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

Υπολογιστικό Νέφος και NUMA Αρχιτεκτονικές: Η Σχέση της Μνήμης με την Απόδοση των Εφαρμογών

Διπλωματικη Εργασια

του

Γεωργίου Ανδρέα Δημητρακόπουλου

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

Αθήνα, Νοέμβριος 2020

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

Υπολογιστικό Νέφος και NUMA Αρχιτεκτονικές: Η Σχέση της Μνήμης με την Απόδοση των Εφαρμογών

Διπλωματικη Εργασια

του

Γεωργίου Ανδρέα Δημητρακόπουλου

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

Εγκρίθηκε από την κάτωθι τριμελή επιτροπή την 20η Νοεμβρίου 2020.

Γεώργιος Γκούμας	Νεκτάριος Κοζύρης	Διονύσιος Πνευματικάτος
Επίκουρος Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Γεώργιος Ανδρέα Δημητρακόπουλος
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Το **υπολογιστικό νέφος** δεν είναι κάτι άλλο παρά πληθώρα διασυνδεδεμένων ηλεκτρονικών υπολογιστών (τους οποίους αποκαλούμε servers), δικτυακά συνδεδεμένους και με πρόσβαση στο διαδίκτυο, στους οποίους εκτελούνται οι διάφορες εφαρμογές (λόγου χάρη κοινωνικά δίκτυα, streaming εφαρμογές, μηχανές αναζήτησης κλπ), και οι οποίοι εξυπηρετούν την μεγάλη μάζα χρηστών που επιχειρούν να τις προσπελάσουν.

Προκειμένου να καταφέρει η κάθε εφαρμογή να εκτελεστεί απρόσκοπτα αλλά και απομονωμένα απ΄ τις υπόλοιπες, χρησιμοποιείται η τεχνική της εικονικοποίησης, και συγκεκριμένα η **εικονική μηχανή**. Πρόκειται για ένα «εικονικό» λειτουργικό σύστημα, το οποίο ζει στο φυσικό μηχάνημα όπως κάθε άλλη διεργασία, και χρησιμοποιεί τους πόρους του, δηλαδή τους επεξεργαστές, μνήμη, δίσκο, δίκτυο κλπ. Κατ΄ αυτόν τον τρόπο, σε κάθε φυσικό μηχάνημα, φιλοξενούνται πολλές εικονικές μηχανές, οι οποίες εξυπηρετούν τις εκάστοτε εφαρμογές, και οι οποίες καταλαμβάνουν και διαμοιράζονται τους πόρους του μηχανήματος

Για να ανταπεξέλθουν οι servers στο φόρτο που προστίθεται απ΄ την εκτέλεση των εικονικών μηχανών, απαραίτητη είναι η χρήση πολυπύρηνων επεξεργαστών. Τα σύγχρονα πολυπύρηνα chips που χρησιμοποιούνται σε υπολογιστικά νέφη, αποτελούνται κυρίως από αρχιτεκτονικές ανομοιόμορφης πρόσβασης μνήμης (**NUMA**). Στις αρχιτεκτονικές αυτές, οι επεξεργαστές και η μνήμη είναι χωρισμένα σε τμήματα, τα οποία αποκαλούνται NUMA nodes, και επικοινωνούν μεταξύ τους χρησιμοποιώντας ειδικούς συνδέσμους.

Σκοπός της παρούσας διπλωματικής εργασίας, είναι η μελέτη της απόδοσης των εφαρμογών που εκτελούνται κάτω απ΄ αυτές τις συνθήκες διαμοιρασμού πόρων, και κυρίως την συμπεριφορά τους όταν η απόδοση της **μνήμης** που χρησιμοποιούν εξαρτάται από την τοπολογία του συστήματος, όπως ακριβώς συμβαίνει με τα NUMA συστήματα.

Για το λόγο αυτό, επιλέξαμε ένα υπολογιστικό σύστημα που ανήκει στο εργαστήριο CSLab του Ε.Μ.Π., στο οποίο δημιουργήσαμε εικονικές μηχανές διαφορετικής δομής και τοπολογίας. Ύστερα επιλέχθηκαν κατάλληλες εφαρμογές προσομοίωσης **benchmarks**, και εκτελέστηκαν στις εικονικές μηχανές, ώστε να προσομοιώσουμε μία cloud υποδομή. Με αυτόν τον τρόπο συλλέξαμε χρήσιμες πληροφορίες που αφορούν τις μεταβολές των επιδόσεων μεταξύ των εικονικών μηχανών, οι οποίες παρουσιάζονται μέσω γραφημάτων και σχολιασμού.

Τέλος παραθέτουμε τα συμπεράσματα που μπορούν να προκύψουν μέσα απ΄ την παραπάνω διαδικασία, καθώς και ιδέες ή προτάσεις βελτίωσης της απόδοσης των εφαρμογών μελλοντικά.

Λέξεις-κλειδιά: Υπολογιστικό Νέφος, NUMA, Εικονική Μηχανή, Μνήμη, Benchmark

Abstract

Cloud computing is nothing more than a large number of computers (called servers), connected by network and having access to the internet, in whom most of the well known applications run (e.g. social networks, streaming applications, search engines etc). These computers serve the large amount of users trying to access these applications.

To make sure each application is able to run independently and isolated from the others, we use virtualization techniques, and more specifically Virtual Machines. A **Virtual Machine** (or a VM) is a seperate OS living in the host machine, just like any other process, able to use and share all the resources granted to it, like CPUs, memory, disk or network connectivity. As a result, a server could potentially have a couple of VMs running on top of it.

In order for servers to handle the load which virtual machines put on them, multicore CPUs, among other components, are required. There are a couple of chip architecture techniques used to create those kind of chips, but the most common and mostly used one is **NUMA** (Non uniform memory access). In this architecture, CPUs and memory are split in a number of NUMA nodes, and communicate with each other via communication links.

The purpose of this diploma thesis is to study and understand the affection resource sharing between Virtual Machines has on the applications' performance. More specifically, we try to understand and optimize applications execution based on differnet **memory** NUMA placements.

To accomplish that, we chose a computer system, belonging to CSlab located in NTUA, and used it to create Virtual Machines of different configuration and NUMA topology. We used these VMs to run **benchmarks** and simulate cloud infrastructure, to collect and analyze information about their execution times and performances. We present all of these via graphs and extended explanations.

Finally, we present all of the conclusions that can be extracted from the above procedure, as well as ideas or propositions for future performance optimization.

Keywords: Cloud Computing, Virtual Machine, NUMA, Memory, Benchmark

Ευχαριστίες

Η εκπόνηση της Διπλωματικής μου εργασίας σηματοδοτεί το τέλος των προπτυχιακών σπουδών μου στη Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου. Πραγματοποιήθηκε στα πλαίσια του Εργαστηρίου Υπολογιστικών Συστημάτων του τομέα Τεχνολογίας Πληροφορικής και Υπολογιστών, με επιβλέποντα επίκουρο καθηγητή τον κ. Γεώργιο Γκούμα, τον οποίο και θα ήθελα να ευχαριστήσω πρωτίστως για την ευκαιρία που μου έδωσε να ασχοληθώ με το συγκεκριμένο επιστημονικό πεδίο, αλλά και για τα άτομα με τα οποία ο ίδιος με έφερε σε επικοινωνία και υπήρξαν αρωγοί στην προσπάθειά μου.

Ιδιαιτέρως θα ήθελα να ευχαριστήσω τον Δρ. Δημήτριο Σιακαβάρα, ο οποίος στάθηκε δίπλα μου από την αρχή, δίνοντας μου τις κατάλληλες συμβουλές και κατευθύνσεις, προκειμένου να έχουμε το επιθυμητό αποτέλεσμα, χάρη στην πολύτιμη βοήθεια του οποίου η συγκεκριμένη διπλωματική εργασία έγινε πραγματικότητα.

Θα ήθελα επίσης να ευχαριστήσω τον καθηγητή Νεκτάριο Κοζύρη και τον καθαγητή Διονύσιο Πνευματικάτο που συμπλήρωσαν την τριμελή επιτροπή.

Θα αποτελούσε παράλειψη να μην ευχαριστήσω όλους τους φίλους μου που με την στήριξή τους με βοηθούν όλα αυτά τα χρόνια να πετύχω τους στόχους μου.

Τέλος, ευχαριστώ βαθύτατα τους γονείς μου Ανδρέα και Δέσποινα καθώς και τον αδελφό μου Φάνη, για την αγάπη, την υπομονή και την στήριξη που μου έχουν προσφέρει όλα αυτά τα χρόνια.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Κεφάλαιο 1

Εισαγωγή

1.1 Υπολογιστικό νέφος και εικονικές μηχανές

Η σύγχρονη εποχή χαρακτηρίζεται από πολλούς ως η Εποχή της Πληροφορίας. Η άποψη αυτή πηγάζει απ΄ το γεγονός πως ολοένα και περισσότεροι άνθρωποι σήμερα, με ποικίλους τρόπους, έχουν εντάξει στη ζωή τους και την καθημερινότητά τους την ψηφιακή πληροφορία. Ο αριθμός των ατόμων, σε παγκόσμια κλίμακα, που χρησιμοποιούν σήμερα ηλεκτρονικούς υπολογιστές ή γενικότερα ηλεκτρονικές συσκευές με πρόσβαση στο διαδίκτυο, σύμφωνα με έρευνα του International Telecommunication Union (ITU)[1] για το έτος 2019 είναι περίπου 54% ή περίπου 4,1 δισεκατομμύρια άνθρωποι. Ο αριθμός αυτός μπορεί να μοιάζει τεράστιος, αλλά δικαιολογείται αν σκεφτεί κανείς την χρηστικότητα που προσφέρει ένας ηλεκτρονικός υπολογιστής. Πληθώρα εφαρμογών, όπως μηχανές αναζήτησης, κοινωνικά δίκτυα, εφαρμογές streaming, απομακρυσμένης σύνδεσης και επικοινωνίας είναι μερικά χαρακτηριστικά παραδείγματα χρήσης ηλεκτρονικού υπολογιστή, τα οποία προσφέρονται δωρεάν στους χρήστες. Ο βασικότερος τρόπος με τον οποίο λειτουργούν όλες αυτές οι εφαρμογές και είναι προσβάσιμες στους χρήστες εύκολα και γρήγορα, είναι με τη χρήση του **υπολογιστικού νέφους (Cloud Computing)**.

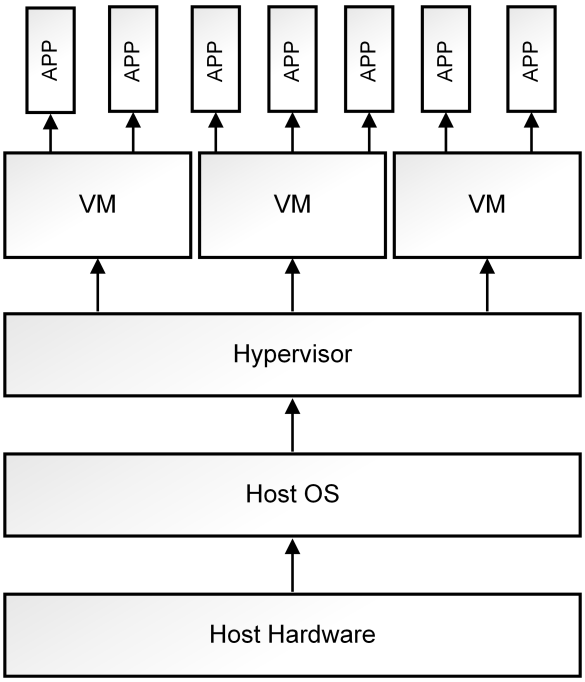
Με τον όρο υπολογιστικό νέφος ή Cloud, εννούμε τη διάθεση υπολογιστικών πόρων μέσω διαδικτύου (π.χ. servers, apps), από κεντρικά συστήματα που βρίσκονται απομακρυσμένα από τον τελικό χρήστη, τα οποία τον εξυπηρετούν αυτοματοποιώντας διαδικασίες, παρέχοντας ευκολίες και ευελιξία σύνδεσης[2]. Ολοένα και περισσότερες υπολογιστικές ανάγκες φιλοξενούνται σε δημόσια clouds, όπως το EC2 της Amazon, το Microsoft Azure και το Compute Engine της Google [1] ή σε ιδιωτικά cloud με χρήση διαχειριστικών εφαρμογών, όπως το ESXi της VMware, το OpenStack[2] και το Mesos.[3] Η χρήση του Cloud προσφέρει τόσο στους τελικούς χρήστες, όσο και στους διαχειριστές ευελιξία και καλή οικονομική διαχείριση. Οι τελικοί χρήστες μπορούν να αρχικοποιήσουν εργασίες που ποικίλλουν από μικρές, χαμηλών υπολογιστικών απαιτήσεων εφαρμογές, μέχρι σύνθετες, κάνοντας χρήση

[1] https://www.itu.int/
[2] https://www.nist.gov/

περισσότερων πόρων του cloud, κοστολογούμενοι κάθε φορά ανάλογα με τις εκάστοτε απαιτήσεις. Αντίστοιχα οι διαχειριστές του cloud, μπορούν να επιτύχουν κλιμακωτή παροχή πόρων με οικονομικό τρόπο[4], κατασκευάζοντας κέντρα δεδομένων (datacenters - DCs)[5][6] καθώς και με διαμοιρασμό των πόρων μεταξύ των τελικών χρηστών.

Η βασική αρχή λειτουργίας του Cloud είναι η χρήση **εικονικών μηχανών (VMs)**. Εικονική μηχανή είναι ένα λογισμικό ή λειτουργικό σύστημα το οποίο όχι μόνο παρουσιάζει τα χαρακτηριστικά μιας πιστής αντιγραφής ενός υπολογιστικού συστήματος, αλλά έχει τη δυνατότητα να εκτελεί λειτουργίες όπως εκτέλεση εφαρμογών και προγραμμάτων, σαν ένας ξεχωριστός υπολογιστής[7]. Μια εικονική μηχανή (ή guest) δημιουργείται και λειτουργεί μέσα σε ένα φυσικό υπολογιστή (ή host). Παραπάνω από μία εικονικές μηχανές μπορούν να ζουν στο ίδιο φυσικό μηχάνημα, συγκροτώντας έτσι ένα cloud. Οι εικονικές μηχανές διαμοιράζονται στους χρήστες, προκειμένου να υλοποιήσουν σε αυτές τις εφαρμογές που επιθυμούν.

Την διαχείριση των VMs αναλαμβάνει ο **(Υπερ)επόπτης (hypervisor)** ή **Ελεγκτής Εικονικών Μηχανών (Virtual Machine Monitor - VMM)**, ο οποίος μπορεί να βρίσκεται είτε σε επίπεδο υλικού (bare metal) είτε σε λογισμικού (hosted). Στο παρακάτω σχήμα 1.1 μπορούμε να δούμε όσα αναφέρθηκαν σε ένα top - down σχήμα.



Σχήμα 1.1: Από την εφαρμογή στον φυσικό εξυπηρετητή

Ο hypervisor χρησιμοποιεί την τεχνική της εικονικοποίησης πλατφόρμας (hardware virtualization) για την λειτουργία των VMs, δηλαδή την απόκρυψη των φυσικών χαρακτηριστικών υλοποίησης και πόρων μιας υπολογιστικής πλατφόρμας από τους πελάτες των πόρων αυτών (πχ εφαρμογές, χρήστες κλπ)[8]. Με τον τρόπο αυτό ανατίθενται από το φυσικό μηχάνημα στο κάθε VM εικονικοί πόροι (CPU, Memory, Disk κλπ) ώστε να

προσομοιώνεται η λειτουργία τους με αυτή του φυσικού μηχανήματος.

Η εικονικοποίηση πλατφόρμας χωρίζεται σε τρεις κατηγορίες:

- **Πλήρης (Full Virtualization):** Η εικονική μηχανή προσομοιώνει επαρκές τμήμα του πραγματικού υποκείμενου υλικού ώστε να επιτρέπει την εκτέλεση επάνω της ενός μη τροποποιημένου, φιλοξενούμενου λειτουργικού συστήματος, σχεδιασμένου για τον ίδιο τύπο επεξεργαστή με τον πραγματικό (π.χ. KVM, ESXi).
- **Παραεικονικοποίηση (Para-Virtualization):** Η εικονική μηχανή δεν προσομοιώνει επακριβώς το υλικό αλλά παρέχει στις εικονικές μηχανές ένα API, μία προγραμματιστική διεπαφή, ώστε να επιτρέπει την εκτέλεση επάνω της ενός τροποποιημένου, φιλοξενούμενου λειτουργικού συστήματος, σχεδιασμένου για εκτέλεση από τον συγκεκριμένο hypervisor (π.χ. Xen, Hyper-V).
- **Επίπεδο Λειτουργικού Συστήματος (Operating-system-level Virtualization):** Η εικονική μηχανή λειτουργεί στον χώρο-χρήστη του υποκείμενου υλικού, καθώς ο πυρήνας του λειτουργικού συστήματος του φυσικού μηχανήματος επιτρέπει πολλαπλούς διαφορετικούς απομονωμένους χώρους-χρήστη να συνυπάρχουν (π.χ. Docker, Solaris).

Στην περίπτωση της παρούσας εργασίας, χρησιμοποιήθηκε ο KVM hypervisor[9] σε συνδυασμό με το λογισμικό QEMU για την εικονικοποίηση και την λειτουργία των εικονικών μηχανών, στα οποία θα αναφερθούμε εκτενέστερα σε επόμενη ενότητα.

