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

Μελέτη επίδρασης ανταγωνισμού για κοινούς πόρους σε περιβάλλον cloud

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΚΩΝΣΤΑΝΤΙΝΟΣ ΠΑΠΑΔΗΜΗΤΡΙΟΥ

Επιβλέπων : Γεώργιος Γκούμας
Αν. Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2018

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

Μελέτη επίδρασης ανταγωνισμού για κοινούς πόρους σε περιβάλλον cloud

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΚΩΝΣΤΑΝΤΙΝΟΣ ΠΑΠΑΔΗΜΗΤΡΙΟΥ

Επιβλέπων : Γεώργιος Γκούμας
Αν. Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 12η Μαρτίου 2018.

.....	.....	.....
Γεώργιος Γκούμας Αν. Καθηγητής Ε.Μ.Π.	Νεκτάριος Κοζύρης Καθηγητής Ε.Μ.Π.	Νικόλαος Παπασπύρου Αν. Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2018

Κωνσταντίνος Παπαδημητρίου

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

Περίληψη

Οι πολυπύρηνες αρχιτεκτονικές αποτελούν σήμερα την αποκλειστική επιλογή σχεδιασμού για κάθε σύγχρονο υπολογιστικό σύστημα. Καθώς γίνεται κοινή χρήση πόρων, όπως ο δίαυλος στην κύρια μνήμη ή οι κρυφές μνήμες, δημιουργείται ανταγωνισμός για την χρήση τους, με αποτέλεσμα σημαντικές καθυστερήσεις στον χρόνο εκτέλεσης των προγραμμάτων. Σύγχρονοι χρονοδρομολογητές, όπως ο CFS του linux, δεν αντιμετωπίζουν με κάποιο τρόπο τέτοια προβλήματα.

Στο περιβάλλον του νέφους είναι ευρέως διαδεδομένες υπηρεσίες που προσφέρουν τη απομακρυσμένη δημιουργία και χρήση εικονικών μηχανών. Ο πάροχος των συγκεκριμένων υπηρεσιών έχει στη διάθεσή του ένα cluster πάνω στο οποίο δημιουργούνται πολύ περισσότερες εικονικές μηχανές απ' ό,τι τα φυσικά μηχανήματα. Αυτό έχει ως αποτέλεσμα οι εικονικές μηχανές να ανταγωνίζονται για τους φυσικούς πόρους. Ανάλογα με τις εφαρμογές που εκτελεί μία εικονική μηχανή έχει διαφορετική συμπεριφορά πάνω σε ένα σύστημα και οι ανάγκες της στους πόρους του ποικίλουν. Η επιλογή πολιτικής δρομολόγησης των εικονικών μηχανών που δεν λαμβάνει υπ' όψιν τους ανταγωνισμούς έχει ως αποτέλεσμα την απρόβλεπτη συμπεριφορά του συστήματος.

Σκοπός της παρούσης εργασίας είναι να μελετηθούν οι επιδράσεις τον παραπάνω ανταγωνισμών στην επίδοση του συστήματος. Γίνεται πρόταση κατηγοριοποίησης των εικονικών μηχανών βάση της συμπεριφοράς τους και στη συνέχει αναλύεται η επίδραση που έχει κάθε κατηγορία στις υπόλοιπες. Ταυτόχρονα προτείνονται λύσεις για την αντιμετώπιση του προβλήματος μέσω χρονοπρογραμματισμού που λαμβάνει υπ' όψιν τους ανταγωνισμούς. Τελικά γίνεται μία σύγκριση των διάφορων πολιτικών χρονοδρομολόγησης.

Λέξεις κλειδιά

Χρονοπρογραμματισμός, χρονοδρομολόγηση, ανταγωνισμός για κοινούς πόρους, εικονικές μηχανές, εικονοποίηση.

Abstract

Multi-core architectures constitute, nowadays, the only option for designing any modern computer system. Given the common use of resources, such as the channel in the main memory or the hidden memories, contention raises for their use, which results to important delays when running a program. Modern process schedulers, such as CFS in Linux, do not encounter such problems in any way.

In the cloud environment, services offering remote creation and use of virtual machines are prevailing. The provider of those services has at their disposal a cluster on which a much bigger number of virtual machines than physical devices is created. As a result, virtual machines compete for material resources. In respect to the applications it runs, a virtual machine alters its behavior upon a system; its needs in resources also vary. The option of a virtual machines scheduling method not taking into account existing contentions results in an unpredictable behavior of the system.

Aim of this thesis is to study the effects of the above-mentioned contentions on the system performance. We suggest a classification method based on virtual machines' behaviors and we then analyze the impact of each class over the rest. In addition, we propose solutions for resolving the problem through a time scheduling method taking into account existing contentions. In the end, we compare the different policies of process scheduling.

Key words

Scheduling, contention, virtualization, virtual machines, cloud computing.

Ευχαριστίες

Ευχαριστώ θερμά τον υποψήφιο διδάκτωρα και επιβλέποντα αυτής της διατριβής κ. Αλέξανδρο Χαριτάτο, για τη συνεχή καθοδήγηση και εμπιστοσύνη του. Θέλω ακόμα να ευχαριστήσω τόσο τους φίλους μου όσο και την κοπέλα μου Αλεξία που με στηρίξανε κατά την εκπόνηση αυτής της εργασίας σε ψυχολογικό αλλά και σε πρακτικό επίπεδο. Θα ήθελα τέλος να ευχαριστήσω την οικογένειά μου και κυρίως τους γονείς μου, οι οποίοι με υποστήριξαν και έκαναν δυνατή την απερίσπαστη ενασχόλησή μου τόσο με την εκπόνηση της διπλωματικής μου, όσο και συνολικά με τις σπουδές μου.

Κωνσταντίνος Παπαδημητρίου,

Αθήνα, 12η Μαρτίου 2018

Η εργασία αυτή είναι επίσης διαθέσιμη ως Τεχνική Αναφορά CSD-SW-TR-42-14, Εθνικό Μετσόβιο Πολυτεχνείο, Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών, Εργαστήριο Τεχνολογίας Λογισμικού, Μάρτιος 2018.

URL: `http://www.softlab.ntua.gr/techrep/`
FTP: `ftp://ftp.softlab.ntua.gr/pub/techrep/`

Περιεχόμενα

Περίληψη 5

Abstract 7

Ευχαριστίες 9

Περιεχόμενα 11

Κατάλογος σχημάτων 13

1. Εισαγωγή 15
1.1 Αρχές πολυπύρηνων αρχιτεκτονικών 15
1.2 Λειτουργικά Συστήματα 15
1.3 Υπολογιστικό Νέφος (Cloud Computing) 16
1.3.1 Μοντέλα Νεφών 16
1.3.2 Πλεονεκτήματα 17
1.3.3 Μειονεκτήματα 17
1.4 Εικονικές Μηχανές (VM) 17
1.4.1 Στιγμιότυπο 19
1.4.2 Migration 19
1.5 Ορισμός του προβλήματος 19
1.5.1 Η δική μας προσέγγιση 20
1.6 Περιγραφή κεφαλαίων 20

2. Κίνητρο και σχετική έρευνα 21
2.1 Θεωρητικό υπόβαθρο 21
2.1.1 Υλικό υπολογιστών και πρόοδος 21
2.1.2 Η έννοια του παραλληλισμού 23
2.1.3 Νεφος - Cloud Computing 24
2.1.4 Εικονοποίηση - Virtualization 25
2.1.5 Χρονοπρογραμματισμός 26
2.2 Το πρόβλημα σε βάθος 28
2.2.1 Τεχνικές διαχείρισης νέφους 29
2.2.2 Σημεία ανταγωνισμού 30
2.2.3 Contention-aware scheduling 30
2.3 Σχετική έρευνα 31
2.3.1 Χρονοπρογραμματισμός και ανταγωνισμός 31
2.3.2 Χρονοπρογραμματισμός και εικονοποίηση 34

3. Χρησιμοποιούμενες υποδομές . . . 35
3.1 Περιβάλλον πειραμάτων . . . 35
3.1.1 Υλικό . . . 35
3.1.2 Scaff . . . 38
3.1.3 Μέθοδος εκτέλεσης πειραμάτων . . . 39
3.1.4 QEMU cpu models . . . 40
3.2 Αναλυτική περιγραφή αλγορίθμων . . . 41
3.2.1 Throughput Balance . . . 41
3.2.2 LCA: ένας αποδοτικός χρονοδρομολογητής . . . 42

4. Αποτελέσματα μετρήσεων . . . 47
4.1 Κατάταξη πειραματικών εφαρμογών . . . 47
4.2 Πειραματική διαδικασία . . . 48
4.2.1 Classification . . . 48
4.2.2 Αναλυτικοί πίνακες αποτελεσμάτων (Heatmaps) . . . 48
4.2.3 Αλγόριθμοι χρονοδρομολόγησης . . . 49
4.2.4 Εκτέλεση πειραμάτων σε εικονικό περιβάλλον . . . 49
4.2.5 Results estimation . . . 49
4.3 Πειράματα απ ευθείας στον host . . . 51
4.4 Πειράματα σε εικονική μηχανή με -cpu QEMU64 . . . 54
4.5 Πειράματα σε εικονική μηχανή με -cpu host . . . 55

5. Συγκρίσεις χρονοδρομολογητών . . . 57
5.1 Τα αποτελέσματα . . . 58
5.2 Γενικές παρατηρήσεις και προτάσεις . . . 59

6. Συμπεράσματα και προτάσεις . . . 71
6.1 Συμπεράσματα . . . 71
6.2 Future Work . . . 72
6.2.1 Χρονοδρομολόγηση με ανταγωνισμούς . . . 72
6.2.2 Χρονοδρομολόγηση σε εικονικές μηχανές . . . 72

Βιβλιογραφία . . . 73

Κατάλογος σχημάτων

2.1 Νόμος του Moore . . . 21
2.2 Πρόοδος CPU - Μνήμης . . . 22
2.3 Τύποι εικονοποίησης . . . 26
2.4 Παράδειγμα: Ιδανική λύση . . . 33
2.5 Παράδειγμα: Ιδανική λύση (συνέχεια) . . . 33

3.1 Intel Xeon E5-4620 . . . 36
3.2 Χαρακτηριστικά Intel Xeon E5-4620 . . . 36
3.3 Πειραματική διάταξη . . . 37
3.4 Κατηγοριοποίηση εφαρμογών . . . 43
3.5 Δέντρο επιλογής κλάσης . . . 44

4.1 Κατάταξη εφαρμογών σε κλάσεις . . . 47
4.2 Επιβραδύνσεις μεταξύ εφαρμογών της κλάσης C . . . 51
4.3 Επιβραδύνσεις μεταξύ εφαρμογών των κλάσεων C και LC . . . 52
4.4 Επιβραδύνσεις μεταξύ εφαρμογών των κλάσεων C και L . . . 53
4.5 Επιβράδυνση σε bm περιβάλλον . . . 54
4.6 Επιβράδυνση με -cpu QEMU64 . . . 55
4.7 Επιβράδυνση με -cpu QEMU64 . . . 55
4.8 Επιβράδυνση με -cpu host . . . 56

5.1 Αποτελέσματα για c-c-c-c-l-l-n-n . . . 61
5.2 Αποτελέσματα για l-l-l-l-c-c-c-c . . . 62
5.3 Αποτελέσματα για l-l-l-l-n-n-n-n . . . 63
5.4 Αποτελέσματα για l-l-lc-lc-lc-lc-c-c . . . 64
5.5 Αποτελέσματα για l-lc-lc-lc-c-c-c-n . . . 65
5.6 Αποτελέσματα για lc-c-c-c-c-c-c-n . . . 66
5.7 Αποτελέσματα για lc-lc-c-c-c-c-n-n . . . 67
5.8 Αποτελέσματα για lc-lc-lc-lc-c-c-c-c . . . 68
5.9 Αποτελέσματα για lc-lc-lc-lc-n-n-n-n . . . 69

Κεφάλαιο 1

Εισαγωγή

1.1 Αρχές πολυπύρηνων αρχιτεκτονικών

Τα τελευταία χρόνια η σχεδίαση και οι αρχές των υπολογιστικών συστημάτων έχουν αλλάξει ραγδαία. Αρχικά ένας επεξεργαστής αποτελούνταν από ένα μόλις πυρήνα ο οποίος αναλάμβανε να εκτελέσει όλη την εργασία. Αυτό φυσικά καθυστερεί ιδιαίτερα τις διάφορες εφαρμογές και δεν εκμεταλλεύεται τις δυνατότητες παραλληλισμού είτε εντός της εφαρμογής είτε μεταξύ δύο ή περισσότερων εφαρμογών. Έτσι η πρόοδος της τεχνολογίας και οι απαιτήσεις για μεγάλη υπολογιστική δύναμη έφεραν τις πολυπύρηνες αρχιτεκτονικές, στις οποίες ο φόρτος εργασίας διαμοιράζεται σε περισσότερους πυρήνες, βελτιώνοντας θεαματικά την απόδοση ενός συστήματος.

Η τεχνολογία των πολυπύρηνων αρχιτεκτονικών ήταν γνωστή και χρησιμοποιούταν από τις προηγούμενες δεκαετίες σε συστήματα μεγάλης κλίμακας όπως υπερυπολογιστές και κέντρα δεδομένων (data centers). Είναι τα τελευταία χρόνια όμως που έχει γνωρίσει μεγάλη άνθηση. Η μεγάλη ζήτηση σε αποδοτικότερα συστήματα έφερε την τεχνολογία σε υπολογιστές και συστήματα γενικού σκοπού όπως laptop και κινητά τηλέφωνα. Ωστόσο, τα συστήματα αυτά συναντάνε νέα προβλήματα τα οποία περιορίζουν την βελτίωση της υπολογιστικής ισχύος. Οι πυρήνες μοιράζονται κοινούς πόρους για τη χρησιμοποίηση των οποίων ανταγωνίζονται. Αυτό έχει ως αποτέλεσμα τον περιορισμό της συνολικής επίδοσης (throughput). Είναι δουλειά λοιπόν του λειτουργικού συστήματος να μοιράσει με τέτοιο τρόπο τους πόρους στις διάφορες εφαρμογές ώστε να ελαχιστοποιήσει αυτόν τον ανταγωνισμό.

1.2 Λειτουργικά Συστήματα

Με τον όρο λειτουργικό σύστημα (Operating System - OS) εννοούμε το πρόγραμμα το οποίο αναλαμβάνει να διαμοιράσει τους πόρους του συστήματος στις διάφορες διεργασίες. Βασικός ρόλος λοιπόν του ΛΣ είναι να δίνει χρόνο για εκτέλεση στις εφαρμογές και να διαμοιράζει την κύρια μνήμη όπως και τους λοιπούς πόρους. Επίσης αναλαμβάνει να διατηρεί τα διάφορα συστήματα αρχείων και να παρέχει στις διεργασίες ό,τι αυτές μπορεί να χρειαστούν.

Για την επίλυση του παραπάνω προβλήματος το ΛΣ πρέπει να λαμβάνει υπ' όψιν τις απαιτήσεις των διεργασιών και να παίρνει διάφορες αποφάσεις αναφορικά με τον διαμοιρασμό των πόρων. Βασικό συστατικό ενός ΛΣ αποτελεί ο χρονοδρομολογητής (scheduler). Ο χρονοδρομολογητής παίρνει αποφάσεις βασισμένες σε μια συγκεκριμένη πολιτική που αφορούν τη χρήση του επεξεργαστή από τις διεργασίες με στόχο τη βέλτιστη απόδοση του συστήματος. Η απόδοση από τη μεριά της μπορεί να μεταφράζεται σε καλή αποκριτικότητα - χαρακτηριστικό απαραίτητο σε συστήματα γενικού σκοπού, σε υψηλό throughput, δηλαδή επεξεργασία μεγάλου όγκου δεδομένων ανα μονάδα χρόνου - απαραίτητο σε συστήματα μεγάλης κλίμακας όπως servers και υπερυπολογιστές ή χαμηλή κατανάλωση ενέργειας, συνηθισμένη απαίτηση σε ενσωματωμένα συστήματα (μαζί με την υψηλή αποκριτικότητα)

Έχουν προταθεί και υλοποιηθεί λοιπόν πολλοί αλγόριθμοι που αποφασίζουν την σειρά με την οποία οι διάφορες διεργασίες θα χρησιμοποιήσουν την κεντρική μονάδα επεξεργασίας. Για συστήματα με μόνο ένα πυρήνα επεξεργασίας, το πρόβλημα ανάγεται στον διαμοιρασμό του χρόνου στις διεργασίες που χρειάζονται τον επεξεργαστή και μετά από έρευνες ετών οι πολιτικές που είχαν προταθεί βελτιστοποιήθηκαν. Με την πρόοδο ωστόσο της τεχνολογίας και την είσοδο στις πολυπύρηνες αρχιτεκτονικές η πολυπλοκότητα του προβλήματος αυξήθηκε καθώς πλέον δεν πρέπει να διαμοιραστεί μόνο ο χρόνος αλλά να αποφασιστεί και ποιος πυρήνας θα διατεθεί στην εκάστοτε διεργασία. Οι υπάρχουσες υλοποιήσεις που χρησιμοποιούνται στα σημερινά λειτουργικά συστήματα δεν λαμβάνουν υπ' όψιν την πολυπλοκότητα του συστήματος και χειρίζονται τους πυρήνες ως ξεχωριστές οντότητες. Αυτή η λογική ενώ είναι αρκετά απλή και εύκολα υλοποιήσιμη, έχει συχνά ως συνέπεια την ταυτόχρονη εκτέλεση διεργασιών που χρησιμοποιούν κοινούς πόρους, με αποτέλεσμα την αύξηση του ανταγωνισμού και τον περιορισμό της απόδοσης του συστήματος.

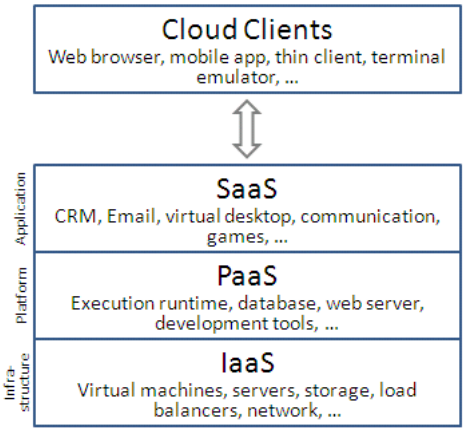
1.3 Υπολογιστικό Νέφος (Cloud Computing)

Τα τελευταία χρόνια προωθείται με νέα τάση, όπου ο χρήστης δεν χρησιμοποιεί μηχανήματα της κατοχής του για υπολογισμούς ή αποθήκευση δεδομένων, αλλά προτιμάει οργανισμούς ή εταιρείες που είναι σε θέση να παρέχουν τις εν λόγω υπηρεσίες. Υπολογιστικό νέφος (Cloud Computing) είναι ένας τύπος υπηρεσίας, βασισμένης στο διαδίκτυο, όπου ένας χρήστης μπορεί να χρησιμοποιήσει απομακρυσμένα συστήματα για προσωπική χρήση. Έτσι μία οντότητα που μπορεί να είναι είτε απλός χρήστης είτε οργανισμός ή εταιρεία έχει τη δυνατότητα να μισθώσει μηχανήματα που δεν έχει στην κατοχή της και να γλιτώσει τουλάχιστον αρχικά το κόστος αγοράς και εγκατάστασης δικού της κέντρου δεδομένων.

1.3.1 Μοντέλα Νεφών

Τα βασικά μοντέλα υπηρεσιών που παρέχει ένα νέφος χωρίζονται σε αφαιρετικά επίπεδα και είναι τα εξής:

Λογισμικό ως Υπηρεσία - Software as a Service (SaaS). Σε αυτόν τον τύπο, ο τελικός χρήστης / πελάτης έχει την δυνατότητα να χρησιμοποιήσει τις εφαρμογές που του δίνει ο πάροχος και τρέχουν στο νέφος. Οι εφαρμογές γίνονται προσβάσιμες από τον χρήστη μέσω κάποιας διεπαφής (πχ web browser, command-line interface κλπ). Ο πελάτης δεν έχει την δυνατότητα να τροποποιήσει τις υποδομές του νέφους (όπως το δίκτυο, τους αποθηκευτικούς χώρους ή το λειτουργικό σύστημα) παρά μόνο την εφαρμογή που του παρέχεται, τις ρυθμίσεις της και το περιβάλλον της. Τυπικά παραδείγματα SaaS είναι εφαρμογές email, εικονικές επιφάνειες εργασίας και online παιχνίδια.



Πλατφόρμα ως Υπηρεσία - Platform as a Service (PaaS). Με τον όρο πλατφόρμα ως Υπηρεσία, πάμε ένα επίπεδο χαμηλότερα. Αυτές οι υπηρεσίες παρέχουν στον χρήστη μία πλατφόρμα πάνω στην οποία μπορεί να αναπτύξει να εκτελέσει και να διαχειριστεί τις δικές του εφαρμογές. Σε αυτόν τον τύπο συνήθως ο πάροχος είναι υπεύθυνος να δημιουργήσει και να προσφέρει τη πλατφόρμα η οποία αποτελείται εκτός από το υλικό κομμάτι (κέντρα δεδομένων, δικτύωση κλπ) αλλά και από το λειτουργικό σύστημα, κάποιες βιβλιοθήκες, η βάση δεδομένων και άλλα εργαλεία. Πάνω σε αυτά βασίζεται ο χρήστης που αναλαμβάνει να δημιουργήσει και να χρησιμοποιήσει την τελική εφαρμογή.

Υποδομή ως Υπηρεσία - Infrastructure as a Service (IaaS). Εδώ ο χρήστης έχει τη δυνατότητα να δημιουργήσει και να ελέγξει το σύστημα που θα χρησιμοποιήσει, δηλαδή να επιλέξει χαρακτηριστικά όπως το λειτουργικό σύστημα η τοπολογία του δικτύου κ.ο.κ αλλά και πάλι δεν μπορεί να τροποποιήσει το υποκείμενο σύστημα πάνω στο οποίο τρέχουν οι υπηρεσίες εκτός ίσως από περιορισμένο αριθμό ρυθμίσεων, όπως το τοίχος προστασίας του host (firewall). Τυπικό παράδειγμα αυτού του τύπου των υπηρεσιών είναι η εικονικές μηχανές και ο απομακρυσμένος αποθηκευτικός χώρος. Στην περίπτωση των εικονικών μηχανών, ο χρήστης δημιουργεί εικονικά μηχανήματα, τα οποία υλοποιούνται με τη βοήθεια ενός προγράμματος επίβλεψης (hypervisor) στο απομακρυσμένο σύστημα και ο πελάτης έχει πλήρη πρόσβαση σε αυτά. Ο τρόπος με τον οποίο τα εικονικά μηχανήματα τοποθετούνται στους επεξεργαστές του host είναι το αντικείμενο μελέτης της παρούσας εργασίας.

1.3.2 Πλεονεκτήματα

Ίσως το σημαντικότερο πλεονέκτημα του νέφους είναι η σημαντική εξοικονόμηση κόστους και ενέργειας. Ο χρήστης έχει τις υπηρεσίες που χρειάζεται, όταν τις χρειάζεται χωρίς να απασχολείται για το που και πως θα βρει την υπολογιστική ισχύ που απαιτείται και χωρίς να πρέπει να αγοράσει υπερκοστολογημένα μηχανήματα. Αντίστοιχα μία εταιρεία μπορεί να είναι πλήρως λειτουργική με σχεδόν μηδενικό κόστος.

Επίσης πολύ σημαντική είναι η ευκολία και η διαθεσιμότητα των υπηρεσιών από οποιοδήποτε σημείο βρίσκεται ο τελικός χρήστης. Με απλές μεθόδους (login) ο χρήστης έχει πλήρη πρόσβαση στις εικονικές μηχανές του ή στα απομακρυσμένα δεδομένα που έχει αποθηκεύσει. Άλλωστε ο πάροχος αναλαμβάνει την προστασία των δεδομένων και την ασφαλή επαναφορά τους μετά από απώλεια, κρατώντας πολλά αντίγραφα στο νέφος. Σε περίπτωση κάποιας αναπόφευκτης αστοχίας, ο τελικός χρήστης δεν θα χρειαστεί να διαθέσει ούτε χρόνο ούτε ανθρώπινο δυναμικό για την επισκευή βλαβών.

Η χρήση του νέφους παρέχει μεγάλη ευελιξία. Το κόστος (είτε χρηματικό, είτε ενεργειακό, είτε εξοπλιστικό) είναι άμεσα συνδεδεμένο με τις - παροδικά μεταβαλλόμενες - ανάγκες του χρήστη. Έτσι αν μία εταιρεία χρειαστεί παραπάνω πόρους για συγκεκριμένο χρονικό διάστημα το μόνο που έχει να κάνει είναι να τους νοικιάσει από κάποιο πάροχο αντί να υποστεί το πλήρες αντίτιμο για αυτούς.

1.3.3 Μειονεκτήματα

Όπως όλες οι νέες τεχνολογίες, εκτός από θετικά υπάρχουν και κάποια αρνητικά. Με τη χρήση του νέφους, υπάρχει ο -έστω και αμυδρός- κίνδυνος τεχνικής βλάβης για την οποία δεν ευθύνεται ο χρήστης. Τυπικά οι πόροι είναι μόνιμα διαθέσιμοι από διαφορετικές τοποθεσίες, ωστόσο η πρόσβαση σε αυτούς γίνεται μέσω διαδικτύου που ακόμα και σήμερα δεν θεωρείται πάντα δεδομένη.

Επίσης σημαντική είναι η ασφάλεια των δεδομένων. Μία εταιρεία εμπιστεύεται μέρος των δεδομένων της σε τρίτους, χωρίς να μπορεί να διασφαλίσει πως αυτά δεν θα υποκλαπούν. Άλλωστε η κρυπτογράφησή τους μπορεί να μειώσει πολύ την απόδοση του εγχειρήματος. Παρόμοια, η ιδιοτικότητα ευαίσθητων δεδομένων δεν μπορεί να θεωρείται δεδομένη.

1.4 Εικονικές Μηχανές (VM)

Η έννοια των εικονικών μηχανών υπάρχει από την δεκαετία του 1960 αλλά είναι τα τελευταία 10 χρόνια που έχει επανέλθει στο προσκήνιο, κυρίως επειδή η υψηλή υπολογιστική ισχύς των σύγχρονων συστημάτων προσφέρει τη δυνατότητα υλοποίησής τους. Με τον όρο

εικονοποίηση, εννοούμε τη δημιουργία εικονικών αντικειμένων (αντί πραγματικών) όπως εικονικές πλατφόρμες, δίκτυα, συσκευές αποθήκευσης κλπ. Τα πλεονεκτήματα από τη χρήση εικονικών μηχανών είναι πολλά. Κάποια είναι περισσότερο προφανή, όπως η εκτέλεση εφαρμογών που δεν έχουν σχεδιαστεί για συγκεκριμένο υπολογιστικό και λειτουργικό σύστημα και η χρήση συσκευών που δεν είναι μέρος ενός συστήματος, αλλά υλοποιημένες σε λογισμικό, ενώ άλλα είναι λιγότερο προφανή και έχουν να κάνουν με την ασφάλεια και την απομόνωση που παρέχει μία εικονική μηχανή. Υπάρχουν τρεις βασικοί τύποι εικονοποίησης: **Full virtualization**, **Partial Virtualization** και **Paravirtualization**.

Ο πρώτος τύπος σηματοδοτεί την πλήρη προσομοίωση του υλικού σε λογισμικό. Αυτό σημαίνει πως το λειτουργικό σύστημα του guest (VM) δεν γνωρίζει πως τρέχει σε προσομοίωση, δηλαδή δεν έχει υποστεί καμία αλλαγή. Αυτού του τύπου η εικονοποίηση παρέχει το πλεονέκτημα ότι το λειτουργικό σύστημα συμπεριφέρεται ακριβώς σαν να έτρεχε σε πραγματικό σύστημα και κατ' επέκταση μπορεί να μελετηθεί αυτή η συμπεριφορά σαν τα ήταν υπό πραγματικές συνθήκες. Επίσης είναι σημαντικό πως η υλοποίηση αυτής της τεχνικής είναι εύκολη συγκριτικά με τις άλλες δύο (το μόνο που χρειάζεται να γίνει είναι να προσομοιωθεί η συμπεριφορά ενός μηχανήματος). Από την άλλη μεριά, αφού όλο το υλικό είναι υλοποιημένο σε λογισμικό, η όλη διαδικασία υστερεί σε απόδοση.

Με την δεύτερη τεχνική η εικονική μηχανή προσομοιώνει επαρκές τμήμα του πραγματικού υποκείμενου υλικού ώστε να επιτρέπει την εκτέλεση ενός μη τροποποιημένου ΛΣ σχεδιασμένου για την ίδια αρχιτεκτονική επεξεργαστή με αυτήν του πραγματικού. Σε αυτήν την περίπτωση δεν χρειάζεται η πλήρης εξομοίωση του συνόλου εντολών του πραγματικού επεξεργαστή και μάλιστα υπό συνθήκες επιτρέπεται απευθείας εκτέλεση των εντολών του φιλοξενούμενου μηχανήματος στο πραγματικό με την προϋπόθεση να μην επηρεάζεται κάποιο υποσύστημα έξω από τον άμεσο έλεγχό του. Σε κρίσιμα τμήματα ωστόσο (πχ σε προσπάθεια πρόσβασης σε συσκευή μέσω κλήσης συστήματος) η εποπτεία του hypervisor είναι αναπόφευκτη.

Με την τρίτη τεχνική, το λειτουργικό σύστημα του guest γνωρίζει την ύπαρξη του hypervisor (host) και συνεργάζεται με αυτόν για την καλύτερη απόδοση του συστήματος. Ο hypervisor σε αυτήν την περίπτωση προσφέρει μία διεπαφή στο guest os που του επιτρέπει την απ' ευθείας χρήση του υλικού. Για την χρήση ωστόσο αυτής της διεπαφής το guest os πρέπει να υποστεί βασικές αλλαγές. Για να γίνει χρήση αυτής της τεχνικής με ικανοποιητική βελτίωση της απόδοσης, απαιτείται πρόσθετη υποστήριξη από το υλικό (Intel VT-x, AMD-V).

Τα πλεονεκτήματα της χρήσης εικονικών μηχανών είναι πολλά και όπως προαναφέρθηκε κάποια είναι περισσότερο προφανή από άλλα:

- Δημιουργία υλικών συσκευών από λογισμικό. Με χρήση εικονικών μηχανών, ένας προγραμματιστής είναι σε θέση να δημιουργήσει συσκευές χωρίς την αναγκαστική αγορά ακριβού υλικού.

- Εκτέλεση εφαρμογών σχεδιασμένων για συστήματα διαφορετικά από αυτό του host. Είναι απλή η προσομοίωση διάφορων αρχιτεκτονικών με συνέπεια ο χρήστης να έχει τη δυνατότητα να χρησιμοποιήσει λογισμικό που δεν είναι σχεδιασμένο για τον host. Με την ίδια λογική μπορεί ο χρήστης να εκτελέσει εφαρμογές γραμμένες για ένα λειτουργικό σύστημα σε κάποιο άλλο (πχ παιχνίδια γραμμένα για Windows μπορούν να τρέξουν σε host με Linux).

- Ασφάλεια του host μηχανήματος. Ο χρήστης έχει πλήρη πρόσβαση (administrator/root) στο δικό του εικονικό μηχάνημα, ωστόσο στον host διατηρεί δικαιώματα απλού χρήστη.

- Απομόνωση των εικονικών μηχανών. Ένα εικονικό μηχάνημα μπορεί να τρέχει στο ίδιο φυσικό μηχάνημα μαζί με πολλά άλλα. Ωστόσο ο χώρος του κάθε ενός είναι ιδιωτικός

και δεν επηρεάζεται από κάποιο άλλο μηχάνημα.

