική βέλτιστη απάντηση στο δεύτερο ερώτημά μας, αιτιολογήσαμε γιατί στα πλαίσια της εργασίας ασχοληθήκαμε με τρεις πολιτικές κατανομής. Τις κατονομάσαμε και παρουσιάσαμε τα βασικά στοιχεία της υλοποίησής τους, πάνω από τον KairosVM hypervisor, δεδομένου του μοντέλου στο οποίο βασιστήκαμε.

Για την αξιολόγηση της προσέγγισής μας προβήκαμε σε μια σειρά πειραμάτων. Μέσω αυτών κάναμε εμφανές το χρονικό διάστημα περιόδων εφαρμογών, στο οποίο τα overheads του virtualization και της υλοποίησης δεν επιφέρουν επιβαρύνσεις στην διενέργεια του συστήματος. Χρησιμοποιώντας εφαρμογές εντός αυτού του διαστήματος συγκρίναμε τις διαφορετικές πολιτικές κατανομής φόρτου εργασίας, από τους guest στους πόρους του συστήματος, τόσο ως προς την αποδοτική εκτέλεση των εφαρμογών, όσο και ως προς την δυνατότητά τους να χρησιμοποιούν τους ελάχιστους αναγκαίους επεξεργαστικούς πόρους, χωρίς απώλειες απόδοσης. Συγκρίναμε, τέλος, τις επιδόσεις του KairosVM hypervisor με τον γνωστό VanillaKVM σε μονο-επεξεργαστικό αλλά και πολυ-επεξεργαστικό σύστημα.

Εν συνεχεία, παραθέσαμε συνοπτικά άλλες γνωστές υλοποιήσεις στον χώρο της δρομολόγησης εφαρμογών πραγματικού χρόνου σε υπολογιστικό νέφος. Σταθήκαμε, ιδιαίτερα, στα στοιχεία εκείνα που την διαφοροποιούν από τη δική μας.

Με λίγα λόγια, δώσαμε μια πλήρη λύση στο ζήτημα της δρομολόγησης εργασιών πραγματικού χρόνου πλαισιωμένη από τις μετρήσεις απόδοσής της σε ένα συγκεκριμένο φάσμα εφαρμογών, που ακολουθούν δεδομένο μοντέλο. Στα πλαίσια της εκτενέστερης αξιολογησής της, ενδεικτικά αναφέρουμε την ανάγκη διενέργειας περαιτέρω πειραμάτων σε υλικό με μεγαλύτερο αριθμό επεξεργαστικών μονάδων, σε μια απόπειρα να προσομοιώσουμε καλύτερα ένα αληθινό σύστημα υπολογιστικού νέφους. Για την επέκτασή της, πιθανές κατευθύνσεις ενασχόλησης θα ήταν α) η χρήση διαφορέτικών πολιτικών κατανομής φόρτου εργασίας στους πόρους του συστήματος, είτε ”fit” πολιτικές με διαφορετικά κριτήρια ταξινόμησης, είτε πολιτικές με άλλη βασική αρχή, β) η υλοποίηση μιας λύσης που ξεφεύγει από τους περιορισμούς του μοντέλου υλικού που χρησιμοποιήθηκε και κυρίως της λογικής της partitioned κατανομής, που ακολουθήθηκε.

Βιβλιογραφία

[1] K. G. Shin and P. Ramanathan. Real-time computing: a new discipline of computer science and engineering. *Proceedings of the IEEE*, 82(1):6–24, Jan 1994. ISSN 0018-9219. doi: 10.1109/5.259423.

[2] Dario Faggioli, Fabio Checconi, Michael Trimarchi, and Claudio Scordino. An edf scheduling class for the linux kernel. Citeseer.

[3] Matthew Dellinger, Piyush Garyali, and Binoy Ravindran. Chronos linux: a best-effort real-time multiprocessor linux kernel. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, pages 474–479. IEEE, 2011.

[4] Peter M. Mell and Timothy Grance. Sp 800-145. the nist definition of cloud computing. Technical report, Gaithersburg, MD, United States, 2011.

[5] SCOPE Alliance. Virtualization: State of the art, 2008.

[6] James E Smith and Ravi Nair. The architecture of virtual machines. *Computer*, 38(5):32–38, 2005.

[7] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. kvm: the linux virtual machine monitor. In *Proceedings of the Linux symposium*, volume 1, pages 225–230, 2007.

[8] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *USENIX Annual Technical Conference, FREENIX Track*, pages 41–46, 2005.