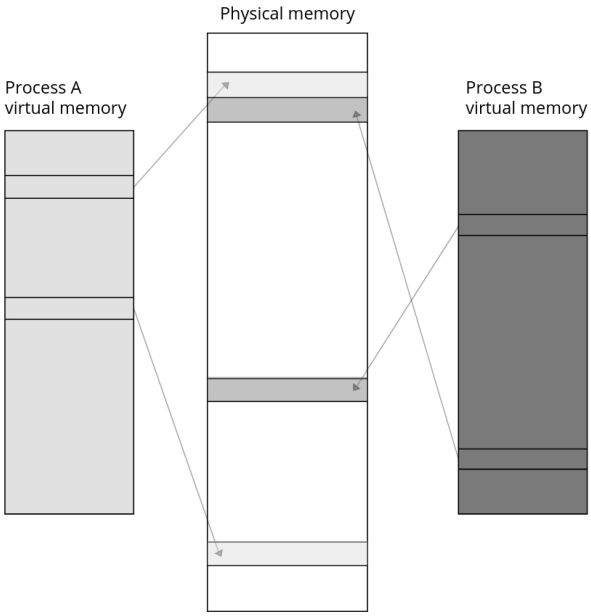
1.2 Διαχείριση Μνήμης σε πολυπύρηνα συστήματα

Στη συνέχεια, περιγράφεται συνοπτικά ο μηχανισμός εικονικής μνήμης, πώς επιτρέπεται στο λειτουργικό σύστημα να διαμοιράζει φυσική μνήμη μεταξύ των εφαρμογών καθώς και πώς το υλικό προσπαθεί να ελαχιστοποιήσει τις καθυστερήσεις (latency) στην πρόσβαση στη μνήμη, μέσω μνήμης cache και prefetchers.

1.2.1 Εικονική Μνήμη

Τα σύγχρονα πολυπύρηνα συστήματα συχνά χρειάζεται να τρέχουν παραπάνω από μία διεργασίες (processes) ταυτόχρονα. Οι διεργασίες ή εφαρμογές θέλουν να αποθηκεύσουν δεδομένα στη μνήμη, χωρίς να χρησιμοποιούν διευθύνσεις μνήμης που ανήκουν σε άλλες διεργασίες. Τη λύση στο πρόβλημα αυτό έρχεται να δώσει η εικονική μνήμη (Virtual Memory). Η εικονική μνήμη επιτρέπει στις διεργασίες να χειρίζονται τη μνήμη σαν να μην υπάρχει άλλος ανταγωνιστής. Το λειτουργικό σύστημα είναι υπεύθυνο να πραγματοποιήσει την αντιστοίχιση από εικονική σε φυσική μνήμη για κάθε διεργασία. Έτσι, όταν μια διεργασία ζητήσει μνήμη για να αποθηκεύσει δεδομένα, το λειτουργικό σύστημα θα της γυρίσει έναν συνεχόμενο εικονικό χώρο διευθύνσεων. Τα δεδομένα αυτά στην κύρια μνήμη δεν είναι

απαραίτητο να είναι συνεχόμενα, αλλά μπορούν να παρεμβάλλονται δεδομένα από διάφορες διεργασίες, όπως παρουσιάζεται και στο σχήμα 1.2



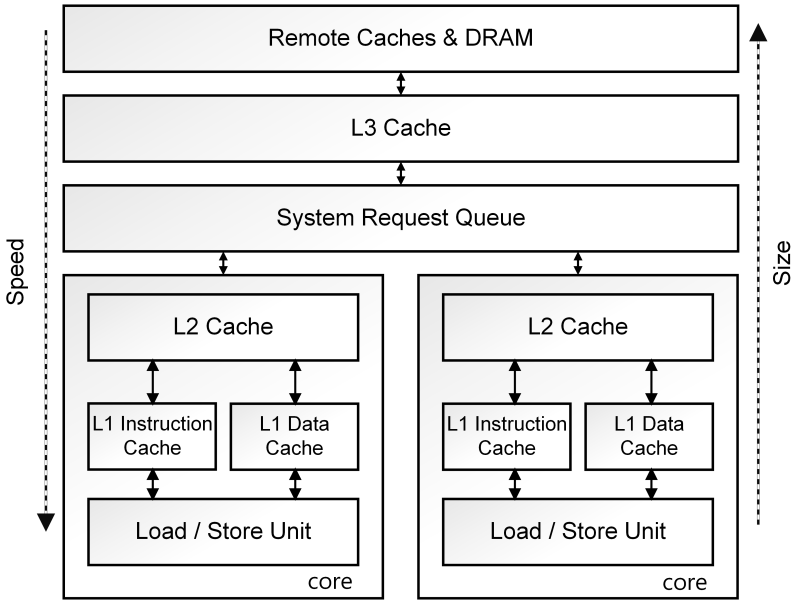
Σχήμα 1.2: Αντιστοίχιση εικονικής σε φυσική μνήμη.

Ο πιο συχνός τρόπος αναπαράστασης τμημάτων μνήμης είναι με τη χρήση της σελιδοποίησης (paging). Ένα page αναπαριστά ένα συνεχόμενο εικονικό ή φυσικό χώρο διευθύνσεων και συνήθως καταλαμβάνει 4KB. Κάθε εικονική σελίδα αντιστοιχίζεται σε οποιαδήποτε φυσική, χωρίς περιορισμό συνεχόμενης αποθήκευσης. Η μετάφραση μεταξύ εικονικών και φυσικών διευθύνσεων γίνεται με τους πίνακες σελιδοποίησης (page tables). Ο πυρήνας του λειτουργικού συστήματος είναι υπεύθυνος για την αποθήκευση του page table. Σήμερα οι περισσότεροι επεξεργαστές αναλαμβάνουν να κάνουν την μετάφραση μεταξύ εικονικών και φυσικών διευθύνσεων σε επίπεδο υλικού, με τη χρήση του Memory Management Unit (MMU). Για την επιτάχυνση της διαδικασίας της μετάφρασης χρησιμοποιείται ο TLB (Translation Lookaside Buffer).

1.2.2 Μνήμη Cache - Κρυφή μνήμη

Σε έναν επεξεργαστή, η αριθμητική και λογική μονάδα (ALU - Arithmetic and Logical Unit) εκτελεί πράξεις πάνω στους καταχωρητές (registers). Η χωρητικότητα ενός τυπικού καταχωρητή σε σύγχρονους επεξεργαστές είναι πολύ μικρή (λιγότερο από 1KB) επομένως χρειάζεται συνεχώς να αποθηκεύουν και να αφαιρούν δεδομένα. Έτσι, για να ελαχιστοποιηθεί η πρόσβαση στην κύρια μνήμη (η οποία κοστίζει περίπου από 100 έως 2000 κύκλους ρολογιού επεξεργαστή, ενώ αντίθετα η πρόσβαση σε καταχωρητή περίπου 0.25/κύκλο) χρησιμοποιήθηκε η κρυφή μνήμη (cache)[10]. Η μνήμη cache κατά κύριο λόγο έχει τρία επίπεδα, την L1, L2 και L3, όπως φαίνεται και στην εικόνα 1.3. Η cache μνήμη χωρίζεται και αξιοποιείται σε τμήματα που ονομάζονται cache lines, συνήθως 64B. Έτσι κάθε μεταφορά μεταξύ των cache ή από και

προς τη μνήμη γίνεται σε κομμάτια των 64B.



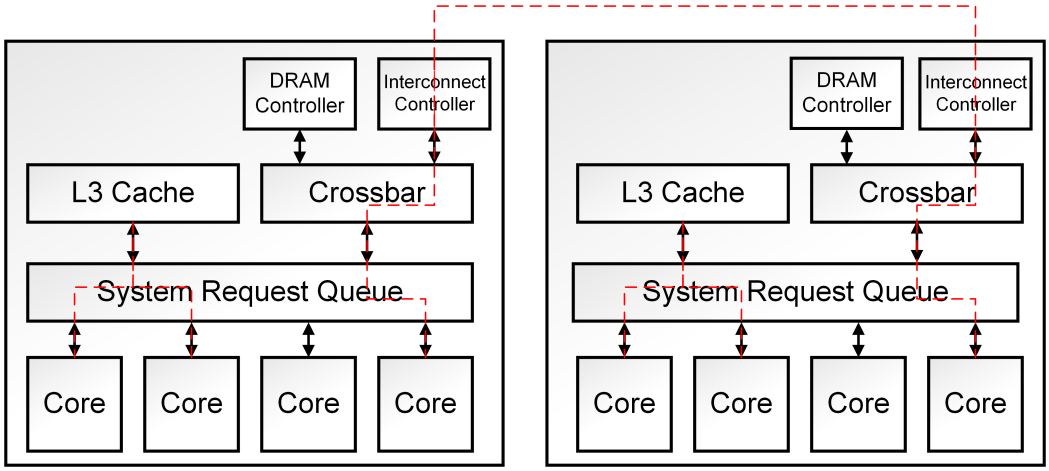
Σχήμα 1.3: Μνήμη cache - Πολλαπλά επίπεδα μνήμης cache σε έναν τυπικό επεξεργαστή. Η πρόσβαση στην cache είναι ταχύτερη απ΄ ότι στην DRAM.

Το πρώτο επίπεδο cache, η L1, είναι μια μικρή μνήμη (συνήθως από 32KB έως 64KB) η οποία είναι προσπελάσιμη ταχύτατα από τον επεξεργαστή (3 ή 4 κύκλους ρολογιού). Παρά το μέγεθός της, επειδή οι περισσότερες εφαρμογές συνήθως παρουσιάζουν χωρική και χρονική τοπικότητα, η L1 έχει συχνά μεγάλο ποσοστό hit rate. Εάν τα δεδομένα που ζητούνται δεν βρεθούν στην L1, τότε ο επεξεργαστής θα τα αναζητήσει στις L2 και L3 caches. Οι caches αυτές είναι τυπικά μεγαλύτερες απ΄ την L1 (από 512KB έως 12MB) και είναι προσβάσιμες με μεγαλύτερη καθυστέρηση (12 και 60 κύκλους ρολογιού αντίστοιχα). Αν τα ζητούμενα δεδομένα δεν βρίσκονται στις caches ο πυρήνας θα στείλει αίτημα στις caches των υπόλοιπων πυρήνων, αλλιώς θα τα αναζητήσει στην κύρια μνήμη.

1.2.3 Επικοινωνία Inter-core

Σε μία πολυπύρηνη μονάδα επεξεργασίας, πολλές φορές προκύπτει η ανάγκη εύρεσης δεδομένων από έναν υπολογιστικό πυρήνα, που βρίσκονται στην cache ενός άλλου. Υπάρχουν δύο περιπτώσεις ανάκτησης δεδομένων στο σενάριο αυτό, όπως φαίνεται και στην εικόνα . Η πρώτη περίπτωση είναι όταν τα δεδομένα που ζητούνται βρίσκονται στην κοινή L3 μνήμη cache, οπότε και μπορεί απευθείας να τα προσπελάσει, ενώ η δεύτερη είναι να στείλει αιτήματα στην cache άλλων πυρήνων. Για την δεύτερη περίπτωση χρησιμοποιούνται εσωτερικές διασυνδέσεις που αποκαλούνται interconnect links.

Μια βασική ιδιότητα που πρέπει να ισχύει καθολικά για σειριακά και παράλληλα προγράμματα που εκτελούνται σε έναν επεξεργαστή, είναι πως κάθε ανάγνωση μιας τοποθεσίας, θα πρέπει να επιστρέφει την τελευταία τιμή που γράφτηκε σε αυτή. Η ιδιότητα



Σχήμα 1.4: Inter-Core communication - Όταν δύο cores μοιράζονται μια cache (π.χ. L3) τότε επικοινωνούν μέσω της μνήμης αυτής. Αλλιώς επικοινωνούν μέσω interconnect links.

αυτή διατηρείται όταν πολλαπλά νήματα εκτελούνται σε έναν επεξεργαστή, καθώς «βλέπουν» την ίδια ιεραρχία μνήμης. Όμως στα συστήματα με παραπάνω επεξεργαστές, όπως αναφέρθηκε, ο κάθε επεξεργαστής έχει τη δική του cache. Η υλοποίηση αυτή μπορεί να οδηγήσει σε πολλαπλά αντίγραφα ενός δεδομένου σε παραπάνω από μία caches, και κάποιοι επεξεργαστές να διαβάζουν τα μη ανανεωμένα δεδομένα απ΄ τις δικές τους. Τη λύση στο πρόβλημα συνάφειας κρυφής μνήμης (Cache Coherence) δίνει το πρωτόκολλο MOESI, στο οποίο βασίζονται οι σύγχρονοι επεξεργαστές. Το πρωτόκολλο αυτό ορίζει πέντε διαθέσιμες καταστάσεις όπου μπορεί να βρίσκεται κάθε φορά μια εγγραφή της cache, ώστε να ανανεωθούν πριν την επόμενη ανάγνωση από τον επεξεργαστή ή να μείνουν ως έχουν. Συνοπτικά, οι πέντε καταστάσεις αυτές είναι οι εξής:

- **M - Modified.** Τα δεδομένα της cache είναι επικαιροποιημένα και βρίσκονται μόνο εκεί. Τα δεδομένα αυτά δεν υπάρχουν στην κύρια μνήμη, οπότε θα χρειαστεί να γραφούν εκεί πριν διαγραφούν.

- **O - Owned.** Τα δεδομένα της cache είναι επικαιροποιημένα και βρίσκονται και σε άλλες caches. Η τιμή τους στην κύρια μνήμη είναι λανθασμένη. Η κατάσταση αυτή μπορεί να υπάρχει μόνο σε ένα απ΄ τα κοινά αντίγραφα των cache lines (τα υπόλοιπα θα βρίσκονται σε κατάσταση shared) και δηλώνει ότι το δεδομένο cache line είναι υπεύθυνο να ενημερώσει την κύρια μνήμη όταν χρειαστεί.

- **E - Exclusive.** Τα δεδομένα της cache είναι επικαιροποιημένα και δεν υπάρχει αντίγραφο σε άλλη cache. Τα δεδομένα στην κύρια μνήμη είναι και αυτά σωστά, οπότε δεν απαιτείται κάποια ενέργεια κατά την διαγραφή τους.

- **S - Shared.** Τα δεδομένα της cache είναι επικαιροποιημένα και βρίσκονται και σε άλλες caches. Εάν όλες οι καταστάσεις που περιέχουν αυτά τα δεδομένα είναι shared, τότε η κύρια μνήμη είναι και αυτή ενημερωμένη. Σε διαφορετική περίπτωση, κάποιο αντίγραφο

έχει κατάσταση owned και θα αναλάβει να ενημερώσει τη μνήμη.

- **I - Invalid.** Τα δεδομένα της cache δεν είναι επικαιροποιημένα.

1.2.4 Prefetching units

Προκειμένου να μειωθεί ο αριθμός των προσβάσεων στη μνήμη από τον επεξεργαστή, και κατά συνέπεια, η καθυστέρηση που προσθέτει, ο επεξεργαστής προσπαθεί να πετύχει όσο το δυνατόν καλύτερο ποσοστό cache-hit, δηλαδή όταν χρειαστεί πρόσβαση σε δεδομένα, να τα βρει στην cache, ώστε να αποφευχθεί το latency που θα προσέθετε η πρόσβαση στη μνήμη. Μια απ΄ τις τεχνικές που ακολουθούνται για την βελτίωση του ποσοστού αυτού, είναι η προσπάθεια να προβλέψει τα δεδομένα που θα χρειαστούν στο κοντινό μέλλον και να τα τοποθετήσει στην cache πριν αυτά ζητηθούν. Τη διαδικασία αυτή αναλαμβάνουν οι prefetchers. Οι prefetchers είναι υλοποιημένοι εξ ολοκλήρου σε επίπεδο υλικού, και το λειτουργικό σύστημα έχει ελάχιστο έως καθόλου έλεγχο πάνω τους.

Οι περισσότεροι επεξεργαστές περιλαμβάνουν πολλαπλούς prefetchers, οι οποίοι χρησιμοποιούν ευρετικές μεθόδους για να προβλέπουν. Δουλεύουν καλύτερα όταν στο εκάστοτε πρόγραμμα υπάρχουν συνηθισμένα μοτίβα, όπως για παράδειγμα διάβασμα πίνακα ή αντιγραφή μνήμης, αλλά μπορούν να προβλέψουν και πιο σύνθετες περιπτώσεις, όπως οι δείκτες. Οι prefetchers σπάνια επιδεινώνουν την απόδοση, και συνήθως πετυχαίνουν σημαντικά ποσοστά βελτίωσης (για τις περισσότερες εφαρμογές συνήθως βελτίωση στον χρόνο εκτέλεσης κατά 5 έως 15%)[11]

1.3 Πολυπύρηνες αρχιτεκτονικές - Από UMA σε NUMA

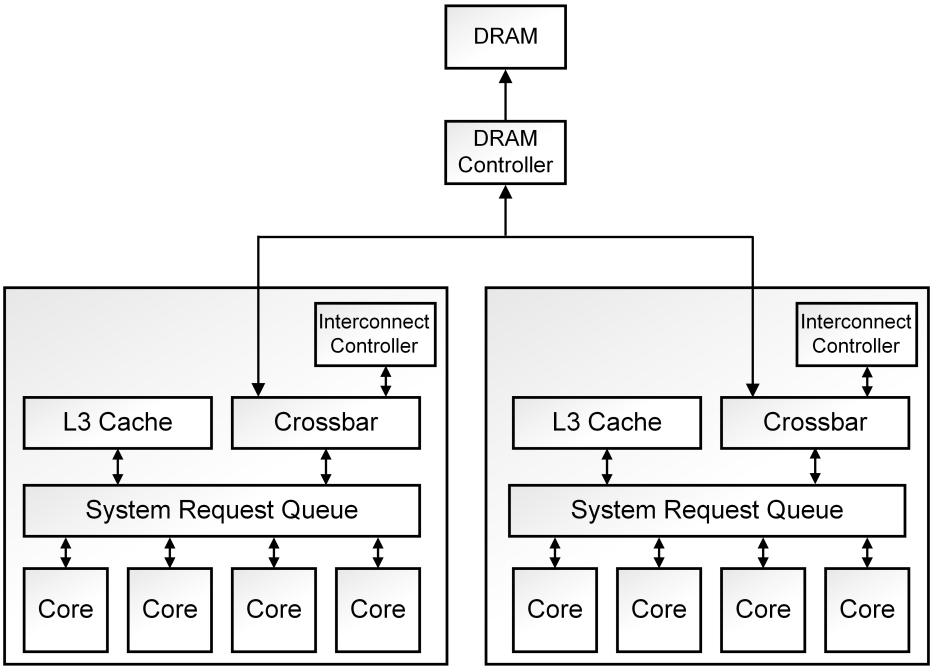
Οι σύγχρονες αρχιτεκτονικές υπολογιστών έχουν μεταβληθεί ραγδαία σε σχέση με την προηγούμενη δεκαετία, τόσο στην σχεδίαση, όσο και στις βασικές αρχές λειτουργίας. Πριν από μερικά χρόνια, οι επεξεργαστές αποτελούνταν από έναν πυρήνα (μονοπύρηνοι), ο οποίος αναλάμβανε και εκτελούσε όλο το φόρτο εργασιών (workload). Η απόδοση των μονοεπεξεργαστικών συστημάτων βελτιωνόταν με μικροαρχιτεκτονικές αναβαθμίσεις, με την επίτευξη μεγαλύτερης συχνότητας ρολογιού διατηρώντας χαμηλά την κατανάλωση ισχύος, και με την εισαγωγή πιο περίπλοκων και αποδοτικών ιεραρχιών μνήμης. Η διαδικασία της συνεχούς βελτίωσης σταμάτησε, καθώς υπήρχαν φυσικά όρια που δεν επέτρεπαν περαιτέρω αναβαθμίσεις, αφού οι δεδομένες κατασκευαστικές τεχνικές οδηγούσαν, μεταξύ άλλων, σε πολύ υψηλές θερμοκρασίες και απώλεια συγχρονισμού δεδομένων. Οι περιορισμοί αυτοί έφεραν μια νέα προσέγγιση στην ανάγκη για μεγαλύτερη υπολογιστική ισχύ, την χρήση περισσότερων από ένα πυρήνα στο ίδιο chip.