- Αποθήκευση μίας κατάστασης και επαναφορά σε αυτήν (πχ ύστερα από κάποια αστοχία (failover)) με χρήση στιγμιοτύπων (snapshot). Η χρήση στιγμιοτύπων επεξηγείται αναλυτικότερα στη συνέχεια.
- Χρήση νέων μηχανημάτων δίχως την αγορά νέου υλικού.
- Πειραματισμός και χρήση στην εκπαίδευση Λειτουργικών Συστημάτων χωρίς τον φόβο της καταστροφής ενός συστήματος από άπειρους χρήστες. Όπως αναφέρθηκε προηγουμένως, η χρήση εικονικών μηχανών προσφέρει ασφάλεια στον host. Όταν λοιπόν θέλουμε να αλλάξουμε το λειτουργικό σύστημα και να το προσαρμόσουμε στις ανάγκες μας, στο χειρότερο σενάριο θα επηρεαστεί μόνο το εικονικό μηχάνημα και όχι ο host.

1.4.1 Στιγμιότυπο

Όταν ο χρήστης θέλει να αποθηκεύσει την κατάσταση της εικονικής μηχανής του, αποθηκεύει ένα **στιγμιότυπο (snapshot)**. Με το στιγμιότυπο το σύνολο των δεδομένων (τόσο από τους σκληρούς δίσκους όσο και από την μνήμη RAM) του εικονικού μηχανήματος αποθηκεύονται και μπορούν σε μελλοντικό χρόνο να επαναφερθούν και η χρήση του μηχανήματος να συνεχίσει από εκείνο το σημείο. Αυτό είναι και από τα χρησιμότερα χαρακτηριστικά μίας εικονικής μηχανής καθώς καθιστά την επαναφορά της σε παρελθοντική κατάσταση ιδιαίτερα εύκολη (πχ μετά από κάποια αστοχία).

1.4.2 Migration

Με τον όρο migration αναφερόμαστε στη μετακίνηση μίας εικονικής μηχανής από ένα φυσικό μηχάνημα σε κάποιο άλλο. Αυτό γίνεται είτε για λόγους ανακατανομής του φόρτου εργασίας είτε για τη συντήρηση του συστήματος είτε λόγω κάποιας αστοχίας του φυσικού μηχανήματος.

Ενώ σε κάποιες περιπτώσεις η μετακίνηση αυτή είναι απαραίτητη (συντήρηση - αστοχίες) σε άλλες είναι αρκετά απλό να αποφευχθεί. Κατά τη μετακίνηση μίας εικονικής μηχανής η κατάσταση της μνήμης αντιγράφεται από το ένα φυσικό μηχάνημα στο άλλο (είτε on the fly είτε μετά από αναστολή της λειτουργίας της). Η διαδικασία αυτή είναι ιδιαίτερα απαιτητική σε χρόνο και ενέργεια, οπότε είναι σημαντικό να γίνεται όσο το δυνατό πιο σπάνια.

Γενικά για να είναι αποδοτικό ένα σύστημα, πρέπει οι πόροι του να βρίσκονται σε χρήση όσο το δυνατό περισσότερο. Μετά από κάποια ώρα όμως είναι πολύ πιθανό να υπάρξει κάποια ανισορροπία στο φόρτο των επεξεργαστών, δηλαδή ενώ κάποιος δεν θα εκτελεί μία εργασία ένας άλλος θα είναι ιδιαίτερα επιβαρυμένος. Σε αυτή την περίπτωση γίνεται ανακατανομή του φόρτου και η εκτέλεση των εργασιών μοιράζεται εκ νέου στα διαθέσιμα μηχανήματα. Είναι ευθύνη του λειτουργικού συστήματος και του χρονοδρομολογητή να κάνει αυτές τις ανισορροπίες και τις επακόλουθες ανακατανομές όσο το δυνατό λιγότερες. Έτσι με μια πολιτική δρομολόγησης μπορεί να έχουμε πολύ συχνές μετακινήσεις εικονικών μηχανών από το ένα μηχάνημα στο άλλο, ενώ με μία καταλληλότερη οι μετακινήσεις αυτές να μειωθούν στο ελάχιστο.

1.5 Ορισμός του προβλήματος

Είναι κατανοητό πως όταν πολλές διεργασίες μοιράζονται κοινούς πόρους συνολική απόδοση του συστήματος ως ένα βαθμό περιορίζεται. Έχουν προταθεί λοιπόν διάφορες πολιτικές

δρομολόγησης ώστε να περιοριστεί ο ανταγωνισμός στο ελάχιστο. Όταν ένας χρήστης μισθώνει εικονικές μηχανές σε έναν πάροχο πληρώνει την ώρα που βρίσκονται οι εικονικές του μηχανές πάνω στη cpu. Κάτι τέτοιο σημαίνει πως είναι σημαντικό αυτή η ώρα να είναι η ελάχιστη δυνατή. Είναι ταυτόχρονα προφανές πως όσο μικρότερος είναι ο ανταγωνισμός, τόσο καλύτερη είναι και χρήση των πόρων ενός συστήματος με αποτέλεσμα την ελαχιστοποίηση τόσο του κόστους λειτουργίας, όσο και της κατανάλωσης ενέργειας. Στόχος μας λοιπόν είναι να μελετηθεί η επίδραση που ασκεί η μία εικονική μηχανή σε αυτή με την οποία μοιράζεται τους πόρους συναρτήσει της γενικότερης συμπεριφοράς τους και των απαιτήσεών τους στους κοινούς πόρους.

1.5.1 Η δική μας προσέγγιση

Γνωρίζοντας εκ των προτέρων λεπτομέρειες της αρχιτεκτονικής του συστήματος, όπως την ιεραρχία της μνήμης και τους κοινούς πόρους είμαστε σε θέση να προβλέψουμε τον ανταγωνισμό και να υπολογίσουμε την επίδραση που θα έχει αυτός στην εκτέλεση των εφαρμογών. Σε αυτή την εργασία, μελετούμε την παράλληλη εκτέλεση εικονικών μηχανών και προσπαθούμε να αναλύσουμε την επίδραση που έχει ο ανταγωνισμός για τους κοινούς πόρους στην φιλοξενία τους σε ένα σύστημα. Για τον σκοπό αυτό χρησιμοποιούμε προγράμματα ελέγχου (benchmarks) τα οποία εκτελούνται σε περιβάλλον εικονοποίησης. Τα προγράμματα αυτά υλοποιούν διάφορους αλγορίθμους, κυρίως γραμμικής άλγεβρας και το καθένα έχει διαφορετικές απαιτήσεις τόσο σε μνήμη όσο και στον δίαυλο προς αυτή. Αφού τα διαχωρίσουμε με βάση την ανάγκες τους σε διαφορετικές κλάσεις (classification), κάνουμε πειράματα με όλους τις πιθανούς συνδυασμούς κλάσεων ώστε να δούμε με ποια πολιτική δρομολόγησης έχουμε την καλύτερη επίδοση. Ιδιαίτερο ενδιαφέρον παρουσιάζει και το κατά πόσο το εικονικό περιβάλλον στο οποίο εκτελούνται επηρεάζει την απόδοσή τους.

Για τους σκοπούς της εργασίας, χρησιμοποιούμε διάφορους χρονοδρομολογητές όπως ο CFS του linux καθώς και πολιτικές που προσπαθούν να εξισορροπήσουν διάφορους παράγοντες που επηρεάζουν την επίδοση όπως τα cache misses. Τέλος δίνουμε ιδιαίτερη βάση στους χρονοδρομολογητές lca και throughput-balance που έχουν υποσχόμενα αποτελέσματα σε φυσικά μηχανήματα.

1.6 Περιγραφή κεφαλαίων

Στο δεύτερο κεφάλαιο αναλύουμε σε μεγαλύτερο βάθος το πρόβλημα διατυπώνοντας τους παράγοντες από τους οποίους αποτελείται. Επίσης αναφέρουμε σχετική έρευνα και το τρόπο που έχουν προσεγγίσει άλλοι για την επίλυσή του από τη βιβλιογραφία.

Στο τρίτο κεφάλαιο περιγράφουμε το σύστημα και τις συνθήκες κάτω από τις οποίες εκτελέστηκαν τα πειράματα και παρουσιάζουμε με συντομία τους υπόλοιπους χρονοδρομολογητές που επιλέξαμε για τις μετρήσεις, ενώ στο τέταρτο παρουσιάζουμε τις μετρήσεις από όλους τους πιθανούς συνδυασμούς κλάσεων καθώς και τα αποτελέσματά τους.

Στο πέμπτο κεφάλαιο περιγράφουμε μία σειρά από σενάρια που επιλέξαμε να εκτελέσουμε χρησιμοποιώντας διαφορετικές πολιτικές τις οποίες προσπαθούμε να συγκρίνουμε. Τα σενάρια είναι σχεδιασμένα για να αποκαλύψουν τα θετικά και τα αρνητικά στοιχεία των διάφορων πολιτικών. Επίσης παρουσιάζουμε τα αποτελέσματά αυτών των μετρήσεων. Τέλος το έκτο κεφάλαιο περιέχει μία ανακεφαλαίωση και τα συμπεράσματα της εργασίας καθώς και μελλοντική δουλειά που μπορεί να ακολουθήσει ώστε να απαντηθούν σημαντικά ερωτήματα.

Κεφάλαιο 2

Κίνητρο και σχετική έρευνα

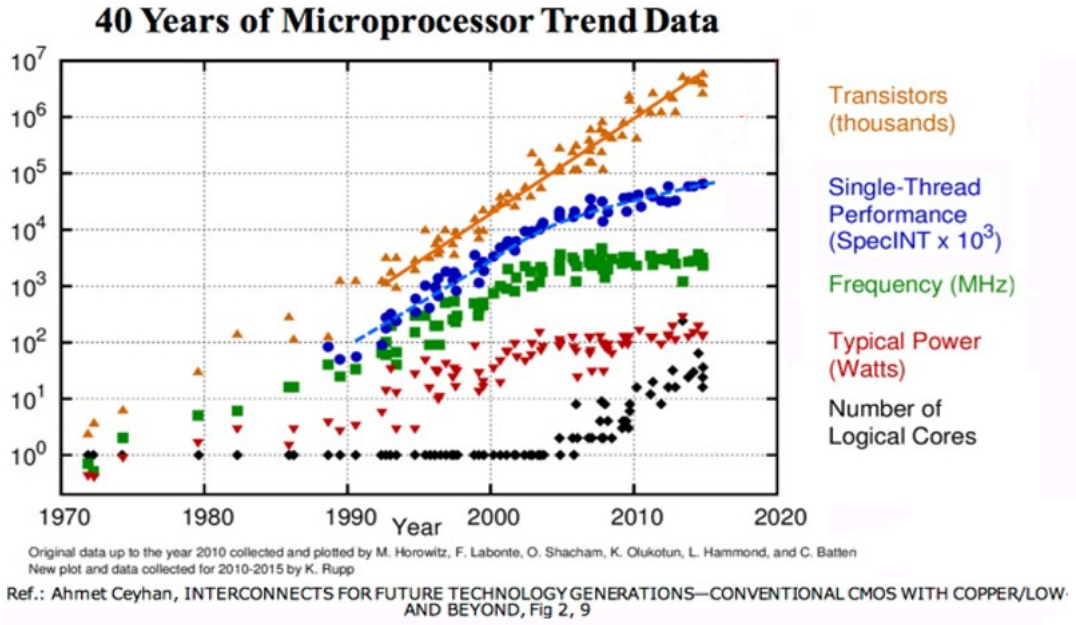
2.1 Θεωρητικό υπόβαθρο

2.1.1 Υλικό υπολογιστών και πρόοδος

Σε αυτό το σημείο θα αναλυθεί με συντομία η πρόοδος που επιτεύχθηκε στο υλικό των υπολογιστών με τη πάροδο των χρόνων, καθώς και η επίπτωση που έχει αυτή στα λειτουργικά συστήματα.

CPU και υπολογιστική ισχύς

Τον Απρίλιο του 1965, όταν του ζητήθηκε ο Gordon E. Moore -ένας από τους συνιδρυτές της εταιρείας κατασκευής ολοκληρωμένων κυκλωμάτων Intel- προέβλεψε πως ο αριθμός των τρανζίστορ σε θα διπλασιαζόταν κάθε χρόνο, ενώ αργότερα αναθεώρησε στα δύο χρόνια. Η σημαντικότητα αυτής της πρόβλεψης (που καθιερώθηκε γνωστή ως **Νόμος του Moore**) φαίνεται από το γεγονός πως ενώ αναφερόταν για μία δεκαετία, πολλά χρόνια μετά φάνηκε πως ήταν ακόμα σε ισχύ.



Σχήμα 2.1: Νόμος του Moore

Όπως φαίνεται και από το παραπάνω διάγραμμα (Σχήμα 2.1) η απόδοση των επεξεργαστών μέχρι ένα χρονικό σημείο ήταν παράλληλη με τον αριθμό των τρανζίστορ και την αντίστοιχη αύξηση της συχνότητας του ρολογιού. Το σημείο αυτό είναι το σημείο που φυσικοί περιορισμοί άρχισαν να εμποδίζουν την μείωση του μεγέθους των τρανζίστορ με ταυ-

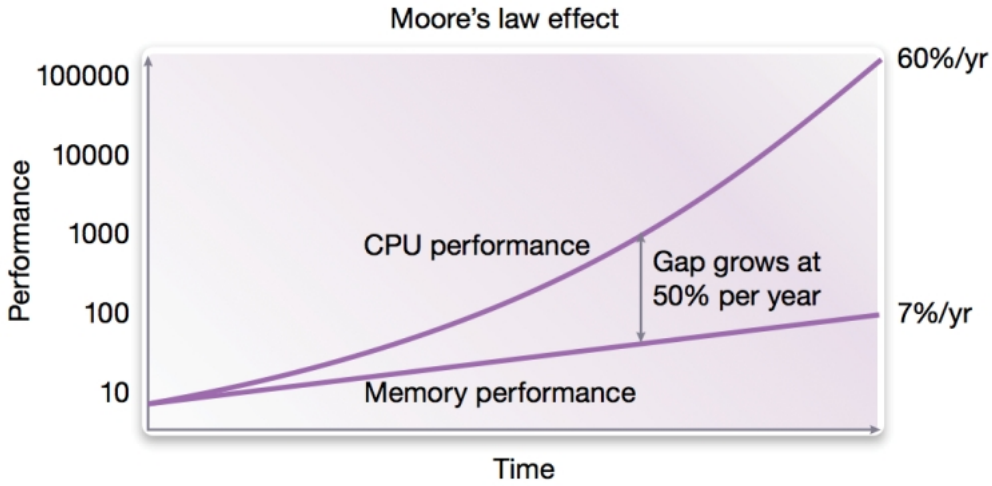
τόχρονη αύξηση της συχνότητας. Η έγκαιρη απομάκρυνση της εκλυόμενης θερμότητας από το τσιπ γίνεται αδύνατη με αποτέλεσμα την κατακόρυφη αύξηση της θερμοκρασίας και κατ' επέκταση την εμφάνιση κβαντομηχανικών φαινομένων που καθιστούν τη λειτουργία του επεξεργαστή αδύνατη.

Η απάντηση των αρχιτεκτόνων ήταν η υιοθέτηση των πολυπύρηνων αρχιτεκτονικών. Πλέον η λύση της προόδου έπαψε να βρίσκεται στην αύξηση της συχνότητας και το ζήτημα της απόδοσης ενός υπολογιστικού συστήματος πέρασε σε πολλές διαστάσεις. Τώρα από τη πλευρά της αρχιτεκτονικής και του υλικού το ζητούμενο είναι η σωστή και αποδοτική συνεργασία μεταξύ των πυρήνων, με το μεγαλύτερο μέρος να είναι κυρίως στην επικοινωνία και έπειτα στην επέκταση των δυνατοτήτων των κρυφών μνημών (διαμοιρασμός των block μεταξύ των πυρήνων για γρηγορότερη προσπέλαση, συνεκτικότητα κρυφής μνήμης με πρωτόκολλο MESIF, κλπ). Ωστόσο, το περισσότερο βάρος πέφτει στον προγραμματισμό του συστήματος. Για να αξιοποιηθεί σε όλο του το εύρος το νέο σύστημα πρέπει να γίνεται πλήρης χρησιμοποίηση του υλικού και των παραλληλισμών που αυτό προσφέρει.

Οι προγραμματιστές λοιπόν είναι αυτοί που έχουν πλέον τον μεγαλύτερο λόγο και την ευθύνη να αξιοποιήσουν το σύστημα κατάλληλα ώστε να γίνεται απόλυτη χρήση των διαθέσιμων πόρων.

Μνήμη και ταχύτητα προσπέλασης

Ο Moore εκτός του περίφημου νόμου για την υπολογιστική ισχύ, διατύπωσε και την πεποίθηση πως μεγάλο πρόβλημα στο μέλλον θα είναι και το λεγόμενο memory bottleneck. Την ραγδαία αύξηση στην απόδοση των CPU δεν μπορεί να ακολουθήσει ο ρυθμός με τον οποίο βελτιώνεται η μνήμη των υπολογιστών. Υπάρχει μία σταθερή διαφορά της τάξης του 50% κάθε χρόνο μεταξύ του ρυθμού βελτίωσης των CPU και του ρυθμού αύξησης της ταχύτητας προσπέλασης στη μνήμη. Κάτι τέτοιο φαίνεται και από το Σχήμα 2.2. Αυτό σημαίνει πως σε βάθος χρόνου η συνολική επίδοση ενός υπολογιστικού συστήματος θα περιορίζεται σημαντικά από τις αιτήσεις στη κεντρική μνήμη. Δηλαδή οι εφαρμογές χαραμίζουν σημαντικό μέρος του χρόνου που εκτελούνται στο να περιμένουν τα δεδομένα που χρειάζονται από τη μνήμη.



Σχήμα 2.2: Πρόοδος CPU - Μνήμης

Όπως και νωρίτερα, έτσι και σε αυτή την περίπτωση, η λύση έρχεται από τον τρόπο με τον οποίο το λογισμικό χρησιμοποιεί του πόρους. Ο σωστός σχεδιασμός ενός λειτουργικού συστήματος, προϋποθέτει την αντιμετώπιση των παραπάνω προβλημάτων. Στην περίπτωση

των μνημών, αναμένεται κατά την διάρκεια μεταφοράς δεδομένων από τα κατώτερα στρώματα της ιεραρχίας της μνήμης κάποια άλλη εφαρμογή να έρθει προς εκτέλεση στον επεξεργαστή.

2.1.2 Η έννοια του παραλληλισμού

Όπως αναφέρθηκε και νωρίτερα με τη χρήση των παράλληλων συστημάτων εμφανίστηκε η ανάγκη δημιουργίας νέου λογισμικού που να εκμεταλλεύεται όλα τα πλεονεκτήματα που αποφέρουν οι νέες αρχιτεκτονικές. Με τις έννοιες του παράλληλου και του πολυνηματικού προγραμματισμού οι μηχανικοί λογισμικού καλούνται να αντιμετωπίσουν τα παραπάνω προβλήματα.

Παράλληλος προγραμματισμός και μοντέλα

Παράλληλος προγραμματισμός είναι ένα μοντέλο που βοηθάει στη δημιουργία παράλληλων εφαρμογών. Ανάλογα με τα εργαλεία και τη γλώσσα προγραμματισμού που χρησιμοποιεί, ο προγραμματιστής είναι σε θέση να επιλέξει αν το λειτουργικό σύστημα θα είναι αυτό που δεσμεύει πόρους και θα τους μοιράζει ανάλογα ή αν θα καθορίζεται από πριν ο τρόπος που τα παράλληλα νήματα μίας εφαρμογής θα δεσμεύουν τους πόρους και ο τρόπος που αυτά θα επικοινωνούν μεταξύ τους.

Διακρίνονται τα εξής μοντέλα παράλληλου προγραμματισμού, το κάθε ένα από τα οποία προτιμάται για διαφορετικές αρχιτεκτονικές.

- Μοιραζόμενη μνήμη (Shared Memory) είναι μοντέλο παράλληλου προγραμματισμού που χρησιμοποιείται κυρίως σε αρχιτεκτονικές όπου οι εφαρμογές έχουν πρόσβαση σε μία καθολική μνήμη. Τα νήματα που χρησιμοποιούν την μοιραζόμενη μνήμη έχουν κοινό χώρο διευθύνσεων όπου μπορούν να διαβάζουν και να γράφουν ασύγχρονα. Προκειμένου να διατηρηθεί η συνάφεια της μνήμης και να ελεγχθεί η παράλληλη προσπέλασή της είναι αναγκαίοι μηχανισμοί ελέγχου και κλειδώματος (monitors, semaphores και locks).

- Μηνύματα (Message Passing). Πρόκειται για ένα μοντέλο που στηρίζεται στην αποστολή μηνυμάτων μεταξύ των διεργασιών. Έτσι μία διεργασία επικοινωνεί με τις υπόλοιπες με το πέρασμα των μηνυμάτων (MPI - Message Passing Protocol). Η επικοινωνία μπορεί να είναι είτε σύγχρονη είτε ασύγχρονη. Συνήθως αυτό το μοντέλο χρησιμοποιείται όταν η αρχιτεκτονική του συστήματος περιλαμβάνει κατανεμημένη μνήμη, μοιρασμένη σε διαφορετικούς πυρήνες.

- Σιωπηλός παραλληλισμός (Implicit parallelism) είναι μοντέλο όπου ο προγραμματιστής δεν έχει καμία γνώση γύρω από τις επαφές των διεργασιών, ενώ οι μεταγλωττιστές αυτόματα εξάγουν τις δυνατότητες παραλληλισμού διάφορων σημείων ενός προγράμματος. Σε αυτή την περίπτωση ο προγραμματιστής δεν χρειάζεται να ασχοληθεί με τη σωστή κατάτμηση του προγράμματος και την επικοινωνία των διάφορων νημάτων. Ενώ είναι το πιο εύκολο μοντέλο για προφανείς λόγους δεν προσφέρει την ευελιξία των προηγούμενων.

Τέλος μπορεί κανείς να διακρίνει δύο σημαντικά επίπεδα όπου μπορεί να προκύψει παραλληλισμός: Παραλληλισμός σε επίπεδο έργου (**Task Parallelism**) και παραλληλισμός σε επίπεδο δεδομένων (**Data Parallelism**).

Με τον πρώτο τύπο, μία εφαρμογή χωρίζεται σε επιμέρους εργασίες που λειτουργούν πάνω στα δεδομένα. Αυτές όσο παραμένουν η μία ανεξάρτητη από τις άλλες, μπορούν να

τρέχουν παράλληλα και ταυτόχρονα, ωστόσο λόγω της χρήσης κοινών δεδομένων υπάρχουν περιπτώσεις που η εκτέλεσή τους πρέπει να σειριοποιείται.

Στον δεύτερο τύπο, τα δεδομένα πάνω στα οποία λειτουργεί και τα οποία επεξεργάζεται η εφαρμογή διαμοιράζονται σε παράλληλους υπολογιστικούς κόμβους. Κάτι τέτοιο σημαίνει πως η ίδια εργασία μπορεί να εκτελεστεί παράλληλα σε διαφορετικά μέρη του συνόλου των δεδομένων. Έτσι για παράδειγμα, όταν πρέπει να γίνουν υπολογισμοί σε έναν πίνακα και κάθε στοιχείο του πίνακα είναι ανεξάρτητο από τα υπόλοιπα, αυτός μπορεί να χωριστεί σε τόσα κομμάτια όσα είναι και οι υπολογιστικοί πυρήνες του συστήματος και οι υπολογισμοί σε κάθε τμήμα να πραγματοποιηθούν ταυτόχρονα.

Ζητήματα - Challenges

Ενώ με τις παράλληλες αρχιτεκτονικές λύνονται πολλά προβλήματα και όπως αναμένεται οι δυνατότητες ενός συστήματος αυξάνονται ραγδαία υπάρχουν δύο βασικά ζητήματα που πρέπει να απαντηθούν ώστε το τελικό σύστημα να είναι πραγματικά αποδοτικό. Το πρώτο έχει να κάνει με το εύρος του διαύλου προς την κύρια μνήμη. Οι διαφορετικοί πυρήνες ανταγωνίζονται για τη πρόσβαση στη μνήμη και αυτός ο ανταγωνισμός είναι ικανός να περιορίσει σημαντικά το σύστημα. Το δεύτερο και εξίσου σημαντικό ζήτημα αφορά την επικοινωνία μεταξύ των πυρήνων και την αποδοτική αξιοποίηση των μοιραζόμενων επιπέδων κρυφής μνήμης. Αν δεν αντιμετωπιστεί θα παρουσιάζεται συχνά το φαινόμενο όπου ο ένας πυρήνας ακυρώνει τα δεδομένα που χρησιμοποιεί ο άλλος (cache thrashing). Η συνάφεια την κρυφής μνήμης διασφαλίζεται με προηγμένα πρωτόκολλα όπως το MESI και πιο πρόσφατα το MESIF.

Και στις δύο περιπτώσεις ο ανταγωνισμός για τους μοιραζόμενους πόρους μπορεί να αποβεί μοιραίος για την απόδοση του συστήματος και η αντιμετώπισή του έχει κρίσιμο ρόλο στη διαμόρφωση του τελικού συστήματος. Ο ανταγωνισμός δεν μπορεί να αντιμετωπιστεί παρά μόνο στα χαμηλότερα επίπεδα και όχι σε αυτό των εφαρμογών. Έτσι είναι ευθύνη του λειτουργικού συστήματος να καθορίσει τον τρόπο με τον οποίο οι κοινοί πόροι μοιράζονται στις εφαρμογές ώστε να περιοριστεί το φαινόμενο κατά το δυνατό. Για τον λόγο αυτό οι ερευνητές έχουν στραφεί στους Contention-aware schedulers που έχουν αποφέρει υποσχόμενα αποτελέσματα όσον αφορά τη συνολική απόδοση.

2.1.3 Νεφος - Cloud Computing

Όπως αναφέρθηκε και στο εισαγωγικό κεφάλαιο, χρησιμοποιούμε τον όρο **Νέφος (cloud)** για να αναφερθούμε σε ένα σύνολο υπηρεσιών που παρέχονται συνήθως μέσω διαδικτύου και δίνει στον τελικό χρήστη τη δυνατότητα να χρησιμοποιήσει απομακρυσμένους υπολογιστικούς πόρους. Οι υπηρεσίες και οι δυνατότητες που προσφέρει ένα νέφος, το κατατάσσουν σε μία από τις τρεις βασικές κατηγορίες που αναπτύχθηκαν στο πρώτο κεφάλαιο: Software as a Service, Platform as a Service ή Infrastructure as a Service.

Ο τομέας του νέφους έχει τεράστια ζήτηση τα τελευταία χρόνια. Είναι τόσο μεγάλο το εύρος των υπηρεσιών που είναι προφανές πως υπάρχουν σημαντικά ζητήματα σε εντελώς διαφορετικά επίπεδα που όμως όλα αφορούν τελικά το νέφος. Έτσι είναι επόμενο πως αυτός ο τομέας έχει τραβήξει το ενδιαφέρον των ερευνητών. Είναι δύσκολο και εκτός θέματος της παρούσης εργασίας να αναλυθούν σε βάθος τα διάφορα χαρακτηριστικά και τα ζητήματα που υπάρχουν, ωστόσο κρίνεται αναγκαίο να αναφερθούν επιγραμματικά μερικά από αυτά.

Οι εφαρμογές του νέφους ποικίλουν και στα ψηλότερα αφαιρετικά επίπεδα μπορούμε να συναντήσουμε web servers, υπηρεσίες email, απομακρυσμένες βάσεις δεδομένων κλπ. Επίσης είναι πολύ διαδεδομένες υπηρεσίες ανάλυσης δεδομένων. Με την ολοένα και περισσότερο αυξανόμενη αγορά των έξυπνων συσκευών τα διαθέσιμα δεδομένα αποκτούν γιγα-

ντιαίες διαστάσεις και η αποδοτική επεξεργασία τους για εξόρυξη και χρήση πληροφοριών γίνεται εφικτή μέσω συστημάτων cloud. Πολλές εταιρείες παρέχουν τις υπηρεσίες εξολοκλήρου βασισμένες σε τέτοια συστήματα. Σε χαμηλότερα επίπεδα, συναντάμε σαν υπηρεσίες ολόκληρες υποδομές (IaaS). Εδώ θα δούμε ενοικιαζόμενες εικονικές μηχανές, servers, αποθηκευτικούς χώρους, δίκτυα συστημάτων κλπ.

Με το νέφος ο τομέας της πληροφορικής μπήκε σε νέα περίοδο, αυτή των απομακρυσμένων υπηρεσιών. Τα οφέλη είναι πάρα πολλά και τα περισσότερα προφανή. Ωστόσο δεν λείπουν και τα μειονεκτήματα με βασικότερο όλων την περιορισμένη ασφάλεια που έχει ο τελικός χρήστης. Τα δεδομένα πλέον παύουν να βρίσκονται στα χέρια του και τα εμπιστεύεται σε χέρια τρίτων, ενώ δεν έχουν λείψει επιθέσεις σε παρόχους τέτοιου τύπου υπηρεσιών με αποτέλεσμα μεγάλες ποσότητες ιδιωτικών και προσωπικών δεδομένων να διαρρεύσουν.

2.1.4 Εικονοποίηση - Virtualization

Το κεφάλαιο της εικονοποίησης έχει αναφερθεί ήδη στην εισαγωγή. Ωστόσο σε αυτό το κεφάλαιο θα εισέλθουμε σε μεγαλύτερο βάθος για να ξεχωρίσουμε επιγραμματικά τους βασικούς τύπους εικονοποίησης. Οι βασικότερες κατηγορίες εικονοποίησης είναι δύο: **Πλήρης εικονοποίηση (Full Virtualization)** και ***παραεικονοποίηση (Paravirtualization)***

- Με την **πλήρη εικονοποίηση** οι εικονικές μηχανές προσομοιώνουν ένα περιβάλλον σχεδόν πανομοιότυπο με ένα φυσικό μηχάνημα. Με τον τρόπο αυτό το εικονικό μηχάνημα μπορεί να ”τρέχει” λειτουργικό σύστημα (guest) διαφορετικό από αυτό που τρέχει στο φυσικό (host). Σε κάθε περίπτωση, με υποβοηθούμενη από το hardware εικονοποίηση τα αποτελέσματα είναι εντυπωσιακά και η απόδοση πλησιάζει αρκετά την αντίστοιχη των bare-metal εκτελέσεων. Ωστόσο η τελική απόδοση είναι σαφώς χειρότερη από αυτή της παραεικονοποίησης ή όπως θα δούμε αργότερα της εικονοποίησης σε επίπεδο λειτουργικού συστήματος.

  Η πλήρης εικονοποίηση, ακριβώς επειδή προσπαθεί να προσομοιώσει ένα ολόκληρο σύστημα παρουσιάζει το μεγαλύτερο επίπεδο ασφάλειας. Ωστόσο έχουν υπάρξει περιπτώσεις που η ασφάλεια τόσο της εικονικής μηχανής όσο και του host συστήματος τίθεται υπό αμφισβήτηση. Ενδεικτικά παραδείγματα αποτελούν το CVE-2009-1244 όπου ήταν δυνατό να ξεφύγει κανείς από το guest μηχάνημα και να εκτελέσει αυθαίρετο κώδικα στο host (VMWare Workstation), όπως και διάφορες αδυναμίες που έχουν εντοπιστεί σε συστήματα όπως το Xen που προκαλούν Denial of Service.

- Η ***παραεικονοποίηση*** προσφέρει ένα παρόμοιο περιβάλλον με το φυσικό μηχάνημα και έχει ως στόχο καλύτερες επιδόσεις. Προκειμένου να μην προσομοιώνονται απαιτητικές διαδικασίες στο εικονικό περιβάλλον, δημιουργούνται οι κατάλληλες διεπαφές ούτως ώστε οι διαδικασίες να εκτελούνται απ’ ευθείας στον host. Κάτι τέτοιο ωστόσο ενώ αποφέρει σημαντικές βελτιώσεις στην επίδοση, έχει και κάποια σημαντικά μειονεκτήματα. Αρχικά, το φιλοξενούμενο λειτουργικό σύστημα για να είναι σε θέση να χρησιμοποιεί το API του host λειτουργικού χρειάζεται κάποιες τροποποιήσεις. Αυτό έχει ως αποτέλεσμα ο τροποποιημένος πυρήνας να διαφέρει και μην αναβαθμίζεται με την ίδια συχνότητα που αναβαθμίζονται mainline εκδόσεις κάτι που έχει ως συνέπεια να χάνει σημαντικές ενημερώσεις ασφαλείας. Επίσης, ακριβώς επειδή πλέον το guest μηχάνημα επικοινωνεί πολύ συχνότερα με τον host είναι πιο εύκολο ο χρήστης να διαφύγει από το εικονικό προς το φυσικό μηχάνημα.