[9] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *ACM SIGOPS operating systems review*, volume 37, pages 164–177. ACM, 2003.

[10] Intel virtualization technology for directed i/o, . URL http://www.intel.com/content/dam/www/public/us/en/documents/product-specifications/vt-directed-io-spec.pdf.

[11] Intel virtualization technology (intel vt), . URL https://www.intel.com/content/www/us/en/virtualization/virtualization-technology/intel-virtualization-technology.html.

[12] Amd, amd virtualization. URL http://www.amd.com/en-us/solutions?utm_campaign=solutions&tm_medium=redirect&utm_source=301.

[13] Jonas Pfoh, Christian Schneider, and Claudia Eckert. Nitro: Hardware-based system call tracing for virtual machines. In *International Workshop on Security*, pages 96–112. Springer, 2011.

[14] Kevin Burns, Antonio Barbalace, Vincent Legout, and Binoy Ravindran. Kairosvm: Deterministic introspection for real-time virtual machine hierarchical scheduling. In *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*, pages 1–8. IEEE, 2014.

[15] Michael S Drescher. *A Flattened Hierarchical Scheduler for Real-Time Virtual Machines*. PhD thesis, Virginia Polytechnic Institute and State University, 2015.

[16] Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

[17] Phillip A Laplante et al. *Real-time systems design and analysis*. Wiley New York, 2004.

[18] The bin packing problem. URL http://ac.informatik.uni-freiburg.de/lak_teaching/ws11_12/combopt/notes/bin_packing.pdf.

[19] Steve Abbott and Paul Hoffman. The man who loved only numbers: The story of paul erdős and the search for mathematical truth, 1998.

[20] Jose Maria López, Manuel García, José Luis Diaz, and Daniel F Garcia. Worst-case utilization bound for edf scheduling on real-time multiprocessor systems. In *Real-Time Systems, 2000. Euromicro RTS 2000. 12th Euromicro Conference on*, pages 25–33. IEEE, 2000.

[21] Loren Merritt and Rahul Vanam. x264: A high performance h. 264/avc encoder. *online] http://neuron2. net/library/avc/overview_x264_v8_5. pdf*, 2006.

[22] Christian Bienia and Kai Li. Parsec 2.0: A new benchmark suite for chip-multiprocessors. In *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, volume 2011, 2009.

[23] Matthew R Guthaus, Jeffrey S Ringenberg, Dan Ernst, Todd M Austin, Trevor Mudge, and Richard B Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, pages 3–14. IEEE, 2001.

[24] Tommaso Cucinotta, Fabio Checconi, George Kousiouris, Kleopatra Konstanteli, Spyridon Gogouvitis, Dimosthenis Kyriazis, Theodora Varvarigou, Alessandro Mazzetti, Zlatko Zlatev, Juri Papay, et al. Virtualised e-learning on the irmos real-time cloud. *Service Oriented Computing and Applications*, 6(2):151–166, 2012.

[25] Luca Abeni and Giorgio Buttazzo. Integrating multimedia applications in hard real-time systems. In *Real-Time Systems Symposium, 1998. Proceedings. The 19th IEEE*, pages 4–13. IEEE, 1998.

[26] Sisu Xi, Justin Wilson, Chenyang Lu, and Christopher Gill. Rt-xen: Towards real-time hypervisor scheduling in xen. In *Embedded Software (EMSOFT), 2011 Proceedings of the International Conference on*, pages 39–48. IEEE, 2011.

[27] Sisu Xi, Meng Xu, Chenyang Lu, Linh TX Phan, Christopher Gill, Oleg Sokolsky, and Insup Lee. Real-time multi-core virtual machine scheduling in xen. In *Embedded Software (EMSOFT), 2014 International Conference on*, pages 1–10. IEEE, 2014.

[28] Like Zhou, Song Wu, Huahua Sun, Hai Jin, and Xuanhua Shi. Virtual machine scheduling for parallel soft real-time applications. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2013 IEEE 21st International Symposium on*, pages 525–534. IEEE, 2013.

[29] Hyeonjoong Cho, Binoy Ravindran, and E Douglas Jensen. T–l plane-based real-time scheduling for homogeneous multiprocessors. *Journal of Parallel and Distributed Computing*, 70(3):225–236, 2010.

[30] Hyeonjoong Cho, Binoy Ravindran, and E Douglas Jensen. An optimal real-time scheduling algorithm for multiprocessors. In *Real-Time Systems Symposium, 2006. RTSS'06. 27th IEEE International*, pages 101–110. IEEE, 2006.