Με την χρήση δύο ή περισσότερων υπολογιστικών πυρήνων το workload μοιράστηκε μεταξύ τους, αυξάνοντας την απόδοση και καλύπτοντας περισσότερες υπολογιστικές ανάγκες. Έτσι, για την ορθή λειτουργία και τον συντονισμό μεταξύ των υλικών πόρων (CPU, memory κλπ) αναπτύχθηκαν κατά καιρούς διάφορες πολυπύρηνες αρχιτεκτονικές. Στη συνέχεια

παρουσιάζεται η Uniform Memory Access ή UMA, μια αρχιτεκτονική βασισμένη στους SMP (Symmetric Multiprocessors), καθώς και η αρχιτεκτονική που τη διαδέχτηκε, η Non Uniform Memory Access ή NUMA.

1.3.1 Uniform Memory Access - UMA

Ένας επεξεργαστής μπορεί να διασυνδέεται με τη μνήμη DRAM ποικιλοτρόπως. Η πιο απλή σχεδιαστική αρχιτεκτονική αποτελείται από έναν memory controller στον οποίο όλοι οι υπολογιστικοί πυρήνες στέλνουν τα αιτήματα πρόσβασης στη μνήμη. Στη σχεδιαστική αυτή, όλοι οι επεξεργαστές χρησιμοποιούν όλη τη μνήμη έχοντας το ίδιο latency. Για το λόγο αυτό η σχεδιαστική αυτή ονομάζεται αρχιτεκτονική ομοιόμορφης πρόσβασης μνήμης ή UMA (uniform memory access). Στο σχήμα 1.5 βλέπουμε ένα τυπικό παράδειγμα της αρχιτεκτονικής UMA για ένα σύστημα με 2 τετραπύρηνους επεξεργαστές που μοιράζονται ένα κοινό bus για την πρόσβαση στη μνήμη.



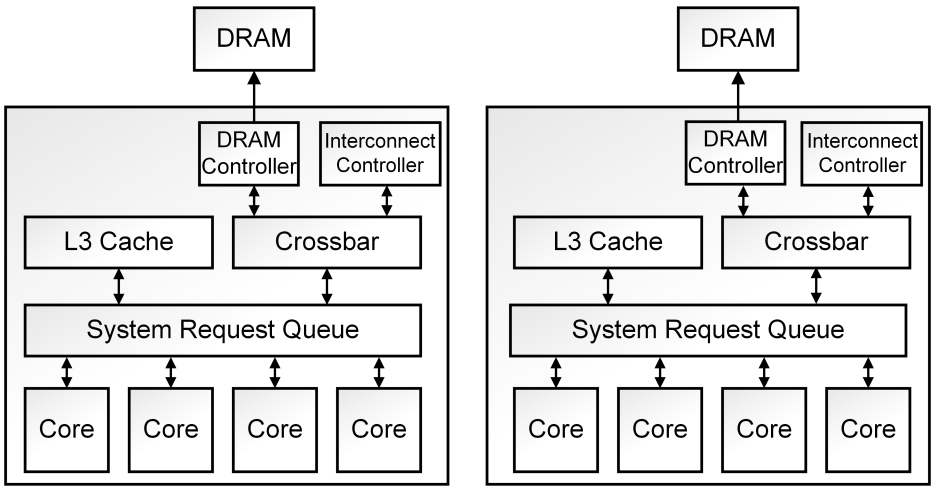
Σχήμα 1.5: Uniform Memory Access - UMA: Όλοι οι επεξεργαστές μοιράζονται ένα bus για την πρόσβαση στη μνήμη.

Σε UMA μηχανήματα, το εύρος πληροφοριών, ή bandwidth, που μεταφέρονται από και προς τη μνήμη, περιορίζεται από το bandwidth που μπορεί να υποστηρίξει ο εκάστοτε memory controller. Ιστορικά, το μέγεθος του bandwidth των memory controllers μεγαλώνει με μικρότερο ρυθμό από το ρυθμό αύξησης των instructions ανά κύκλο ρολογιού που μπορεί να εκτελέσει ένας επεξεργαστής. Αν ένας επεξεργαστής στέλνει περισσότερα αιτήματα στην μνήμη ανά δευτερόλεπτο απ΄ αυτά που μπορεί ο memory controller να υποστηρίξει, τότε το πλεόνασμα θα μπεί σε ουρά προτεραιότητας. Όσο οι ουρές μεγαλώνουν, αυξάνεται και το latency πρόσβασης στη μνήμη. Όπως δείξαμε και σε προηγούμενες υποενότητες, οι

επεξεργαστές προσπαθούν να καλύψουν το latency της πρόσβασης μνήμης (π.χ. caches, prefetchers) αλλά όσο το latency συνεχίζει να αυξάνεται, τόσο δυσκολότερο είναι να περιθωριοποιηθεί. Το πρόβλημα αυτό είναι γνωστό ως Memory Wall και έχει αναλυθεί εκτεταμένα στο παρελθόν[12][13].

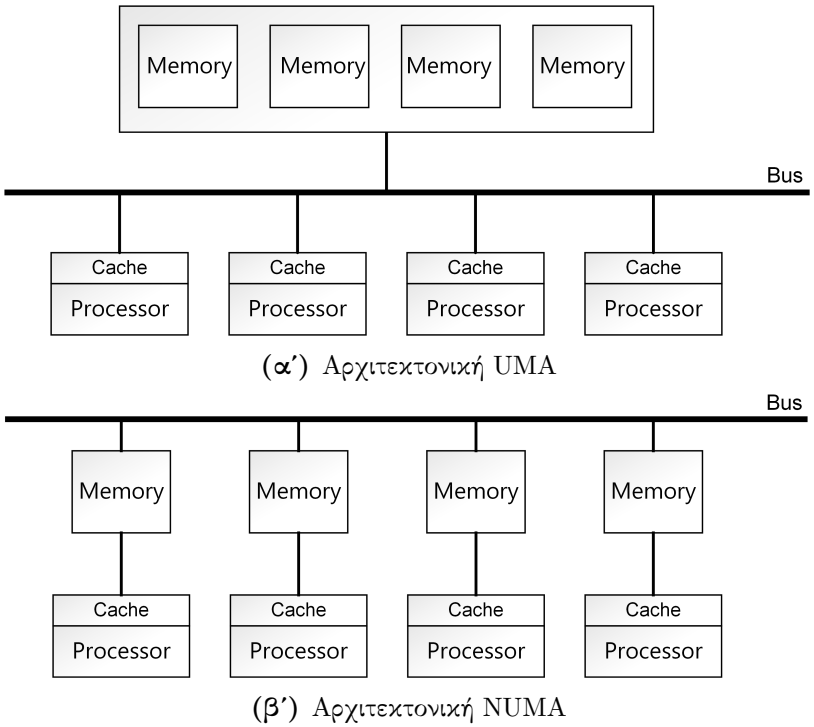
1.3.2 Non Uniform Memory Access - NUMA

Μια διαφορετική προσέγγιση της αρχιτεκτονικής UMA, είναι μια αρχιτετκονική που δεν θα περιορίζεται, έστω σε τόσο μεγάλο βαθμό, από το latency που οφείλεται σε προσβάσεις στη μνήμη. Η αρχιτεκτονική αυτή ονομάζεται αρχιτεκτονική ανομοιόμορφης πρόσβασης μνήμης ή NUMA - Non Uniform Memory Access, και ουσιαστικά είναι αρχιτεκτονική στην οποία η πρόσβαση στη μνήμη εξαρτάται από την τοποθεσία της μνήμης σε σχέση με τον εκάστοτε επεξεργαστή. Ο στόχος της NUMA είναι η αύξηση του bandwidth στη DRAM. Η βασική ιδέα πίσω απ΄ την αρχιτεκτονική αυτή είναι η χρήση περισσότερων από έναν memory controllers. Οι πυρήνες του επεξεργαστή χωρίζονται σε nodes, και κάθε node έχει τον δικό του memory controller, καθώς και τη δικιά του τοπική μνήμη (local memory). Στην εικόνα 1.6 παρουσιάζεται ένα τυπικό παράδειγμα της αρχιτεκτονικής NUMA για δύο nodes, από 4 πυρήνες το καθένα, με την αντίστοιχη τοπική τους μνήμη. Για την πρόσβαση στην μνήμη που διαχειρίζονται άλλοι memory controllers (remote memory), ο πυρήνας στέλνει memory requests μέσω interconnect links. Η χρήση της διάταξης αυτής προσφέρει μικρότερο latency για πρόσβαση στην τοπική μνήμη σε σχέση με την πρόσβαση στην απομακρυσμένη.



Σχήμα 1.6: Non Uniform Memory Access - NUMA: Οι επεξεργαστές χωρίζονται σε nodes. Κάθε node έχει ξεχωριστό memory controller και τμήμα της DRAM

Το μέγιστο διαθέσιμο bandwidth που μπορεί να επιτευχθεί στην τοπολογία αυτή, είναι το άθροισμα των μέγιστων bandwidth που μπορούν να υποστηρίξουν οι memory controllers. Σε ένα μοντέρνο σύστημα, το μέγιστο bandwidth μπορεί να παρατηρηθεί όταν όλοι οι πυρήνες έχουν αποκλειστική πρόσβαση στην τοπική τους μνήμη, οπότε και δε χρειάζεται να στείλουν



(α′) Αρχιτεκτονική UMA

(β′) Αρχιτεκτονική NUMA

Σχήμα 1.7: Η διαφορά μεταξύ των αρχιτεκτονικών UMA 1.7α′ και NUMA 1.7β′

cache coherency μηνύματα, και κατά συνέπεια, να χρησιμοποιήσουν interconnect links.

1.4 Remote access penalty - Σκοπός εργασίας

Η χρήση της αρχιτεκτονικής NUMA κατά κόρον στα περισσότερα πολυπύρηνα συστήματα, έχει προσφέρει σημαντικές βελτιώσεις στην απόδοση των εφαρμογών και προγραμμάτων, χωρίς να σημαίνει ότι είναι μια τέλεια υλοποίηση. Συγκεκριμένα, η πρόσβαση στην απομακρυσμένη μνήμη από έναν υπολογιστικό πυρήνα, προσθέτει μεγάλο latency, αφού περισσότερα hops απαιτούνται για την πρόσβαση, καθώς και μειωμένο memory bandwidth, μιας και τα interconnect links χαρακτηρίζονται από μικρότερο bandwidth σε σχέση με τα local DIMMs. Η NUMA αρχιτεκτονική, όπως αναφέρθηκε προηγουμένως, δουλεύει βέλτιστα όταν όλοι οι επεξεργαστές έχουν τα δεδομένα που θα χρειαστούν στην τοπική τους μνήμη. Όμως, το σενάριο αυτό, για τη βέλτιστη λειτουργία, δεν είναι πάντα εφικτό, ιδιαίτερα όταν πρόκειται για υπολογιστές που χρησιμοποιούνται σε Cloud υπηρεσίες, και πρέπει οι πόροι τους να διαμοιράζονται μεταξύ των χρηστών. Κατασκευάζοντας πολλαπλές εικονικές μηχανές σε ένα πολυπύρηνο σύστημα, στις οποίες δίνεται αποκλειστική πρόσβαση σε ένα τμήμα των πόρων του φυσικού μηχανήματος, δηλαδή έναν αριθμό επεξεργαστών ή ένα ποσοστό μνήμης και δίσκου, δεν είναι δύσκολο να φανταστεί κανείς ότι μπορούν εύκολα να προκύψουν σενάρια στα οποία η μνήμη και οι επεξεργαστές ενός VM βρίσκονται σε διαφορετικά nodes. Ένα τέτοιο σενάριο θα αποφέρει ένα overhead στον τελικό χρήστη, το οποίο είναι μη επιθυμητό, και πρέπει να αποφεύγεται.

Έτσι, σκοπός της εργασίας είναι η μελέτη του φαινομένου που περιγράψαμε, όταν αυτό συμβαίνει σε ένα cloud περιβάλλον. Πραγματοποιήθηκαν μετρήσεις σε κατάλληλες συνθήκες στο εργαστήριο, απ΄ τις οποίες έχουμε εξάγει χρήσιμα συμπεράσματα για το ποσοστό μείωσης της απόδοσης και το αντίκτυπο που έχει το φαινόμενο αυτό σε πραγματικές συνθήκες.

1.5 Διάρθρωση διπλωματικής εργασίας

Στο δεύτερο κεφάλαιο παρατέθηκαν συγγενείς εργασίες και έρευνες που έχουν γίνει στο θέμα με το οποίο ασχοληθήκαμε, καθώς και την δικιά μας σκοπιά και προσέγγιση. Κατόπιν, στο τρίτο κεφάλαιο γίνεται μια αναφορά στους πόρους, τα μηχανήματα και τις τεχνολογίες που χρησιμοποιήσαμε στην εργασία προκειμένου να πραγματοποιήσουμε πειραματικές μετρήσεις. Στη συνέχεια, στο τέταρτο κεφάλαιο παρουσιάζουμε τα αποτελέσματα των μετρήσεων και σχολιάζουμε τα αποτελέσματα. Τέλος, στο πέμπτο κεφάλαιο βρίσκονται τα συμπεράσματα που μπορούν να προκύψουν, καθώς και τα περιθώρια που υπάρχουν για μελλοντική έρευνα πάνω στο θέμα.

Κεφάλαιο 2

Συγγενείς εργασίες - Η δικιά μας προσέγγιση

Στο συγκεκριμένο κεφάλαιο παρουσιάζεται η προϋπάρχουσα έρευνα και εργασία που έχει γίνει στο latency της ανομοιόμορφης πρόσβασης στη μνήμη, που παρουσιάζει η αρχιτεκτονική NUMA. Επιπλέον θα περιγράψουμε την σκοπιά απ΄ την οποία προσεγγίσαμε εμείς το φαινόμενο αυτό, σε σενάρια cloud υποδομής, τις ενέργειες και μελέτες που πραγματοποιήσαμε, καθώς και τις πληροφορίες που μπορούμε να αντλήσουμε από τη διαδικασία.

2.1 Σχετικές Έρευνες

Το θέμα με το οποίο ασχοληθήκαμε στην παρούσα εργασία, έχει απασχολήσει ένα ευρύ επιστημονικό κοινό στο πρόσφατο παρελθόν. Πιο συγκεκριμένα, μια απ΄ τις σπουδαιότερες έρευνες που έχουν γίνει είναι η εργασία Memory Management in NUMA Multicore Systems: Trapped between Cache Contention and Interconnect Overhead των Zoltan Majo και Thomas R. Gross του πανεπιστημίου ETH στη Ζυρίχη[14]. Στα πλαίσια της εργασίας αυτής μελετήθηκε η συμπεριφορά των εφαρμογών όταν μεταβάλλεται το data locality σε NUMA αρχιτεκτονικές, αλλά και πώς ο διαμοιρασμός των πόρων της CPU, όπως η cache, επίσης επηρεάζει την απόδοση. Η μελέτη που έγινε είναι πολύ ενδιαφέρουσα καθώς δείχνει ότι η τοπικότητα των δεδομένων δεν είναι πάντα η βέλτιστη λύση για την επίτευξη μέγιστης απόδοσης, αφού υπάρχουν περιπτώσεις κατά τις οποίες η κοινή χρήση της cache από διεργασίες που ζουν στο ίδιο NUMA node και κατά συνέπεια η συμφόρηση που προκύπτει στην πρόσβασή της, μπορεί να αποφέρει μικρότερες επιδόσεις απ΄ ότι αν δεν διαμοιράζονταν cache και είχαν πρόσβαση σε τοπολογικά απομακρυσμένη μνήμη. Το συμπέρασμα που προέκυψε από την εργασία αυτή είναι πως αξίζει να προσπαθήσουμε να επιτύχουμε data locality όταν η μνήμη που χρειάζεται να δεσμεύσουν οι διεργασίες είναι περίπου η ίδια για όλες. Σε αντίθετη περίπτωση πρέπει δοθεί προτεραιότητα στην αποφυγή της συμφόρησης στην cache προκειμένου να μεγιστοποιηθεί η απόδοση των διεργασιών. Παρόμοια έρευνα έχει γίνει και στις εργασίες που ακολουθούν. [15]–[17]

Ακόμα πιο κοντά στο θέμα μας, είναι η εργασία των Yuxia Cheng και Wenzhi Chen με τίτλο Evaluation of virtual machine performance on NUMA multicore systems[18]. Στη μελέτη αυτή πραγματοποιήθηκαν πειραματικές μετρήσεις σε εικονικές μηχανές πάνω σε πολυπύρηνα NUMA συστήματα, κατασκευασμένες με τον KVM hypervisor, με διαφορετική τοπολογία η κάθε μια, και παρατηρήθηκε μεταβολή της απόδοσης των εκτελεσθέντων μετροπρογραμμάτων που αφορούσε τόσο την τοπικότητα της μνήμης, όσο και την κοινή χρήση της cache του επεξεργαστή.