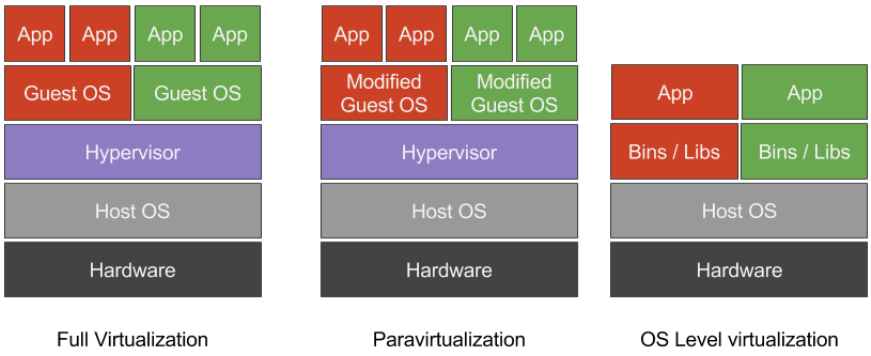
  Αυτή η μορφή υποστηρίζεται από πολλές πλατφόρμες όπως το Xen που βασίζεται κυρίως στο Qemu, καθώς και από το KVM που παρέχει υποστήριξη για αρκετές συσκευές χρησιμοποιώντας το VirtIO API. Το τελευταίο παρέχει διεπαφές για άμεση εκτέλεση

IO αιτημάτων που προσεγγίζουν την bare metal επίδοση και ξεπερνούν κατά πολύ τα αντίστοιχα IO της πλήρους εικονοποίησης.

- Τέλος με την χρήση **εικονοποίηση επιπέδου λειτουργικού συστήματος** ο πυρήνας του λειτουργικού επιτρέπει την ύπαρξη πολλαπλών χώρων χρήστη (containers) οι οποίοι είναι ανεξάρτητοι ο ένας από τον άλλον. Αυτό πρακτικά σημαίνει πως οι εφαρμογές νομίζουν πως εκτελούνται σε ξεχωριστούς υπολογιστές, πλήρως απομονωμένες από αυτές που τρέχουν σε διαφορετικό container. Αυτός ο τύπος εικονοποίησης έχει βρει μεγάλη άνθηση τα τελευταία χρόνια, με χαρακτηριστικότερο παράδειγμα το Docker.

  Από τη μία φέρει πολλά πλεονεκτήματα, όπως ελαστικότητα και πολύ καλή απόδοση. Οι εφαρμογές παρουσιάζουν ελάχιστο έως καθόλου overhead αφού πλέον χρησιμοποιούν άμεσα τις κλήσεις συστήματος του λειτουργικού χωρίς να χρειάζονται απαιτητικές προσομοιώσεις. Ταυτόχρονα, αποτελεί έναν πολύ κομψό και εύκολο τρόπο απομόνωσης των εφαρμογών αφού κάθε εφαρμογή χρησιμοποιεί ξεχωριστό χώρο που ενδεχομένως να έχει πρόσβαση σε διαφορετικές συσκευές και πόρους από άλλες που ο χρήστης θέλει να προστατέψει. Από την αρνητική πλευρά, δεν προσφέρει σε καμία περίπτωση την ευελιξία των προηγούμενων τύπων. Αφού δεν προσομοιώνει κάποιο σύστημα, είναι υποχρεωμένος να περιορίζεται σε εφαρμογές σχεδιασμένες για το συγκεκριμένο σύστημα (με την ίδια αρχιτεκτονική κλπ). Επίσης το θέμα της ασφάλειας είναι σημαντικό καθώς φαντάζει εύκολο μία εφαρμογή να καταφέρει να ξεπεράσει τα όρια του χώρου της και να αποκτήσει πρόσβαση σε ξένα containers (διαδικασία γνωστή ως jailbreaking).

Ο κάθε τύπος εικονοποίησης έχει δικά του θετικά και αρνητικά. Η επιλογή του κατάλληλου τύπου εικονοποίησης εξαρτάται καθαρά από τις ανάγκες του χρήστη. Στο επόμενο σχήμα βλέπουμε γραφικά τις διαφορές του κάθε τύπου.



Σχήμα 2.3: Τύποι εικονοποίησης

2.1.5 Χρονοπρογραμματισμός

Ένα από τα βασικά ζητήματα που καλείται να αντιμετωπίσει ένα λειτουργικό σύστημα είναι ο διαμοιρασμός του χρόνου που οι διεργασίες θα έχουν στον επεξεργαστή. Ανάλογα με τη χρήση για την οποία προορίζεται ένα σύστημα απαιτείται διαφορετική πολιτική για την εύρυθμη λειτουργία του. Για παράδειγμα ένας υπολογιστή γενικού σκοπού είναι απαραίτητο να είναι αποκρίσιμος στην είσοδο του χρήστη, ενώ ένας υπερυπολογιστής στον οποίο ανατίθενται διάφορες εργασίες πρέπει να καταναλώνει την ελάχιστη δυνατή ενέργεια, οπότε

απαιτεί πολιτική που να εξασφαλίζει πως οι υπολογισμοί θα ολοκληρωθούν το συντομότερο δυνατό.

Βασικά χαρακτηριστικά χρονοδρομολογητή

Κάθε πολιτική χρονοδρομολόγησης έχει κάποια χαρακτηριστικά που είναι ικανά να κρίνουν πότε αυτή είναι ικανή να αντιμετωπίσει ένα πρόβλημα ή όχι. Αυτά είναι:

- **Throughput**: Δηλώνει τη συνολική πρόοδο που σημειώνεται ανα μονάδα χρόνου.
- **Latency**: Χρόνος από την υποβολή ενός αιτήματος μέχρι να παραχθεί η πρώτη απόκριση.
- **Fairness**: Είναι η προσπάθεια της πολιτικής να είναι δίκαιη προς τις διεργασίες, είτε με το να τους αναθέτει ίσο χρόνο στον επεξεργαστή είτε με το να ακολουθεί συγκεκριμένο πρόγραμμα βάση της προτεραιότητας που έχει κάθε διεργασία.
- **Waiting time**: Είναι ο μέσος χρόνος που μία διεργασία περιμένει στην ουρά των προς εκτέλεση διεργασιών (ready queue).

Έτσι όπως περιγράφηκε παραπάνω, σε ένα σύστημα γενικού σκοπού ενδιαφέρει το χαμηλό waiting time το fairness και το latency - χαρακτηριστικά που καθιστούν το σύστημα αποκρίσιμο, ενώ σε έναν υπερυπολογιστή του υψηλό throughput έχει προτεραιότητα - αφού ενδιαφέρει ανα μονάδα χρόνου να παράγεται σημαντική ποσότητα έργου.

Βασικές πολιτικές χρονοδρομολόγησης

- **First Come-First Serve (FCFS)**: Πρόκειται για την απλούστερη μέθοδο. Ταξινομεί τις διεργασίες με τη σειρά που έρχονται στη λίστα με τις προς εκτέλεση εφαρμογές και στη συνέχεια τους δίνει χρόνο στη cpu με την ίδια σειρά. Το overhead των χρονοδρομολογητών υλοποιούν αυτή την πολιτική είναι πολύ μικρό, ακριβώς επειδή δεν έχουν να πάρουν κάποια δύσκολη απόφαση και οι λίστες ή οι προτεραιότητες δεν αλλάζουν στην πορεία. Ωστόσο υστερεί σημαντικά σε άλλους τομείς. Το throughput θα είναι χαμηλό αφού διεργασίες θα καταλαμβάνουν τον επεξεργαστή ακόμα και όταν περιμένουν για δεδομένα, ενώ για τον ίδιο λόγο θα παρουσιάζονται υψηλά latency και waiting time. Τελικά είναι μία αρκετά μη-αποδοτική μέθοδος αφού το σύστημα δύσκολα θα ανταποκρίνεται σε διορίες λόγω έλλειψης προτεραιοτήτων.
- **Shortest Job First (SJF)**: Είναι μία πολιτική που εγγυάται το ελάχιστο χρόνο αναμονής. Αυτή η μέθοδος απαιτεί γνώση τουλάχιστον κατά προσέγγιση του χρόνου που χρειάζονται οι διεργασίες για να ολοκληρωθούν. Στην περίπτωση που προστεθεί μία διεργασία με χρόνο εκτέλεσης μικρότερο από αυτόν της διεργασίας που εκτελείται εκείνη τη στιγμή θα πρέπει να γίνει αντικατάστασή της (context switch) που σημαίνει αύξηση του μέσου overhead. Επίσης υπάρχει σημαντικός κίνδυνος οι διεργασίες με μεγαλύτερο εκτιμώμενο χρόνο εκτέλεσης να μην πάρουν ποτέ χρόνο στον επεξεργαστή (starvation) αν έρχονται συνεχώς προς εκτέλεση ”μικρότερες” διεργασίες.
- **Fixed priority preemptive scheduling**: Εδώ ο χρονοδρομολογητής ταξινομεί τις διεργασίες βάση προτεραιοτήτων. Όταν μία διεργασία έρθει στην ουρά των έτοιμων προς εκτέλεση διεργασιών και έχει μεγαλύτερη προτεραιότητα από αυτή που εκτελείται τη δεδομένη χρονική στιγμή, θα πάρει τη θέση της στον επεξεργαστή. Έχει πολλά θετικά,

καθώς αντιμετωπίζει τα προβλήματα των υπόλοιπων πολιτικών. Το overhead που παρατηρείται δεν είναι το ελάχιστο δυνατό, αλλά δεν είναι και σημαντικά υψηλό. Οι διορίες μπορούν να επιτευχθούν με το να δοθεί μεγαλύτερη προτεραιότητα στις αντίστοιχες διεργασίες. Τέλος το φαινόμενο της λιμοκτονίας (starvation) μπορεί να αποφευχθεί με το να αυξάνεται η προτεραιότητα των διεργασιών όσο περνάει ο χρόνος.

- **Round Robin**: Με την Round Robin πολιτική ο χρόνος διαιρείται σε μικρά τμήματα (time quantum) και κάθε διεργασία καταναλώνει από ένα τμήμα στον επεξεργαστή. Η εναλλαγή των διεργασιών γίνεται κυκλικά. Συγκριτικά με τις προηγούμενες πολιτικές, οι διεργασίες που απαιτούν μικρό χρονικό διάστημα για να εκτελεσθούν ολοκληρώνονται συντομότερα απ' ό,τι με τη FCFS ενώ αυτές που απαιτούν μεγάλο χρονικό διάστημα ολοκληρώνονται συντομότερα απ' ό,τι με τη SJF. Έτσι το μέσο bandwidth βρίσκεται κάπου μεταξύ των δύο πολιτικών. Οι διεργασίες δεν απειλούνται από λιμοκτονία, καθώς δίνεται σε όλες χρόνος για εκτέλεση κυκλικά.

Χρονοπρογραμματισμός σε πολυπύρηνα συστήματα - space scheduling

Το πρόβλημα του χρονοπρογραμματισμού σε πολυπύρηνους επεξεργαστές φέρνει νέες διαστάσεις, αφού για τη λύση του πρέπει να ληφθούν υπ' όψιν θέματα που ενδεχομένως να προκαλέσουν προβλήματα που δεν υπάρχουν στον χρονοπρογραμματισμό μονοπύρηνων συστημάτων. Οι διεργασίες που εκτελούνται ταυτόχρονα στους πυρήνες μοιράζονται κοινούς πόρους και όπως είναι λογικό ανταγωνίζονται γι αυτούς, με αποτέλεσμα να περιορίζεται η απόδοση του συστήματος. Ταυτόχρονα, για την πλήρη αξιοποίηση της αρχιτεκτονικής είναι προφανές πως όλοι οι πυρήνες του επεξεργαστή πρέπει να εκτελούν κάποιο έργο ανα πάσα στιγμή. Είναι σύνηθες με επιλογή λανθασμένης πολιτικής, να ανατεθούν πολλές διεργασίες σε έναν πυρήνα τη στιγμή που ένας άλλος παραμένει αδρανής. Ο κατάλληλος scheduler πρέπει να είναι ικανός να ισομοιράζει το φόρτο εργασίας σε όλους τους πυρήνες. Τέλος όταν οι διεργασίες προσπαθούν να αποκτήσουν πρόσβαση σε κοινές δομές για να διατηρηθεί η συνάφεια της μνήμης πρέπει να υπάρχει κάποια σειριοποίηση στη πρόσβασή της ή να χρησιμοποιούνται μηχανισμοί κλειδώματος. Αυτό έχει σαν συνέπεια - λόγω κακού σχεδιασμού - να δημιουργούνται **αδιέξοδα** (deadlocks) με καταστροφικά αποτελέσματα.

Ενώ το τελευταίο είναι ένα πρόβλημα η μόνη λύση του οποίου είναι η σωστή χρήση των μηχανισμών κλειδώματος, οι ανταγωνισμοί με προσεκτική μελέτη μπορούν να προβλεφθούν και ως ένα σημείο με τον κατάλληλο σχεδιασμό να αποφευχθούν, ενώ το ίδιο μπορεί να συμβεί και με τον ακανόνιστο διαμοιρασμό του φόρτου εργασίας. Έτσι έρχεται στο προσκήνιο η έννοια της **συνχρονοδρομολόγησης** (co-scheduling). Τώρα δεν γίνεται διαμοιρασμός (μόνο) του χρόνου αλλά και του χώρου (space scheduling). Δηλαδή ο χρονοδρομολογητής είναι υπεύθυνος να καθορίσει ποιες διεργασίες θα καταλαμβάνουν τους διάφορους πυρήνες ανα πάσα στιγμή.

2.2 Το πρόβλημα σε βάθος

Όπως αναφέρθηκε νωρίτερα, ένα από τα μεγαλύτερα προβλήματα που καλείται να αντιμετωπίσει ένας πάροχος υπηρεσιών είναι αυτό του ανταγωνισμού για κοινούς πόρους. Όταν κάποιος (πχ μια εταιρεία) αποφασίσει να έχει στην ιδιοκτησία του τις υποδομές στις οποίες θα στηριχθεί, είναι εύκολο να ρυθμίσει τη χρήση τους και να μοιράσει τις διάφορες εργασίες σε αυτές ανάλογα με τις δικές του ανάγκες. Αυτό φαντάζει σχετικά απλό από τη στιγμή που ο ίδιος ξέρει ακριβώς τις ανάγκες της κάθε εργασίας. Μπορεί λοιπόν από πριν να αναλύσει τις δοσοληψίες που θα γίνουν μεταξύ των διεργασιών και με δεδομένες τις εξαρτήσεις που

ενδεχομένως θα παρουσιαστούν, να καθορίσει τη σειρά με την οποία αυτές θα εκτελεστούν ώστε να μειωθεί ο ανταγωνισμός και να αυξηθεί η συνολική παραγωγικότητα.

Κάτι τέτοιο είναι σαφώς πιο δύσκολο στο περιβάλλον του νέφους και είναι από τις βασικότερες αιτίες για τις οποίες οι εταιρείες αποφεύγουν αυτή τη λύση. Είναι υποχρέωση λοιπόν του παρόχου να μπορεί να απομονώσει τις εργασίες που είναι ιδιαίτερα απαιτητικές και θα επηρεάσουν την απόδοση των υπόλοιπων ώστε να διασφαλίσει την καλή ποιότητα της υπηρεσίας του. Τα επιμέρους προβλήματα από τα οποία συνίσταται και το βασικό είναι αρκετά και οι λύσεις τους όχι πάντα προφανείς.

Στην περίπτωση του IaaS, οι εικονικές μηχανές αντιμετωπίζονται ως μαύρα κουτιά: δεν είμαστε σε θέση να γνωρίζουμε τί εκτελούν και κατ' επέκταση τις απαιτήσεις τους σε πόρους. Αυτό έχει ως αποτέλεσμα να μην μπορούμε να προβλέψουμε από πριν τις επιπτώσεις που θα υπάρξουν αν δρομολογήσουμε ταυτόχρονα δύο μηχανές. Αν η επιλογή γίνει με τυχαιοκρατική πολιτική υπάρχει σοβαρή πιθανότητα οι δύο μηχανές να ανταγωνίζονται είτε για τον δίαυλο προς τη μνήμη είτε για τις κρυφές μνήμες με αποτέλεσμα η μία να επιβραδύνει σε μεγάλο βαθμό την άλλη. Στην πραγματικότητα το γεγονός ότι δεν είμαστε σε θέση να ξέρουμε από πριν τη συμπεριφορά των εικονικών μηχανών έχει ιδιαίτερο βάρος στο συνολικό εγχείρημα.

2.2.1 Τεχνικές διαχείρισης νέφους

Το ζητούμενο του χρονοπρογραμματισμού είναι να αποφασίσουμε πού θα τοποθετηθεί κάθε VM και πότε θα έρθει στη CPU ώστε να μπορεί να χρησιμοποιήσει αποδοτικά το σύστημα να επηρεάζει τα υπόλοιπα όσο γίνεται λιγότερο. Για να αποφασίσουμε κάτι τέτοιο πρέπει να είμαστε σε θέση να γνωρίζουμε τις ανάγκες της κάθε εικονικής μηχανής για τους διάφορους πόρους. Για παράδειγμα, γνωρίζοντας πόσο συχνά θα καταφεύγει στη μνήμη για να φέρει δεδομένα προς επεξεργασία καθώς και σε τί ποσοστό θα τα επαναχρησιμοποιεί θα κρίνουμε αν είναι ασφαλές να τη τοποθετήσουμε δίπλα σε μία που έχει μεγάλη ανάγκη τις κρυφές μνήμες ή αν πρέπει να δρομολογηθεί μαζί με μία που περιορίζεται στα ιδιωτικά μέρη του πυρήνα (registers και ιδιωτικές κρυφές μνήμες)

Γενική ιδέα πίσω από τις λύσεις που έχουν προταθεί είναι η τοποθέτηση των διάφορων διεργασιών (στην περίπτωση που μελετάμε, τοποθέτηση των εικονικών μηχανών) σε διάφορες κατηγορίες, έτσι ώστε σε κάθε κατηγορία να βρίσκονται διεργασίες που έχουν κοινά χαρακτηριστικά. Ύστερα αντιμετωπίζεται η εκάστοτε κλάση ανάλογα με τη συμπεριφορά που παρουσιάζει.

Έως ένα βαθμό ο εκάστοτε πελάτης μπορεί να βοηθήσει στην κατηγοριοποίηση του δικού του VM καθώς συνήθως γνωρίζει τη λειτουργία του. Ωστόσο αυτό δεν αρκεί και πρέπει να βρεθεί τρόπος να χαρακτηρίζεται με αυτόματο τρόπο μία εικονική μηχανή. Έτσι έχουν προταθεί διάφορα μοντέλα και τεχνικές που είναι σε θέση να μελετήσουν την ευρύτερη συμπεριφορά ενός μηχανήματος και με τρόπο δυναμικό και αυτόματο να το τοποθετήσουν σε ξεχωριστές κλάσεις ανάλογα με τις απαιτήσεις του στους διάφορους πόρους. Και αν στο μέλλον κριθεί απαραίτητο υπάρχει πάντα η επιλογή να αλλάξουν την κλάση του.

Ο πιο εύκολος τρόπος να μελετηθεί η συμπεριφορά είναι με performance counters. Πρόκειται για μετρητές υλοποιημένους σε hardware, οι οποίοι παρέχουν τη δυνατότητα για καταγραφή διάφορων τιμών, όπως οι αστοχίες των κρυφών μνημών, οι εντολές που εκτελούνται αν κύκλο ρολογιού ή το εύρος του διαύλου που χρησιμοποιεί μία διεργασία. Στη συνέχεια, με χρήση του κατάλληλου αλγορίθμου, οι διεργασίες τοποθετούνται στην αντίστοιχη κλάση. Συγκεκριμένο παράδειγμα αναφέρεται στο επόμενο κεφάλαιο, όπου και περιγράφεται αναλυτικά ο βασικός αλγόριθμος που χρησιμοποιήθηκε στην παρούσα εργασία.

2.2.2 Σημεία ανταγωνισμού

Για να αντιμετωπίσουμε το πρόβλημα πρέπει πρώτα να βρούμε τα κυριότερα σημεία στα οποία συναντάται ο περισσότερος ανταγωνισμός. Μελετώντας περισσότερο την αρχιτεκτονική των συστημάτων που χρησιμοποιούνται παρατηρούμε πως ο κύριος ανταγωνισμός εντοπίζεται σε τρία σημεία:

- τελευταίο επίπεδο cache (llc): Το φαινόμενο κατά το οποίο πολλά νήματα ή διεργασίες ανταγωνίζονται για το llc έχει διερευνηθεί εκτενώς σε προηγούμενες εργασίες. Η συνηθέστερη πολιτική αντικατάστασης που χρησιμοποιούν οι κρυφές μνήμες και έχει σημαντικά αποτελέσματα σε μονοπύρηνα συστήματα είναι η LRU. Είναι σχεδιασμένη για να αξιοποιεί στο μέγιστο το temporal locality αφού φροντίζει να κρατάει στη cache τα δεδομένα που έχουν χρησιμοποιηθεί πιο πρόσφατα. Όταν όμως η LLC είναι μοιραζόμενη, η πολιτική αυτή αντιμετωπίζει τα misses από όλα τα threads με τον ίδιο τρόπο. Κάτι τέτοιο σημαίνει πως όταν ένα thread έχει κάποια αστοχία θα φέρει από τη μνήμη δεδομένα και θα αντικαταστήσει ένα block το οποίο ενδεχομένως χρησιμοποιείται από άλλη διεργασία. Με τον τρόπο αυτό, η λειτουργία μίας διεργασίας μπορεί να επηρεάσει σημαντικά την επίδοση των γειτονικών της.

- Δίαυλος προς την κεντρική μνήμη (Memory bus): Είναι ίσως το προφανέστερο σημείο για το οποίο θα υπάρξει ανταγωνισμός. Για σχεδιαστικούς λόγους ο δίαυλος μπορεί ανα πάσα στιγμή να χρησιμοποιηθεί από περιορισμένο αριθμό διεργασιών. Κάτι τέτοιο σημαίνει πως όταν δύο εφαρμογές προσπαθούν να φέρουν δεδομένα από τη μνήμη, θα υπάρξουν καθυστερήσεις εξ αιτίας του ανταγωνισμού.

- DRAM Controller: Προφανές σημεία ανταγωνισμού αποτελούν και οι ελεγκτές της κύριας μνήμης. Ο ελεγκτής είναι η συσκευή που είναι υπεύθυνη να παρακολουθεί τις δοσοληψίες με τη κύρια μνήμη και να διασφαλίζει την εύρυθμη λειτουργία της. Κάθε πρόσβαση στη μνήμη περνάει από τον αντίστοιχο ελεγκτή οπότε φαίνεται πως οι διεργασίες που τη χρησιμοποιούν θα ανταγωνιστούν για αυτόν. Ωστόσο φαίνεται πως αυτός ο ανταγωνισμός δεν θα επηρεάσει ιδιαίτερα τη μελέτη μας, καθώς θα είναι άμεσα συνδεδεμένος με τον αντίστοιχο για τον δίαυλο προς τη μνήμη. Δηλαδή οι διεργασίες που ανταγωνίζονται για τον δίαυλο θα ανταγωνιστούν αργότερα και για τον ελεγκτή. Έτσι είναι ασφαλές να θεωρήσουμε πως οι δύο ανταγωνισμοί μπορούν να συμπτυχθούν σε έναν.

Αξίζει να σημειωθεί πως οι ανταγωνισμοί περιορίζονται όταν η μελέτη ξεφεύγει από ένα package όπου οι πυρήνες μοιράζονται να παραπάνω σημεία. Σε αυτή την περίπτωση οι ανταγωνισμοί που παρατηρούνται, συνήθως εντοπίζονται σε προσβάσεις στις περιφερειακές συσκευές ή σε συγκρούσεις στα δίκτυα.

2.2.3 Contention-aware scheduling

Δεδομένου πως γνωρίζουμε τα σημεία στα οποία απαντάται ο περισσότερος ανταγωνισμός και έχοντας τρόπο να τον εντοπίσουμε είμαστε σε θέση να προσαρμόσουμε με τέτοιο τρόπο το σύστημα ώστε να τον ελαττώσουμε στο ελάχιστο. Με κατάλληλη πολιτική χρονοπρογραμματισμού οι εικονικές μηχανές που ανταγωνίζονται περισσότερο μπορούν να απομακρυνθούν μεταξύ τους. Ο χρονοδρομολογητής που λαμβάνει υπ' όψιν τον ανταγωνισμό για κοινούς πόρους ονομάζεται contention-aware scheduler. Εδώ είναι σημαντικό να σημειωθεί πως ένας τέτοιος χρονοδρομολογητής διαμοιράζει περισσότερο τον "χώρο" -δηλαδή τους πυρήνες της CPU- στις εικονικές μηχανές παρά τον χρόνο σε αυτούς. Μιλάμε δηλαδή για space scheduling.

Στη συνέχεια περιγράφουμε εν συντομία τα κύρια σημεία που αποτελούν έναν τέτοιο χρονοδρομολογητή.

Χρησιμοποιούμε τον όρο **σκοπό** (objective) για να περιγράψουμε το σημείο (metric) που προσπαθούμε να βελτιστοποιήσουμε. Αυτό συνήθως είναι είτε το συνολικό throughput (δηλαδή η ποσότητα του έργου που παράγεται ανα μονάδα χρόνου - IPC) είτε η ποιότητα υπηρεσιών (QoS) ή πιο απλά δικαιοσύνη μεταξύ των εικονικών μηχανών. Συνήθως ένας τυπικός scheduler προσπαθεί να μοιράσει τη συνολική δουλειά στους πυρήνες χρησιμοποιώντας κάποια τεχνική load-balancing. Τα διάφορα νήματα μπορούν να τοποθετηθούν με πολλούς συνδυασμούς στους πυρήνες. Κάθε συνδυασμός μπορεί να έχει είτε μικρό ανταγωνισμό και καλή απόδοση είτε το αντίθετο. Κάτι τέτοιο με τους παραδοσιακούς χρονοδρομολογητές δεν μπορεί ούτε να ελεγχθεί ούτε να προβλεφθεί. Αντίθετα αν ο χρονοδρομολογητής λαμβάνει υπ' όψιν τους ανταγωνισμούς δημιουργεί τους κατάλληλους συνδυασμούς ώστε να περιορίσει τόσο την επίδρασή τους όσο την απρόβλεπτη συμπεριφορά του συστήματος.

Έχοντας συγκεκριμένο αριθμό από νήματα και πυρήνες οι δυνατοί συνδυασμοί που αντιστοιχούν τα μεν με τους δε είναι τόσοι που καθιστούν την δοκιμή κάθε ενός από αυτούς αδύνατη. Δηλαδή είναι μη αποδοτικό το να ελέγχουμε τη συμπεριφορά όλων των δυνατών συνδυασμών, ώστε να καταλήξουμε στον καλύτερο. Αυτό σημαίνει πως πρέπει να έχουμε έναν μηχανισμό να κάνουμε μία **πρόβλεψη** αναφορικά με την απόδοση κάθε ενός συνδυασμού. Έχουν γίνει πολλές έρευνες στο πεδίο, με τις περισσότερες να προτείνουν λύσεις που ελέγχουν τις αστοχίες της LLC καθώς και το ρυθμό πρόσβασης στη μνήμη και ανάλογα προβλέπουν τη συμπεριφορά κάθε συνδυασμού.

Έπειτα ένας contention-aware scheduler πρέπει να πάρει μία **απόφαση** αναφορικά με το ποιός συνδυασμός είναι ο καταλληλότερος στο να επιτευχθεί ο σκοπός. Ωστόσο, ακολουθώντας ακόμα και ένα ιδανικό και ακριβές μοντέλο πρόβλεψης είναι αποδεδειγμένο πως όταν έχουμε να κάνουμε με παραπάνω από 2 πυρήνες πρέπει να αντιμετωπίσουμε ένα NP-complete πρόβλημα [YJia08].

Τέλος ο χρονοδρομολογητής έχει ένα μηχανισμό **επιβολής** της πολιτικής ο οποίος είναι υπεύθυνος να αναθέσει το κάθε thread σε κάποια CPU. Είναι εύκολο να υλοποιηθούν ακόμα και σε user space καθώς τα σύγχρονα λειτουργικά παρέχουν τις αντίστοιχες κλήσεις συστήματος.

Το πιο σημαντικό κομμάτι ωστόσο είναι ο χαρακτηρισμός της κάθε διεργασίας (classification). Επιλέγοντας την κατάλληλη τεχνική και παρακολουθώντας ίσως τη τη συμπεριφορά μίας διεργασίας, πρέπει να είμαστε σε θέση να τη τοποθετήσουμε σε κάποια κατηγορία με άλλες που έχουν παρόμοιες απαιτήσεις σε πόρους. Έτσι τελικά το πρόβλημα θα περιοριστεί στο να βρεθεί ο ιδανικός συνδυασμός κλάσεων που μπορούν να τρέχουν παράλληλα.

Πάντως, ανάλογα με το σκοπό για τον οποίο προορίζεται ένα σύστημα, διαφορετική προσέγγιση μπορεί να επιλεχθεί. Έτσι σε κάποιες περιπτώσεις που η υψηλή διαθεσιμότητα είναι απαραίτητο συστατικό των υπηρεσιών που προσφέρει το σύστημα, μπορούν οι ανταγωνισμοί να μη ληφθούν υπ' όψιν με το περιοριστούν οι απαιτήσεις σε συγκεκριμένη απόδοση. Από την άλλη, όταν σημαντικότερο ρόλο παίζει η υψηλή απόδοση μίας διεργασίας όσο αυτή βρίσκεται σε κάποιον πυρήνα, ίσως είναι προτιμότερο να παραμείνει σε αδράνεια για μεγαλύτερο χρονικό διάστημα μία άλλη η οποία ενδεχομένως να την ανταγωνιστεί ισχυρά για κοινούς πόρους.

2.3 Σχετική έρευνα

2.3.1 Χρονοπρογραμματισμός και ανταγωνισμός

Το πρόβλημα του ανταγωνισμού είναι πολλά χρόνια γνωστό και έχει μελετηθεί εκτενώς από την ερευνητική κοινότητα. Οι λύσεις που έχουν προταθεί είναι πολλές και δεν περιο-

ρίζονται μόνο στους schedulers. Μία ιδιαίτερα ενδιαφέρουσα απάντηση έχει να κάνει με τη διάτμηση μέσω λογισμικού της κρυφής μνήμης [SKim04]. Αυτή η τεχνική στοχεύει στο να απομονώσει τα νήματα που χρησιμοποιούν τη cache και το κάθε ένα να έχει ένα υποσύνολο από τα block της llc για αποκλειστική χρήση. Για να επιτευχθεί κάτι τέτοιο, πρέπει όταν ένα νήμα προσπαθήσει να δεσμεύσει ένα κομμάτι μνήμης, να του ανατεθεί αυτό που θα αντιστοιχηθεί στο κατάλληλο block στην κρυφή μνήμη ανάλογα με το associativity της τελευταίας. Προφανώς μία τέτοια λύση για να είναι αποδοτική, το μέγεθος του κάθε τμήματος την κρυφής μνήμης θα διαφέρει από τα υπόλοιπα και κάθε thread ανάλογα με τις απαιτήσεις του σε cache θα έχει και το κατάλληλο κομμάτι. Πάντως ενώ η τεχνική θεωρείται υποσχόμενη έχει δύο μεγάλα εμπόδια. Πρώτον προϋποθέτει σημαντικές αλλαγές στο σύστημα διαχείρισης της εικονικής μνήμης - ένα πολύπλοκο μέρος του λειτουργικού και έπειτα απαιτεί την αντιγραφή ολόκληρων περιοχών της μνήμης αν για κάποιο λόγο πρέπει να μεταβληθεί ένα κομμάτι της cache (πχ σε μέγεθος). Εκτός αυτού, αξίζει να σημειωθεί πως μία τέτοια τεχνική βοηθάει στην αντιμετώπιση του ανταγωνισμού κυρίως στο επίπεδο της κρυφής μνήμης που ενώ αποτελεί μεγάλος μέρος του προβλήματος, ο ανταγωνισμός στον δίαυλο προς τη κεντρική μνήμη και στον ελεγκτή αυτής παραμένει ο ίδιος.

Ακριβώς επειδή είναι τόσο σημαντικός ο ανταγωνισμός στο επίπεδο της LLC εκεί επικεντρώθηκαν οι περισσότεροι ερευνητές τουλάχιστον στα πρώτα στάδια των μελετών τους. Οι Xie και Loh πρότειναν μία μέθοδο ταξινόμηση βασισμένη στο ζωικό βασίλειο[Xie08]. Συγκεκριμένα χώρισαν τις εφαρμογές σε τέσσερις κατηγορίες: χελώνες, πρόβατα, λαγοί και δαίμονες της Τασμανίας. Εφαρμογές τοποθετούνται στις κατηγορίες ανάλογα με τη χρήση της LLC και το κατα πόσο είναι ευαίσθητες ή τα cache misses. Οι Lin et al, χρησιμοποίησαν cache partitioning και χαρακτήρισαν τις εφαρμογές βασιζόμενοι στην μείωση της απόδοσης όταν αυτές έτρεχαν σε 4MB cache και όταν έτρεχαν με 1MB αντίστοιχα [JLin08]. Έτσι όταν μία εφαρμογή είχε ζημιά μεγαλύτερη του 20% ανήκει στη κόκκινη κλάση. Για ζημιά μεταξύ του 5% και 20% ανήκει στη κίτρινη κλάση και για ζημιά μικρότερη του 5% οι εφαρμογές χωρίζονται σε πράσινες και μαύρες ανάλογα με το miss rate.

Η ιδανική λύση

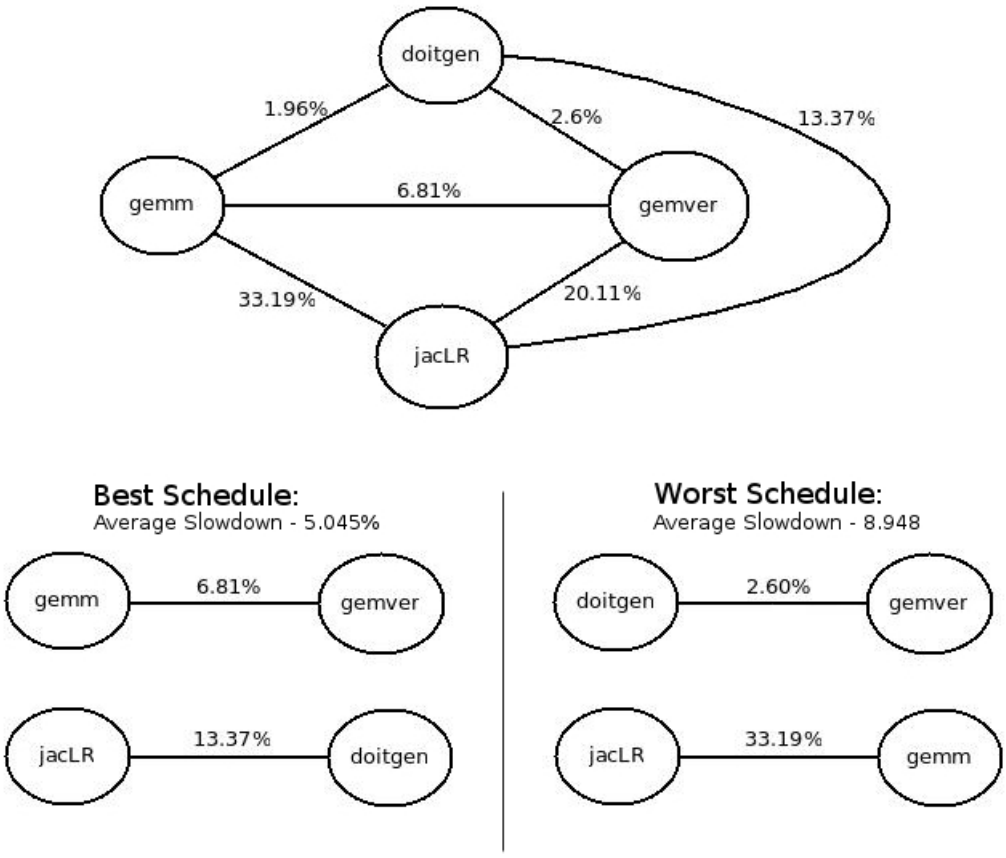
Χρησιμοποιούμε τον όρο επιβράδυνση για να περιγράψουμε το πόσο μία διεργασία επηρεάζεται από μία άλλη που εκτελείται σε γειτονικό πυρήνα, σε σχέση με το αν έτρεχε μόνη της.

Ο Jiang (et al, 2008) κατασκεύασε το γράφημα που κορυφές έχει τις διεργασίες και το βάρος κάθε ακμής ισοδυναμεί με την επιβράδυνση που θα υποστούν οι διεργασίες που αυτή ενώνει αν τοποθετηθούν ταυτόχρονα στον πυρήνα [YJia08]. Έδειξε δε πως ιδανική λύση του προβλήματος ισοδυναμεί με λύση του min-weight perfect matching problem. Παρακάτω φαίνεται μία οπτική απεικόνιση της βέλτιστης λύσης του προβλήματος. Στο παράδειγμα φαίνονται οι επιβραδύνσεις που παρουσιάστηκαν από την συν-εκτέλεση τεσσάρων εφαρμογών.

	doitgen	gemm	gemver	jacLR
doitgen	0.36%	1.33%	0.92%	4.29%
gemm	0.63%	1.91%	1.50%	5.68%
gemver	1.68%	5.31%	7.77%	20.11%
jacLR	9.08%	27.51%	24.34%	78.14%

Σχήμα 2.4: Παράδειγμα: Ιδανική λύση

Στη συνέχεια κατασκευάζεται το αντίστοιχο γράφημα με τις πιθανές αλληλεπιδράσεις και διατυπώνουμε τον καλύτερο καθώς και τον χειρότερο συνδυασμό από αυτές.



Σχήμα 2.5: Παράδειγμα: Ιδανική λύση (συνέχεια)

Φυσικά ο όρος ’ιδανική λύση’ δεν είναι τόσο μονοσήμαντος. Για την ακρίβεια η παραπάνω διατύπωση εκφράζει μόνο μέρος της αλήθειας, καθώς σε μία τέτοια περίπτωση αυτό που αναζητούμε είναι καθαρά η ελαχιστοποίηση του χρόνου που απαιτείται για την εκτέλεση του συνολικό έργου. Πολλές φορές όμως (και κυρίως σε περιβάλλον cloud) η ελαχιστοποίηση του χρόνου δεν είναι το μόνο που μας ενδιαφέρει. Ιδιαίτερο ρόλο έχει και η **ποιότητα υπηρεσιών**. Ενδιαφέρει δηλαδή η ύπαρξη μίας εγγύησης πως μία εφαρμογή θα υποστεί επιβράδυνση μέχρι μία προκαθορισμένη και αποδεκτή τιμή.

2.3.2 Χρονοπρογραμματισμός και εικονοποίηση

Σημαντική έρευνα έχει γίνει και στον χώρο της εικονοποίησης και σε περιβάλλον νέφους (cloud). Στη προσπάθειά τους να βελτιστοποιήσουν την υπηρεσία τους οι πάροχοι έχουν επενδύσει πολύ στην έρευνα του χρονοπρογραμματισμού, στοχεύοντας τόσο σε καλύτερη ποιότητα υπηρεσίας (QoS) όσο και σε ενεργειακά ή οικονομικά οφέλη. Έτσι έχουν προταθεί QoS-aware, energy-aware και resource-aware πολιτικές που υπόσχονται να δώσουν μία λύση στο πρόβλημα.

Παρακάτω βρίσκονται διάφοροι αλγόριθμοι που έχουν είτε έχουν προταθεί, είτε χρησιμοποιούνται ως μέτρα σύγκρισης για την αξιολόγηση άλλων χρονοδρομολογητών.

1. Round robin: Πρόκειται για τον απλούστερο αλγόριθμο. Ο χρονοδρομολογητής αναθέτει ένα-ένα τα εικονικά μηχανήματα σε κόμβους μέχρι κάθε κόμβος να απασχολείται από ένα μηχάνημα. [KPra13]

2. Genetic aglorithm: Εδώ, εφαρμόζεται μία αρχική αντιστοίχηση μεταξύ vm και πυρήνων. Στη συνέχεια, ο αλγόριθμος χρησιμοποιεί γενετικούς τελεστές ώστε να δημιουργήσει τη νέα - αποδοτικότερη - αντιστοίχηση για το σύστημα. Ο αλγόριθμος αρχικά υπολογίζει την επίδραση που θα έχει μία λύση στο σύστημα χρησιμοποιώντας τόσο τη τρέχουσα αντιστοίχηση όσο και στατιστικά και ιστορικά δεδομένα και τελικά επιλέγει τη λύση με την ελάχιστη επίδραση στο σύστημα [Yu06].

3. Match-Making algorithm: Τώρα, αρχικά οι απορρίπτονται οι κόμβοι οι οποίοι δεν ικανοποιούν τις ανάγκες των εικονικών μηχανών (όπως CPU, μνήμη κλπ). Για τους εναπομείναντες κόμβους ορίζεται ένας βαθμός ανάλογα με τις πληροφορίες που έχουν συλλεχθεί. Οι βαθμοί έρχονται ως ορίσματα στον χρονοδρομολογητή, ο οποίος στη συνέχεια αποφασίζει ανάλογα την κατάλληλη απεικόνιση.

4. Memory-aware cloud scheduler: Ο τελευταίος αλγόριθμος μελετάει τη συμπεριφορά των εικονικών μηχανών και μέσω του μηχανισμού της μετανάστευσης, προσπαθεί να περιορίσει όποιες επιβραδύνσεις προέρχονται από την αύξηση των αστοχιών της κρυφής μνήμης, [14]. Ένας τέτοιος χρονοδρομολογητής δρα είτε τοπικά, οπότε παρακολουθεί και προσπαθεί να περιορίσει τα cache-misses εντός του node (cache-aware scheduler), είτε καθολικά οπότε λαμβάνει υπ' όψιν τις κρυφές μνήμες όλων των κόμβων (NUMA-aware scheduler) [AJeo12].

Τέλος, έχουν προταθεί ακόμη πιο προηγμένες μεθόδου διαχείρισης του νέφους. Ομάδα ερευνητών προτείνει μεθόδους, όπου με χρήση μηχανισμών δρομολόγησης και μετανάστευσης ένα σύστημα εξισορροπείται όταν τείνει να βρεθεί σε ανισορροπία [MAlA15].

Κεφάλαιο 3

Χρησιμοποιούμενες υποδομές

Στο τρίτο κεφάλαιο περιγράφονται τόσο το σύστημα όσο και οι συνθήκες κάτω από τις οποίες εκτελέστηκαν τα πειράματα και οι μετρήσεις. Στο τέλος του κεφαλαίου βρίσκονται αναλυτικότερες περιγραφές δύο αλγορίθμων που χρησιμοποιήθηκαν για μελέτη των ανταγωνισμών και δείχνουν υποσχόμενοι για την αντιμετώπιση των ανισορροπιών που προκαλούν οι ανταγωνισμοί.

3.1 Περιβάλλον πειραμάτων

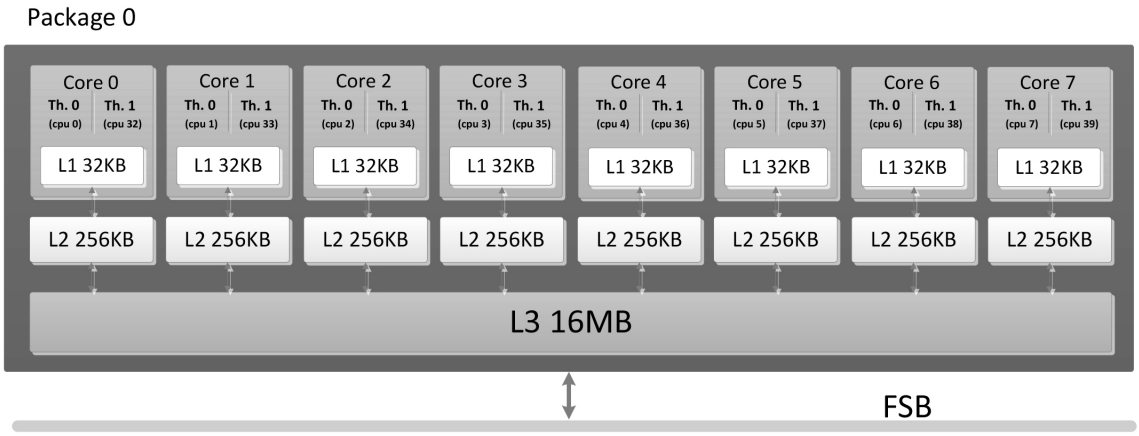
Σε κάθε σύστημα, ανάλογα με την αρχιτεκτονική του τα σημεία ανταγωνισμού διαφέρουν και κυρίως επιδρούν σε διαφορετικό ποσοστό στην τελική απόδοση. Έτσι όταν γίνεται προσπάθεια υλοποίησης μίας υπηρεσίας είναι σημαντικό να γίνει κατάλληλη μελέτη του συστήματος. Για να αποκαλυφθούν οι ανταγωνισμοί σε όλο τους το εύρος, πρέπει να γίνει σωστή επιλογή των προγραμμάτων που θα χρησιμοποιηθούν για τις μετρήσεις (για παράδειγμα να επιδρούν σε αρκετά μεγάλο dataset που να μην χωράει στη μοιραζόμενη κοινή μνήμη με αποτέλεσμα να καταφεύγουν συχνά στην κεντρική μνήμη μέσω του διαύλου ή να έχουν τόσο μικρό dataset που να περιορίζονται στα ιδιωτικά μέρη του πυρήνα και να μην προκαλούν ανταγωνισμούς).

Επίσης σημαντική είναι και η επιλογή του υλικού πάνω στο οποίο θα εκτελεστούν τα πειράματα. Στη συνέχεια γίνεται μία σύντομη παρουσίασή του.

3.1.1 Υλικό

Για την εκτέλεση των πειραμάτων χρησιμοποιήθηκε cluster που υλοποιεί NUMA αρχιτεκτονική με 4 κόμβος NUMA. Ο κάθε κόμβος αποτελείται από επεξεργαστή Intel Xeon E5-4620 (αρχιτεκτονική Intel Sandy Brigde) με 8 πυρήνες, 32 KB instruction cache και 32 KB Level 1 cache και 256 KB Level 2 Cache. Οι 8 πυρήνες μοιράζονται την κοινή 16MB και 16-way associative Level 3 Cache (LLC). Τόσο η L1 cache όσο και η L2 είναι 8-way associative. Το μέγεθος μίας γραμμής cache (cache line size) είναι 64 bytes. Τέλος ο επεξεργαστής είναι hyperthreaded και κάθε ένας από τους 8 πυρήνες υποστηρίζει δύο νήματα (Σχήματα 3.1 και 3.2).

Κάθε πακέτο επικοινωνεί με τη κεντρική μνήμη μέσα από 4 κανάλια για ταυτόχρονη προσπέλαση, με μέγιστο εύρος περίπου 14 GB/s για κάθε πυρήνα και 18 GB/s για κάθε πακέτο (Σχήμα 3.3).

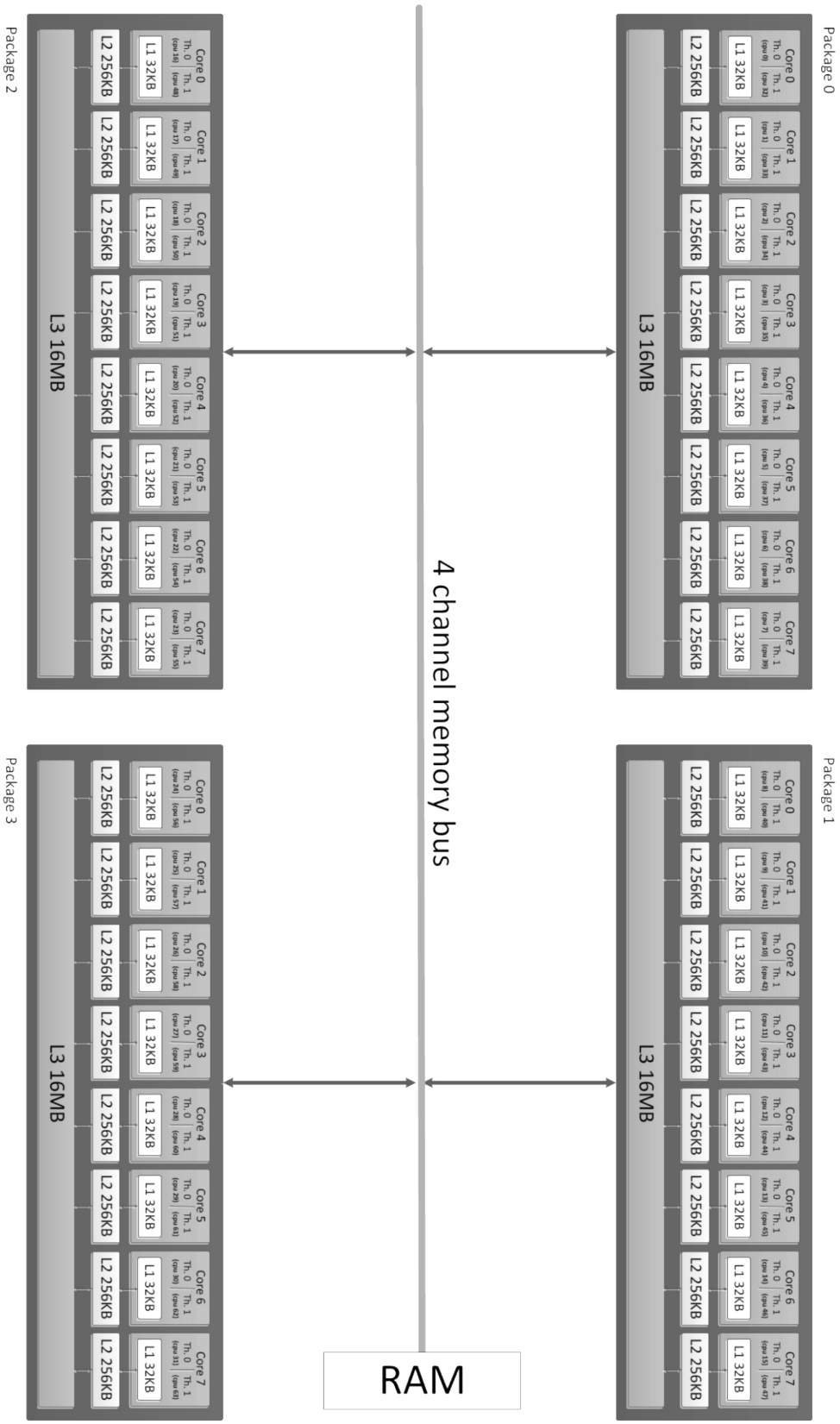


Σχήμα 3.1: Intel Xeon E5-4620

Τέλος οι επεξεργαστές παρέχουν δυνατότητα παρατήρησης της απόδοσης των εφαρμογών μέσω μετρητών (performance counters). Υπάρχει έτσι η δυνατότητα να μελετηθεί αφ' ενός η συμπεριφορά των εφαρμογών σε βάθος καθιστώντας ευκολότερη την κατηγοριοποίησή τους και εφ' ετέρου η επίδραση που έχει στην απόδοση του συστήματος ο ανταγωνισμός και τα αποτελέσματα της ταυτόχρονης εκτέλεσής τους. Η αρχιτεκτονική των Sandy Bridge παρέχει μεγάλο εύρος μετρητών που μπορούν να συνδυαστούν ώστε να υπολογίσουμε τα αντίστοιχα "μετρικά". Οι μετρητές που θα χρησιμοποιηθούν και ο τρόπος υπολογισμού των μετρικών α αναλυθεί περισσότερο στη συνέχεια. Στο σύνολό του είναι ένα σύστημα υψηλής επίδοσης και θεωρείται κατάλληλο να αξιολογήσει το μοντέλο αντιμετώπισης ανταγωνισμού που θα περιγραφεί στη συνέχεια του κεφαλαίου. Στον παρακάτω πίνακα αναφέρονται περιληπτικά τα χαρακτηριστικά του συστήματος.

# of packages	4
Cores/Socket	8
Threads/Core	2
CPU frequency	2.2 GHz (TurboBoost™ up to 2.6 GHz)
L1 Cache	32KB data + 32KB instr., private per core, 8-way
L2 Cache	256KB private per core, 8-way
L3 Cache	16MB shared, 16-way
RAM	256GB DDR3, 4-channel bus

Σχήμα 3.2: Χαρακτηριστικά Intel Xeon E5-4620



Σχήμα 3.3: Πειραματική διάταξη

3.1.2 Scaff

Για όλα τα πειράματα χρησιμοποιήθηκε το πρόγραμμα scaff. Το scaff είναι ένα πρόγραμμα που αναπτύχθηκε από το CSLab και είναι σχεδιασμένο να συντονίζει την εκτέλεση προκαθορισμένου έργου το οποίο αποτελείται από πολυνηματικές εφαρμογές σε πολυπύρηνα συστήματα. Εφαρμόζει τη δική του πολιτική χρονοδρομολόγησης την οποία ο χρήστης είναι σε θέση να καθορίσει και εκτελείται στο user space συστημάτων linux. Χρησιμοποιείται για να παρέχει έναν μηχανισμό επικοινωνίας μεταξύ του scheduler και των προγραμμάτων που εκτελεί.

Το scaff αποτελείται από δύο συστήματα: τον executor και τον scheduler. Το πρώτο χειρίζεται διάφορα συμβάντα της εκτέλεσης, για παράδειγμα την έναρξη και την λήξη της εκτέλεσης ενός προγράμματος ή τον χειρισμό των ρολογιών. Το δεύτερο είναι υπεύθυνο για τις αποφάσεις που αφορούν τον διαμοιρασμό των πόρων και εφαρμόζει την πολιτική που θα επιλέξει το χρήστης. Υπάρχουν προκαθορισμένες πολιτικές που μπορούν να εφαρμοσθούν, πχ ο default scheduler του linux, αλλά ο χρήστης είναι σε θέση να γράψει και να εφαρμόσει τη δική του.

Μηχανισμοί kernel και cgroups

Για να επιβάλλει την επιλεχθείσα πολιτική το scaff χρησιμοποιεί τεχνολογίες που παρέχονται από τον πυρήνα του linux. Μία από αυτές είναι τα cgroups τα οποία παρέχουν έναν μηχανισμό δημιουργίας συνόλων προγραμμάτων. Τα προγράμματα που ανήκουν σε ένα συγκεκριμένο σύνολο έχουν συγκεκριμένη και καθορισμένη συμπεριφορά. Ένας χρήστης μπορεί να παρέμβει στην εκτέλεση των διεργασιών ενός συνόλου με το να παραμετροποιήσει και να αλλάξει τα κατάλληλα αρχεία (configuration files) του συνόλου. Τα cgroups είναι διαθέσιμα μέσω ενός εικονικού συστήματος αρχείων και είναι ελέγξιμα από τον χώρο χρήστη (userspace). Αυτό είναι που τα καθιστά κατάλληλα για τον χειρισμό διεργασιών από εφαρμογή που τρέχει στο userspace η οποία καθορίζει τη συμπεριφορά των προγραμμάτων σε ένα σύνολο με την αλλαγή τιμών σε αυτά τα αρχεία.

Χαρακτηριστικό παράδειγμα μηχανισμού που χρησιμοποιεί τα cgroups είναι τα cpuset τα οποία περιορίζουν την εκτέλεση των προγραμμάτων σε συγκεκριμένες cpu. Τυπικά, δημιουργούνται τα εικονικά αρχεία 'tasks' και 'cpus' στα οποία ο χρήστης τοποθετεί το PID μίας διεργασίας και το cpu id στο οποίο επιθυμεί να εκτελεστεί η διεργασία αντίστοιχα. Ωστόσο με τα cpusets ο χρήστης μπορεί να καθορίσει και άλλες παραμέτρους εκτός από την αντιστοιχία διεργασίας - cpu.

Το scaff δεσμεύει μία νέα δομή (aff_prog_t) για κάθε πρόγραμμα που εκτελεί, την οποία χρησιμοποιεί για να αποθηκεύσει τις αναγκαίες πληροφορίες για την εκτέλεση του προγράμματος. Η ίδια δομή περιέχει έναν δείκτη προς το κομμάτι της μνήμης που μοιράζονται ο executor και το πρόγραμμα για τη μεταξύ τους επικοινωνία. Τέλος περιέχει και ένα πεδίο για να αναφέρεται στο αντίστοιχο cpuset. Το scaff ξεκινάει την εκτέλεση ενός προγράμματος με χρήση της κλήσης συστήματος fork και δημιουργεί ένα νέο cgroup στο σύστημα αρχείων των cpuset για κάθε πρόγραμμα.

Τέλος, για τον χειρισμό των προγραμμάτων χρησιμοποιεί σήματα για τα οποία έχει τους αντίστοιχους χειριστές (signal handlers). Η τυπική έξοδος ενός προγράμματος προκαλεί ένα σήμα SIGCHLD, ενώ ένας ανώμαλος τερματισμός θα προκαλέσει το SIGTERM.

Για την υλοποίηση διάφορων contention-aware πολιτικών που έπρεπε να λάβουν υπ' όψιν τις αποδόσεις των εφαρμογών, το scaff χρησιμοποιεί performance counters που είναι διαθέσιμοι από τις κατασκευαστικές εταιρείες των επεξεργαστών. Φυσικά ύστερα από επιλογή του χρήστη τυπώνει τους counters ώστε να μπορεί στη συνέχεια να μελετηθεί είτε η συμπεριφορά μίας εφαρμογής (με ενδεχόμενη τοποθέτησή της σε κάποια κλάση), είτε η επίδοση της συγκεκριμένης πολιτικής.

3.1.3 Μέθοδος εκτέλεσης πειραμάτων

Για την εκτέλεση των πειραμάτων, αρχικά έπρεπε να επιλεχθούν τα κατάλληλα προγράμματα που θα χρησιμοποιηθούν ως σημεία αναφοράς (benchmarks).

Στη συνέχεια τα προγράμματα, ανάλογα με τη συμπεριφορά τους και την πολιτική που βρίσκεται σε δοκιμασία, τοποθετούνται σε διαφορετικές κλάσεις. Τα περισσότερα προγράμματα που χρησιμοποιήθηκαν προέρχονται από το PARSEC benchmark και υλοποιούν διάφορους αλγορίθμους γραμμικής άλγεβρας κλπ. Κάποια βασίζονται σε επαναλαμβανόμενες πράξεις πάνω σε πίνακες και αποκομίζουν μεγάλο κέρδος από την εκμετάλλευση της κρυφής μνήμης ενώ άλλα μεταφέρουν διαρκώς δεδομένα από την κύρια μνήμη ασκώντας πίεση στον δίαυλο. Τέτοιο παράδειγμα αποτελεί το stream benchmark, το οποίο είναι σχεδιασμένο να μετρήσει το συνολικό εύρος του διαύλου.

Ανταγωνιστικές πολιτικές

- Linux: Η πρώτη πολιτική με την οποία συγκρίνουμε τα αποτελέσματα είναι αυτή του default scheduler του Linux. Από την έκδοση 2.6.23 του πυρήνα και ύστερα αυτός είναι ο Completely Fair Scheduler (CFS). Ο αλγόριθμος χρησιμοποιεί RB-δέντρα για να κρατάει ταξινομημένες τις διεργασίες, ανάλογα με τον χρόνο που έχουν χρησιμοποιήσει τον επεξεργαστή. Οι εφαρμογές που τον έχουν χρησιμοποιήσει λιγότερο έχουν υψηλότερη προτεραιότητα. Ο αλγόριθμος θεωρείται δίκαιος ως προς την αντιμετώπιση των διεργασιών, ωστόσο δεν υπολογίζει καθόλου τους ανταγωνισμούς που προκύπτουν.

- LLC miss Balance: Χρησιμοποιώντας σαν μετρικό τα LLC misses, δημιουργούνται ζεύγη ανάλογα, ώστε να ελαττώσουμε όσο μπορούμε τον ανταγωνισμό στη LLC. Ο μηχανισμός αυτός αναμένουμε να βελτιώσει την εικόνα στις cache, αλλά δεν αντιμετωπίζει τον ανταγωνισμό στον δίαυλο. Είναι βασικό να σημειωθεί πως η εξισορρόπηση γίνεται με βάση τα cache misses που έχουν οι εφαρμογές όταν τρέχουν μόνες τους. Για καλύτερα αποτελέσματα, μετρήσεις για τα misses θα πρέπει να παρθούν σε χρόνο εκτέλεσης.

- Throughput Balance: Πρόκειται για έναν χρονοδρομολογητή που βασικό στόχο έχει να εξισορροπήσει όσο γίνεται το throughput των εφαρμογών. Το πώς ακριβώς θα το πετύχει θα αναλυθεί στη συνέχεια του κεφαλαίου.

- LCA : Πρόκειται για αλγόριθμο που θα αναλυθεί στη συνέχεια του παρόντος κεφαλαίου. Έχει φέρει υποσχόμενα αποτελέσματα σε bare metal περιβάλλον και αναμένουμε αντίστοιχα σε virtualization περιβάλλον. Ακολουθεί τη λογική του LLC miss balance, ωστόσο προσπαθεί να αντιμετωπίσει τον ανταγωνισμό σε δύο επίπεδα.

Ένα θέμα στο οποίο παρουσιάζεται διαφορετική συμπεριφορά ανάμεσα στον CFS και τους υπόλοιπους χρονοδρομολογητές είναι ο ανταγωνισμός κλειδώματος (**lock contention**). Ο CFS αντιμετωπίζει κάθε thread σαν ξεχωριστή διεργασία. Αυτό σημαίνει πως το ανταγωνισμός για κάποιο lock μεταξύ των νημάτων της ίδιας εφαρμογής θα έχει πολύ βαρύτερες συνέπειες αφού το thread που το κρατάει είναι πολύ πιθανό να βρίσκεται εκτός επεξεργαστή.

Από την άλλη, οι χρονοδρομολογητές που τοποθετούν τις εφαρμογές και τούπλες και τις δρομολογούν ταυτόχρονα στον επεξεργαστή, με αποτέλεσμα να σπαταλούν πόρους όταν παρουσιάζονται ανταγωνισμοί εντός της εφαρμογής.

Τελικά να σημειωθεί πως ο CFS κάνει ιδιαίτερα καλή δουλειά να ισομοιράζει τον χρόνο στις διεργασίες οπότε παρέχει δικαιοσύνη ως προς αυτήν την οπτική. Ωστόσο αποτυγχάνει να προσφέρει την ίδια δικαιοσύνη από την οπτική της επιρροής λόγω ανταγωνισμών. Οι διεργασίες που χρησιμοποιούν κοινούς πόρους θα υποστούν δυσανάλογες επιβραδύνσεις σε σύγκριση με αυτές που περιορίζονται στα ιδιωτικά μέρη του επεξεργαστή.