Τέλος, παραθέτουμε πηγές από μερικές ακόμα εργασίες στις οποίες έγιναν εκτεταμένες έρευνες για την αρχιτεκτονική NUMA, τα χαρακτηριστικά της, και την συμπεριφορά που παρουσιάζει στις ανομοιόμορφες προσβάσεις στη μνήμη, τις οποίες συμβουλευθήκαμε για την συγγραφή της παρούσας εργασίας. [19]–[22]

2.2 Η προσέγγιση της παρούσας εργασίας

Όπως προαναφέρθηκε, σκοπός της συγκεκριμένης εργασίας είναι η μελέτη του remote access penalty, δηλαδή μεγαλύτερο latency και μικρότερο bandwidth στην πρόσβαση της μνήμης, όταν η μνήμη και ο επεξεργαστής βρίσκονται σε διαφορετικά NUMA nodes. Πιο συγκεκριμένα, προσεγγίσαμε το φαινόμενο απ΄ την μεριά του cloud, καθώς θέλαμε να ανακαλύψουμε πώς συμπεριφέρονται οι cloud εφαρμογές και υπηρεσίες, στις οποίες απαιτείται συνεχώς διαμοιρασμός και διαχωρισμός πόρων. Μέσα από τη μελέτη αυτή, στοχεύουμε στην καταγραφή αποτελεσμάτων και συμπερασμάτων τα οποία μπορούν να φανούν σημαντικά για την βελτίωση της απόδοσης του cloud infrastructure.

Αρχικά μελετήσαμε τον KVM hypervisor καθώς και το QEMU λογισμικό προκειμένου να τα χρησιμοποιήσουμε για την δημιουργία εικονικών μηχανών, οι οποίες θα προσομοιώσουν το cloud περιβάλλον. Αποκτήσαμε πρόσβαση σε ένα πολυπύρηνο σύστημα, αρχιτεκτονικής NUMA, του εργαστηρίου CSlab που βρίσκεται στο Εθνικό Μετσόβιο Πολυτεχνείο. Στο μηχάνημα αυτό στήσαμε τις εικονικές μηχανές που χρησιμοποιήσαμε για να εκτελέσουμε πειραματικές μετρήσεις. Χρησιμοποιήσαμε το λογισμικό διεπαφής εικονικοποίησης libvirt, για να κατασκευάσουμε τρεις εικονικές μηχανές, στις οποίες οι πόροι κατανεμήθηκαν με διαφορετικό τρόπο κάθε φορά, ώστε να συγκρίνουμε τις επιδόσεις των εκτελεσθέντων εφαρμογών. Κατόπιν, επιλέξαμε μερικά κατάλληλα για το σενάριό μας benchmarks, τα οποία εκτελέσαμε κάτω απ΄ τις ίδιες συνθήκες κάθε φορά σε κάθε εικονική μηχανή ξεχωριστά, καταγράφοντας κάθε φορά τα αποτελέσματα προκειμένου να συγκριθούν με τα αντίστοιχα των υπόλοιπων VM's. Έτσι, αντιμετωπίζοντας την κάθε εικονική μηχανή σαν μια πιθανή περίπτωση που μπορεί να προκύψει σε cloud infrastructure, θα προσπαθήσουμε να αποφανθούμε εάν, και σε τι βαθμό, επηρεάζει η αρχιτεκτονική του επεξεργαστή την επίδοση των cloud εφαρμογών.

Για τα χαρακτηριστικά του φυσικού μηχανήματος, των εικονικών μηχανών που δημιουργήθηκαν, και τις εφαρμογές που επιλέξαμε να χρησιμοποιήσουμε για την προσομοίωση, ακολουθεί εκτεταμένη περιγραφή και διασαφήνιση.

Κεφάλαιο 3

Υλικό και Λογισμικό

Στο κάτωθι κεφάλαιο παρουσιάζουμε το υλικό που χρησιμοποιήθηκε για τις μετρήσεις, ακολουθούμενο από το λογισμικό κατασκευής εικονικών μηχανών και προσομοιώσεων.

3.1 Χαρακτηριστικά χρησιμοποιηθέντος υλικού

Προκειμένου να πραγματοποιήσουμε μετρήσεις και να αναλύσουμε τη συμπεριφορά των cloud based εφαρμογών σε πολυπύρηνα NUMA συστήματα, ήταν απαραίτητη η πρόσβαση σε ένα φυσικό μηχάνημα, server, ο οποίος είχε αρκετούς πόρους για τις ζητούμενες μετρήσεις, αλλά προφανώς και CPU με NUMA αρχιτεκτονική. Ένα τέτοιο μηχάνημα δανειστήκαμε από το εργαστήριο CSlab - Computer Systems Laboratory του Εθνικού Μετσόβιου Πολυτεχνείου.

Το μηχάνημα που επιλέξαμε αποτελείται από δύο AMD Opteron 6378 CPUs[1] και 256GB μνήμη RAM. Το ΛΣ του μηχανήματος είναι Debian Linux 8 (jessie). Στους παρακάτω δύο πίνακες, 3.1 και 3.2, ακολουθούν αναλυτικές πληροφορίες για την CPU.

General Info		**Bus**		**Memory Controller**	
Data width	64 bits	Architecture	HyperTransport 3.1	Mem. Controllers	1
Cores	16	Transfer rate	6400 MT/s	Mem. Channels	4
Threads	16	Clock Speed	3200 MHz	DIMMs per chan.	3
Frequency	2400 MHz			Max Memory bandwidth	512 GB/s

Πίνακας 3.1: Βασικά χαρακτηριστικά επεξεργαστή

Όπως φαίνεται και στους πίνακες, η CPU έχει 16 cores, και το σύστημά μας αποτελείται από 32 cores. Ο κάθε επεξεργαστής χωρίζει τα 16 cores του σε 2 numa nodes. Έτσι, συνολικά το μηχάνημα έχει 4 numa nodes με τοπική μνήμη 64GB και 8 cores σε κάθε node. Οι αποστάσεις μεταξύ των nodes είναι ίδιες, οπότε και το latency πρόσβασης δεδομένων ενός node προς οποιοδήποτε άλλο, είναι ακριβώς το ίδιο. Στην εικόνα 3.1 φαίνεται ακριβώς η κατανομή και η τοπολογία του επεξεργαστή.

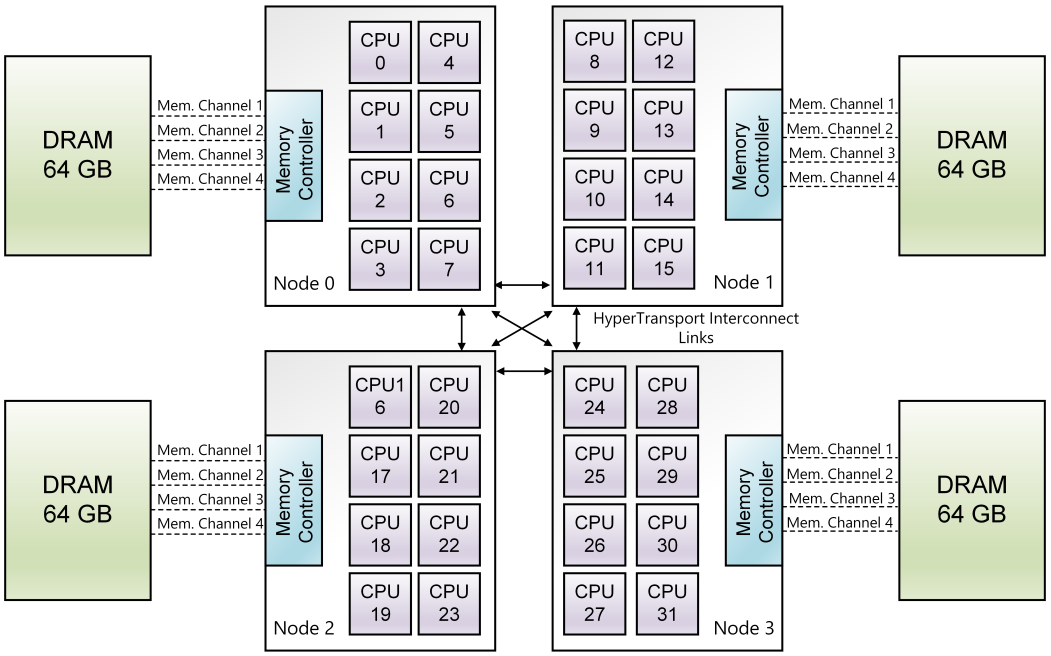
[1] https://www.amd.com/en/products/cpu/6378

Cache Details				
Cache:	L1 data	L1 instruction	L2	L3
Size:	16 x 16 KB	8 x 64 KB	8 x 2 MB	2 x 6 MB
Associativity:	4-way set associative	2-way set associative	16-way set associative	48-way set associative
Line size:	64 bytes	64 bytes	64 bytes	64 bytes
Placemet Policy:	Direct-mapped	Direct-mapped	Non-inclusive Direct-mapped	Non-inclusive Direct-mapped
Distribution:	1 cache per 1 core	1 cache per 2 cores	1 cache per 2 cores	1 cache per 8 cores

Πίνακας 3.2: Χαρακτηριστικά cache

Στην CPU, όπως βλέπουμε στον παραπάνω πίνακα, παρέχονται 768KB L1 μνήμη cache, 16MB L2 και 12MB L3. Η κρυφή μνήμη είναι κατανεμημένη μεταξύ των επεξεργαστικών πυρήνων, ώστε να υπάρχει 64KB L1 data cache ανά επεξεργαστή, 64KB L1 instruction cache και 2MB L2 cache ανά δύο επεξεργαστές, καθώς και 6MB L3 ανά οχτώ επεξεργαστές.

Επιπλέον, η διασύνδεση μεταξύ των numa nodes πραγματοποιείται με την τεχνολογία HyperTransport της AMD[23]. Η τεχνολογία αυτή αποτελείται από αμφίδρομα, high bandwidth, μικρού latency, point-to-point links. Τα links χρησιμοποιούνται για διασύνδεση των nodes του επεξεργαστή καθώς και επεξεργαστή και I/O, και προσφέρουν πολύ μεγαλύτερες ταχύτητες σε σχέση με το τυπικό bus που υπήρχε στο παρελθόν. Πιο συγκεκριμένα, το HyperTransport 3.1 αποτελείται από 32 bit links, ρολόι συχνότητας 3200 MHz και προσφέρει συνολικά έως και 51.2 GB/s bandwidth. Η τεχνολογία αυτή μπορεί επίσης να χρησιμοποιηθεί σε routers ή switches, καθώς και για την διασύνδεση co-processors με την κεντρική CPU.



Σχήμα 3.1: Τοπολογία επεξεργαστών φυσικού μηχανήματος

3.2 Χαρακτηριστικά χρησιμοποιηθέντος λογισμικού

Στην παρούσα υποενότητα θα γίνει αναλυτική αναφορά σε όλα τα εργαλεία που χρησιμοποιήθηκαν κατά τη διάρκεια της διπλωματικής εργασίας προκειμένου να κατασκευαστούν τα απαιτούμενα σενάρια και να γίνουν οι κατάλληλες μετρήσεις. Η περιγραφή που ακολουθεί, είναι της μορφής «bottom - up», ξεκινώντας δηλαδή από τα κατώτερα επίπεδα λογισμικού για την εικονικοποίηση έως και τα ανώτερα για την προσομοίωση cloud εφαρμογών.

3.2.1 QEMU - KVM

Η εικονικοποίηση ήταν και είναι ένα θέμα που έχει απασχολήσει ευρύ κοινό στον τομέα της επιστήμης των υπολογιστών. Η χρησιμότητα που προσφέρει πηγάζει από ζητήματα όπως απομόνωση και διαχωρισμός των server και εικονικά περιβάλλοντα για δοκιμές και προσομοιώσεις. Έτσι, οι κατασκευαστές επεξεργαστών (π.χ. Intel, AMD), βλέποντας την ανάγκη αυτή, τα τελευταία χρόνια έχουν προσθέσει στο instruction set των chip τους εντολές, οι οποίες εξυπηρετούν στην ταχύτερη και ευκολότερη εκμετάλλευση της τεχνολογίας της εικονικοποίησης (Intel VT, AMD-V).

Τις επεκτάσεις αυτές στις εντολές εκμεταλλεύεται το KVM ή Kernel-based Virtual Machine, ένα virtualization module που επιτρέπει στον πυρήνα (kernel) να λειτουργεί ως hypervisor τύπου 1, δηλαδή bare metal hypervisor. Ο KVM σχεδιάστηκε αρχικά για επεξεργαστές αρχιτεκτονικής x86 και i386, ενώ στη συνέχεια επεκτάθηκε και σε διαφορετικούς, όπως ARM, PowerPC, S/390. Το 2007 το KVM εντάχθηκε στον κώδικα του linux kernel στην έκδοση 2.6.20

Το KVM λειτουργεί σαν ένα τυπικό process όπως και τα υπόλοιπα που τρέχουν στο ΛΣ, μπορεί να διαχειρίζεται, να δημιουργεί και να τρέχει πολλαπλές εικονικές μηχανές, και προσφέρει απαραίτητες λειτουργίες οι οποίες περιλαμβάνουν memory manager, process scheduler, I/O stack και πολλά περισσότερα που απαιτούνται για την ορθή λειτουργία των VM. Το KVM κάνει διαθέσιμο ένα device node, το `/dev/kvm` , που προσφέρει τις εξής λειτουργίες:

- Κατασκευή εικονικής μηχανής
- Δέσμευση μνήμης για μια εικονική μηχανή
- Διάβασμα και γράψιμο σε εικονικούς CPU registers
- Αποστολή interrupt σε μια εικονική CPU
- Εκτέλεση μιας εικονικής CPU

Παράλληλα, υπάρχει μηχανισμός με τον οποίο εικονικοποιείται η μνήμη και η MMU, ώστε να γίνεται η αντιστοίχιση μεταξύ των μνημών guest virtual σε guest physical καθώς και guest physical σε host physical. Χρησιμοποιώντας τις επεκτάσεις του υλικού, δημιουργείται νέο

page table που αποθηκεύει την αντιστοίχιση guest virtual σε host physical και συγχρόνως φροντίζει να συγχρονίζει το guest page table με το νέο shadow page table. Εκτός από MMU προσομοιώνεται και η λειτουργία του TLB για τη βέλτιστη απόδοση μνήμης του guest, και δίνει τη δυνατότητα για hot plug vCPUs, δυναμική διαχείριση μνήμης και live migration.

Ενώ το KVM είναι ικανό να εκμεταλλεύεται τις επεκτάσεις υλικού, και να είναι ένας απ΄ τους ταχύτερους hypervisors, με ταχύτητες εκτέλεσης εντολών εικονικών μηχανών σχεδόν ίδιες με αυτές των native εφαρμογών, δεν μπορεί να λειτουργήσει αυτόνομα, και να εκτελέσει όλες τις απαραίτητες ενέργειες τις εικονικοποίησης. Για παράδειγμα, δεν μπορεί να κάνει emulate έναν επεξεργαστή ή άλλα περιφερειακά υλικά που μπορεί να απαιτούνται. Και εδώ είναι που το QEMU έρχεται να δώσει λύση και να συνεργαστεί με το KVM για ένα ολοκληρωμένο αποτέλεσμα εικονικοποίησης.[24]

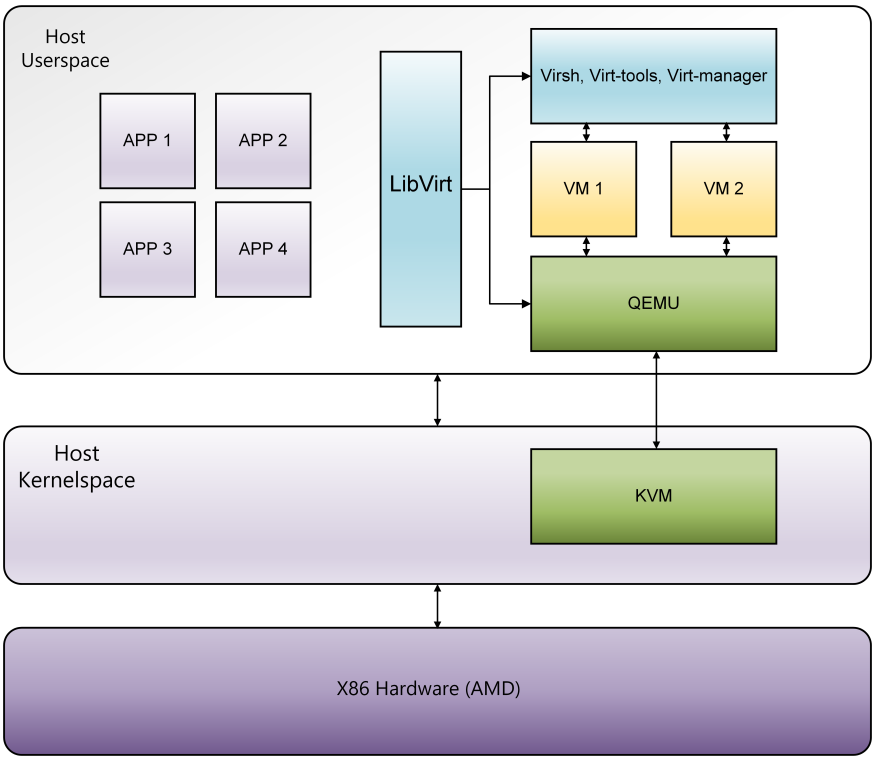
Το QEMU (Quick EMUlator) είναι ένα δωρεάν λογισμικό ανοιχτού κώδικα, γραμμένο στη γλώσσα προγραμματισμού C, το οποίο πραγματοποιεί hardware virtualization. Ανήκει στην κατηγορία των type 2 hypervisors, δηλαδή ζει και λειτουργεί στο userspace. Το hardware virtualization που προσφέρει σημαίνει ότι μπορεί να δημιουργήσει δίσκους, δίκτυο, VGA, PCI, USB και ένα μεγάλο πλεονέκτημα που έχει είναι πως κάνει emulate επεξεργαστές μέσω dynamic binary translation (DBT), επιτρέποντας έτσι κώδικα που είναι γραμμένος για έναν τύπο επεξεργαστή, να εκτελεστεί σε διαφορετικό (π.χ. ARM σε x86). Το QEMU είναι standalone hypervisor, δηλαδή μπορεί να λειτουργήσει αυτόνομα και να κατασκευάσει εικονικές μηχανές, αλλά όταν συνυπάρχει με το KVM, οι εικονικές μηχανές που κατασκευάζονται έχουν πολύ μεγαλύτερες επιδόσεις. Επιπλέον το QEMU χρησιμοποιείται εκτός από την εικονικοποίηση και για προσομοίωση user-level διεργασιών, ώστε να γίνονται compile σε διαφορετικές αρχιτεκτονικές επεξεργαστών.