Μετρήσεις

Αφού διακρίθηκαν οι βασικοί αλγόριθμοι που θα χρησιμοποιηθούν, έπρεπε να βρεθεί η συμπεριφορά των εφαρμογών όταν αυτές τρέχουν σε bare metal περιβάλλον και πως αυτή θα επηρεαστεί από την εκτέλεσή τους σε εικονικές μηχανές. Έτσι έγιναν οι αντίστοιχες μετρήσεις και φανερώθηκε πως το περιβάλλον της εικονοποίησης ελάχιστα επηρεάζει την συμπεριφορά τους. Έπειτα δημιουργήθηκαν διάφορα workload με χρήση των διαθέσιμων εφαρμογών και αυτά εκτελέστηκαν με χρήση κάθε ενός από τους προαναφερθέντες schedulers. Για την προσομοίωση περιβάλλοντος cloud χρησιμοποιήθηκε το πρόγραμμα QEMU που με χρήση του -παρεχόμενου από τον πυρήνα του Linux- KVM έχει εντυπωσιακά αποτελέσματα σε θέματα εικονοποίησης. Για την εύκολη καταγραφή των μετρήσεων χρησιμοποιήθηκαν κατάλληλα bash script (για την αυτόματη εκκίνηση των benchmark όταν σηκώνεται το VM) και το εργαλείο time του Linux για να καταγραφεί ο χρόνος που απαιτείται για την εκτέλεση κάθε εφαρμογής.

Τέλος για την ολοκλήρωση των μετρήσεων, δημιουργήθηκαν σενάρια με όλους τους δυνατούς συνδυασμούς που προκύπτουν από τις διαθέσιμες προς μέτρηση εφαρμογές, όταν αυτές τοποθετούνται σε ζεύγη για εκτέλεση. Έτσι προέκυψαν πίνακες με τους χρόνους που χρειάστηκε κάθε εφαρμογή όταν εκτελούταν στον ίδιο επεξεργαστή ταυτόχρονα με οποιαδήποτε άλλη. Οι πίνακες (heatmap) που προέκυψαν βρίσκονται στο επόμενο κεφάλαιο μαζί με την ανάλυση των μετρήσεων.

3.1.4 QEMU cpu models

Για την δημιουργία των εικονικών μηχανών χρησιμοποιήθηκε η πλατφόρμα εικονοποίησης Qemu-KVM. Οι μετρήσεις που έγιναν, πραγματοποιήθηκαν τόσο σε baremetal περιβάλλον (το περιβάλλον του host) όσο και σε εικονικό (περιβάλλον του guest). Με το qemu παρουσιάζονται πολλές δυνατότητες αφού υποστηρίζει ένα μεγάλο εύρος μοντέλων επεξεργαστών. Ανάλογα με τη χρήση για την οποία προορίζεται ένα σύστημα, διαφορετικό μοντέλο κρίνεται κατάλληλο. Είναι πολύ σημαντικό να σημειωθεί πως ενώ το qemu είναι σε θέση να προσομοιώσει τον εικονικό επεξεργαστή, η υποστήριξη του υποκείμενου υλικού είναι κομβική για την τελική απόδοση. Αυτό πρακτικά σημαίνει πως το qemu θα χρησιμοποιήσει όποια βοήθεια μπορεί από το υλικό ώστε να χρειαστεί να προσομοιώσει όσο το δυνατόν λιγότερες λειτουργίες.

Παρακάτω ακολουθεί σύντομη περιγραφή δύο μοντέλων που είναι τα συνηθέστερα με τη χρήση του qemu.

- -cpu qemu64: Πρόκειται για την default επιλογή. Το qemu64 υλοποιεί μικρό αριθμό από cpu features. Είναι εύκολο ένας τέτοιος επεξεργαστής να φιλοξενηθεί σε διαφορετικά φυσικά μηχανήματα χωρίς να χρειάζεται το qemu να προσομοιώσει κάποιες από τις λειτουργίες, αφού το πιθανότερο είναι αυτές να είναι υλοποιημένες στον φυσικό επεξεργαστή. Αυτό τον κάνει πολύ πιο ευέλικτο από άλλες επιλογές.

- -cpu host: Με αυτή την επιλογή το qemu δίνει στον εικονικό επεξεργαστή όλα τα features τα οποία υποστηρίζει ο πραγματικός επεξεργαστής. Με τον τρόπο αυτό αυξάνεται πολύ η απόδοση του εικονικού μηχανήματος, αφού λειτουργίες που χρειάζεται να χρησιμοποιήσει τις παρέχει απ' ευθείας ο host.

Όπως παρουσιάστηκε στην εισαγωγή, μία σημαντική δυνατότητα των εικονικών μηχανών σε περιβάλλον νέφους είναι η μετανάστευση (migration). Για να είναι όμως ρεαλιστική η μετακίνηση μίας εικονικής μηχανής, χωρίς να παρουσιαστεί κάποια μείωση στην απόδοση, πρέπει το φυσικό μηχάνημα που θα τη φιλοξενήσει να μπορεί να παρέχει τις δυνατότητες που αυτή χρειάζεται. Αν λοιπόν το φυσικό μηχάνημα στο οποίο βρισκόταν είχε κάποιο feature

που το καινούριο δεν έχει το qemu θα αναγκαστεί να το προσομοιώσει· οπότε και η μείωση της απόδοσης να είναι αναμενόμενη.

Από την άλλη, ανάλογα με τις ανάγκες της εικονικής μηχανής και του έργου που έχει να εκτελέσει, μπορεί οι δυνατότητες του φυσικού μηχανήματος να φανούν ιδιαίτερα χρήσιμες και να αυξήσουν πολύ την απόδοσή της. Σε τέτοια περίπτωση συμφέρει να προωθηθούν όλα τα χαρακτηριστικά του φυσικού επεξεργαστή στον εικονικό.

Οπότε γνωρίζοντας τη χρήση για την οποία προορίζονται οι εικονικές μηχανές και κυρίως τις υποδομές τις οποίες έχουμε στη διάθεσή μας επιλέγουμε τη καλύτερη λύση. Φυσικά το qemu είναι σε θέση να χρησιμοποιήσει περισσότερο μοντέλα, στο πλαίσιο της παρούσης εργασίας ωστόσο θα μελετηθούν τα δύο μοντέλα που θεωρούνται τα πιο αντιπροσωπευτικά και καταλληλότερα για υλοποίηση υπηρεσιών IaaS.

3.2 Αναλυτική περιγραφή αλγορίθμων

3.2.1 Throughput Balance

Με τον συγκεκριμένο χρονοδρομολογητή γίνεται απόπειρα να υπάρξει μία κανονικοποίηση στο IPC των εικονικών μηχανών. Αυτό σημαίνει πως οι εικονικές μηχανές θα έχουν παραπλήσιο IPC.

Βασική ιδέα πίσω από την πολιτική του αλγορίθμου είναι εντοπίζει το πόσο επηρεάζει μία διεργασία ο ανταγωνισμός για κοινούς πόρους με το να συγκρίνει το throughput που πετυχαίνει με το ιδανικό (αυτό που θα είχε αν έκανε αποκλειστική χρήση των πόρων). Ύστερα κατατάσσει τις διεργασίες με βάση τη διαφορά $IPC_{real} - IPC_{alone}$ και δίνει χρόνο στις περισσότερο αδικημένες.

Ο αλγόριθμος χρησιμοποιεί τη συνάρτηση quicksort για να ταξινομήσει τις εφαρμογές με βάση την πρόοδο που έχουν κάνει όσο βρίσκονται στον επεξεργαστή. Για την ταξινόμηση χρησιμοποιείται σαν κριτήριο η ”ηλικία” της εκάστοτε εφαρμογής όπως κάνει και ο CFS, ωστόσο πλέον προστίθεται σαν παράμετρος και η πρόοδος που έκανε η εφαρμογή στο προηγούμενο κβάντο χρόνου. Η πρόοδος ορίζεται ως:

$$Progress = \frac{IPC_{last\_quantum}}{IPC_{alone}}$$

Στη συνέχεια υπολογίζεται η ποινή ανταγωνισμού ως:

$$CP = 1 - Progress$$

Και η νέα ηλικία της εφαρμογής:

$$Age = \frac{WT}{n \cdot TQ} + CP$$

όπου WT είναι ο χρόνος που η εφαρμογή περιμένει εκτός επεξεργαστή (waiting time), ενώ TQ είναι το κβάντο χρόνου του συστήματος (time quantum). Το πρώτο μέρος είναι η γήρανση λόγω αναμονής ενώ το δεύτερο η γήρανση λόγω ανταγωνισμών. Αφού ο χρονοδρομολογητής ταξινομήσει τις εφαρμογές (στη περίπτωση που εξετάζουμε τις εικονικές μηχανές) με βάση την προσαρμοσμένη ηλικία τους, τοποθετεί προς εκτέλεση τις δύο ”γηραιότερες”. Με τον τρόπο αυτό διασφαλίζεται δικαιοσύνη τόσο ως προς τον χρόνο αναμονής, όσο και ως προς την πρόοδο σε κάθε κβάντο.

Το αρνητικό του συγκεκριμένου αλγορίθμου είναι πως πρέπει να υπάρχει μηχανισμός να δίδεται το ιδανικό IPC των εφαρμογών ως παράμετρος για τον σωστό υπολογισμό της προόδου.

Τέλος ο χρονοδρομολογητής έχει ένα ιδιαίτερο χαρακτηριστικό. Δεν κατηγοριοποιεί τις εφαρμογές όπως κάνουν οι περισσότεροι, αλλά δρα εντελώς ανεξάρτητα από το μέρος και

τον τρόπο που εντοπίζονται οι ανταγωνισμοί. Αντιθέτως το μόνο που λαμβάνει υπ' όψιν είναι η επιβράδυνση της κάθε εφαρμογής σε σύγκριση με το άμα έτρεχε μόνη της, κάτι το οποίο του δίνει μεγάλο πλεονέκτημα αφού προσπαθεί να αντιμετωπίσει τους ανταγωνισμούς σε όποιο επίπεδο και αν παρουσιάζονται.

3.2.2 LCA: ένας αποδοτικός χρονοδρομολογητής

Σαν λύση στο πρόβλημα προτάθηκε ένας αρκετά απλός χρονοδρομολογητής που είχε υποσχόμενες μετρήσεις σε baremetal περιβάλλον. Το πρώτο πράγμα που πρέπει να κοιτάξουμε είναι ο τρόπος με τον οποίο κατηγοριοποιούμε τις διεργασίες. Ο LCA παρακολουθεί τον τρόπο με τον οποίο χρησιμοποιείται όλη η ιεραρχία της μνήμης, ώστε να προβλέψει προβλήματα αλληλεπίδρασης και κοινής χρήσης πόρων. Προσπαθεί τελικά να αντιμετωπίσει τον ανταγωνισμό τόσο στη κρυφή μνήμη όσο και στον δίαυλο προς την κεντρική μνήμη.

Κατηγοριοποίηση (Classification)

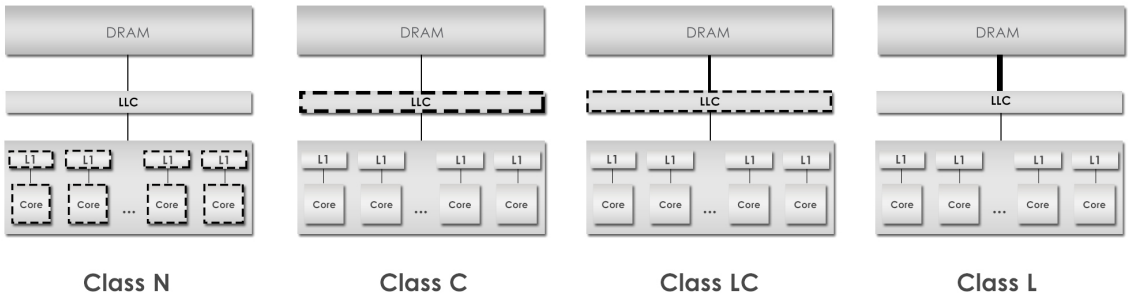
Για την αποφυγή του ανταγωνισμού οι διεργασίες χωρίζονται σε τέσσερις διαφορετικές κλάσεις (N, C, LC, L) χρησιμοποιώντας πληροφορίες που συλλέγονται από performance counters που είναι διαθέσιμοι από τους περισσότερους σύγχρονους επεξεργαστές:

- *Κλάση N*: Περιέχει διεργασίες και εφαρμογές που χρησιμοποιούν ελάχιστα ή καθόλου τους μοιραζόμενους πόρους της ιεραρχίας και περιορίζονται στα ιδιωτικά μέρη του επεξεργαστή. Συνήθως πρόκειται για εφαρμογές που εκτελούν πράξεις πάνω σε δομές δεδομένων πολλές φορές, ενώ οι ίδιες οι δομές είναι αρκετά μικρές ώστε να "χωράνε" στα κατώτερα επίπεδα της ιεραρχίας των κρυφών μνημών (μή μοιραζόμενα επίπεδα).

- *Κλάση C*: Όπως και στη κλάση N, οι εφαρμογές αυτής της κλάσης περιέχουν εφαρμογές που επαναχρησιμοποιούν πολύ τα δεδομένα τους με μία βασική διαφορά: οι δομές δεδομένων είναι αρκετά μεγάλες ώστε να μην μπορούν να περιοριστούν στα πρώτα επίπεδα της κρυφής μνήμης. Έτσι γίνεται μεγάλη χρήση της llc με αποτέλεσμα να υπάρχει μεγάλος ανταγωνισμός σε αυτό το επίπεδο. Οι εφαρμογές αυτές είναι ιδιαίτερα ευαίσθητες στον ανταγωνισμό.

- *Κλάση LC*: Στη κλάση αυτή τοποθετούνται οι εφαρμογές που χρησιμοποιούν τόσο την llc, όσο και τον δίαυλο της κεντρικής μνήμης. Είναι συνήθως εφαρμογές που χρειάζονται μεγάλες δομές δεδομένων στις οποίες εκτελούν μεγάλο όγκο επεξεργασίας. Για τον λόγο αυτό εμφανίζουν ανταγωνισμό τόσο στο επίπεδο της κρυφής μνήμης όσο και στον δίαυλο.

- *Κλάση L*: Η τελευταία κλάση περιέχει εφαρμογές που χρησιμοποιούν ιδιαίτερα τον δίαυλο προς τη μνήμη για να φέρουν δεδομένα στον επεξεργαστή που δεν θα επαναχρησιμοποιήσουν. Είναι εφαρμογές με υψηλά ποσοστά miss rate, εμφανίζουν υψηλό ανταγωνισμό στο επίπεδο του διαύλου και συνήθως είναι καταστροφικές για τις εφαρμογές της κλάσης C (αφού μολύνουν διαρκώς τα block της llc).



Σχήμα 3.4: Κατηγοριοποίηση εφαρμογών

Τα παρακάτω θα αναλυθούν περισσότερο σε επόμενα κεφάλαια, αλλά από κάθε κλάση μπορούμε να κάνουμε κάποιες εκτιμήσεις αναφορικά με τη συμπεριφορά των εφαρμογών που τις αποτελούν.

Για τη κλάση N περιμένουμε πως ο ανταγωνισμός θα είναι ελάχιστος και οι εφαρμογές θα επηρεάζουν και θα επηρεάζονται από ελάχιστα έως καθόλου.

Οι εφαρμογές της κλάσης C είναι ίσως οι δυσκολότερες να εκτιμηθούν, καθώς ανάλογα με τις εφαρμογές που έχουν δίπλα τους, μπορεί να επηρεαστούν από ελάχιστα (κλάσεις N και ίσως C) έως πολύ, αν ανταγωνίζονται για εφαρμογές που ακυρώνουν συνεχώς την cache. Αντίστοιχα αν δεν αντιμετωπίσουν ισχυρό ανταγωνισμό για την llc, οι ίδιες μπορούν να περάσουν σχετικά απαρατήρητες. Σε αντίθετη περίπτωση θα είναι υποχρεωμένες να καταφεύγουν στην κεντρική μνήμη για να επαναφέρουν τα δεδομένα με αποτέλεσμα να εμφανίζουν ανταγωνισμό και στον δίαυλο.

Για τις εφαρμογές της κλάσης LC ισχύουν αντίστοιχες παρατηρήσεις, καθώς έχουν να ανταγωνιστούν τόσο εφαρμογές της κλάσης C (στην LLC), εφαρμογές της κλάσης LC (στην LLC και στον δίαυλο) αλλά το μεγαλύτερη καθυστέρηση τη συναντούν όταν εκτελούνται δίπλα σε εφαρμογές της κλάσης L. Σε αυτήν την περίπτωση θα υποστούν επιβράδυνση τόσο εξαιτίας του ανταγωνισμού στον δίαυλο, όσο και επειδή δεν καταφέρνουν να αξιοποιήσουν επαρκώς την κρυφή μνήμη.

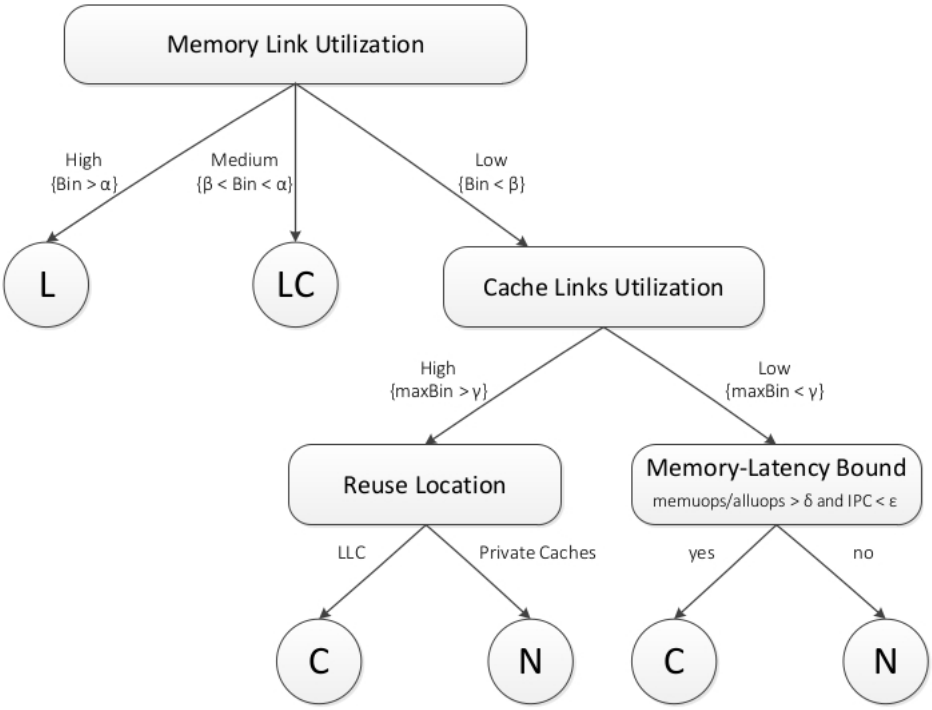
Τέλος οι εφαρμογές στην κλάση L αναμένεται να αντιμετωπίσουν ελάχιστο ως λίγο ανταγωνισμό για τον δίαυλο από εφαρμογές των κλάσεων N και C (στη δεύτερη περίπτωση, όπως αναφέρθηκε και παραπάνω οι αντίστοιχες εφαρμογές αναγκάζονται να καταφύγουν στη κεντρική μνήμη μέσω του διαύλου, αφού δεν καταφέρνουν να αξιοποιήσουν τα διαθέσιμα δεδομένα στη κρυφή μνήμη). Αντίστοιχα η επιβράδυνση θα είναι αισθητά μεγαλύτερη όταν μοιράζονται τον επεξεργαστή με εφαρμογές των κλάσεων LC και L.

Αξίζει να σημειωθεί πως η επιβράδυνση που θα υποστεί μία εφαρμογή της κλάσης C όταν δίπλα της τοποθετηθεί μία ανταγωνίστρια αναμένεται να είναι αισθητά μεγαλύτερη από την επιβράδυνση μίας άλλης κλάσης. Αυτό θα συμβεί επειδή ο χρόνος που θα απαιτηθεί για να μεταφερθούν δεδομένα από τη κύρια μνήμη στη cache είναι πολύ μεγαλύτερος από τον αντίστοιχο για μεταφορά από τη LLC στον επεξεργαστή (miss penalty) και το miss rate θα αυξηθεί ραγδαία. Επίσης να αναφέρουμε πως οι επιπτώσεις των ανταγωνισμών πρέπει να είναι ανάλογες με τα δεδομένα στα οποία εργάζεται μία διεργασία. Έτσι μπορεί μία εφαρμογή να χαρακτηριστεί ως C, επειδή επαναχρησιμοποιεί αρκετά τα δεδομένα της, ωστόσο άμα αυτά καταλαμβάνουν ένα σχετικά μικρό μέρος της κοινής κρυφής μνήμης δεν αναμένεται ο ανταγωνισμός να έχει ιδιαίτερο αντίκτυπο στη συνολική απόδοση.

Για την τοποθέτηση μίας εφαρμογής στην αντίστοιχη κλάση έχουν θεσπιστεί πέντε όρια που καθορίζουν το ποσοστό στο οποίο η εφαρμογή χρησιμοποιεί τη LLC και το memory bus.

$\alpha = 0.5 \cdot B_{max}$	Υψηλή χρήση διαύλου προς τη μνήμη.
$\beta = 0.025 \cdot B_{max}$	Μέτρια χρήση διαύλου προς τη μνήμη.
$\gamma = 0.15 \cdot B_{max}$	Υψηλή χρήση του διαύλου μεταξύ llc και cpu.
$\delta = \frac{mem\_uops}{all\_uops} = 0.25$	Όριο που δείχνει τη χρήση της μνήμης από την εφαρμογή
$\epsilon = 0.25 \cdot IPC_{max}$	Όριο που αν ξεπεραστεί σημαίνει πως μία εφαρμογή περιορίζεται στη CPU (κλάση N).

Αφού λοιπόν καθοριστούν οι αντίστοιχες παράμετροι, η κάθε εφαρμογή τοποθετείται στην κατάλληλη κλάση σύμφωνα με το ακόλουθο σχήμα.



Σχήμα 3.5: Δέντρο επιλογής κλάσης

Ο αλγόριθμος

Ο αλγόριθμος σε πρώτο στάδιο θα ταιριάξει όλες τις εφαρμογές της κλάσης N όσες περισσότερες εφαρμογές πρώτα της κλάσης L και ύστερα της κλάσης C ώστε να τις προστατέψει από τους ανταγωνισμούς. Στη συνέχεια χρησιμοποιεί τη κλάση LC και αν περισσεύουν τη κλάση N όπου δεν εντοπίζεται ανταγωνισμός. Έπειτα, δεδομένου πως δεν υπάρχουν εφαρμογές N, θα αντιστοιχίσει τις C πρώτα με εφαρμογές LC και ύστερα με C. Τις εφαρμογές LC θα τις ταιριάξει με εφαρμογές L και ύστερα LC ενώ τελικά τις L μπορεί να τις ταιριάξει μόνο με L.

Ο αλγόριθμος είναι σχεδιασμένος έτσι ώστε να αποφεύγονται όσο το δυνατόν περισσότερο συνδυασμοί L - C, L - L και L - LC. Είναι άπληστος με πολυπλοκότητα $O(n)$ και προσπαθεί να φτιάξει ζευγάρια με την προκαθορισμένη σειρά που αναπτύχθηκε παραπάνω.

Algorithm 1: LCA co-scheduling algorithm

Input: L,LC,C,N: List of applications in classes L, LC, C, N, respectively

foreach *x in N* **do**
    $y \leftarrow popFromFirstNonEmpty(L, C, LC, N)$;
    co-schedule(x, y);

foreach *x in C* **do**
    $y \leftarrow popFromFirstNonEmpty(LC, C)$;
    co-schedule(x, y);

foreach *x in LC* **do**
    $y \leftarrow popFromFirstNonEmpty(L, LC)$;
    co-schedule(x, y);

foreach *x in L* **do**
    $y \leftarrow popFrom(L)$;
    co-schedule(x, y);

Κεφάλαιο 4

Αποτελέσματα μετρήσεων

Στο τέταρτο κεφάλαιο θα καταγραφούν τα αποτελέσματα των μετρήσεων και θα γίνει ένας σύντομος σχολιασμός αυτών.

4.1 Κατάταξη πειραματικών εφαρμογών

Για την εκτέλεση των πειραμάτων χρησιμοποιήθηκαν κάποιες εφαρμογές με διαφορετική συμπεριφορά κάθε φορά. Κάποιες χρησιμοποιούν και επωφελούνται ιδιαίτερα από τη κοινή κρυφή μνήμη, κάποιες άλλες χρησιμοποιούν πολύ τον δίαυλο προς τη μνήμη, ενώ δεν έλειπαν και αυτές που δεν χρησιμοποιούν καθόλου τους κοινούς πόρους του συστήματος οπότε και περιορίζονται στα ιδιωτικά μέρη του επεξεργαστή. Για τον διαχωρισμό τους και τη τοποθέτησή τους στις διαφορετικές κλάσεις που περιγράψαμε παραπάνω χρησιμοποιήθηκαν μετρητές (performance counters) που είναι διαθέσιμοι από τη CPU. Έτσι, με χρήση του scaff καταγράφηκαν διάφορες μετρήσεις που προέκυψαν από την εκτέλεση των εφαρμογών. Για την εκτέλεση χρησιμοποιήθηκαν 4 πυρήνες για κάθε εφαρμογή και έγινε καταγραφή των μετρικών που αναφέρθηκαν σε προηγούμενο κεφάλαιο. Ανάλογα με τις τιμές αυτές και με χρήση του παραπάνω αλγορίθμου έγινε η κατάταξη των εφαρμογών στις αντίστοιχες κλάσεις.

Στον πίνακα του σχήματος 4.1 φαίνονται για κάθε εφαρμογή, η κλάση τους καθώς και οι μετρήσεις που τις κατέταξαν στην εκάστοτε κλάση.

BENCHMARK	TIME	IPC	MEM_INT	L1 BW	L2 BW	L3 BW	L2 REUSE	L3 REUSE
2mm	11522	0,36244	161463,9	24,2906	23,8575	0,01211	1,02	2976,56
3mm	16875	0,34959	144555,7	24,7463	24,587	0,01044	1,01	3540,53
cholesky	6454	1,93181	11988,6	24,07	31,6936	0,00232	0,76	13695,15
doitgen	11149	2,28343	674106	157,255	0,51484	0,35949	305,44	1,43
durbin	6720	0,17094	3173099	6,41373	4,83663	0,53151	1,33	9,55
ep.A.g	10329	0,69053	540,1111	0,96102	0,99908	0,00035	0,96	2873,81
ft.a	3218	1,48384	25106030	30,8976	5,90815	2,71128	5,11	2,13
gemm	14788	0,37459	140618,3	23,6442	23,4134	0,01239	1,01	2737,11
gemver	8800	0,50098	15242015	16,8101	12,9777	2,88935	1,3	4,49
gesummv	7476	0,76509	3125849	6,24137	4,97074	4,73299	1,26	1,05
gramschmidt	13712	0,15105	1123,833	12,8834	12,8203	7,7E-05	1	180420,57
jacLR	12467	0,6496	31537204	15,7279	10,4819	10,2223	1,5	1,03
mvt	11142	0,3602	15232444	16,0193	13,7676	2,39423	1,16	5,75
stream_d0	6869	0,7618	15163878	13,2704	13,275	12,9883	1	1,02
syr2k	16143	1,99351	2613654	24,1557	22,1015	2,48797	1,09	8,88
syrk	15552	2,53967	144740,3	22,2814	22,2844	0,02923	1	768,55
trisolv	8064	0,58659	3372052	2,88374	2,63103	2,189	1,1	1,2
trmm	14673	2,38154	977742,2	21,5023	26,1521	0,43184	0,82	60,58

Σχήμα 4.1: Κατάταξη εφαρμογών σε κλάσεις

4.2 Πειραματική διαδικασία

Σκοπός είναι να μελετηθεί η επίδραση που έχει ο ανταγωνισμός για κοινούς πόρους που προκύπτει όταν δύο (ή περισσότερες) εικονικές μηχανές εκτελούνται παράλληλα σε ένα σύστημα.

4.2.1 Classification

Για την κατάταξη των εφαρμογών σε κλάσεις, γίνεται συλλογή από διάφορους performance counters για στοιχεία που φανερώνουν τη συμπεριφορά της εφαρμογής κατά την εκτέλεσή της. Το πρώτο βήμα είναι να εκτελέσουμε τις εφαρμογές αυτόνομα, σε έναν πυρήνα του συστήματος ώστε να συλλέξουμε συγκεκριμένες πληροφορίες που θα μας βοηθήσουν στη συνέχεια στην κατάταξη.

- LLC misses: Με τη μελέτη των αστοχιών της LLC, μπορούμε να δούμε τη χρήση του διαύλου της μνήμης που κάνει η εφαρμογή. Αφού κάθε αστοχία σημαίνει μεταφορά μίας γραμμής από τη μνήμη (64 bytes) το bandwidth που χρησιμοποιείται θα είναι $64 \cdot \frac{llc\_misses}{T}$ (B/s), όπου T είναι η περίοδος στην οποία μετρήθηκαν οι αστοχίες.
- L1 lines: Αυτή η μέτρηση μας δείχνει πόσες γραμμές μεταφέρθηκαν από την L2 cache στην L1 ανα μονάδα χρόνου T. Με γραμμή 64 bytes το υπολογιζόμενο bandwidth μεταξύ L2 και L1 είναι $64 \cdot \frac{l1lines}{T}$ (B/s).
- L2 lines: Όμοια με παραπάνω, αυτή η μέτρηση μας βοηθάει να δούμε το bandwidth μεταξύ των L3 (LLC) και L2 cache που χρησιμοποιεί η εφαρμογή. Αυτό είναι $64 \cdot \frac{l2lines}{T}$ (B/s).
- Per core bandwidth: Με αυτή τη μέτρηση, βλέπουμε τα δεδομένα (σε 64B lines) που μεταφέρονται από τη μνήμη σε συγκεκριμένο πυρήνα το χρονικό διάστημα T. Το bandwidth υπολογίζεται ως $64 \cdot \frac{per\_core\_bandwidth}{T}$ (B/s).
- Bandwidth: Πρόκειται για τα συνολικά δεδομένα (ξανά σε lines των 64B) που μεταφέρονται από τη μνήμη στη cpu το διάστημα T. Για τον υπολογισμό του συνολικού bandwidth χρησιμοποιούμε τον τύπο $64 \cdot \frac{bandwidth}{T}$ (B/s).
- Power: Σημαντικός μετρητής που δείχνει την ισχύ που καταναλώνει η cpu. Στη παρούσα εργασία δεν θα χρησιμοποιηθεί και αναφέρεται επιγραμματικά.

Το επόμενο βήμα είναι να γίνει μία κατάταξη των εφαρμογών στις κλάσεις που περιγράφει σαν παραπάνω. Με χρήση των μετρήσεων που κάνουμε υπολογίζουμε τις παραμέτρους του αλγορίθμου που περιγράφηκε σε προηγούμενο κεφάλαιο και είμαστε σε θέση να κατατάξουμε τις εφαρμογές. Έτσι για παράδειγμα, αν δούμε πως μια εφαρμογή έχει πολύ υψηλή τιμή L1 lines αλλά χαμηλή τιμή L2 lines και χαμηλή τιμή bandwidth καταλαβαίνουμε πως περιορίζεται στα ιδιωτικά μέρη του επεξεργαστή (κλάση N). Αντίστοιχα αν δούμε χαμηλή τιμή στο bandwidth αλλά υψηλή L2 lines σημαίνει πως η εφαρμογή χρησιμοποιεί και αξιοποιεί την L3 cache η οποία είναι μοιραζόμενη (κλάση C).