3.2.2 LibVirt

Ένα απ΄ τα κυριότερα σημεία της παρούσας εργασίας είναι η ορθή κατασκευή των εικονικών μηχανών και η τροποποίηση της τοπολογίας τους, ώστε να ταιριάζουν στα σενάρια που θέλουμε να προσομοιωθούν. Προκειμένου να επιτευχθεί αυτό, χρησιμοποιήθηκε το λογισμικό LibVirt το οποίο ανέλαβε την επικοινωνία με τον hypervisor. Το LibVirt είναι μια προγραμματιστική διεπαφή (API) ανοιχτού κώδικα, δαίμονας (libvirtd) και διαχειριστικό εργαλείο (virsh) για hypervisors. Χρησιμοποιείται για την διαχείριση διάφορων hypervisor μέσα στους οποίους και ο QEMU/KVM. Σκοπός του LibVirt είναι να προσφέρει στο χρήστη έναν απλό τρόπο να διαχειρίζεται διάφορους virtualization providers/hypervisors με ένα εργαλείο, χωρίς να χρειάζεται να γνωρίζει αυτά που παρέχει ο εκάστοτε hypervisor. Έτσι, χωρίς να γίνει χρήση QEMU/KVM εργαλείων, δημιουργήθηκαν τα κατάλληλα VMs. Στο παρακάτω σχήμα 3.2 φαίνεται ακριβώς η διασύνδεση μεταξύ libvirt, QEMU, KVM σε ένα φυσικό linux μηχάνημα.

Το LibVirt, όπως αναφέρθηκε, είναι δωρεάν λογισμικό ανοιχτού κώδικα, το οποίο συντηρείται και αναβαθμίζεται από το development team της Red Hat, ενώ έχουν συμβάλει ανά διαστήματα περισσότεροι φορείς. Είναι γραμμένο στη γλώσσα προγραμματισμού C και

περιλαμβάνει βιβλιοθήκες με bindings ώστε να μπορούν να χρησιμοποιηθούν από διάφορες γλώσσες προγραμματισμού, όπως Python, Perl, OCaml, Java και PHP.



Σχήμα 3.2: Εικονικοποίηση με χρήση LibVirt - QEMU - KVM

Χρησιμοποιώντας τις δυνατότητες που προσφέρει το libvirt καταφέραμε να δημιουργήσουμε τις τρεις ζητούμενες εικονικές μηχανές, και να τροποποιήσουμε τις numa τοπολογίες μνήμης και επεξεργαστών ώστε να κατασκευάσουμε τα ζητούμενα τρία σενάρια εκτέλεσης μετροπρογραμμάτων.

3.2.3 Benchmarks

Μετά την κατασκευή των εικονικών μηχανών, σκοπός ήταν να μελετήσουμε την επίδοση πρόσβασης στη μνήμη σε πραγματικές συνθήκες, οι οποίες συναντώνται κυρίως σε cloud infrastructures. Έτσι, χρησιμοποιήσαμε μετροπρογράμματα ή benchmarks, προγράμματα δηλαδή κατασκευασμένα για να προσομοιώνουν τη λειτουργία ρεαλιστικής εφαρμογής, τα οποία χρησιμοποιούνται για αξιολόγηση και εξαγωγή συμπερασμάτων που αφορούν τις επιδόσεις των συστημάτων στα οποία εκτελούνται. Για την παρούσα εργασία, επιλέχθηκαν δύο σουίτες τέτοιων μετροπρογραμμάτων. Η σουίτα Cloudsuite και η Parsec.

3.2.3.1 Cloudsuite

Η Cloudsuite[2] είναι μια σουίτα που αποτελείται από οκτώ benchmarks[25], [26]. Τα benchmakrs αυτά, βασίζονται στις πιο συνήθεις εφαρμογές που συναντώναι στα data centers,

[2] https://www.cloudsuite.ch/

ώστε να ανταποκρίνονται ακριβώς στις πραγματικές εφαρμογές που συναντάμε σε cloud infrastructures. Η σουίτα δημιουργήθηκε και συντηρείται από το εργαστήριο PARSA (Parallel Systems Architecture Lab) του πανεπιστημίου EPFL στη Λωζάνη της Ελβετίας. Τα benchmarks είναι τα εξής:

1. **Data Analytics**

   Η συνεχής αύξηση των ανθρωπογενών πληροφοριών, απαιτεί αυτοματοποιημένη αναλυτική επεξεργασία, με σκοπό την ομαδοποίηση και φιλτράρισμα των πληροφοριών αυτών. Το μοντέλο map-reduce[27] εξελίχθηκε σε μια απ΄ τις πιό δημοφιλείς προσεγγίσεις σε large-scale ανάλυση, αναθέτοντας requests σε ένα cluster από μηχανήματα, τα οποία αρχικά πραγματοποιούν φιλτράρισμα και διαχωρισμό των δεδομένων (map), και στη συνέχεια αθροίζουν τα αποτελέσματα (reduce). Το Data Analytics benchmark συμπεριλήφθηκε στη σουίτα Cloudsuite, αλλά και στην παρούσα εργασία, ώστε να καλύψει τη συνεχώς αυξανόμενη σημασία που έχουν οι Machine Learning αλγόριθμοι στην ανάλυση και επεξεργασία των δεδομένων στα Data Centers, κάνοντας χρήση του map-reduce μοντέλου. Το benchmark αποτελείται από ένα Hadoop cluster[3], μια υλοποίηση ανοιχτού κώδικα δηλαδή του map-reduce μοντέλου, στο οποίο εκτελείται ο Naive Bayes classifier, ένας Machine Learning αλγόριθμος που βρίσκεται στη Mahout[4] βιβλιοθήκη, για ένα Wikimedia dataset.

2. **Data Caching**

   Προκειμένου να βελτιστοποιήσουν την απόδοσή τους, οι σύγχρονοι web servers χρησιμοποιούν object caching systems ώστε να αποθηκεύουν τα αποτελέσματα από «ακριβούς» χρονικά υπολογισμούς. Το Data Caching benchmark χρησιμοποιεί τον Memcached data caching server[5], το οποίο είναι ένα σύγχρονο, ανοιχτού κώδικα, object caching σύστημα. Χρησιμοποιείται από web servers για την αποθήκευση χρονοβόρων ερωτημάτων (queries) σε βάσεις δεδομένων. Είναι ένα εντελώς in-memory key-value αποθηκευτικό εργαλείο. Επιπλέον, το benchmark υλοποιεί εικονικούς clients, οι οποίοι στέλνουν αιτήματα στον Memcached server. Ο κάθε client στέλνει αιτήματα και εξετάζει τα αποτελέσματα ανεξάρτητα, ώστε να σημειώνει τους χρόνους απόκρισης του εκάστοτε server. Το benchmark προσομοιώνει τη λειτουργία ενός ή περισσότερων Twitter data caching servers χρησιμοποιώντας ένα Twitter dataset καθώς και clients που ζητούν πρόσβαση στα δεδομένα αυτά. Η μετρική που χρησιμοποιείται για την αξιολόγηση της προσομοίωσης είναι ο αριθμός των requests που εξυπηρετήθηκαν, ανά δευτερόλεπτο.

3. **Data Serving**

[3] https://hadoop.apache.org/
[4] https://mahout.apache.org/
[5] http://memcached.org/

Διάφορες NoSQL βάσεις δεδομένων έχουν σχεδιαστεί τα τελευταία χρόνια ώστε να εξυπηρετούν ως αποθηκευτικός χώρος δεδομένων για large-scale web applications, όπως το Facebook, Google Earth, Netflix, E-bay και πολλά ακόμα. Τα NoSQL συστήματα τεμαχίζουν εκατοντάδες TB δεδομένων σε πολλά τμήματα και αυξάνονται οριζοντίως (horizontally scale out) σε μεγάλα clusters. Οι περισσότερες ενέργειες που απαιτούνται για την εύρεση δεδομένων γίνονται με χρήση indexes που υποστηρίζουν γρήγορο lookup, ενώ παράλληλα είναι σχεδιασμένα με τέτοιο τρόπο ώστε το κάθε query στη βάση να εκτελείται από ένα μηχάνημα, και τα αποτελέσματα που απαιτούν πληροφορίες από πολλά μηχανήματα να οργανώνονται από το middleware. Απ΄ τις πιό γνωστές NoSQL βάσεις δεδομένων είναι η MongoDB[6], η Apache's Cassandra[7] και η DynamoDB[8]. Το Data Serving benchmark χρησιμοποιεί το εργαλείο της Yahoo! YCSB (Yahoo! Cloud Serving Benchmark)[28], που χρησιμοποιείται για benchmarking data store συστημάτων. Το YCSB έρχεται με τα κατάλληλα interfaces για populating και stressing πολλών γνωστών data serving συστημάτων, απ΄ τα οποία επιλέχθηκε το Apache's Cassandra. Χρησιμοποιώντας το benchmark δίνεται η δυνατότητα στο χρήστη να κατασκευάσει έναν ή περισσότερους Cassandra data store servers, και ύστερα να τους κάνει populate και stress επιλέγοντας τον αριθμό των operations που θα εκτελεστούν.

4. **Graph Analytics**

   Το Graph Analytics benchmark βασίζεται στο Apache's Spark framework[9], για να πραγματοποιήσει graph analytics σε large scale datasets. Όπως αναφερθήκαμε και προηγουμένως στο Hadoop MapReduce, έτσι και το Spark είναι ένα αντίστοιχο framework, που χρησιμοποιείται για γενικού περιεχομένου κατανεμημένο cluster computing, διανέμει δηλαδή τα δεδομένα σε μηχανήματα ενός cluster, ώστε μετά να τα επεξεργαστεί παράλληλα. Χρησιμοποιεί in-memory τεχνικές για να μειώσει το latency, και είναι κατάλληλο για real time επεξεργασία δεδομένων, σε αντίθεση με το hadoop που εξυπηρετεί καλύτερα batch processing ανάγκες. Μπορεί να επεξεργάζεται δεδομένα από διάφορα data stores, όπως το HDFS (Hadoop Distributed File System), NoSQL και σχεσιακές βάσεις δεδομένων, και πολλά ακόμα. Στο συγκεκριμένο benchmark, χρησιμοποιείται το framework GraphX[10], το οποίο είναι σχεδιασμένο να τρέχει πάνω στο Spark, και εκτελεί τον αλγόριθμο PageRank[29] σε ένα Twitter dataset. Ο χρήστης έχει τη δυνατότητα να ορίσει πόση μνήμη θα κάνει allocate το Spark, ώστε να τρέξει η προσομοίωση εντελώς in-memory.

5. **In-Memory Analytics**

[6] https://www.mongodb.com/
[7] http://cassandra.apache.org/
[8] https://aws.amazon.com/dynamodb/
[9] https://spark.apache.org/
[10] https://spark.apache.org/graphx/

Τα τελευταία χρόνια, μια απ΄ τις πιο σημαντικές εφαρμογές που λαμβάνουν χώρα σε cloud infrastructures είναι τα recommender systems. Σκοπός των συστημάτων είναι να προβλέψουν τη βαθμολογία ή την προτίμηση του χρήστη για κάποιο προϊόν. Παραδείγματα τέτοιων προϊόντων είναι οι ταινίες, βιβλία, γεγονότα, άρθρα, μουσική και γενικότερα ό,τι άλλο μπορεί να χαρακτηρισθεί προϊόν που ένας χρήστης καλείται να χρησιμοποιήσει/αγοράσει/βαθμολογήσει. Το In-Memory Analytics benchmark, ομοίως με το προηγούμενο Graph Analytics, χρησιμοποιεί το Spark framework και το MLlib[11], μια βιβλιοθήκη με scalable Machine Learning αλγορίθμους, προκειμένου να εφαρμόσει τον αλγόριθμο ALS (Alternating Least Square) σε μια λίστα με κριτικές χρηστών σε ταινίες. Σκοπός είναι η εκπαίδευση ενός νευρωνικού δικτύου χρησιμοποιώντας ένα dataset με κριτικές, και έπειτα η αξιολόγηση άλλων ταινιών με βάση τις προτιμήσεις του χρήστη. Το benchmark τρέχει εξ ολοκλήρου in-memory, και η μετρική που εξάγουμε είναι ο χρόνος που απαιτείται για να γίνουν οι εκτιμήσεις προτίμησης των ταινιών.

6. **Media Streaming**

   Η εύκολη δια-συνδεσιμότητα των χρηστών στο διαδίκτυο με χρήση high-bandwidth δικτύων, τόσο στο σπίτι όσο και στα κινητά τηλέφωνα, έχει αναγκάσει τις media streaming εφαρμογές, όπως το YouTube και το Netflix, να είναι πανταχού παρούσες. Το streaming video είναι ένας τρόπος παράδοσης περιεχομένου, κατά τον οποίο τα video παραδίδονται στον τελικό χρήστη παράλληλα με την προβολή τους, και ο χρήστης δεν χρειάζεται να προβεί σε λήψη ολόκληρου του αρχείου προτού αρχίσει την προβολή. Το Media Streaming benchmark καλύπτει την ανάγκη αυτή, καθώς χρησιμοποιεί τον Nginx web server[12], στον οποίο αποθηκεύει video με διαφορετικά χαρακτηριστικά, όπως η διάρκεια και η ποιότητα, και παράλληλα χρησιμοποιώντας το httperf[13] εργαλείο, ένα εργαλείο το οποίο αποσκοπεί στη μέτρηση της απόδοσης των web server, παράγει ένα mix αιτήσεων video στον server, διαφορετικής διάρκειας και ποιότητας.

7. **Web Search**

   Οι μηχανές αναζήτησης στις μέρες μας, όπως η Google, Bing, Yahoo! κλπ, πραγματοποιούν indexing σε πολλά TB δεδομένων τα οποία συλλέγουν από διάφορες online πηγές. Προκειμένου να υποστηρίζουν έναν τεράστιο αριθμό από συνεχόμενα, παράλληλα, latency sensitive queries αναζήτησης στο ευρετήριο, χωρίζουν τα δεδομένα σε shards τα οποία βρίσκονται in-memory, και διαμοιράζονται στα index serving nodes (ISN), με κάθε ISN να είναι υπεύθυνο αποκλειστικά για τα δικά του shards. Έτσι, ένα frontend μηχάνημα το οποίο λαμβάνει το ερώτημα προς αναζήτηση, θα το μεταφέρει σε όλα τα ISN παράλληλα, λαμβάνει τις απαντήσεις, και σχηματίζει κατάλληλη απάντηση για να επιστραφεί στον χρήστη. Εκατοντάδες ερωτήματα ανά δευτερόλεπτο φτάνουν στο κάθε ISN, τα οποία είναι άσχετα μεταξύ τους, γι΄ αυτό και είναι απαραίτητο τα

[11] https://spark.apache.org/mllib/
[12] https://www.nginx.com/
[13] https://github.com/httperf/httperf

shards να βρίσκονται in-memory, αλλά και να υπάρχουν ρέπλικες των ISN, ώστε να γίνεται load balancing, και να υπάρχει πολύ χαμηλό latency. Το σενάριο αυτό κατασκευάζει το Web Search benchmark, στο οποίο δημιουργείται ένα ISN χρησιμοποιώντας το Apache Solr[14], όπου υπάρχει ένα index 12GB αποθηκευμένο in-memory, από διάφορες online ιστοσελίδες. Έπειτα προσομοιώνουμε έναν αριθμό χρηστών, οι οποίοι στέλνουν παράλληλα ερωτήματα προς το ISN, και καταγράφουν το χρόνο που χρειάστηκε για να λάβουν απάντηση.

8. **Web Serving**

   Το Web Serving benchmark, προσομοιώνει τη λειτουργία ενός online κοινωνικού δικτύου, όπως το Facebook. Σε ένα κοινωνικό δίκτυο ένας χρήστης μπορεί να κάνει εγγραφή, να συνδεθεί με άλλους χρήστες (επιλέγοντας να γίνουν «φίλοι»), να ανταλλάξει μηνύματα με άλλους χρήστες, καθώς και να έχει έναν προσωπικό «τοίχο», στον οποίο μπορεί να προσθέτει εικόνες, συνδέσμους ή κείμενο το οποίο είναι φανερό σε όλους τους φίλους του. Το benchmark καλύπτει όλες αυτές τις λειτουργίες, χρησιμοποιώντας το εργαλείο Elgg[15], το οποίο είναι ένα εργαλείο ανοιχτού κώδικα, γραμμένο στη γλώσσα προγραμματισμού PHP, και παρέχει όλες τις λειτουργίες που περιγράφηκαν παραπάνω, παρόμοια με το Facebook. Επιπλέον χρησιμοποιείται η βάση δεδομένων MySQL[16], για την αποθήκευση των δεδομένων αλλά και memcached server[17] για να γίνονται όσες ενέργειες είναι δυνατόν in-memory. Τέλος, δημιουργούνται clients, οι οποίοι ζητουν να πραγματοποιήσουν όλες τις ενέργειες που περιγράφηκαν, και σημειώνουν το χρόνο που έλαβε η κάθε ενέργεια να πραγματοποιηθεί.

3.2.3.2 Parsec

Η Parsec (Princeton Application Repository for Shared-Memory Computers) είναι σουίτα μετροπρογραμμάτων η οποία δημιουργήθηκε από τους Christian Bienia και Kai Li, στα πλαίσια του εργαστηρίου Computer Science του πανεπιστημίου του Πρίνστον στις ΗΠΑ.[30]–[32] Η Parsec περιλαμβάνει 13 benchmarks που καλύπτουν ένα ευρύ φάσμα σύγχρονων εφαρμογών, που αφορούν κυρίως HPC (High Performance Computing), και είναι κατασκευασμένα με χρήση C/C++. Για κάθε benchmark δίνεται στο χρήστη η επιλογή να το τρέξει είτε σειριακά είτε παράλληλα, χρησιμοποιώντας περισσότερα CPUs και threads, καθώς και η επιλογή του workload input, ώστε να εξαντλούνται πιο αποδοτικά οι πόροι του εκάστοτε συστήματος που τα εκτελεί. Από τα 13 benchmarks που προσφέρονται στη σουίτα, επιλέχθηκαν τα 10 προς εκτέλεση στην παρούσα εργασία, και ακολουθεί μια σύντομη περιγραφή για το καθένα απ΄ αυτά.

[14] https://lucene.apache.org/solr/
[15] https://elgg.org/
[16] https://www.mysql.com/
[17] https://memcached.org/

1. **Blackscholes**

   Οι ηλεκτρονικοί υπολογιστές έχουν αποδειχθεί τεχνολογία «κλειδί» για τους υπολογισμούς και τις συναλλαγές στον χρηματοοικονομικό τομέα. Πιο συγκεκριμένα, απαιτείται πολύ συχνά ο υπολογισμός των παραγώγων, που συναντώνται με μεγάλη συχνότητα στις περισσότερες μαθηματικές εξισώσεις που αφορούν οικονομικά. Μια τέτοια διάσημη εξίσωση είναι αυτή των F. Black και M. Scholes, η οποία ονομάζεται Black-Scholes equation[33], και χρησιμοποιείται για την αξιολόγηση μιας επενδυτικής επιλογής ώστε να μειωθεί το ρίσκο. Το Blackscholes benchmark είναι ένα πρόγραμμα το οποίο χρησιμοποιεί την εξίσωση αυτή σε ένα αρχείο εισόδου με 10.000.000 επιλογές τις οποίες καλείται να αξιολογήσει, πράγμα που σημαίνει ότι πρόκειται για ένα heavy computational benchmark στο οποίο χρειάζεται να γίνουν πολλά εκατομμύρια μαθηματικοί υπολογισμοί. Το benchmark επιλέχθηκε ώστε να εκπροσωπήσει το ευρύ πεδίο των αναλύσεων με χρήση διαφορικών εξισώσεων γενικώς, αλλά κυρίως στο τεχνοοικονομικό πεδίο.

2. **Bodytrack**

   Δεν είναι λίγες οι φορές όπου ένα μηχάνημα στηρίζει τη λειτουργία του σε επεξεργασία οπτικού σήματος από υπολογιστή, ώστε να αλληλεπιδρά με το περιβάλλον του. Η διαδικασία αυτή είναι απαιτητική σε resources, δύσκολο να την κατασκευάσεις προγραμματιστικά, και συχνά χρειάζεται να γίνεται σε πραγματικό χρόνο. Παραδείγματα αποτελούν το video surveillance, το character animation και το computer interfaces. Το Bodytrack benchmark είναι ένα τέτοιο computer vision benchmark, το οποίο εντοπίζει μια 3D φιγούρα ενός ανθρώπου σε βίντεο από τέσσερις κάμερες, που απεικονίζουν τον ίδιο άνθρωπο από διαφορετική γωνία λήψης, και προσθέτει πλαίσια στα μέλη του (χέρια, πόδια, κεφάλι και κορμό) ώστε να το ελέγξει ένας παρατηρητής. Στο benchmark δίνεται είσοδος βίντεο από τέσσερις κάμερες, αποτελούμενο από 261 frames στα οποία και πραγματοποιεί την ανάλυση.

3. **Canneal**

   Τα προβλήματα βελτιστοποίησης (optimization) είναι απ΄ τα πιο συχνά προβλήματα που καλείται να επιλύσει ένας υπολογιστής. Τέτοιο πρόβλημα αποτελεί και το Electronic Design Automation (EDA) στο οποίο σκοπός είναι να βρεθεί η βέλτιστη διάταξη και διαδρομή των κυκλωμάτων και transistor σε ένα microchip. Το Canneal benchmark είναι μια εφαρμογή που επιλύει αυτό το πρόβλημα, πραγματοποιώντας optimizations σε ένα chip με περίπου 2.500.000 στοιχεία πάνω στο κύκλωμα, τα οποία αναδιατάσσει συνεχώς, προσπαθώντας να βρει τη βέλτιστη λύση που δεν θα ξεπερνάει μια δεδομένη θερμοκρασία.

4. **Facesim**

   Η ραγδαία βελτίωση των γραφικών στα βιντεοπαιχνίδια και τις ταινίες τα τελευταία χρόνια, οφείλεται κατά μεγάλο ποσοστό στην βελτίωση των computer animations. Ένα

απ΄ τα πιο δύσκολα και απαιτητικά ζητήματα που αντιμετωπίζεται, είναι η προσομοίωση του ανθρώπινου προσώπου, το οποίο έχει αντιμετωπισθεί με μεγάλη επιτυχία, και τα περισσότερα ανθρώπινα πρόσωπα στα animations που βλέπουμε έχουν μεγάλη λεπτομέρεια και αληθοφάνεια. Το Facesim benchmark προσομοιώνει οπτικά την κίνηση ανθρώπινου προσώπου. Είσοδος του προγράμματος είναι ένα μοντέλο ανθρώπινου προσώπου, και μια χρονική ακολουθία από μϋϊκές κινήσεις που πραγματοποιεί, και υπολογίζει ένα οπτικά ρεαλιστικό animation του προσώπου, χρησιμοποιώντας κανόνες φυσικής.

5. **Ferret**

   Η αύξηση, σε παγκόσμια κλίμακα, των ψηφιακών δεδομένων, έχει δημιουργήσει πιο απαιτητικές ανάγκες για την αναζήτηση και την κατηγοριοποίηση των δεδομένων αυτών. Ένα απ΄ τα δυσκολότερα ζητήματα που αντιμετωπίζουμε συχνά είναι η εύρεση πανομοιότυπων εικόνων. Το Ferret benchmark πραγματοποιεί αναζήτηση σε μια βάση δεδομένων με εικόνες, προκειμένου να βρει παρόμοιες με αυτή που δίνεται ως είσοδος σε κάθε ερώτημα. Το benchmark έχει αποθηκευμένη μια βάση με 59.695 εικόνες και πραγματοποιεί 3.500 ερωτήματα προς τη βάση, για τα οποία επιστρέφει τις 50 πιο σχετικές εικόνες που βρήκε.

6. **Fluidanimate**

   Ένα απ΄ τα πιο απαιτητικά features που συναντώνται στα βιντεοπαιχνίδια και τις ταινίες, είναι τα physics simulations που εξυπηρετούν τη δημιουργία πολύ πιο ρεαλιστικών animations. Παρόμοια με όσα αναφέρθηκαν για το Facesim benchmark, ένα εξίσου δυσεπίλυτο πρόβλημα είναι η προσομοίωση της κίνησης του ρευστού (fluid animation). Το Fluidanimate benchmark χρησιμοποιεί το μοντέλο SPH (Smoothed Particle Hydrodynamics) για να προσομοιώσει την κίνηση ενός ρευστού, σε animation εφαρμογές. Το πρόγραμμα υπολογίζει με κανόνες φυσικής τις κινήσεις κάθε σωματιδίου του ρευστού, τα οποία αλληλεπιδρούν μεταξύ τους. Είσοδος του προγράμματος είναι 500 frames με 500.000 σωματίδια.

7. **Freqmine**

   Όπως έχει ήδη αναφερθεί σε προυηγούμενα benchmarks, το data analysis στη σύγχρονη εποχή είναι θέμα μείζονος σημασίας. Πιο συγκεκριμένα, η ανάλυση δεδομένων για την εύρεση μοτίβων (pattern analysis), έχει απασχολήσει ευρύ επιστημονικό κύκλο, καθώς οι εφαρμογές του βρίσκουν θέση σε πολλούς διαφορετικούς τομείς, όπως το computer security, το ηλεκτρονικό εμπόριο ή την υπολογιστική βιολογία. Το Freqmine benchmark είναι μια τέτοια data mining εφαρμογή, η οποία ψάχνει να βρει μοτίβα σε μια βάση δεδομένων, όπου αποθηκεύει τα δεδομένα χρησιμοποιώντας ένα FP tree (Frequent-Pattern tree). Είσοδος του προγράμματος είναι μια δομή 250.000 HTML σελίδων, διασυνδεδεμένων μεταξύ τους.

8. **Raytrace**

   Το Raytrace benchmark είναι μια εφαρμογή βασισμένη στην τεχνική ray tracing. Η τεχνική αυτή δημιουργεί μια οπτικά ρεαλιστική εικόνα, ακολουθώντας την πορεία της δέσμης φωτός μέσα σε μία σκηνή. Πρόκειται για τεχνική που χρησιμοποιείται για rendering σκοπούς, η οποία έχει το πλεονέκτημα απέναντι σε άλλες αντίστοιχες τεχνικές, να δημιουργεί μια πολύ ρεαλιστική εικόνα σε ό,τι έχει να κάνει με το φως, προσθέτοντας βέβαια το κόστος της πολύπλοκης και απαιτητικής σε πόρους υλοποίησης. Τα σημεία στα οποία ξεχωρίζει η ποιότητα της εικόνας είναι στις σκιές, τους καθρεπτισμούς και τις διαθλάσεις φωτός, οι οποίες είναι πολύ δύσκολο να κατασκευαστούν ορθά με προγραμματιστικές τεχνικές. Χρησιμοποιείται στις περισσότερες κάρτες γραφικών για βέλτιστα αποτελέσματα σε gaming και video editing. Είσοδος του προγράμματος είναι 200 frames, 1.920 x 1.080 pixels στις οποίες πραγματοποιεί rendering.

9. **Streamcluster**

   Το Streamcluster benchmark είναι ένα πρόγραμμα το οποίο υπολογίζει μια εκτίμηση του βέλτιστου clustering για ένα stream από data points που δέχεται ως είσοδο. Πιο συγκεκριμένα, για ένα stream από data points υπολογίζει έναν προκαθορισμένο αριθμό από medians, ώστε κάθε point να ανατίθεται στο πιο κοντινό κέντρο. Το stream clustering είναι μια συνηθισμένη τεχνική, που εξυπηρετεί στην οργάνωση των δεδομένων εισόδου σε πραγματικό χρόνο, και χρησιμοποιείται κυρίως σε περιπτώσεις intrusion detection, pattern recognition και data mining. Είσοδος του προγράμματος είναι ένα stream από 1.000.000 points, με 128 διαστάσεις, και υπολογίζονται 10-20 κεντρικά points.

10. **Swaptions**

    Το Swaptions benchmark είναι ένα computational finance πρόγραμμα, στο οποίο δίνοντας είσοδο ένα portfolio με derivatives, χρησιμοποιεί το Heath-Jarrow-Morton (HJM) framework ώστε να αξιολογήσει τα δοσμένα swap options. Το HJM framework αξιολογεί το ενδιαφέρον και την εξέλιξη της τιμής βασιζόμενο στο risk και asset liability management. Επιπλέον, το benchmark χρησιμοποιεί το μοντέλο Μόντε-Κάρλο, προκειμένου να πραγματοποιήσει πολλαπλές προσομοιώσεις για τη ζητούμενη αξιολόγηση. Είσοδος του προγράμματος είναι 128 swaptions για τα οποία πραγματοποιεί 1.000.000 προσομοιώσεις.

Κεφάλαιο 4

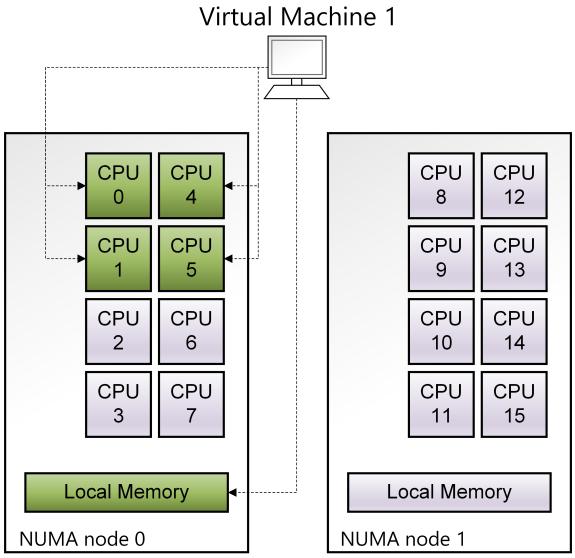
Αποτελέσματα προσομοιώσεων

Στο κεφάλαιο που ακολουθεί, αφού δείξουμε ακριβώς την τοπολογία και την αντιστοίχιση των πόρων της εικονικής μηχανής με το φυσικό μηχάνημα, παρουσιάζουμε τα αποτελέσματα που λάβαμε απ΄ την εκτέλεση των μετροπρογραμμάτων. Για την καλύτερη κατανόηση και παρουσίαση των αποτελεσμάτων, όπου κρίθηκε απαραίτητο, κατασκευάστηκαν κατάλληλα διαγράμματα ή πίνακες που συμπληρώνουν την εν λόγω παρουσίαση.

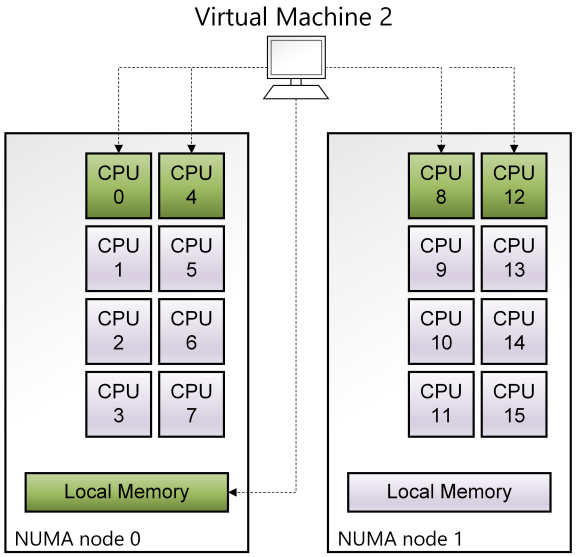
4.1 Αρχιτεκτονική Εικονικών Μηχανών

Στην παρακάτω εικόνα 4.1, φαίνεται πώς έγινε η αντιστοίχηση εικονικών και φυσικών επεξεργαστών μεταξύ guest και host, καθώς και η μνήμη που δεσμεύτηκε σε κάθε περίπτωση, για κάθε μία απ΄ τις τρεις εικονικές μηχανές. Η κάθε εικονική μηχανή αποτελείται από 4 CPUs, 64GB RAM και 30GB δίσκο. Στην πρώτη περίπτωση, τα CPUs του VM αντιστοιχήθηκαν σε επεξεργαστές που βρίσκονται στο ίδιο NUMA node, και η μνήμη που χρησιμοποιήθηκε ήταν μέρος της τοπικής μνήμης του ίδιου node. Κατόπιν, κρατώντας τη μνήμη σταθερή, μετακινήθηκαν τα μισά CPUs σε διπλανό node. Τέλος, για το τρίτο σενάριο, τα CPUs βρίσκονται στο ίδιο node και η μνήμη σε διαφορετικό. Έτσι το πρώτο σενάριο είναι θεωρητικά το ιδανικό, αφού παρουσιάζει το μικρότερο latency και το μεγαλύτερο bandwidth, το δεύτερο είναι χειρότερο, αφού μόνο 2 απ΄ τους 4 πυρήνες εκμεταλλεύονται τα βελτιωμένα latency και bandwidth, ενώ το τρίτο είναι το χειρότερο δυνατό, καθώς αντιμετωπίζουν όλα τα cores το remote access penalty. Αναλυτικές λεπτομέρειες και παραδείγματα configuration αρχείων libvirt που χρησιμοποιήθηκαν για το σκοπό αυτό, βρίσκονται στο παράρτημα B΄

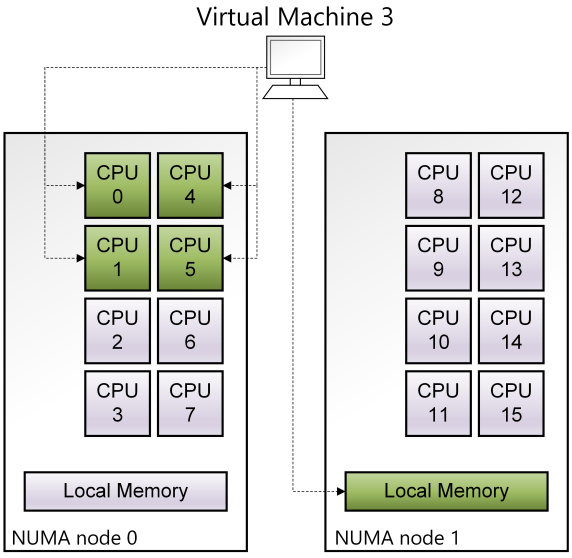
Στην υπόλοιπη εργασία, όπου αναγράφεται Virtual Machine 1 ή VM1, πρόκειται για την εικονική μηχανή που παρουσιάζεται στο 4.1α΄, όπου αναγράφεται Virtual Machine 2 ή VM2, πρόκειται για την εικονική μηχανή που παρουσιάζεται στο 4.1β΄, και τέλος όπου αναγράφεται Virtual Machine 3 ή VM3, πρόκειται για την εικονική μηχανή που παρουσιάζεται στο 4.1γ΄.