4.2.2 Αναλυτικοί πίνακες αποτελεσμάτων (Heatmaps)

Αφού βάλουμε σε κλάσεις τις εφαρμογές, πρέπει να δούμε πως αλληλεπιδρούν αυτές μεταξύ τους. Για τον σκοπό αυτό τοποθετήθηκαν σε ένα package (8 πυρήνες που μοιράζονται την ίδια L3 cache και το ίδιο δίαυλο προς τη μνήμη) ανα δύο οι εφαρμογές. Έτσι δημιουργήθηκαν $17 \cdot 17 = 289$ συνδυασμοί. Κάθε μία από τις δύο εφαρμογές που έτρεχαν παράλληλα

είχε στη διάθεσή της 4 από τους 8 πυρήνες. Με αυτόν τον τρόπο βλέπουμε τη επίδραση που έχει μία εφαρμογή σε κάθε άλλη και κατ επέκταση μία κλάση σε κάθε άλλη. Στη συνέχεια θα γίνει μία ανασκόπηση των αποτελεσμάτων που αναμένουμε να δούμε και σε επόμενο στάδιο θα παρουσιαστούν τα αποτελέσματα που είχαμε μαζί με σύντομο σχολιασμό τους.

Σαν ορισμός της επιβράδυνσης δίνεται ο εξής:

$$Slowdown = \frac{t_{cosched}}{t_{alone}} - 1$$

όπου $t_{cosched}$ είναι ο χρόνος που χρειάστηκε μία εφαρμογή για να ολοκληρώσει την εκτέλεσή της όσο έτρεχε μαζί με μία άλλη, ενώ t_{alone} ο αντίστοιχος χρόνος όσο έτρεχε μόνη της.

4.2.3 Αλγόριθμοι χρονοδρομολόγησης

Το επόμενο βήμα είναι να επιλέξουμε τους αλγόριθμους χρονοδρομολόγησης που θα συγκρίνουμε. Έχουμε ήδη περιγράψει επιγραμματικά τις διαφορετικές πολιτικές που μελετήθηκαν σε αυτήν την εργασία, με μεγαλύτερη έμφαση να δίνεται στους χρονοδρομολογητές LCA και Throughput Balance. Ο τελευταίος αναμένεται να έχει σημαντική επίδραση στις εφαρμογές και να είναι αξιοποιήσιμος καθώς είναι σε θέση να παρέχει εγγύηση για την ποιότητα υπηρεσιών (QoS). Κάτι τέτοιο σημαίνει πως κάθε εικονική μηχανή θα έχει εγγυημένη πρόοδο ανα μονάδα χρόνου, αφού ο χρόνος που βρίσκεται στον επεξεργαστή θα είναι ανάλογος με την επίδραση που θα έχει πάνω της ο ανταγωνισμός.

4.2.4 Εκτέλεση πειραμάτων σε εικονικό περιβάλλον

Κεντρικός σκοπός αυτής της εργασίας είναι η μελέτη της επίδρασης που έχει ο ανταγωνισμός στην εκτέλεση εικονικών μηχανών. Για την πειραματική διαδικασία, χρησιμοποιήθηκε και πάλι το scaff με το οποίο ξεκινούν δύο εικονικές μηχανές. Σε κάθε εικονική μηχανή ανατίθενται 4 πυρήνες περιορίζοντας την εκτέλεση σε ένα package. Με χρήση των performance counters είμαστε σε θέση να παρακολουθούμε την επίδοση του συστήματος και να καθορίσουμε τον βέλτιστο συνδυασμό. Για την παραγωγή των αναλυτικών πινάκων ξεκινάμε δύο μηχανήματα όπου το πρώτο θα εκτελεί τα 17 benchmark για 1530 δευτερόλεπτα το κάθε ένα ενώ το δεύτερο εκτελεί τα 17 benchmark για 90 δευτερόλεπτα το κάθε ένα (1530 δευτερόλεπτα συνολικά) από 17 φορές. Για την παρακολούθηση των εφαρμογών χρησιμοποιήθηκε το εργαλείο time του linux ενώ τα αποτελέσματα καταγράφονταν σε αρχείο μέσα στα εικονικά μηχανήματα ώστε να ανακτηθούν αργότερα. Για τον καλύτερο συγχρονισμό χρησιμοποιήθηκαν dummy αρχεία σαν locks τα οποία ήταν διακριτά και από τα δύο μηχανήματα με χρήση του sshfs. Με αυτόν το τρόπο ήμασταν σε θέση να γνωρίζουμε πότε τελείωσαν τα 17 benchmarks της δεύτερης μηχανής ώστε να προχωρήσουμε στην επόμενη επανάληψη.

Όπως αναφέρθηκε παραπάνω, για να χρησιμοποιηθεί ο δρομολογητής thbal, χρειάζεται το ιδανικό IPC (IPC που θα είχε η εικονική μηχανή, άμα έτρεχε μόνη της στον επεξεργαστή). Για να βρεθεί το ιδανικό IPC, ένα εικονικό μηχάνημα με 4 πυρήνες ξεκίνησε από το scaff και έτρεξε όλα τα benchmarks το ένα μετά το άλλο (με ένα κενό διάστημα 5 δευτερολέπτων ώστε να ξεχωρίζουν οι μετρήσεις μεταξύ τους). Στη συνέχεια έγινε ανάλυση των αποτελεσμάτων ώστε να βρεθεί ο μέσος όρος του IPC που είχε το εικονικό μηχάνημα κατά τη διάρκεια που εκτελούσε το κάθε benchmark.

4.2.5 Results estimation

Στις επόμενες παραγράφους, θα αναλυθούν συνοπτικά αυτά που περιμέναμε να δούμε από τις μετρήσεις. Θα συζητηθούν όλοι οι πιθανοί συνδυασμοί εφαρμογών που ανήκουν σε κάποια κλάση.

- N - Any: Από τη στιγμή που οι διεργασίες θα εκτελούνται σε διαφορετικούς πυρήνες (αυτό συμβαίνει πάντα στη παρούσα εργασία), οι περισσότερες εφαρμογές αυτής της κλάσης δεν αναμένεται να επηρεάζουν ή να επηρεάζονται από άλλες. Παρ' όλα αυτά, μπορεί να υπάρξουν περιπτώσεις που εφαρμογές με μικρή χρήση της L3 cache θα χαρακτηριστούν σαν N. Σε αυτές τις περιπτώσεις, αναμένεται να παρατηρηθεί μία σχετικά μικρή επιβράδυνση που θα είναι πολύ μικρότερη από 2.

- C - C: Στη κλάση C, ταξινομούνται εφαρμογές που μπορεί να διαφέρουν πολύ μεταξύ τους. Δηλαδή μπορεί μία εφαρμογή να κάνει βαριά χρήση της L3 cache και να επιταχύνεται ιδιαίτερα από τη χρήση της, οπότε και να είναι ιδιαίτερα ευάλωτη στον ανταγωνισμό. Από την άλλη μία εφαρμογή μπορεί να βρεθεί σε αυτή τη κλάση, κάνοντας μία σχετικά μικρή χρήση της LLC ή να χρησιμοποιεί μόνο ένα τμήμα αυτής. Σε κάθε περίπτωση πάντως, αναμένουμε να δούμε τις εφαρμογές αυτής της κλάσης να επηρεάζονται σε μεγάλο βαθμό από όμοιες εφαρμογές. Και οι δύο στη προσπάθειά τους να αξιοποιήσουν την αρχιτεκτονική του συστήματος θα ακυρώνουν τα δεδομένα που χρειάζεται η μία ή η άλλη με αποτέλεσμα να χάνεται κάθε πλεονέκτημα που προσφέρει η cache.

- C - LC: Σε αυτό το σενάριο, υπάρχει περιμένουμε ανταγωνισμό κυρίως στη LLC. Από αυτόν τον ανταγωνισμό περισσότερο θα επηρεαστεί η εφαρμογή της κλάσης C και λιγότερο η LC. Όμως, εκτός από τον ανταγωνισμό για την l3 cache, είναι πιθανό να εμφανιστεί ανταγωνισμός και στον δίαυλο, αν η εφαρμογή της κλάσης C χρησιμοποιεί μεγάλο τμήμα της cache. Σε αυτή την περίπτωση θα καταφεύγει συνεχώς στη μνήμη για να αναπληρώσει τις αστοχίες που προκαλούνται οπότε θα χρησιμοποιεί περισσότερο τον δίαυλο για τον οποίο και θα ανταγωνίζεται. Αντίστοιχα, η εφαρμογή της κλάσης LC, αναμένεται να παρουσιάσει μία σχετικά μικρή επιβράδυνση λόγω του ανταγωνισμού τόσο στην LLC όσο και στον δίαυλο.

- C - L: Το χειρότερο δυνατό σενάριο για μία εφαρμογή της κλάσης C, είναι να μοιράζεται την L3 cache με εφαρμογή L. Σε αυτή τη περίπτωση αναμένεται μεγάλη επιβράδυνση της πρώτης, αφού τα δεδομένα που θα φέρνει στην κρυφή μνήμη θα ακυρώνονται συνεχώς με αποτέλεσμα να χάνεται κάθε πλεονέκτημα που προσφέρει η cache. Όπως και στη περίπτωση C-LC, κάποιος περιορισμένος ανταγωνισμός θα παρουσιαστεί και στον δίαυλο, προερχόμενος από την προσπάθεια της εφαρμογής C να φέρει δεδομένα από τη κεντρική μνήμη στη LLC. Η εφαρμογή L αναμένεται να επηρεαστεί σε μικρό βαθμό από αυτόν τον ανταγωνισμό.

- LC - LC: Πρόκειται για ένα συνδυασμός η συμπεριφορά του οποίου είναι δύσκολο να προβλεφθεί με ακρίβεια. Και οι δύο εφαρμογές χρησιμοποιούν ως ένα βαθμό τόσο την μοιραζόμενη κρυφή μνήμη όσο και τον δίαυλο προς την κεντρική μνήμη. Αυτό σημαίνει πως στο χειρότερο σενάριο θα παρατηρείται μία επιβράδυνση κοντά στο 100%, ενώ στη μέση περίπτωση θα αναμένεται μία πιο ήπια αλλά και πάλι σημαντική επιβράδυνση. Αυτό θα εξαρτάται κυρίως από το πόσο οι δύο εφαρμογές αξιοποιούν την L3 cache.

- LC - L: Σε αυτή την περίπτωση ανταγωνισμός θα παρουσιαστεί κυρίως στον δίαυλο. Η επιβράδυνση δεν αναμένεται να είναι δραματική, ωστόσο αν η LC εφαρμογή χρησιμοποιεί σε μεγάλο βαθμό την LLC, τα δεδομένα της θα ακυρώνονται συνεχώς με αποτέλεσμα να αυξάνεται ακόμα περισσότερο ο ανταγωνισμός στον δίαυλο. Παρομοίως η εφαρμογή της κλάσης L, αναμένεται να παρουσιάσει μία ήπια επιβράδυνση, κυρίως λόγω του πρωτογενούς ανταγωνισμού στον δίαυλο (δηλαδή του ανταγωνισμού

που προκαλείται από την πρώτη μεταφορά δεδομένων από τη μνήμη και όχι από τη μεταφορά δεδομένων που βρίσκονταν στη cache αλλά ακυρώθηκαν στη συνέχεια).

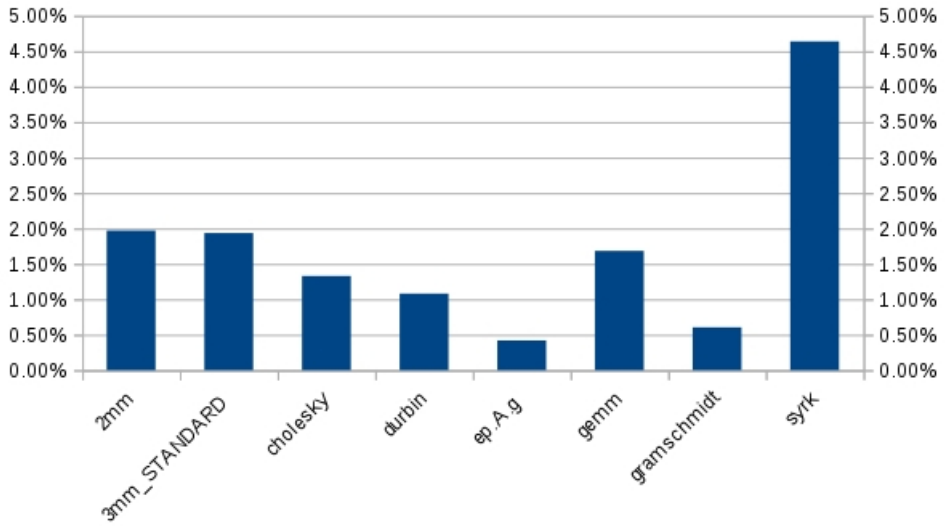
- L - L: Πρόκειται για τον συνδυασμό που παρουσιάζει το μικρότερο ενδιαφέρον, αφού η συμπεριφορά είναι εύκολο να προβλεφθεί, αλλά και τις μεγαλύτερες επιβραδύνσεις. Οι δύο εφαρμογές κάνουν βαριά χρήση του διαύλου και θα είναι υποχρεωμένες να τον μοιράζονται (όχι αναγκαστικά ισοδύναμα). Έτσι θα τον χρησιμοποιούν τον μισό χρόνο κατά μέσο όρο από ότι αν έτρεχαν μόνες τους. Έτσι αναμένεται να παρουσιάσουν επιβραδύνσεις της τάξης του 100%. Τέλος αξίζει να σημειωθεί, πως όταν τον δίαυλο τον μοιράζονται εφαρμογές L, σε μέση τιμή ο παράγοντας επιβράδυνσης θα είναι ανάλογος με το πλήθος των εφαρμογών που μοιράζονται τον δίαυλο.

4.3 Πειράματα απ ευθείας στον host

Στην επόμενη παράγραφο θα καταγράψουμε τα αποτελέσματα που προέκυψαν από τις μετρήσεις που έγιναν απευθείας στο host μηχάνημα χωρίς το εικονικό περιβάλλον να παρεμβάλλεται μεταξύ των εφαρμογών και του υλικού. Ταυτόχρονα γίνεται προσπάθεια να επεξηγηθεί κάθε μέτρηση, ψάχνοντας την αιτία πίσω από αποτελέσματα διαφορετικά από αυτά που αναμέναμε.

- N - Any: Αντιλαμβανόμαστε από το διάγραμμα του σχήματος 4.5 πως όλες οι κλάσεις επηρεάζονται ελάχιστα (στα όρια του στατιστικού λάθους) από τις εφαρμογές της κλάσης N. Συμπέρασμα αναμενόμενο αφού ο ανταγωνισμός είναι ελάχιστος. Αντίστοιχα και οι εφαρμογές της κλάσης N επηρεάζονται ελάχιστα από όλες τις άλλες.

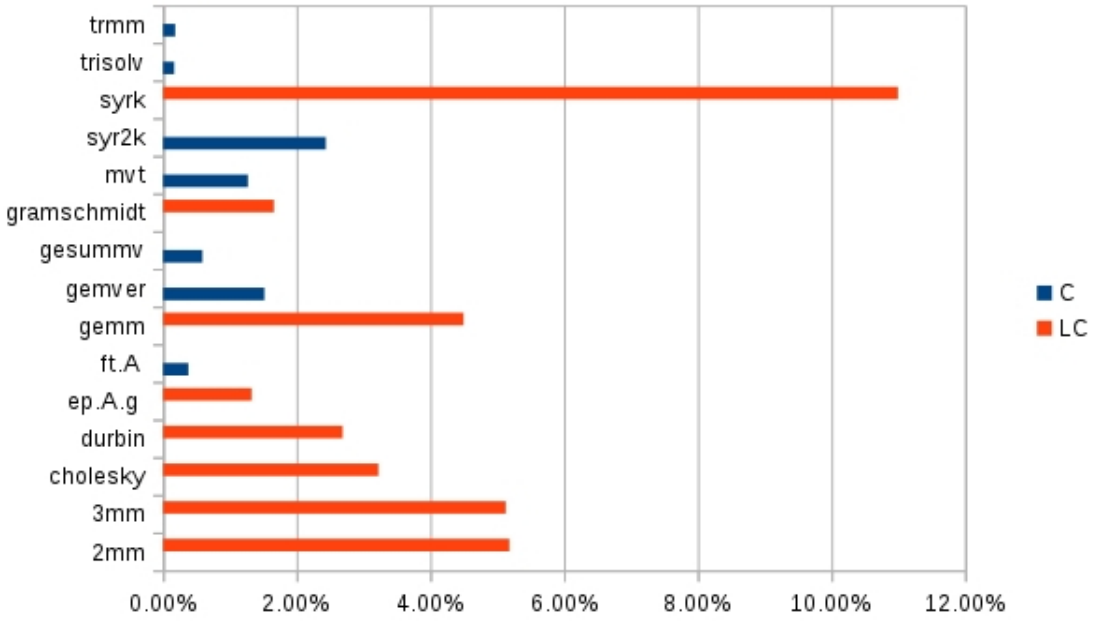
- C - C: Όπως φαίνεται και στο διάγραμμα μία επιβράδυνση της τάξης του 7% παρατηρείται όταν δύο εφαρμογές της κλάσης C μοιράζονται τον επεξεργαστή. Αυτά τα αποτελέσματα είναι σύμφωνα με την εκτίμηση που έγινε νωρίτερα. Επίσης πρέπει να σημειωθεί πως επιβεβαιώνεται πως η κλάση C ένα αρκετά μεγάλο εύρος από εφαρμογές, που άλλες αξιοποιούν περισσότερο και άλλες λιγότερη την LLC. Αυτό γίνεται αντιληπτό από το γεγονός πως κάποιες εφαρμογές επηρεάζονται αρκετά περισσότερο από άλλες. Στο σχήμα 4.2 φαίνονται οι επιβραδύνσεις της εκάστοτε εφαρμογής όταν αυτή εκτελεστεί παράλληλα με μία εφαρμογή κλάσης C.



Σχήμα 4.2: Επιβραδύνσεις μεταξύ εφαρμογών της κλάσης C

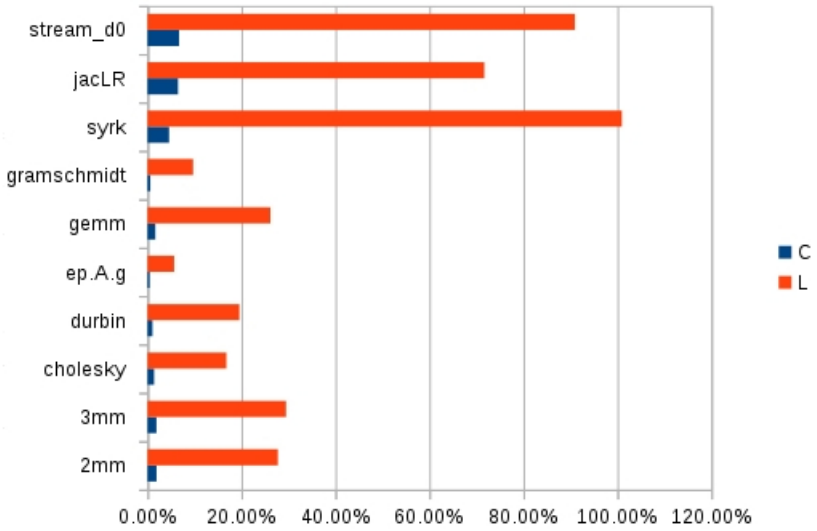
- C - LC: Και πάλι τα αποτελέσματα επιβεβαιώνουν τις προβλέψεις μας, αφού φαίνεται και απ' το διάγραμμα μία περιορισμένη επιβράδυνση και στις δύο εφαρμογές. Αυτή που επηρεάζεται κυρίως από τους ανταγωνισμούς είναι η εφαρμογή της κλάσης C, αφού είναι αυτή που χάνει θα χάσει τα δεδομένα της από την LLC. Η επιβράδυνση είναι της τάξης του 10% για την εφαρμογή C και της τάξης του 7% για αυτή της LC.

Στο σχήμα 4.3 παρουσιάζονται γραφικά οι επιβραδύνσεις των εφαρμογών της κλάσης C όταν τρέχουν με εφαρμογές της κλάσης LC και αντίστοιχα αυτές της κλάσης LC όταν τρέχουν με εφαρμογές της C.



Σχήμα 4.3: Επιβραδύνσεις μεταξύ εφαρμογών των κλάσεων C και LC

- C - L: Σε αυτή την περίπτωση τα αποτελέσματα παρουσιάζουν ιδιαίτερο ενδιαφέρον. Οι εφαρμογές της κλάσης C επηρεάζονται ανάλογα με το πόσο χρησιμοποιούν την κοινή cache. Έτσι βλέπουμε πως κάποιες από τις εφαρμογές επιβραδύνονται ελάχιστα, ενώ άλλες υποφέρουν από τον ανταγωνισμό πολλαπλασιάζοντας τον χρόνο εκτέλεσής τους. Από την άλλη η εφαρμογές L όπως είναι λογικό επηρεάζονται σε μικρό αλλά αξιοσημείωτο βαθμό. Αρχικά δεν φαίνεται να βρίσκουν κάποιον ανταγωνισμό, αφού η εφαρμογή της κλάσης C δεν δείχνει να χρησιμοποιεί τον δίαυλο. Αυτό όμως αλλάζει γρήγορα, αφού θα αναγκαστεί να καταφύγει στην κεντρική μνήμη για να ξαναφέρει στις cache τα block που έχουν ακυρωθεί. Για τον λόγο αυτό, βλέπουμε αυτό το μικρό slowdown και στις εφαρμογές L.



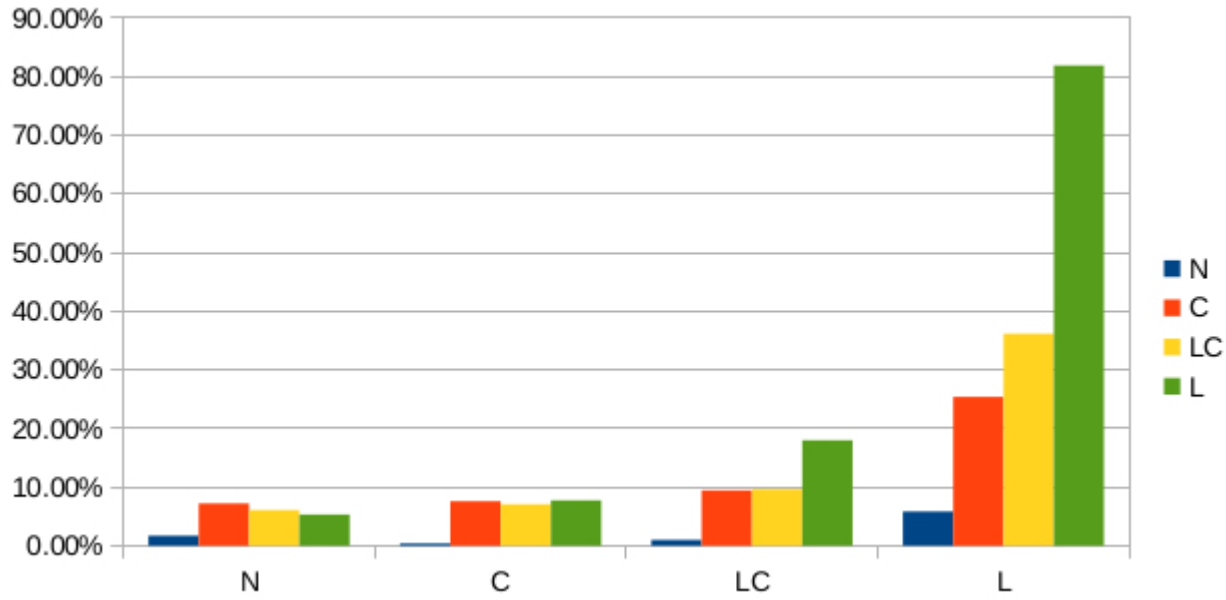
Σχήμα 4.4: Επιβραδύνσεις μεταξύ εφαρμογών των κλάσεων C και L

- LC - LC: Από τα αποτελέσματα παρατηρούμε πως οι επιβραδύνσεις είναι ανάλογες με τις αναμενόμενες. Δεν έχουμε κάποια επιβράδυνση σε υπερβολικό βαθμό αλλά πάντα είναι αισθητή.

- LC - L: Αν μία εφαρμογή της κλάσης L τοποθετηθεί δίπλα σε εφαρμογή LC, θα παρατηρηθεί σημαντική επιρροή και από την μία και από την άλλη πλευρά. Ανάλογα με την χρήση της L3 cache που κάνει η LC θα έχουμε και τις αντίστοιχες επιβραδύνσεις. Όπως και στον συνδυασμό C - L, έτσι και εδώ παρατηρείται επιπλέον ανταγωνισμός στον -έτσι κι αλλιώς περιζήτητο- δίαυλο λόγω των μπλοκ της cache που ακυρώνονται από την L εφαρμογή.

- L - L: Όπως περιμέναμε, σε αυτή την περίπτωση θα παρουσιάζεται πολύ υψηλός ανταγωνισμός στον δίαυλο προς τη μνήμη με τις δύο εφαρμογές να περιορίζονται και να παρουσιάζουν επιβραδύνσεις συχνά μεγαλύτερες του 90%. Στην γενική περίπτωση η επιβράδυνση όταν δύο εφαρμογές L μοιράζονται τον δίαυλο θα είναι κοντά στο 100% με χειρότερη περίπτωση να ξεπερνά αυτό το νούμερο αφού ο διαμοιρασμός δεν είναι ισόποσος.

Στο διάγραμμα του σχήματος 4.5 φαίνονται συνοπτικά οι επιβραδύνσεις που είχαν οι εφαρμογές των τεσσάρων κλάσεων ανάλογα με τις εφαρμογές που εκτελούνταν ταυτόχρονα σε γειτονικούς πυρήνες.



Σχήμα 4.5: Επιβράδυνση σε bm περιβάλλον

4.4 Πειράματα σε εικονική μηχανή με -cpu QEMU64

Αφού αναλύθηκαν τα αποτελέσματα των μετρήσεων σε baremetal περιβάλλον, πρέπει να περάσουμε και στα αποτελέσματα των εικονικών μηχανών. Σε γενικές γραμμές αναμέναμε παραπλήσια αποτελέσματα. Πράγματι όπως φαίνεται στο διάγραμμα, οι επιβραδύνσεις είναι ανάλογες με αυτές της baremetal εκτέλεσης. Ωστόσο υπάρχουν μερικές διαφορές που τράβηξαν το ενδιαφέρον και χωράνε περαιτέρω ανάλυση.

Συγκεκριμένα βλέπουμε πως οι εφαρμογές της κλάσης LC δεν επηρεάζονται όσο πριν ειδικά όταν τρέχουν δίπλα σε εφαρμογές N, C ή και LC. Συγκεκριμένα βλέπουμε επιβραδύνσεις από 0 έως 3.5%. Αντίθετα όταν εκτελούνται μαζί με εφαρμογή τάξης L, παρουσιάζουν την ίδια συμπεριφορά με πριν. Αντίστοιχα φαίνεται πως οι C εφαρμογές δεν επηρεάζονται τόσο πολύ μεταξύ τους.

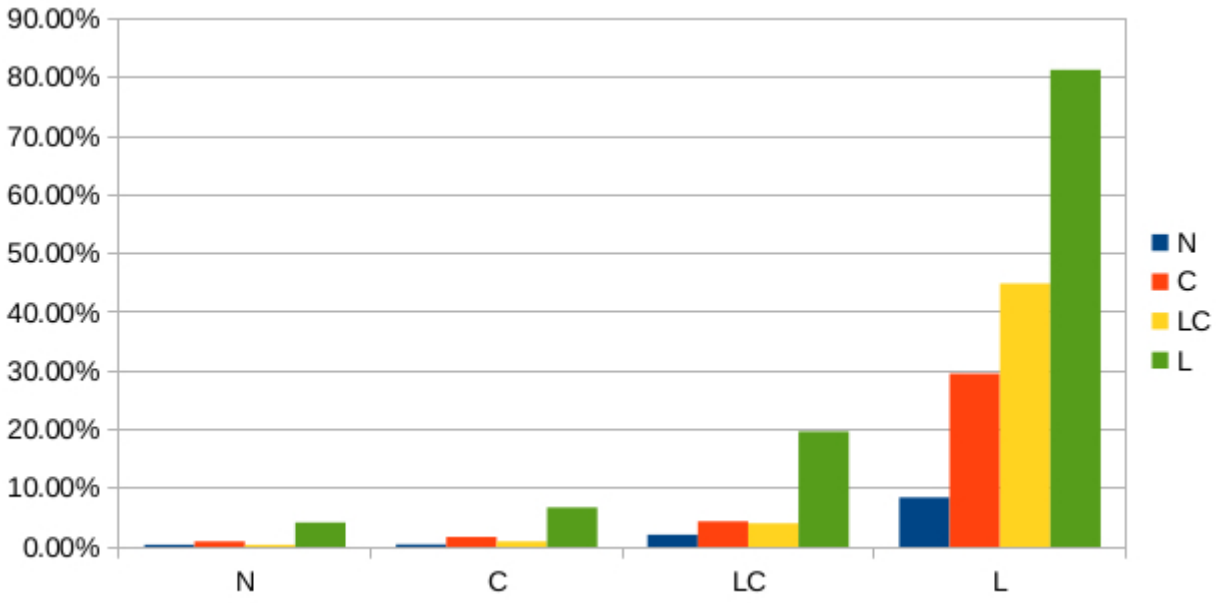
Σε κάθε άλλη περίπτωση, δεν παρουσιάζεται κάποια θεαματική αλλαγή, πράγμα που μας οδηγεί στο συμπέρασμα πως όταν οι εφαρμογές έχουν και το στρώμα της εικονοποίησης, εξακολουθούν να αντιμετωπίζουν ανταγωνισμούς, οι οποίοι αυξάνουν αρκετά τον χρόνο εκτέλεσης. Οι σχετικά μικρές διαφορές που παρατηρούνται οφείλονται κυρίως στο στρώμα εικονοποίησης που βρίσκεται μεταξύ των εφαρμογών και του host. Όπως θα φανεί και στο σχήμα 4.6 (σε σύγκριση με το σχήμα 4.1) οι εφαρμογές έχουν ελαφρώς διαφορετική συμπεριφορά όταν τρέχουν μέσα από εικονικό περιβάλλον. Κάτι τέτοιο είναι λογικό και αναμενόμενο, καθώς τώρα εκτελείται ανα διαστήματα και κώδικας του QEMU που είτε θα αλλοιώνει τις κρυφές μνήμες, είτε θα μεταβάλει τις δοσοληψίες μέσω του διαύλου.

Εκτός αυτού, οι δύο εικονικές μηχανές που βρίσκονται στον επεξεργαστή είναι πολύ πιθανό να μοιράζονται βιβλιοθήκες, ειδικά από τη στιγμή που δύο στιγμιότυπα του QEMU εκτελούνται ταυτόχρονα. Αυτό δίνει τη δυνατότητα στο σύστημα να εκμεταλλευτεί τυχόν ομοιότητες που παρουσιάζονται μεταξύ των εικονικών μηχανών, ώστε να βελτιώσει την απόδοσή τους.