(α′) Virtual Machine 1
Cores και Memory στο ίδιο node



(β′) Virtual Machine 2
Cores και Memory μοιρασμένα

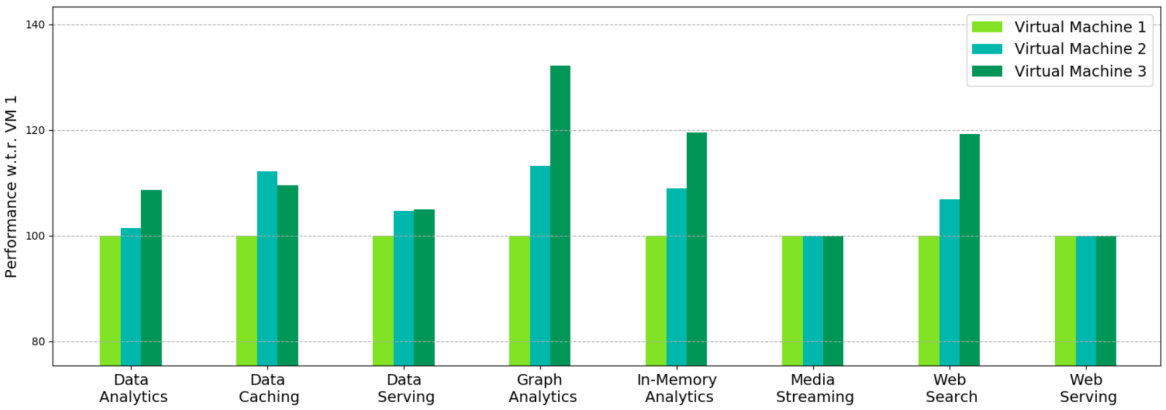


(γ′) Virtual Machine 3
Cores και Memory σε διαφορετικό node

Σχήμα 4.1: Τα τρία VMs που κατασκευάστηκαν

4.2 Αποτελέσματα Cloudsuite

Στην υποενότητα αυτή παρουσιάζονται τα αποτελέσματα που λάβαμε μετά την εκτέλεση των Cloudsuite benchmarks.



Σχήμα 4.2: Κανονικοποιημένα αποτελέσματα Cloudsuite

Στο παραπάνω διάγραμμα 4.2 παρουσιάζονται τα αποτελέσματα των μετρήσεων κανονικοποιημένα ως προς τα αποτελέσματα του VM 1. Τα αποτελέσματα του VM 1 θεωρήθηκαν τα βέλτιστα και κάθε φορά αντιστοιχήθηκαν στην τιμή 100, ενώ για όλα τα υπόλοιπα υπολογίστηκε το ποσοστό απόκλισης από την τιμή που βρέθηκε στο VM 1, και προστέθηκε στο 100, εάν πρόκειται για χειρότερη απόδοση, ή αφαιρέθηκε αν πρόκειται για καλύτερη. Έτσι, όσο πιο πάνω απ΄ το 100 βρίσκεται ένα διάγραμμα, τόσο χειρότερη απόδοση είχε. Να αναφέρουμε στο σημείο αυτό, ότι σε περίπτωση που είχαμε αποτελέσματα για παραπάνω από μια περιπτώσεις εκτέλεσης (π.χ. για διάφορες τιμές client στο Web Search benchmark), επιλέχθηκε ο μέσος όρος όλων των εκτελέσεων. Η διαδικασία της κανονικοποίησης είναι χρήσιμη καθώς η απόδοση δεν είχε παντού ίδια μονάδα (για παράδειγμα ms, sec, operations, operations/sec), και έτσι μας δίνεται η δυνατότητα να δούμε μια συνολική εικόνα για το πώς συμπεριφέρθηκαν σε γενικές γραμμές οι προσομοιώσεις.

Παρατηρώντας με μια πρώτη ματιά το διάγραμμα, γίνεται αντιληπτό, πως σχεδόν σε όλες τις περιπτώσεις (με εξαίρεση τα Media Streaming και Web Serving όπου δεν φαίνεται κάποια αισθητή μεταβολή της απόδοσης μεταξύ των τριών διαφορετικών εκτελέσεων), η απόδοση φαίνεται πως χειροτερεύει στα VMs 2,3 σε σχέση με το VM 1, όπως ακριβώς δηλαδή περιμέναμε. Έτσι, εξ΄ αρχής είναι ασφαλές να πούμε, πως σίγουρα η τοπολογία των επεξεργαστών και μνήμης ανάμεσα σε φυσική και εικονική μηχανή, έχει μεγάλες επιπτώσεις στην απόδοση, και πως όντως η περίπτωση εκτέλεσης του Virtual Machine 1 φαίνεται πως είναι η καλύτερη ανάμεσα στις 3.

Ακολουθεί περιγραφή των αποτελεσμάτων για κάθε benchmark ξεχωριστά, όπου και θα αποπειραθούμε να εξηγήσουμε εάν και γιατί υπάρχει μεταβολή στην απόδοση, και ποιο είναι τελικά το ιδανικό σενάριο εκτέλεσης της cloud εφαρμογής σε ό,τι αφορά την NUMA αρχιτεκτονική.

4.2.1 Data Analytics

Το Data Analytics benchmark πραγματοποιεί ανάλυση δεδομένων με χρήση του map-reduce σε ένα wikimedia dataset. Το αποτέλεσμα του benchmark μετά την εκτέλεσή του είναι ο χρόνος ολοκλήρωσης. Εκτελέσαμε το benchmark την πρώτη φορά με 2 slaves και έπειτα με 4, ώστε να έχουμε μια καλύτερη εικόνα. Οι slaves ουσιαστικά είναι οι εφαρμογές που εκτελούν τους mappers και reducers, και συνεργάζονται με το κυρίως πρόγραμμα που οργανώνει την εκτέλεση. Τα αποτελέσματα που λάβαμε παρουσιάζονται στον παρακάτω πίνακα 4.1.

Number of Slaves	**VM1**	**VM2**	**VM3**
2	56m10sec	57m15sec	1h2m5sec
4	57m40sec	58m10sec	1h2m34sec

Πίνακας 4.1: Αποτελέσματα εκτέλεσης Data Analytics benchmark

Βλέποντας τον παραπάνω πίνακα, παρατηρούμε πως ο χρόνος εκτέλεσης αυξάνεται όσο το setup των εικονικών μηχανών γίνεται όλο και λιγότερο αποδοτικό. Ελέγχοντας την κατάσταση των εικονικών μηχανών κατά τη διάρκεια της εκτέλεσης του benchmark, παρατηρήθηκε χρήση των CPU που άγγιζε το 100%, αλλά και δεσμευμένη μνήμη περίπου 3 έως 5GB. Έτσι συμπεραίνουμε πως πρόκειται για μια CPU bound εφαρμογή, η οποία κάνει και αρκετή χρήση της μνήμης προκειμένου να αποθηκεύει δεδομένα, όπως π.χ. τα αποτελέσματα που παράγει η διαδικασία του mapping προκειμένου να αθροιστούν στη συνέχεια με το reducing. Έτσι, δικαιολογείται και η μικρή, αλλά αισθητή μεταβολή στην απόδοση της εφαρμογής. Ιδανική περίπτωση επομένως εκτέλεσης της εφαρμογής είναι η πρώτη περίπτωση όπου τα CPUs και η μνήμη βρίσκονται στο ίδιο NUMA node.

4.2.2 Data Caching

Το Data Caching benchmark χρησιμοποιεί το Memcached εργαλείο για την αποθήκευση δεδομένων από ένα Twitter dataset. Με τη χρήση του Memcached μπορούμε να αποθηκεύουμε πληροφορίες, όπως απαντήσεις ερωτημάτων σε βάσεις δεδομένων, οι οποίες κανονικά χρειάζονται παραπάνω χρόνο για να εξαχθούν, κάνοντας χρήση της RAM. Στο benchmark μπορεί ο χρήστης να ορίσει τον αριθμό των servers που εξυπηρετούν τους clients, τη συνολική μνήμη που δεσμεύουν για την αποθήκευση των δεδομένων, αλλά και τον αριθμό των clients που στέλνουν τα ερωτήματα.

Στην παρούσα εργασία επιλέξαμε να χρησιμοποιήσουμε 4 servers, που δεσμεύουν συνολικά 4GB και 8GB μνήμης, και τους στέλνουν ερωτήματα 4 και 8 clients ταυτόχρονα. Έτσι εκτελέσαμε το benchmark από 4 φορές για κάθε διαφορετικό συνδυασμό, στο κάθε setup εικονικών μηχανών.

Τα αποτελέσματα που παράγει το benchmark μετά την εκτέλεση είναι δύο. Το πρώτο αφορά τον αριθμό των requests που εξυπηρετήθηκαν, ανά δευτερόλεπτο, ενώ το δεύτερο αφορά

το latency, δηλαδή το χρόνο που έκανε ένα request να εξυπηρετηθεί. Προφανώς, το βέλτιστο αποτέλεσμα είναι όσο το δυνατόν περισσότερα requests ανά δευτερόλεπτο, με το χαμηλότερο δυνατό latency. Βέβαια επειδή αυτό είναι πρακτικά αδύνατον να συμβεί, υπάρχουν κανόνες για το quality of service και το το μέγιστο επιτρεπτό latency που μπορεί να φτάσει, οπότε και το benchmark προσφέρει tunning εργαλεία, ώστε να μειώσει τον αριθμό των requests που εξυπηρετούνται, για να μειωθεί και το latency. Στην παρούσα εργασία, αυτό αγνοήθηκε, καθώς δεν μας ενδιέφερε να βελτιστοποιήσουμε την εμπειρία του client, αλλά να εξαντλήσουμε όσο το δυνατόν περισσότερο τους πόρους του συστήματος. Έτσι, εξετάζεται το πόσα requests κατάφεραν να εξυπηρετήσουν οι web servers ανά δευτερόλεπτο, σε συνδυασμό και με το χρόνο που έλαβε η ολοκλήρωση.

Επιπλέον, το benchmark δεν έχει από μόνο του συνθήκη τερματισμού, δηλαδή συνεχίζει και στέλνει requests προς τους servers μέχρι ο χρήστης να το διακόψει, και εμφανίζει ομαδοποιημένα τα αποτελέσματα στην έξοδο ανά δευτερόλεπτο. Στα πλαίσια της εργασίας, η δειγματοληψία έγινε θέτοντας ίδιο χρονικό περιορισμό εκτέλεσης σε όλα τα σενάρια. Στη συνέχεια, κατασκευάστηκαν κατάλληλα βοηθητικά εργαλεία (scripts) ώστε να γίνει parsing των αποτελεσμάτων, και να υπολογιστούν μέσοι όροι.

Ακολουθεί ο πίνακας 4.2 στον οποίο φαίνονται οι μέσοι όροι των requests και latency για κάθε σενάριο εκτέλεσης.

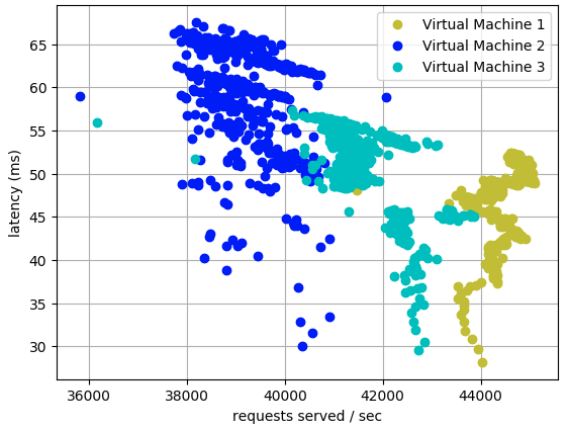
Server's Memory	*Number of Clients*	*Evaluation Metric*	**VM1**	**VM2**	**VM3**
4GB	4	**Average Requests/sec**	44.560	39.258	41.642
		Average Latency	47,85	58,47	50,72
	8	**Average Requests/sec**	42.159	39.165	41.373
		Average Latency	51,10	64,81	51,79
8GB	4	**Average Requests/sec**	43.827	39.687	42.277
		Average Latency	49,15	53,46	50,30
	8	**Average Requests/sec**	43.211	39.329	41.559
		Average Latency	49,20	57,31	51,60

Πίνακας 4.2: Αποτελέσματα εκτέλεσης Data Caching benchmark

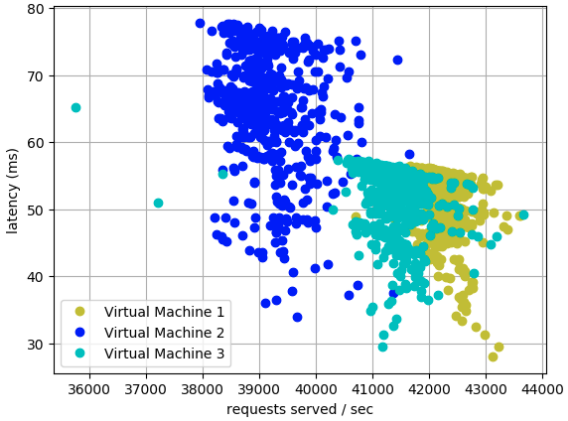
Παρατηρώντας τον πίνακα, βλέπουμε πως η απόδοση χειροτερεύει όταν οι clients αυξάνονται, και πως υπάρχει μια βελτίωση, είτε πιο μικρή είτε πιο μεγάλη, καθώς διπλασιάζουμε τη μνήμη που παραχωρείται στους servers. Τα συμπεράσματα αυτά αφορούν και τα requests και το latency, και φαίνεται πως ισχύουν και για τις 3 διαφορετικές εικονικές μηχανές. Συγκρίνοντας όμως τα αποτελέσματα μεταξύ τους, παρατηρούμε πως τα αποτελέσματα του VM1 είναι τα βέλτιστα, με μικρή όμως διαφορά από τα αποτελέσματα του VM3, ενώ το δεύτερο VM φαίνεται πως είχε συνολικά τη χειρότερη επίδοση.

Το ίδιο φαινόμενο παρατηρούμε και στην εικόνα 4.3, όπου έχουμε οπτικοποιήσει τα αποτελέσματα απ΄ τα οποία πάρθηκαν οι μέσοι όροι, και δείχνουμε τον αριθμό των requests ανά δευτερόλεπτο, σε σχέση με το average latency που έκαναν να εξυπηρετηθούν.

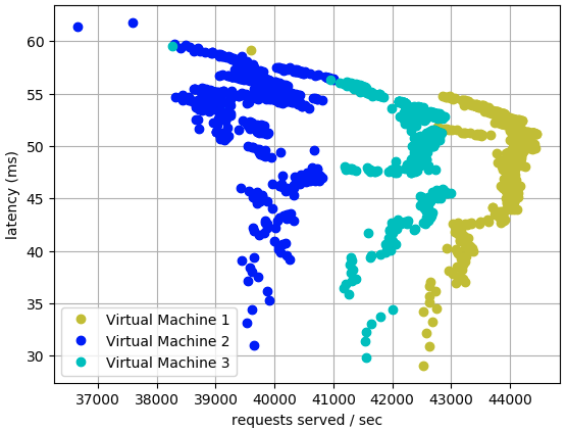
Καταλαβαίνει κανείς ότι, επειδή θέλουμε το μικρότερο latency (άξονας y) και τα περισσότερα requests (άξονας x), τα βέλτιστα αποτελέσματα είναι αυτά που βρίσκονται πιο κοντά στην κάτω δεξιά γωνία του γραφήματος. Έτσι γίνεται αντιληπτό το ίδιο φαινόμενο και στις 4 περιπτώσεις, δηλαδή ότι η απόδοση του VM1 είναι η καλύτερη με μικρή διαφορά απ΄ αυτή του VM3, και σαφώς καλύτερη απ΄ αυτή του VM2.



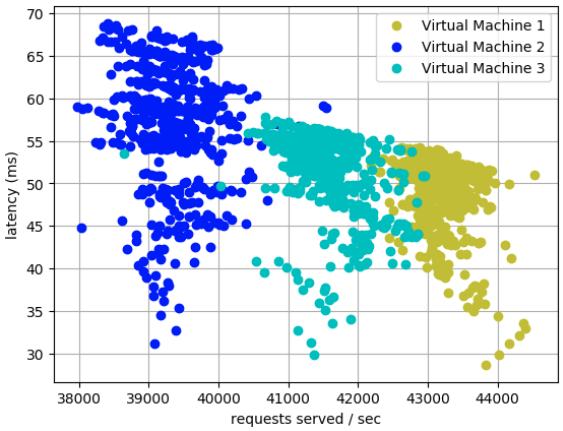
(α΄) 4 servers with 4GB dedicated memory 4 Clients



(β΄) 4 servers with 4GB dedicated memory 8 Clients



(γ΄) 4 servers with 8GB dedicated memory 4 Clients



(δ΄) 4 servers with 8GB dedicated memory 8 Clients

Σχήμα 4.3: Αποτελέσματα εκέλεσης Data Caching benchmark: Κάθε σημείο της γραφικής παράστασης απεικονίζει το αποτέλεσμα εξόδου του μετροπρογράμματος, τα οποία μας δίνονται χωρισμένα ανά δευτερόλεπτο.

Το φαινόμενο αυτό, αν και αρχικά φαίνεται «περίεργο» αφού έχουμε ορίσει «καλύτερες» περιπτώσεις εκτέλεσης με τη σειρά VM1, VM2 και VM3, εδώ βλέπουμε πως δεν ισχύει κάτι τέτοιο. Όμως η συμπεριφορά που παρατηρούμε είναι λογική και εξηγείται αν σκεφτεί κανείς τις διαφορές στις τρεις εικονικές μηχανές. Ενώ η πρόσβαση στη μνήμη γίνεται όλο και λιγότερο αποδοτική από εικονική μηχανή σε εικονική μηχανή, η περίπτωση του VM2 έχει μια σημαντική διαφορά με τις άλλες δύο, και αυτή αφορά τη μνήμη cache του επεξεργαστή. Χωρίζοντας τα φυσικά CPUs του επεξεργαστή σε δύο διαφορετικά nodes, απ΄ τη μια έχουμε διπλασιάσει την

LLC (Last Level Cache), η οποία είναι κοινή για τα CPUs στο ίδιο node, απ΄ την άλλη όμως, έχουμε προσθέσει το overhead του συγχρονισμού (Coherency) μεταξύ των δύο LLC, το οποίο πρακτικά σημαίνει ότι σε μια εφαρμογή που απαιτεί συνεχή επικοινωνία μεταξύ των CPUs, η απόδοση θα μειωθεί. Στην περίπτωσή μας, κάτι τέτοιο μπορεί να συμβεί όταν πολλοί clients ζητούν να διαβάσουν τα ίδια δεδομένα, τα οποία μπορεί να βρίσκονται στην cache αφού είχαν εξυπηρετηθεί από διαφορετικούς επεξεργαστές, και έτσι απαιτείται συγχρονισμός στην cache ώστε και οι υπόλοιποι επεξεργαστές να τα βλέπουν.