Το σχετικό διάγραμμα φαίνεται στο σχήμα 4.7

BENCHMARK	L1 BW (MB)	L2 BW (MB)	L3 BW (MB)	L2 REUSE	L3 REUSE
2mm	16.205	15.923	0.011	1.018	2439.286
3mm	16.597	16.378	0.009	1.014	3270.632
cholesky	23.535	31.107	0.007	0.758	13550.758
doitgen	117.341	0.458	0.274	256.288	1.671
durbin	5.235	3.955	0.433	1.324	9.918
ep.A	0.957	1.018	0.001	0.940	1901.845
ft.A	24.010	4.115	2.369	5.636	1.783
gemm	15.876	15.577	0.010	1.019	2680.700
gemver	15.272	11.973	2.603	1.277	4.603
gesummv	6.026	4.774	4.551	1.261	1.050
gramschmidt	8.359	8.232	0.002	1.014	100920.642
jacLR	15.454	10.297	10.044	1.500	1.024
mvt	14.194	11.919	2.101	1.192	5.680
stream_d0	11.613	11.634	11.332	0.997	1.031
syr2k	23.280	21.553	2.357	1.081	9.156
syrk	21.981	21.988	0.032	1.000	740.712
trisolv	2.801	2.550	2.131	1.099	1.196
trmm	19.482	23.011	2.064	0.847	11.190

Σχήμα 4.6: Επιβράδυνση με -cpu QEMU64

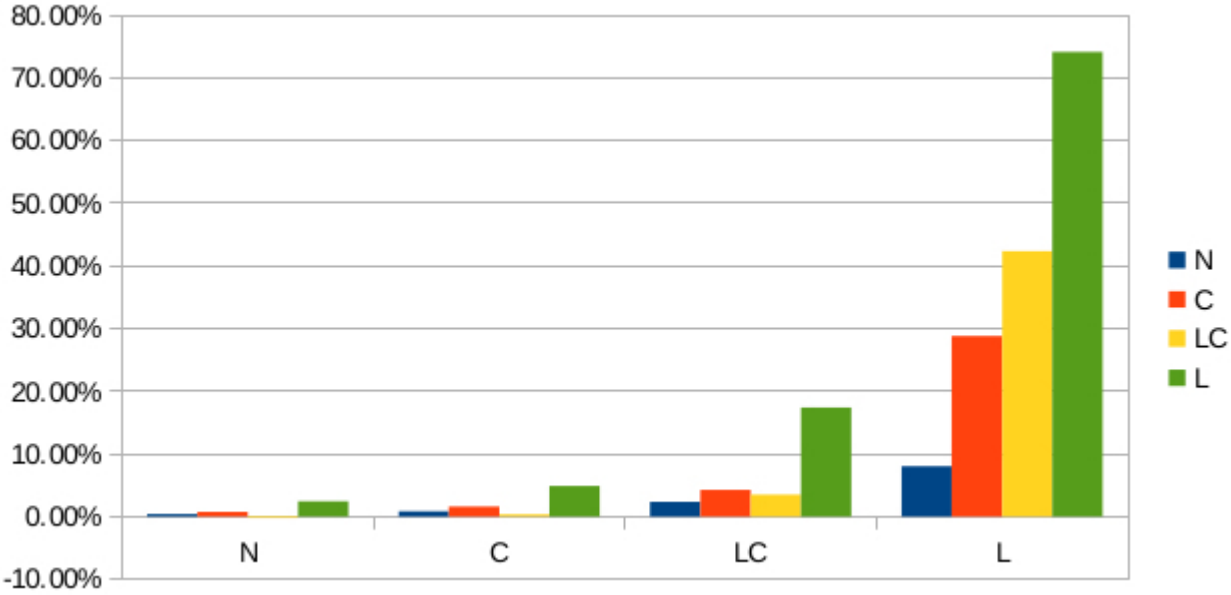


Σχήμα 4.7: Επιβράδυνση με -cpu QEMU64

4.5 Πειράματα σε εικονική μηχανή με -cpu host

Όπως αναφέρθηκε σε προηγούμενο κεφάλαιο, κρίθηκε απαραίτητο να γίνουν οι αντίστοιχες μετρήσεις με το default cpu model του QEMU (QEMU64) ούτως ώστε να μελετηθούν τυχούσες διαφορές μεταξύ των συστημάτων. Τώρα ο guest βλέπει τα cpu features του host, οπότε και τα αποτελέσματα είναι πιο κοντά σε BM περιβάλλον. Ξανά βλέπουμε κάποιες ελάχιστες αλλαγές, οι οποίες όμως δεν διαφέρουν σε μεγάλο βαθμό από αυτές με τα default cpu

features του qemu. Το μόνο αξιοσημείωτο αποτέλεσμα είναι όταν δύο εφαρμογές της κλάσης L εκτελούνται μαζί, οπότε και η επιβράδυνση έπεσε από 81.5% σε 74%. Τα αποτελέσματα είναι εμφανή στο διάγραμμα του σχήματος 4.8



Σχήμα 4.8: Επιβράδυνση με -cpu host

Κεφάλαιο 5

Συγκρίσεις χρονοδρομολογητών

Σε αυτό το σημείο χρειάζεται να επιβεβαιωθεί πως μία πολιτική, η οποία λαμβάνει υπ' όψιν τους ανταγωνισμούς μπορεί να φέρει καλύτερα αποτελέσματα από μία που τους αγνοεί. Επίσης, με τα πειράματα που θα βοηθήσουν στο παραπάνω, μπορεί να γίνει και μία σύγκριση μεταξύ κάποιων βασικών πολιτικών που είναι είτε αρκετά απλές είτε πιο πολύπλοκες.

Για την πραγματοποίηση των μετρήσεων δημιουργήθηκαν διάφορα σενάρια όπου χρειάζεται να εκτελεστούν εφαρμογές από τις τέσσερις κλάσεις με τη χρήση της επιλεγμένης πολιτικής. Οι πολιτικές που χρησιμοποιούνται έχουν αναλυθεί στα κεφάλαια 2 και 3. Παρακάτω παρουσιάζονται συνοπτικά τα workload που επιλέχθηκαν και που έγινε προσπάθεια να είναι είτε αρκετά αντιπροσωπευτικά είτε να καταδεικνύουν τα προβλήματα των ανταγωνισμών.

Κάθε σενάριο χρησιμοποιεί workload που αποτελούνται από 8 εφαρμογές που κάθε μία χρησιμοποιεί τέσσερις πυρήνες. Τα σενάρια θα μας βοηθήσουν να δούμε σε πραγματικές συνθήκες πως συμπεριφέρεται ένα σύστημα και πως οι διάφορες πολιτικές θα επηρεάσουν την απόδοση. Κάποια από αυτά έχουν σκοπό να φανερώσουν τη συμπεριφορά κάθε πολιτικής σε γενικές καταστάσεις, όπου οι εφαρμογές προέρχονται από τουλάχιστον τρεις διαφορετικές κλάσεις, ενώ τα υπόλοιπα είναι σχεδιασμένα έτσι ώστε να φανερώσουν κάποιες ειδικές καταστάσεις, όπου οι επιλογή κάποιας πολιτικής μπορεί να έχει καταστροφικά αποτελέσματα έναντι κάποιας άλλης που ενδεχομένως να πλησιάζει αρκετά στη βέλτιστη λύση.

1. **c-c-c-c-l-l-n-n**: Στο πρώτο σενάριο έχουμε εφαρμογές από τις τρεις κλάσεις L, C και N. Από τα διαγράμματα μπορούμε να προβλέψουμε πότε θα παρουσιαστούν οι ανταγωνισμοί, οπότε μένει να δούμε ποια πολιτική θα αντιμετωπίσει το πρόβλημα με τον καλύτερο τρόπο.

2. **l-l-l-l-c-c-c-c**: Εδώ έχουμε την περίπτωση που οι μισές εφαρμογές είναι κλάσης L και οι υπόλοιπες είναι κλάσης C. Όπως φάνηκε από τα διαγράμματα του προηγούμενου κεφαλαίου, οι ανταγωνισμοί κορυφώνονται όταν οι L εφαρμογές τρέχουν παράλληλα, αλλά σημαντικές επιβραδύνσεις παρουσιάζονται στις C εφαρμογές, όταν βρεθούν δίπλα σε L. Σκοπός είναι να βρεθεί ποια πολιτική μειώνει στο ελάχιστο το μέσο όρο εκτέλεσης, καθώς και ποια είναι πιο δίκαιη απέναντι στις ευαίσθητες εφαρμογές της κλάσης C.

3. **l-l-l-l-n-n-n-n**: Σε αυτό το workload τέσσερις εφαρμογής κλάσης L εκτελούνται μαζί με τέσσερις της κλάσης N. Αυτό έχει σκοπό να δείξει πως όταν ακολουθηθεί η σωστή πολιτική το αποτέλεσμα θα είναι κοντά στο βέλτιστο, αφού κάθε εφαρμογή N θα τοποθετηθεί δίπλα σε μία L, οπότε οι ανταγωνισμοί θα ελαττωθούν στο ελάχιστο. Αντίθετα, με λάθος πολιτική είναι πολύ εύκολο οι εφαρμογές N να μπουν μαζί στον επεξεργαστή με αποτέλεσμα να χαθεί η ευκαιρία να εξαφανιστούν οι ανταγωνισμοί που θα προκύψουν από στις L εφαρμογές.

4. **l-l-lc-lc-lc-lc-c-c**

5. **l-lc-lc-lc-c-c-c-n**

6. **lc-c-c-c-c-c-c-n**

7. **lc-lc-c-c-c-c-n-n**

8. **lc-lc-lc-lc-c-c-c-c**: Όπως στο σενάριο 2, έχουμε 4 εφαρμογές που χρησιμοποιούν πολύ την cache (C) μαζί με 4 εφαρμογές που τείνουν να ακυρώνουν τη cache (LC). Σε αυτή την περίπτωση, οι εφαρμογές LC χρησιμοποιούν ως ένα σημείο την cache, οπότε θέλουμε να δούμε τις συμπεριφορές και τους ανταγωνισμούς που θα σημειωθούν σε αυτό το σημείο.

9. **lc-lc-lc-lc-n-n-n-n**: Αντίστοιχα με το σενάριο 3, θέλουμε να δούμε πως ανταποκρίνονται οι διάφορες πολιτικές όταν έχουμε εφαρμογές που ανταγωνίζονται ισχυρά - τόσο για την κοινή κρυφή μνήμη, όσο και για τον δίαυλο, αλλά παρουσιάζεται η δυνατότητα να μηδενιστεί ο ανταγωνισμός, αν τοποθετηθούν δίπλα στις N που δεν χρησιμοποιούν ούτε τη cache ούτε τον δίαυλο.

Κάθε ένα από τα παραπάνω σενάρια που παρουσιάζει κάποιο ενδιαφέρον, αναλύθηκε συντόμως ώστε να είμαστε σε θέση να σχολιάσουμε αργότερα τα αποτελέσματα των πειραμάτων. Τα υπόλοιπα (σενάρια 4, 5, 6 και 7) όπως αναφέρθηκε παραπάνω, είναι γενικού ενδιαφέροντος και είναι σχεδιασμένα ώστε να αποκαλύψουν τη συμπεριφορά των πολιτικών σε μία γενική περίπτωση.

5.1 Τα αποτελέσματα

Σε αυτήν την παράγραφο θα καταγραφούν και θα σχολιαστούν σύντομα τα αποτελέσματα που προέκυψαν από την εκτέλεση των 9 workload με κάθε έναν από τους χρονοδρομολογητές. Σε αυτό το σημείο και συγκεκριμένα για τον CFS αξίζει να σημειωθεί πως τα αποτελέσματα από εκτέλεση σε εκτέλεση διαφέρουν πράγμα που καθιστά τη παρακολούθησή τους δυσκολότερη. Αυτό συμβαίνει επειδή η πολιτική που ακολουθεί δεν λαμβάνει υπ' όψιν τους ανταγωνισμούς που προκύπτουν από τις ταυτόχρονες εκτελέσεις οπότε η συμπεριφορά του είναι απρόβλεπτη και ακανόνιστη ως προς αυτόν τον τομέα.

1. **c-c-c-c-l-l-n-n**: Για τους χρονοδρομολογητές lca και link bal στο συγκεκριμένο σενάριο η κάθε εφαρμογή L θα τοποθετηθεί στο ίδιο gang με μία N ώστε να μειωθεί στο ελάχιστο ο ανταγωνισμός στον δίαυλο. Στη συνέχεια οι εφαρμογές C τοποθετούνται ανα δύο στο ίδιο gang. Αναμενόμενα λοιπόν θα φέρουν παρόμοια αποτελέσματα, όπως επιβεβαιώνεται και στα διαγράμματα.

2. **l-l-l-l-c-c-c-c**: Αντίθετα με την προηγούμενη περίπτωση, εδώ η συμπεριφορά του lca και του link bal θα είναι ακριβώς αντίθετη. Από τη μία η πολιτική του link bal είναι να εξισορροπεί τη χρήση του διαύλου μεταξύ των gang, οπότε σε κάθε gang θα τοποθετείται ένα VM κλάσης L και ένα κλάσης C. Αντίθετα η πολιτική του lca είναι να αποφεύγει όσο γίνεται συνδρομολογήσεις εικονικών μηχανών L με C, οπότε θα τοποθετήσει στο ίδιο gang τις μηχανές L μεταξύ τους, κάνοντας το ίδιο και στις μηχανές C. Από το διάγραμμα φαίνεται πως ο lca έχει ελαφρώς καλύτερα αποτελέσματα από τον link bal. Δεν παρουσιάζεται κάποια εντυπωσιακή μεταβολή, καθώς όλοι οι συνδυασμοί που προκύπτουν και στις δύο περιπτώσεις είναι αρκετά επιβλαβείς για τις εφαρμογές.

   Από την άλλη, οι cfs και throughput balance δεν τοποθετούν από πριν σε gang τις εικονικές μηχανές αλλά προσαρμόζουν τους συνδυασμούς ανάλογα με την πρόοδο των εκτελέσεων.

3. **l-l-l-l-n-n-n-n**: Σε αυτό το σενάριο οι δύο gang schedulers θα τοποθετήσουν τις εικονικές μηχανές σε ζεύγη L και N. Με αυτόν τον τρόπο ελαχιστοποιούν τους ανταγωνισμούς. Ο throughput balance από την άλλη θα δώσει περισσότερο χρόνο στις εφαρμογές l, τοποθετώντας μάλιστα ταυτόχρονα στον επεξεργαστή. Με τον τρόπο αυτό θα παρέχει δικαιοσύνη όσον αφορά την πρόοδο των διεργασιών, αλλά το συνολικό throughput του συστήματος θα μειωθεί.

   Από την άλλη σε αυτή την εκτέλεση φαίνεται η τυχαιοκρατική πολιτική που ακολουθεί ο CFS, καθώς σε θεωρητικά παρόμοιο περιβάλλον εκτέλεσης που οι υπόλοιποι χρονοδρομολογητές έχουν σχετικά σταθερή συμπεριφορά αυτός παρουσιάζει μεγαλύτερες αποκλίσεις.

4. **l-l-lc-lc-lc-lc-c-c**: Γενικού ενδιαφέροντος

5. **l-lc-lc-lc-c-c-c-n**: Γενικού ενδιαφέροντος

6. **lc-c-c-c-c-c-c-n**: Γενικού ενδιαφέροντος

7. **lc-lc-c-c-c-c-n-n**: Γενικού ενδιαφέροντος

8. **lc-lc-lc-lc-c-c-c-c**: Σε αυτό το σενάριο, τόσο ο link bal όσο και ο lca, θα τοποθετήσουν σε κάθε gang από μία εικονική μηχανή LC και μία C. Από τα αποτελέσματα φαίνεται πως οι δύο πολιτικές φέρουν παρόμοια αποτελέσματα με τις μικρές διαφορές να βρίσκονται στα όρια του στατιστικού λάθους. Ξανά ο throughput balance δείχνει να βελτιώνει αισθητά τις συνθήκες στις οποίες εκτελούνται οι εικονικές μηχανές με το να δίνει περισσότερο χρόνο σε αυτές που αδικούνται λόγω ανταγωνισμών.

9. **lc-lc-lc-lc-n-n-n-n**: Ακριβώς όπως στο προηγούμενο σενάριο οι link bal και lca βάζουν στο ίδιο gang από μία μηχανή LC και μία N. Οι μηχανές κλάσης N παρέχουν την δυνατότητα να εξαλειφθούν οι ανταγωνισμοί με τα αποτελέσματα μεταξύ των χρονοδρομολογητών να είναι πολύ κοντά στην ιδανική περίπτωση. Ο throughput balance δεν θα μπορούσε να κάνει κάτι καλύτερο από αυτό και όντως δείχνει να αυξάνει τη μέση επιβράδυνση, ωστόσο παρέχει δικαιοσύνη στην πρόοδο που κάνουν οι διεργασίες. Έτσι βλέπουμε στα διαγράμματα τους χρονοδρομολογητές να έχουν παρόμοια αποτελέσματα. Άλλωστε το σενάριο είναι παρόμοιο με το l-l-l-l-n-n-n-n

5.2 Γενικές παρατηρήσεις και προτάσεις

Φαίνεται στα αποτελέσματα πως οι ανταγωνισμοί για τους κοινούς πόρους όντως έχουν επίδραση στην απόδοση του συστήματος. Σε όλες τις περιπτώσεις ο throughput balance αντιμετωπίζει σε σημαντικό βαθμό αυτή την επίδραση ενώ και ο lca δείχνει ικανός να βελτιώσει τα αποτελέσματα. Σε κάποιες περιπτώσεις δεν έχει καλύτερη συμπεριφορά από τον cfs, κάτι το οποίο μπορεί να εξηγηθεί με ποικίλους τρόπους.

Για την δημιουργία και συντήρηση μίας εικονικής μηχανής εκτελείται κώδικας QEMU. Αυτή η εκτέλεση μεταβάλει τη συμπεριφορά των εφαρμογών εντός της μηχανής με το να ”ομαλοποιεί” τόσο τις προσβάσεις στη llc όσο και στον δίαυλο. Όταν μία εφαρμογή εκτελείται σε bare-metal περιβάλλον μπορεί να αξιοποιεί σε μεγάλο βαθμό την llc. Το επίπεδο εικονοποίησης που προστίθεται ωστόσο είναι ικανό να περιορίσει τα οφέλη που προσφέρει η κρυφή μνήμη αφού το ίδιο το QEMU θα ακυρώνει κάποια από τα δεδομένα της εφαρμογής. Αντίστοιχα η χρήση του διαύλου από μία εφαρμογή -που υπό κανονικές συνθήκες θα είναι ιδιαίτερα αυξημένη- μπορεί να περιορίζεται. Έτσι οι εφαρμογές C και L θα τείνουν να πλησιάσουν τη συμπεριφορά των LC. Τέλος είναι λογικό πως το εικονικό επίπεδο θα επηρεάσει

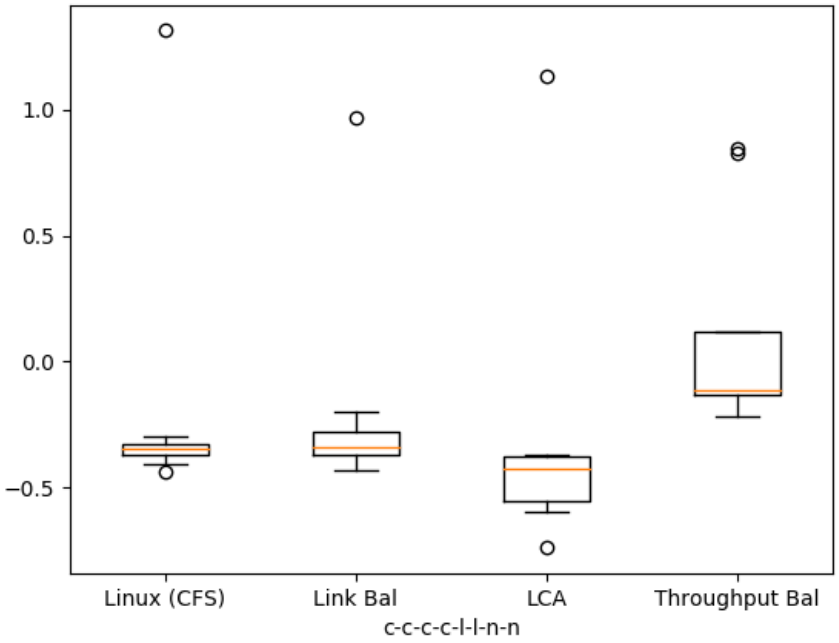
και τις εφαρμογές κλάσης N, αφού ο κώδικας του QEMU δεν θα περιορίζεται κατ' ανάγκη στα ιδιωτικά μέρη του επεξεργαστή.

Όλα τα παραπάνω περιορίζουν την ικανότητα του lca να επιλύει τους ανταγωνισμούς με την ίδια ευχέρεια που θα το έκανε σε bare-metal περιβάλλον. Ταυτόχρονα ο lca δεν είναι σχεδιασμένος να αντιμετωπίζει διάφορα stalls στον επεξεργαστή, οπότε όταν η μία εικονική μηχανή περιμένει δεδομένα δεν θα αντικατασταθεί άμεσα από μία έτοιμη προς εκτέλεση, με αποτέλεσμα να δημιουργούνται καθυστερήσεις τις οποίες ο cfs αντιμετωπίζει αποτελεσματικά.

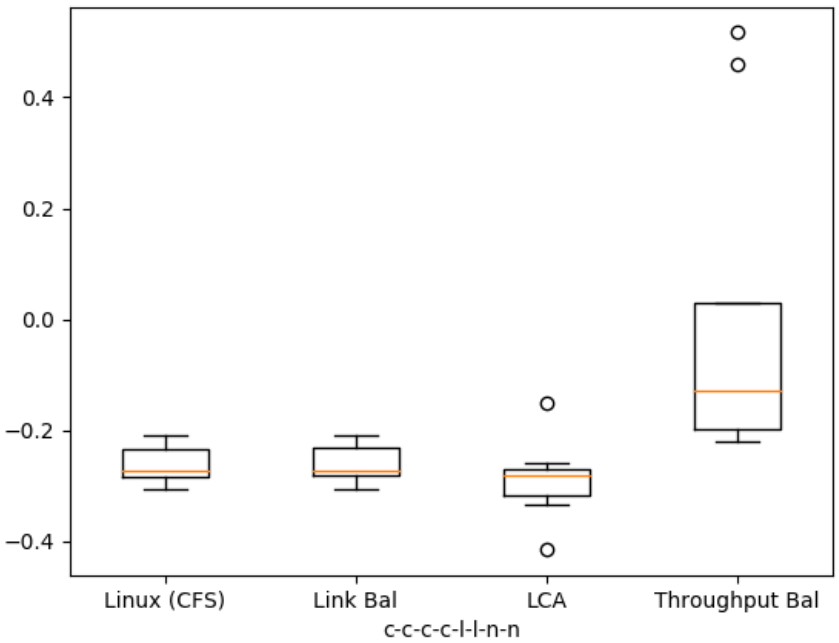
Φυσικά φαίνεται και στα διαγράμματα που ακολουθούν πως σε αρκετές περιπτώσεις η απόδοσή του είναι καλύτερη από αυτή του cfs, γεγονός που τον κατατάσσει στις αξιόπιστες απαντήσεις στο πρόβλημα του ανταγωνισμού.

Εκτός των παραπάνω, από τις μετρήσεις εξάγονται κάποια άλλα ενδιαφέροντα συμπεράσματα. Σε κάποιες περιπτώσεις παρατηρείται ακόμα και αρνητική επιβράδυνση. Κάτι τέτοιο συμβαίνει λόγω χρησιμοποίησης κοινών βιβλιοθηκών από τις εικονικές μηχανές. Τέτοια συμπεριφορά έχει παρατηρηθεί και σε bare metal περιβάλλον [SZhu10], ακόμα περισσότερο όμως εδώ τρέχει ταυτόχρονα πολλές φορές το QEMU που είναι λογικό οι διεργασίες του να χρησιμοποιούν κοινές δυναμικές βιβλιοθήκες.

Ένα πρώτο συμπέρασμα που βγαίνει λοιπόν είναι πως οι χρονοδρομολογητές που λαμβάνουν υπ' όψιν τους ανταγωνισμούς όντως είναι σε θέση να αντιμετωπίσουν κάποια από τα προβλήματα και να βελτιώσουν την επίδοση ενός συστήματος. Σαν συνολικό αποτέλεσμα μπορεί να μην είναι ιδανικό οπότε η απάντηση βρίσκεται μάλλον στη μέση. Ο ιδανική πολιτική θα πρέπει να ακολουθεί τις βασικές αρχές ενός χρονοδρομολογητή που λαμβάνει υπ' όψιν τους ανταγωνισμούς ενώ ταυτόχρονα θα αντιμετωπίζει με πιο εκλεπτυσμένους μηχανισμούς τα προβλήματα που ενδεχομένως να προκύπτουν.

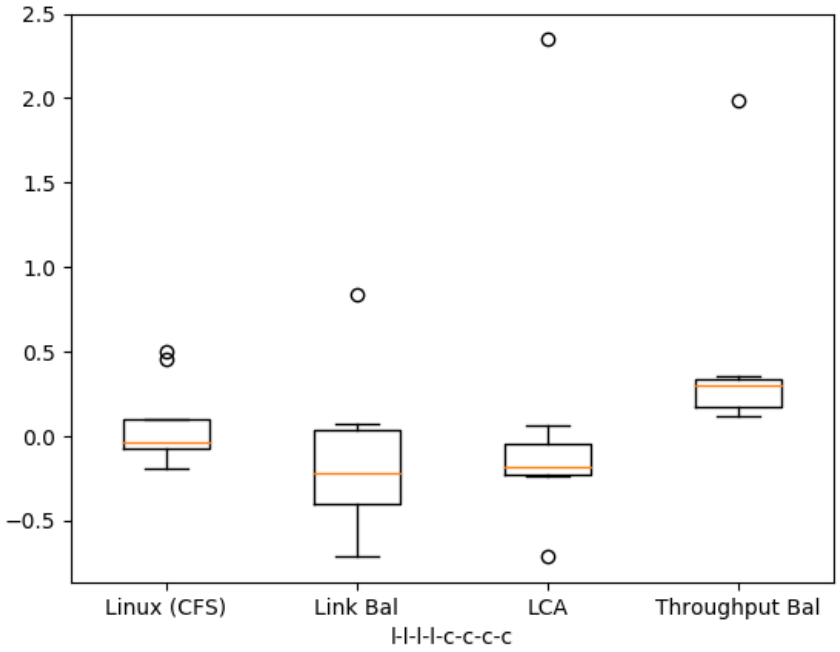


(a) -cpu host

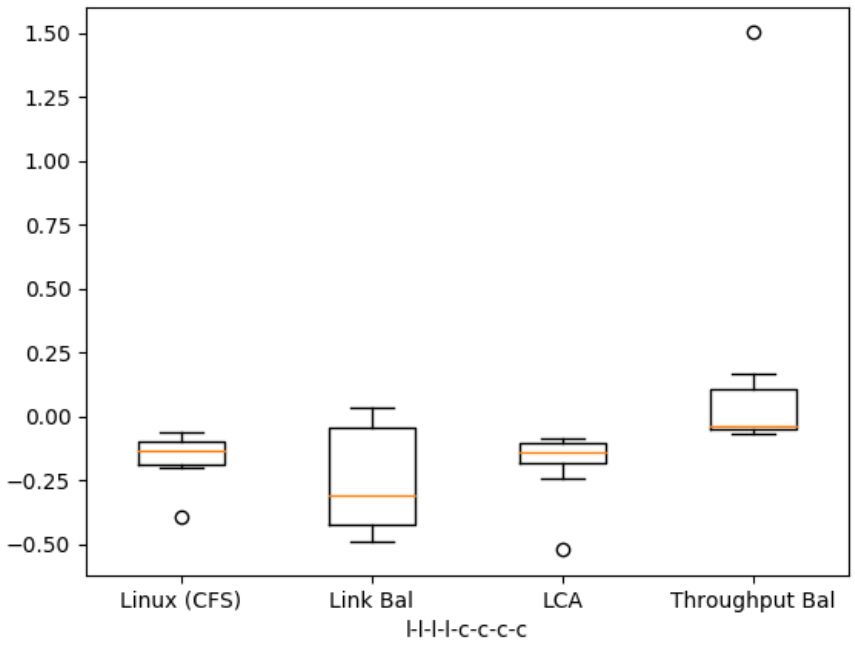


(b) -cpu QEMU64

Σχήμα 5.1: Αποτελέσματα για c-c-c-c-l-l-n-n

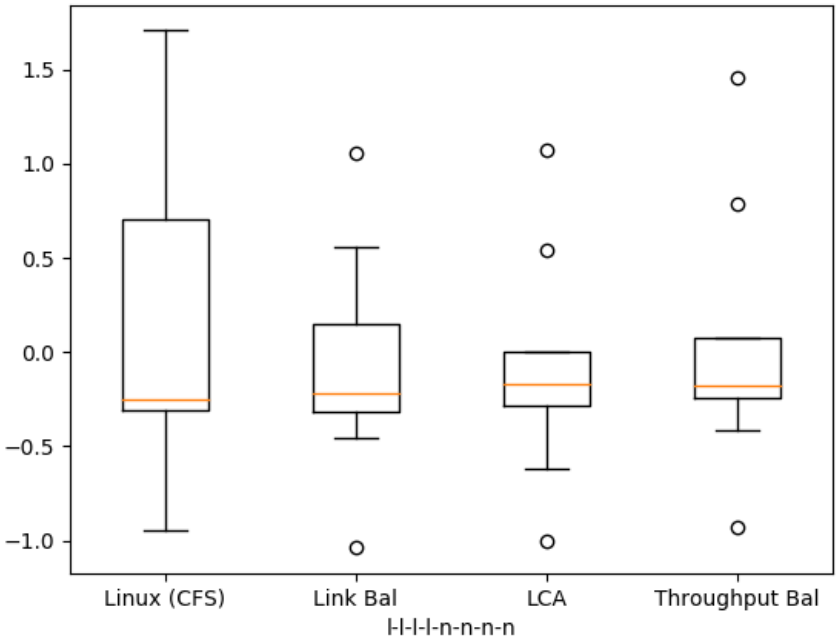


(a) -cpu host

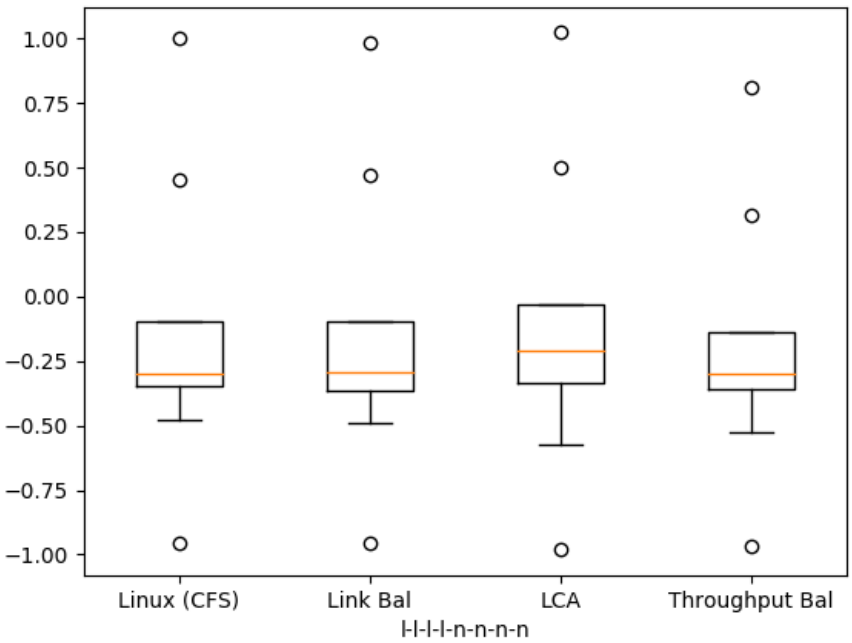


(b) -cpu QEMU64

Σχήμα 5.2: Αποτελέσματα για l-l-l-l-c-c-c-c

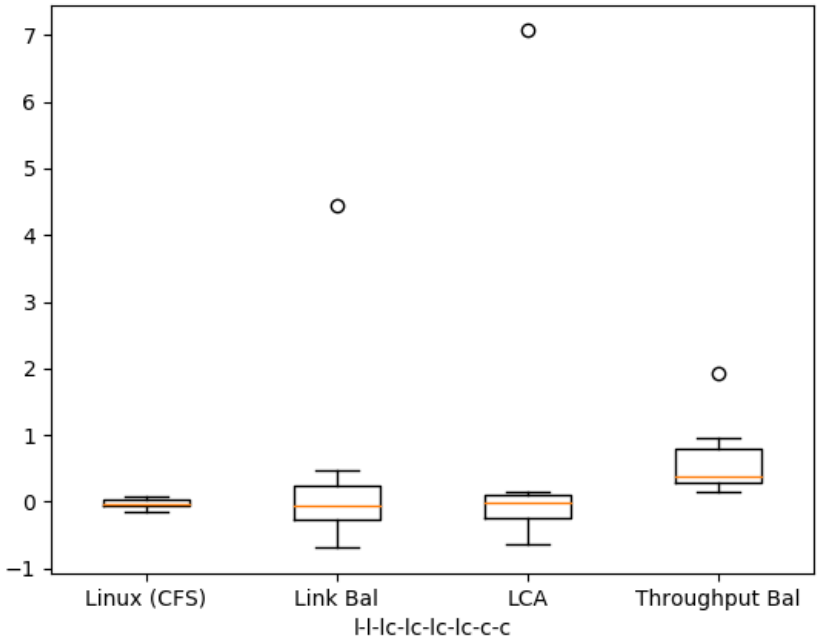


(a) -cpu host

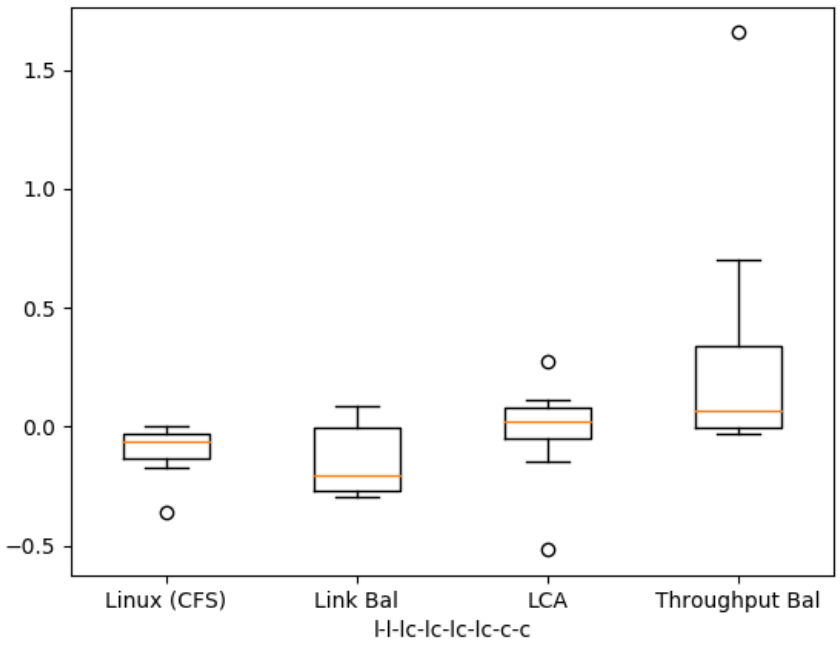


(b) -cpu QEMU64

Σχήμα 5.3: Αποτελέσματα για l-l-l-l-n-n-n-n

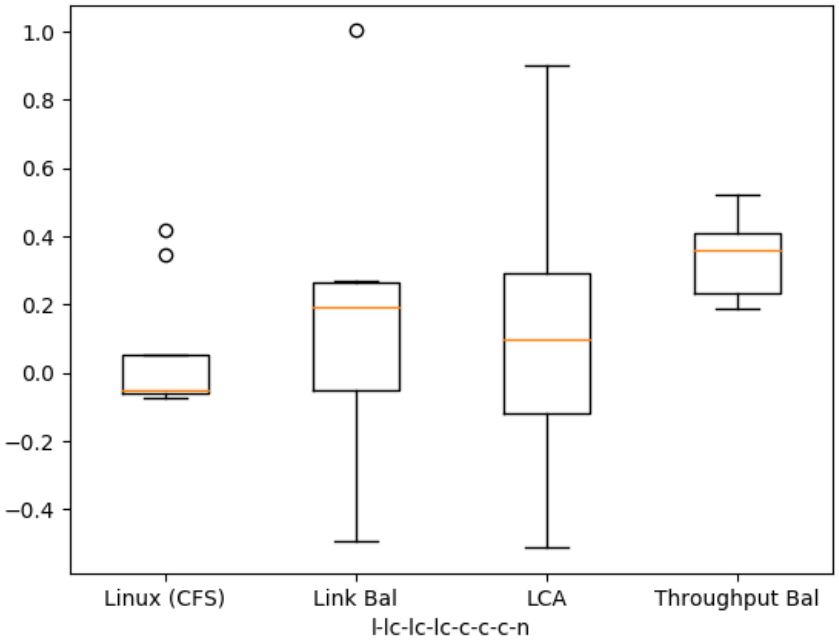


(a) -cpu host

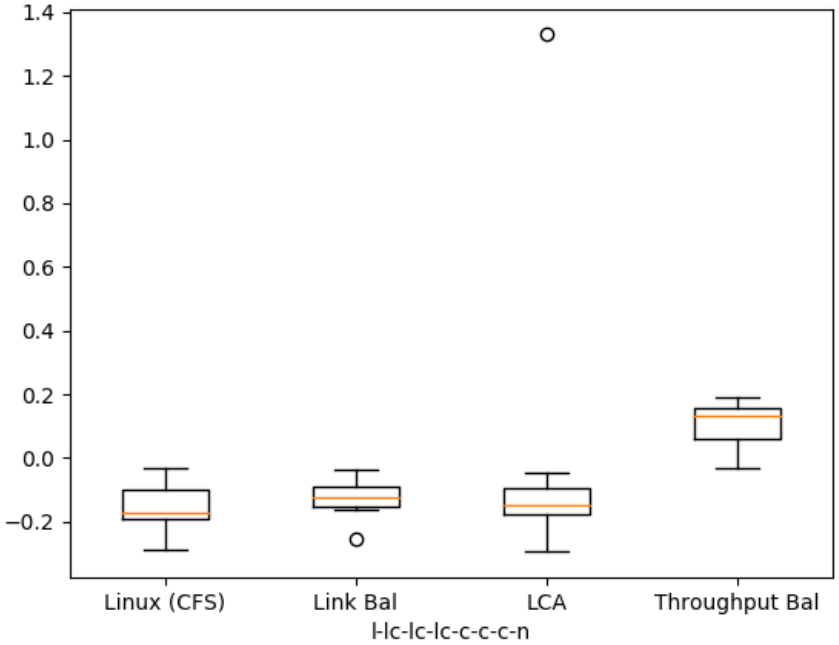


(b) -cpu QEMU64

Σχήμα 5.4: Αποτελέσματα για l-l-lc-lc-lc-lc-c-c

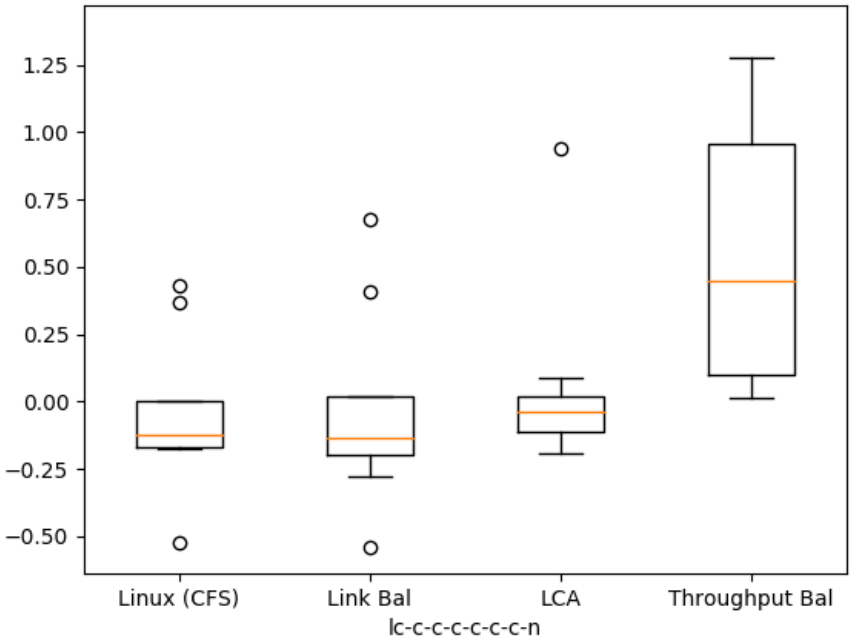


(a) -cpu host

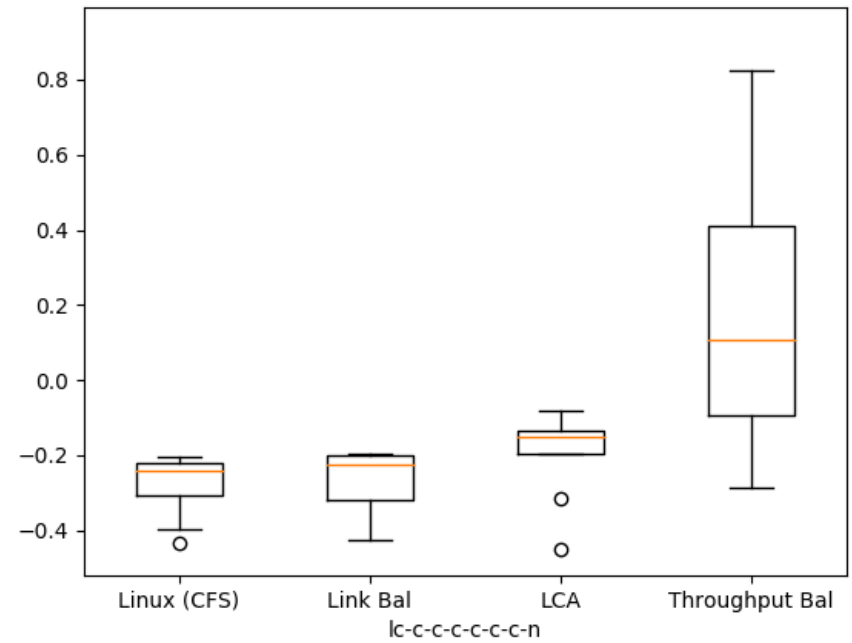


(b) -cpu QEMU64

Σχήμα 5.5: Αποτελέσματα για l-lc-lc-lc-c-c-c-n

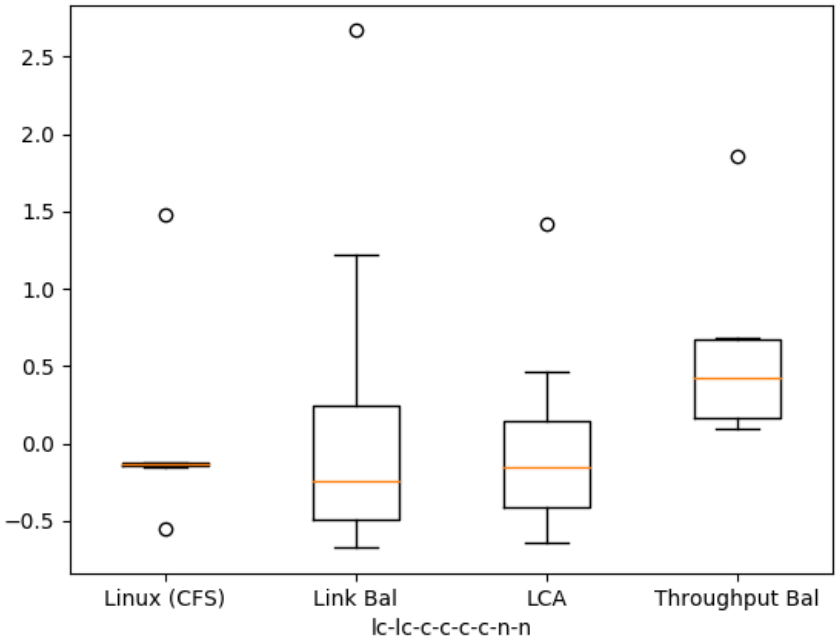


(a) -cpu host

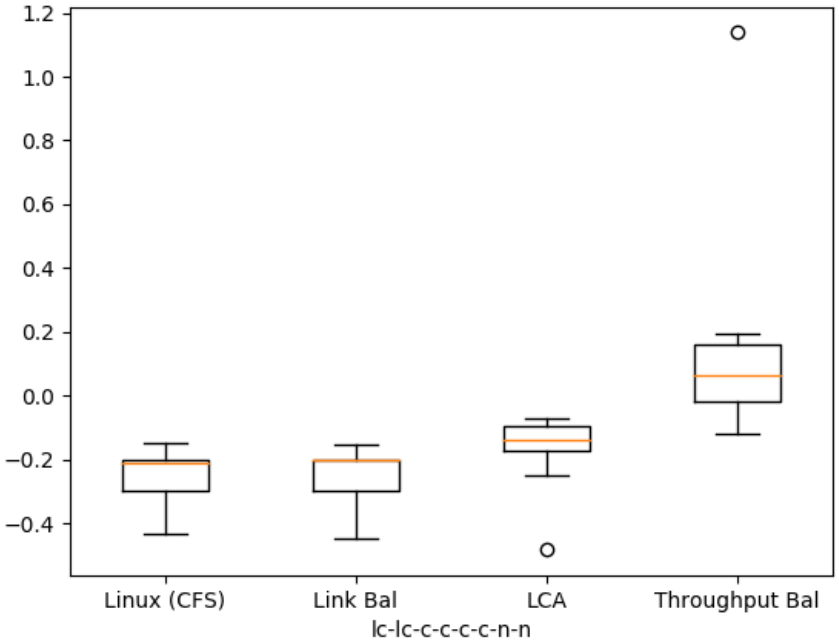


(b) -cpu QEMU64

Σχήμα 5.6: Αποτελέσματα για lc-c-c-c-c-c-c-n

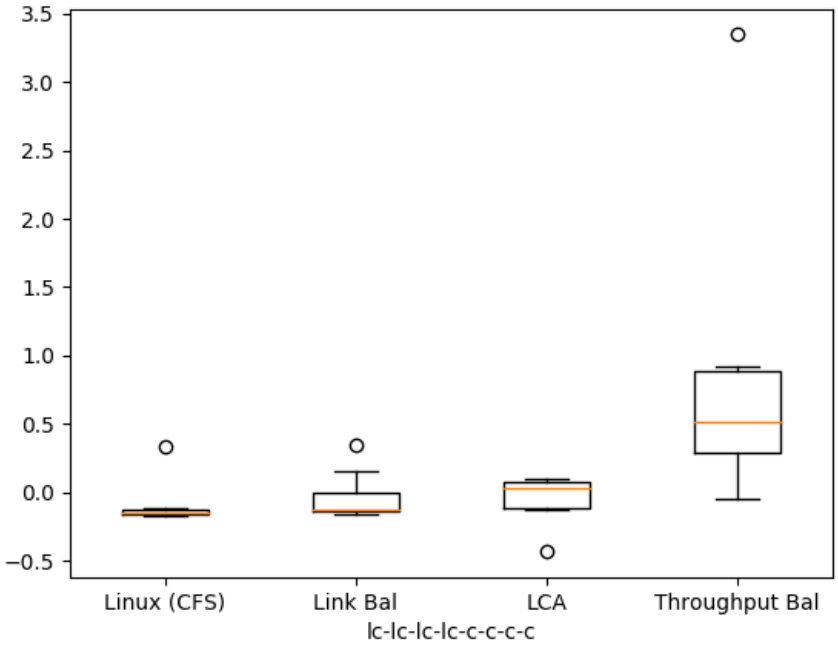


(a) -cpu host

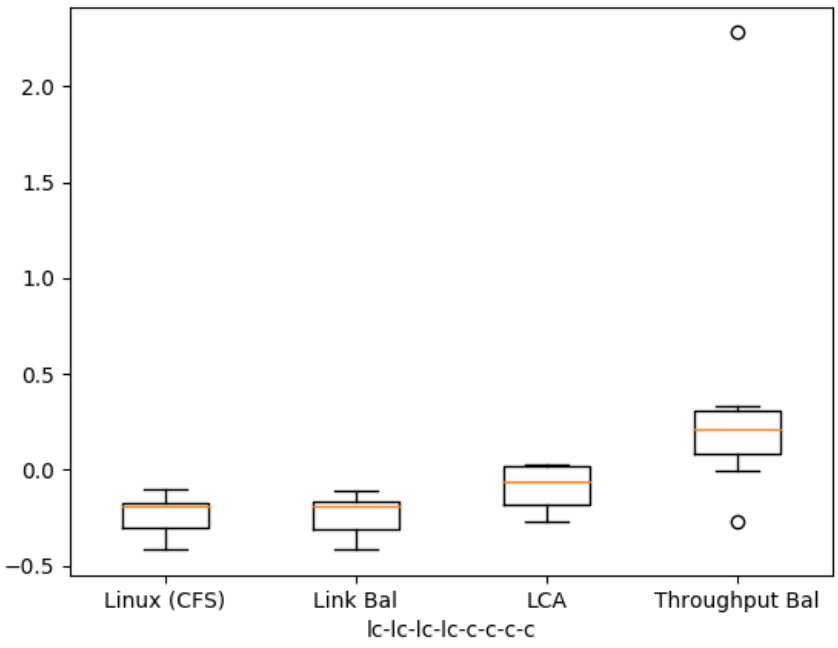


(b) -cpu QEMU64

Σχήμα 5.7: Αποτελέσματα για lc-lc-c-c-c-c-n-n

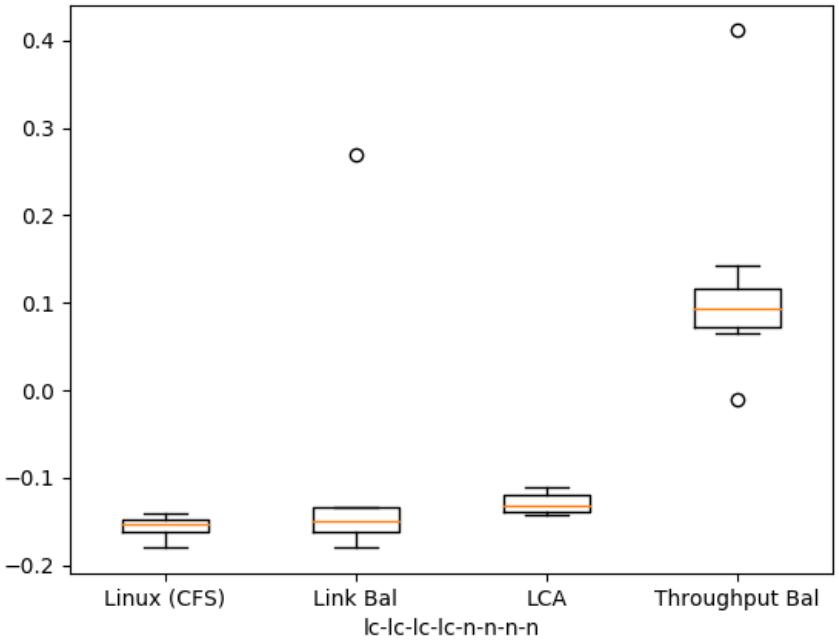


(a) -cpu host

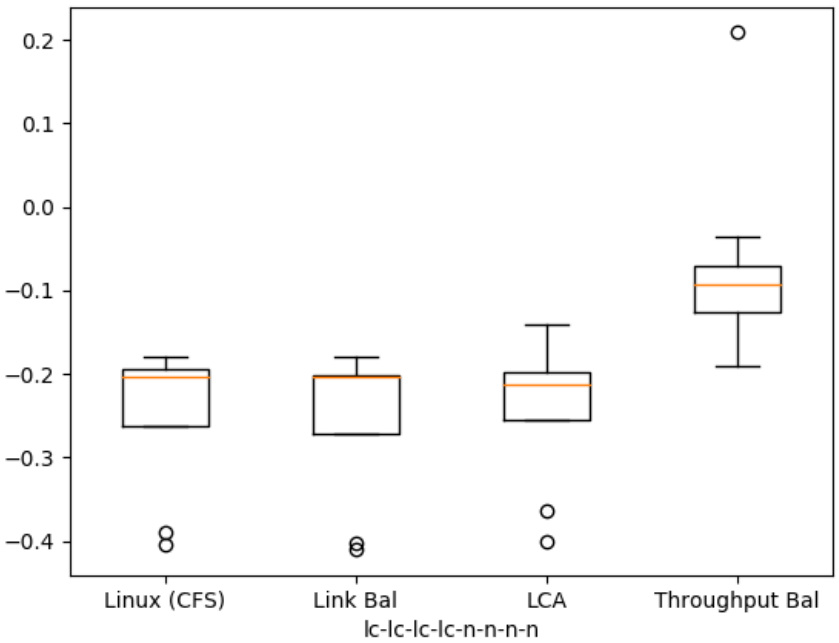


(b) -cpu QEMU64

Σχήμα 5.8: Αποτελέσματα για lc-lc-lc-lc-c-c-c-c



(a) -cpu host



(b) -cpu QEMU64

Σχήμα 5.9: Αποτελέσματα για lc-lc-lc-lc-n-n-n-n

Κεφάλαιο 6

Συμπεράσματα και προτάσεις

Στο τελευταίο κεφάλαιο θα καταγραφούν τα συμπεράσματα που βγήκαν από την εκπόνηση αυτής της διπλωματικής εργασίας δίνοντας μεγαλύτερη έμφαση στις διαφορές που παρατηρήθηκαν μεταξύ των αποτελεσμάτων σε host και σε guest μηχανήματα. Ωστόσο είναι σημαντικό να σχολιαστούν και τα αποτελέσματα των διαφορετικών πολιτικών που χρησιμοποιήθηκαν.

Τέλος θα αναφερθούν συνοπτικά πιθανές επεκτάσεις που μπορούν να εφαρμοστούν στη χρονοδρομολόγηση εικονικών μηχανών και περαιτέρω έρευνα που μπορεί να εκπονηθεί.

6.1 Συμπεράσματα

Μέσα από αυτήν την εργασία και τη μελέτη των αποτελεσμάτων, είναι εμφανές το γεγονός πως ο ανταγωνισμός των εφαρμογών που εκτελούνται σε ένα σύστημα είναι υπαρκτό πρόβλημα που χρήζει ιδιαίτερης προσοχής. Αυτό επεκτείνεται περισσότερο όταν περνάμε σε συστήματα ευρείας κλίμακας όπου η αύξηση της απόδοσης με κατοχυρωμένη τη ποιότητα υπηρεσιών ενός βαθμού είναι κρίσιμης σημασίας. Έτσι αναλύθηκε η συμπεριφορά διαφόρων πρακτικών που υπόσχονται να περιορίσουν το πρόβλημα και μέσω των αποτελεσμάτων προτάθηκαν λύσεις ανάλογα με τις ανάγκες του εκάστοτε χρήστη.

Έχει φανεί σε προηγούμενες εργασίες πως οι ανταγωνισμοί έχουν αρνητικές συνέπειες στην απόδοση ενός συστήματος. Όταν περνάμε σε εικονικό περιβάλλον παρατηρούμε πως το overhead που προσθέτει η εικονική μηχανή στις εφαρμογές που εκτελούνται εξισορροπεί λίγο την κατάσταση. Κάτι τέτοιο σημαίνει πως θα είναι δυσκολότερο να παρουσιαστούν ακραίες καταστάσεις.

Μελετήθηκαν τέσσερις χρονοδρομολογητές, ένας από τους οποίους είναι αυτός του linux (cfs). Μετά από σύγκριση των αποτελεσμάτων φανερώθηκε πως σε κάποιες περιπτώσεις δίνει εξίσου καλά αποτελέσματα με άλλους που λαμβάνουν υπ' όψιν τους ανταγωνισμούς. Ωστόσο σε αρκετές άλλες οι πολιτικές που μελετούμε έφεραν πολύ καλύτερα αποτελέσματα.

Τέλος στη παρούσα εργασία μελετήθηκαν οι επιπτώσεις που έχει σε ένα σύστημα η επιλογή του εκάστοτε μοντέλου επεξεργαστή στο πρόγραμμα QEMU. Φάνηκε πως όσον αφορά το κομμάτι του ανταγωνισμού, αυτή η επιλογή δεν θα έχει θεαματικές διαφορές στα αποτελέσματα. Παρουσιάζονται ωστόσο κάποιες επιδράσεις που φαίνεται πως προκύπτουν από τη διαφορετική αντιμετώπιση που έχει το QEMU απέναντι στα διάφορα μοντέλα. Όταν μία λειτουργία ή ένα χαρακτηριστικό πρέπει να προσομοιωθεί, παραπάνω κώδικας QEMU θα εκτελεστεί με αναμενόμενες επιδράσεις στη συμπεριφορά των εφαρμογών που τρέχουν μέσα στις εικονικές μηχανές.

6.2 Future Work

6.2.1 Χρονοδρομολόγηση με ανταγωνισμούς

Σαν επέκταση της εργασίας, περισσότερα πειράματα που προσομοιώνουν καινούρια σενάρια μπορούν να εκτελεστούν. Για παράδειγμα θα πρέπει να μελετηθεί η συμπεριφορά ενός συστήματος όταν περισσότερες των δύο εφαρμογές εκτελούνται στο ίδιο package. Επίσης σημαντικό είναι να μελετηθούν άλλα σημεία στα οποία παρουσιάζεται ανταγωνισμός (πχ ο ελεγκτής της κύριας μνήμης - DRAM controller).

Επίσης, ιδιαίτερο ενδιαφέρον παρουσιάζει η μελέτη νέων πολιτικών που προσπαθούν να βελτιστοποιήσουν την απόδοση ως προς κάποιον (έναν ή περισσότερους) παράγοντα. Για παράδειγμα ένα σημαντικό κομμάτι που κοιτάει ένας χρήστης για το σύστημά το οποίο σχεδιάζει ή χρησιμοποιεί είναι αυτό της ενεργειακής κατανάλωσης. Άλλο κομμάτι είναι αυτό της ποιότητας υπηρεσιών. Πρέπει να σχεδιαστούν λοιπόν οι κατάλληλες πολιτικές που θα στοχεύουν άμεσα σε αυτά τα κομμάτια.

Τέλος, ενώ έχει γίνει αρκετή δουλειά προς την κατεύθυνση της αντιμετώπισης των ανταγωνισμών και έχουν προταθεί πολλές μέθοδοι κατηγοριοποίησης των εφαρμογών, για να γίνει πρακτική μία λύση πρέπει αυτή η κατηγοριοποίηση να γίνεται σε πραγματικό χρόνο και να είναι γρήγορη και αποδοτική για να μην αυξάνει το overhead του αλγορίθμου. Όταν η εκτέλεση γίνεται σε baremetal περιβάλλον είναι σχετικά εύκολη η παρακολούθηση μίας εφαρμογής και με μέτρηση των κατάλληλων performance counters να τοποθετηθεί σε μία κλάση. Από την άλλη πλευρά όμως, μία εικονική μηχανή αντιμετωπίζεται ως μαύρο κουτί και είναι πολύ δυσκολότερη η τοποθέτησή της σε κάποια κατηγορία. Αυτό δυσχεραίνει το πρόβλημα καθώς όντας ολόκληρο σύστημα πλέον που μπορεί να εκτελεί παραπάνω από μία εφαρμογές πρέπει να υπάρχει ένας αποδοτικός τρόπος αυτή να παρακολουθείται ανα τακτά χρονικά διαστήματα και ενδεχομένως να χρειάζεται να αλλάζει κλάσεις κατά τη διάρκεια της εκτέλεσής της. Κάτι τέτοιο δεν έχει μελετηθεί και είναι ένα ερώτημα που πρέπει να απαντηθεί, ώστε να υπάρξει σαν ρεαλιστική απάντηση απέναντι στους ανταγωνισμούς μία πολιτική χρονοδρομολόγησης.

6.2.2 Χρονοδρομολόγηση σε εικονικές μηχανές

Στα πλαίσια αυτής της διπλωματικής εργασίας, χρησιμοποιήθηκαν εικονικές μηχανές που είχαν πανομοιότυπο αρχιτεκτονικό σχεδιασμό. Είχαν δηλαδή τον ίδιο αριθμό πυρήνων οι οποίοι μάλιστα είχαν και τα ίδια χαρακτηριστικά. Για να γίνει ωστόσο σαφής η συμπεριφορά της εικονοποίησης όσον αφορά τον ανταγωνισμό των εικονικών μηχανών, η έρευνα πρέπει να επεκταθεί ακόμη περισσότερο. Αυτό σημαίνει έλεγχος της συμπεριφοράς με μηχανές που χρησιμοποιούν διαφορετικό αριθμό επεξεργαστών, διαφορετική ιεραρχία μνήμης και γενικότερα διαφορετική αρχιτεκτονική.

Έπειτα, όπως αναφέρθηκε και στην εισαγωγή σημαντικός όρος της εικονοποίησης είναι η **μετανάστευση**. Στη παρούσα εργασία έγινε μελέτη των ανταγωνισμών και των επιπτώσεών τους σε εικονικές μηχανές που εκτελούνται στον ίδιο επεξεργαστή. Είναι σημαντικό να μελετηθεί η επίδραση που θα έχει σε ένα σύστημα μία ενδεχόμενη μεταφορά της εικονικής μηχανής σε άλλον επεξεργαστή με τέτοιον τρόπο ώστε σε κάθε επεξεργαστή να τοποθετούνται συνδυασμοί μηχανών που παρουσιάζουν τους ελάχιστους ανταγωνισμούς.

Τελικά, τα αποτελέσματα πρέπει να δεθούν μεταξύ τους ώστε να δημιουργηθεί η συνολική εικόνα μία καλύτερης πολιτικής χρονοδρομολόγησης εικονικών μηχανών από τις ήδη υπάρχουσες. Άλλωστε όπως έχει αναφερθεί αρκετές φορές, είναι ουτοπικό να μιλάμε για τη βέλτιστη πολιτική, αφού ανάλογα με τις ανάγκες του εκάστοτε χρήστη διαφορετική προσέγγιση μπορεί να είναι η βέλτιστη.

Βιβλιογραφία

[AHHa14] N. Anastopoulos K. Nikas K. Kourtis A-H. Haritatos, G. Goumas and N. Koziris, "LCA: a memory link and cache-aware co-scheduling approach for CMPs", in *Proceedings of the 23rd international conference on Parallel architectures and compilation*, pp. 469–470, 2014.

[AJeo12] H. Jaeung C. Young-Ri A. Jeongseob, K. Changdae and H. Jaehyuk, "Dynamic Virtual Machine Scheduling in Clouds for Architectural Shared Resources", in *Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Ccomputing*, pp. 19–19, USENIX Association, 2012.

[JLin08] X. Ding Z. Zhang J. Lin, Q. Lu and P. Sadayappan, "Gaining Insights into Multicore Cache Partitioning: Bridging the Gap between Simulation and Real Systems", in *Proceedings of the 14th International Symposium on High Performance Computer Architecture*, pp. 367–478, 2008.

[KPra13] M. Karamta K. Prajapati, P. Raval and M.B. Potdar, "Comparison of Virtual Machine Scheduling Algorithms in Cloud Computing", *International Journal of Computer Applications*, vol. 83, pp. 12–14, 2013.

[MAlA15] M. Daraghmeh M. Al-Ayyoub, Y. Jararweh and Q. Althebyan, "Multi-agent Based Dynamic Resource Provisioning and Monitoring for Cloud Computing Systems Infrastructure", *Cluster Computing*, vol. 18, no. 2, pp. 919–932, 2015.

[SKim04] D. Chandra S. Kim and Y. Solihin, "Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture", in *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*, pp. 111–122, 2004.

[SZhu10] S. Blagodurov S. Zhuravlev and A. Fedorova, "Addressing shared resource contention in multicore processors via scheduling", in *Proceedings of the fifteenth edition of ASPLOS on Architectural support for programming languages and operating systems*, pp. 129–142, 2010.

[Xie08] Yuejian Xie and Gabriel H Loh, "Dynamic classification of program memory behaviors in CMPs", *2nd Workshop on Chip Multiprocessor Memory Systems and Interconnects*, 01 2008.

[YJia08] J. Chen Y. Jiang, X. Shen and R. Tripathi, "Analysis and approximation of optimal co-scheduling on chip multiprocessors", in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, pp. 220–229, 2008.

[Yu06] J. Yu and R. Buyya, "Scheduling Scientific Workflow Applications with Deadline and Budget Constraints Using Genetic Algorithms", *Scientific Programming*, vol. 14, pp. 217–230, 2006.