Έτσι συμπεραίνουμε ότι στο Data Caching benchmark η μνήμη cache έχει μεγάλη σημασία, καθώς χρειάζεται να βρίσκει τα δεδομένα απ΄ ευθείας χωρίς να χρειαστεί να τα φέρει απ΄ τη μνήμη, κάτι το οποίο εξηγεί τη χειρότερη απόδοση του VM2. Φυσικά και γίνεται πρόσβαση στη μνήμη, γι΄ αυτό και το ιδανικό σενάριο εκτέλεσης της εφαρμογής είναι αυτό του VM1, αλλά το Data Caching benchmark είναι ένα παράδειγμα που καταρρίπτει τον «κανόνα» για το βέλτιστο και χείριστο σενάριο που θέσαμε στην αρχή.

4.2.3 Data Serving

Το Data Serving benchmark χρησιμοποιεί το YCSB εργαλείο της Yahoo! προκειμένου να κάνει populate και stress την βάση δεδομένων Apache's Cassandra. Το benchmark δίνει τη δυνατότητα να επιλέξουμε τον αριθμό των servers που θα τρέχουν τη βάση, καθώς και τον αριθμό των operations που θα πραγματοποιήσει το YCSB για την προσομοίωση. Τα operations που πραγματοποιούνται χωρίζονται σε UPDATE, όπου ζητείται στη βάση να τροποποιήσει δεδομένα, READ, για να διαβάσει δεδομένα και WRITE, για την προσθήκη δεδομένων.

Στα πλαίσια της παρούσας εργασίας, επιλέχθηκε να χρησιμοποιηθούν 4 servers που τρέχουν τη βάση δεδομένων, και το YCSB τους στρεσάρει με 1.000, 10.000 και 1.000.000 operations. Συνολικά λοιπόν, στο κάθε setup εικονικών μηχανών εκτελέστηκαν από 3 φορές, μία για το κάθε σενάριο.

Στον παρακάτω πίνακα 4.3 παρουσιάζονται τα αποτελέσματα εκτέλεσης. Σημειώνεται εδώ ότι τα αποτελέσματα του benchmark ήταν πολύ πιο αναλυτικά, και περιείχαν πληροφορία για κάθε τύπο operation ξεχωριστά, από τα οποία επιλέξαμε να χρησιμοποιήσουμε τα πιο βασικά, τα οποία είναι ο συνολικός χρόνος εκτέλεσης (runtime), και το throughput, δηλαδή operations / sec. Προφανώς ισχύει η εξής ισότητα,

$$Throughput = \frac{Operations}{Runtime} \tag{4.1}$$

η οποία επαληθεύεται στα παρακάτω αποτελέσματα.

Παρατηρώντας τα αποτελέσματα του πίνακα 4.3, γίνεται αντιληπτό πως η απόδοση της μνήμης αποτέλεσε σημαντικό παράγοντα για τις επιδόσεις του benchmark. Συγκεκριμένα, σε όλες τις περιπτώσεις, παρατηρήθηκε μείωση στο throughput και αύξηση στο runtime των εικονικών μηχανών 2 και 3, σε σχέση με την 1. Το αποτέλεσμα είναι αναμενόμενο, καθώς το benchmark δεσμεύει μνήμη απ΄ τον υπολογιστή προκειμένου να επιτελέσει τις λειτουργίες

		VM1	VM2	VM3
1K OPS	**Runtime (ms)**	7.372	7.821	8.145
	Throughput (ops/sec)	135,65	127,86	122,77
10K OPS	**Runtime (ms)**	39.528	40.408	43.339
	Throughput (ops/sec)	252,98	247,475	230,74
1M OPS	**Runtime (ms)**	1.825.962	1.933.893	1.865.239
	Throughput (ops/sec)	547,656	517,091	536,12

Πίνακας 4.3: Αποτελέσματα εκτέλεσης Data Serving benchmark

γρηγορότερα, όπως παρατηρήθηκε και με κατάλληλα εργαλεία την ώρα της εκτέλεσης. Προφανώς, γνωρίζοντας ότι το input που βάζει στη βάση το YCSB είναι 10GB, και συγκρίνοντας με το commited memory που ήταν λιγότερο, καταλαβαίνουμε ότι δεν εκτελέστηκε εντελώς in-memory το benchmark, γι΄ αυτό και οι χρόνοι ολοκλήρωσης operations είναι κατανεμημένοι σε ένα ευρύ φάσμα. Βέλτιστο σενάριο και στην περίπτωση αυτή είναι αυτό του VM1, με καλύτερη επίδοση που κυμαίνεται από +5,7% έως +10%.

4.2.4 Graph Analytics

Στο Graph Analytics benchmark, εκτελείται ο αλγόριθμος PageRank σε ένα Twitter dataset. Το benchmark χρησιμοποιεί το Apache's Spark framework και δίνει τη δυνατότητα στον χρήστη να επιλέξει πόση μνήμη θα αξιοποιήσει το Spark. Το dataset που περιλαμβάνει τους Twitter users (για την ακρίβεια περιλαμβάνει influences των χρηστών, ανάλογα με τα follows του καθενός, όπου πάνω σε αυτά τα δεδομένα εφαρμόζεται PageRank), είναι περίπου 25GB. Για το λόγο αυτό επιλέχθηκε να τρέξει δύο φορές το benchmark στα VMs, μία φορά με 5GB dedicated memory στο Spark, και μία με 50GB. Έτσι στη δεύτερη εκτέλεση είμαστε βέβαιοι ότι το πρόγραμμα θα τρέξει εντελώς in-memory, οπότε και αναμένουμε σημαντικές μεταβολές στην απόδοση των εκτελέσεων.

Στον παρακάτω πίνακα 4.4 παρουσιάζονται τα αποτελέσματα που λάβαμε. Το αποτέλεσμα της προσομοίωσης είναι ο χρόνος ολοκλήρωσης, οπότε με βάση αυτόν θα γίνει και η σύγκριση της επίδοσης.

Dedicated Memory	**Runtime (min)**		
	VM1	**VM2**	**VM3**
5 GB	24,30	28,52	40,43
50 GB	17,34	19,43	21,10

Πίνακας 4.4: Αποτελέσματα εκτέλεσης Graph Analytics benchmark

Παρατηρώντας τα αποτελέσματα του παραπάνω πίνακα, αμέσως καταλαβαίνουμε το μέγεθος της επίδρασης της μνήμης για την εν λόγω εφαρμογή, τόσο όταν εκτελείται όλη στη μνήμη σε σχέση με τη χρήση και disk I/O λειτουργιών, όσο και όταν αλλάζει η απόδοση στην πρόσβαση της μνήμης. Βλέπουμε σταδιακή μείωση της απόδοσης, δηλαδή αύξηση του χρόνου

ολοκλήρωσης, από το setup του VM1, έως και αυτό του VM3. Επιπλέον φαίνεται η σημασία της μνήμης λόγω της ραγδαίας βελτίωσης στην επίδοση όταν όλο το benchmark έτρεξε in-memory (50GB), όπου παρατηρείται βελτίωση της απόδοσης που αγγίζει το 80-100%. Η επίδραση της μνήμης σε τόσο μεγάλο βαθμό οφείλεται στο ότι για την εκτέλεση του αλγορίθμου PageRank χρειάζεται να μοντελοποιηθεί το πρόβλημα με δομές δεδομένων, όπως συνδεδεμένες λίστες για αναπαράσταση γράφων, οι οποίες μπορούν να αποθηκεύονται στη μνήμη, όταν η μνήμη επαρκεί, και έτσι, να επιταχύνεται η πρόσβαση σε αυτές κατά την εκτέλεση του προγράμματος. Βέλτιστο σενάριο και σε αυτή την περίπτωση είναι αυτό του VM1, και μάλιστα με μεγάλη διαφορά, που ξεπερνά το 30%.

4.2.5 In-Memory Analytics

Το In-Memory Analytics benchmark χρησιμοποιεί το Apache's Spark framework, και τη βιβλιοθήκη MLlib, προκειμένου να αναπτύξει ένα νευρωνικό δίκτυο, του οποίου ο σκοπός είναι να βαθμολογήσει ταινίες ανάλογα με τις προτιμήσεις του χρήστη. Έξοδος του προγράμματος είναι η λίστα με τις ταινίες που προτείνει, καθώς και ο χρόνος ολοκλήρωσης της εκτέλεσης. Έτσι, η μετρική που χρησιμοποιήθηκε για την σύγκριση των αποτελεσμάτων μεταξύ των διαφορετικών εκτελέσεων είναι ο χρόνος ολοκλήρωσης.

Στον παρακάτω πίνακα 4.5 παρουσιάζονται τα αποτελέσματα της εκτέλεσης.

	VM1	**VM2**	**VM3**
Execution Time (ms)	49.147	53.997	61.111

Πίνακας 4.5: Αποτελέσματα εκτέλεσης In-Memory Analytics benchmark

Το συγκεκριμένο benchmark δεν είχε tuning επιλογές. Η διαδικασία εκτέλεσης ήταν σαφής και ξεκάθαρη, χρειάστηκε απλά η επανάληψη στα τρία διαφορετικά σενάρια. Ελέγχοντας τους πόρους που δέσμευσε η προσομοίωση κατά τη διάρκεια της εκτέλεσής της, από την εκάστοτε εικονική μηχανή, παρατηρήθηκε συνεχής χρήση των CPU, και δεσμευμένη μνήμη περίπου στα 6GB, αφού, όπως μαρτυρά και ο τίτλος του benchmark, τρέχει εξ ολοκλήρου in-memory. Τα αποτελέσματα του παραπάνω πίνακα επιβεβαιώνουν το ζητούμενο του benchmark, το οποίο και είναι η μέτρηση της απόδοσης της μνήμης, και δείχνουν πως υπήρξε μείωση της απόδοσης σε κάθε εκτέλεση μετά απ΄ αυτή του VM1, με αυτή του VM3 να είναι η χειρότερη κατά 20% απ΄ τη βέλτιστη του VM1. Γενικότερα, σε αυτό το benchmark όπως και το προηγούμενο (Graph Analytics), η απόδοση της μνήμης έχει τρομερή σημασία, καθώς εκτελούνται αλγόριθμοι σε μεγάλο όγκο δεδομένων, για τους οποίους έχουμε σχεδιάσει συστήματα ώστε να εκμεταλλεύονται τη μνήμη του υπολογιστή, να αποφευχθεί η πρόσβαση στο δίσκο και κατά συνέπεια να έχουμε καλύτερες επιδόσεις. Οπότε, είναι απολύτως λογικό να παρατηρούμε τη μεγάλη αυτή διαφορά στην απόδοση. Τα benchmarks αυτά επιβεβαιώνουν τους ισχυρισμούς που κάναμε σε αυτή την εργασία, και δείχνουν το πόσο

σημαντικό ρόλο έχει η βέλτιστη και πιο αποδοτική αντιστοίχηση φυσικών πόρων στις εικονικές μηχανές, που πρέπει να εξετάζεται σε cloud infrastructures.

4.2.6 Media Streaming

Το Media Streaming benchmark, χρησιμοποιεί το httperf εργαλείο, για να στείλει requests σε έναν nginx web server, που έχει αποθηκευμένα video διαφόρων χαρακτηριστικών και τα προσφέρει στον χρήστη μέσω streaming. Το benchmark δεν προσφέρει tuning λειτουργίες, οπότε εκτελέστηκε τρεις φορές, από μία σε κάθε σενάριο εκτέλεσης. Το benchmark εκτελείται αυτόματα 4 φορές, «ζορίζοντας» κάθε φορά περισσότερο το σύστημα που το φιλοξενεί. Τα αποτελέσματα που παράγει στην έξοδό του για κάθε εκτέλεση, είναι ο αριθμός των συνδέσεων στον web server ανά δευτερόλεπτο (conn/sec), ο χρόνος που χρειάστηκε για να πραγματοποιηθεί η κάθε σύνδεση, κατά μέσο όρο (conn. time (ms)), καθώς και η πιο σημαντική μετρική, που είναι ο αριθμός των απαντήσεων ανά δευτερόλεπτο, δηλαδή ο μέγιστος αριθμός χρηστών μπορούν να εξυπηρετηθούν ταυτόχρονα, σε κάθε δευτερόλεπτο (replies/sec).

Στον παρακάτω πίνακα 4.6 ακολουθούν τα αποτελέσματα της εκτέλεσης του benchmark.

		VM1	**VM2**	**VM3**
First Execution	*conn/sec:*	1,5	1,4	1,5
	conn. time (ms):	2,5	1,1	3,3
	replies/sec:	25,6	24,1	25,7
Second Execution	*conn/sec:*	1,5	1,4	1,5
	conn. time (ms):	2,1	4,7	125,4
	replies/sec:	25,6	24,1	25,1
Third Execution	*conn/sec:*	1,5	1,4	1,5
	conn. time (ms):	3,4	2,1	94,4
	replies/sec:	25,6	24,1	25,2
Fourth Execution	*conn/sec:*	1,5	1,4	1,5
	conn. time (ms):	5,4	1,8	84,5
	replies/sec:	25,6	24,1	25,2

Πίνακας 4.6: Αποτελέσματα εκτέλεσης Media Streaming benchmark

Παρατηρώντας τον παραπάνω πίνακα, μπορούμε να δούμε πως δεν υπάρχουν ιδιαίτερες διαφορές στην απόδοση της εφαρμογής από το ένα σενάριο εκτέλεσης στο άλλο. Ελέγχοντας το ποσοστό αξιοποίησης των πόρων της εικονικής μηχανής κατά τη διάρκεια εκτέλεσης του benchmark, παρατηρήθηκε πολύ μικρό ποσοστό χρήσης τόσο σε CPU, όσο και σε memory. Έτσι, συμπεραίνουμε ότι η παρούσα προσομοίωση δεν κατάφερε να εξασθενήσει αρκετά τα μηχανήματά μας, γι΄ αυτό και δεν παρατηρούμε επαρκείς διαφορές στην απόδοση ώστε να ξεχωρίσουμε κάποιο σενάριο εκτέλεσης ως βέλτιστο ή χείριστο. Βλέποντας το ζήτημα από θεωρητικό υπόβαθρο, ο nginx web server είναι event-based, δηλαδή για κάθε νέο request που λαμβάνει δεν χρειάζεται να δημιουργεί καινούριο process ή thread, καθώς είναι πολύ καλά

optimized (σε αντίθεση με άλλους web servers όπως ο Apache), οπότε και καταναλώνει ελάχιστη μνήμη στο εκάστοτε μηχάνημα. Άρα, δεν περιμένουμε να δούμε αισθητές διαφορές στην απόδοση του benchmark, καθώς μεταβάλλουμε την αποδοτικότητα της μνήμης.

4.2.7 Web Search

Το Web Search benchmark βασίζεται στο εργαλείο Apache's Solr search engine framework, στο οποίο αποθηκεύεται in-memory ένα 12GB index, και παράλληλα δημιουργούνται clients, οι οποίοι στέλνουν requests και καταγράφουν τον χρόνο που χρειάστηκε να λάβουν απάντηση. Τα αποτελέσματα του benchmark ήταν διάφορες μετρικές σε ένα xml output, απ΄ τα οποία στην εργασία επιλέξαμε να κρατήσουμε τον συνολικό αριθμό των operations που κατάφεραν να εξυπηρετηθούν, τα operations/second, καθώς και τον μέσο όρο του χρόνου απόκρισης του index node στα requests των χρηστών. Το benchmark διαρκεί συνολικά 60 δευτερόλεπτα, και βέλτιστο αποτέλεσμα είναι αυτό που έχει εξυπηρετήσει τα περισσότερα requests με το μικρότερο δυνατό χρόνο απάντησης.

Επιπλέον, δίνεται η επιλογή στον χρήστη να επιλέξει πόσοι clients θα δημιουργηθούν, οι οποίοι θα στέλνουν παράλληλα ερωτήματα. Στα πλαίσια της παρούσας εργασίας, το benchmark εκτελέστηκε τρεις φορές σε κάθε setup, για 50, 100 και 200 clients.

Στον παρακάτω πίνακα 4.7 παρουσιάζονται τα αποτελέσματα που λάβαμε.

		VM1	**VM2**	**VM3**
50 clients	*ops/sec*	25,417	24,833	24,400
	total ops	1.525	1.490	1.464
	avg response time (sec)	0,099	0,128	0,129
100 clients	*ops/sec*	47,700	49,067	49,350
	total ops	2.862	2.944	2.961
	avg response time (sec)	0,153	0,135	0,141
200 clients	*ops/sec*	78,533	73,150	63,450
	total ops	4.712	4.389	3.807
	avg response time (sec)	1,140	1,387	1,910

Πίνακας 4.7: Αποτελέσματα εκτέλεσης Web Search benchmark

Παρατηρώντας τα αποτελέσματα του παραπάνω πίνακα, παρατηρούμε ότι στις δύο πρώτες εκτελέσεις, για 50 και 100 clients, η διαφορά στον αριθμό των operations και τους χρόνους διεκπεραίωσης είναι αμελητέα. Στην τρίτη εκτέλεση όμως, με 200 concurrent clients, καταλαβαίνουμε πως η μνήμη είχε πιο σημαντικό ρόλο στην εκτέλεση, καθώς παρατηρείται αισθητή μείωση του συνολικού αριθμού των requests και αύξηση του latency από το πρώτο σενάριο στα επόμενα δύο. Το αποτέλεσμα αυτό είναι και το πλέον λογικό, καθώς εξ αρχής γνωρίζαμε ότι τα δεδομένα είναι αποθηκευμένα in-memory, οπότε και η πρόσβαση στη μνήμη κατά τη διάρκεια της εκτέλεσης θα είναι σε αυξημένα επίπεδα. Ο λόγος για τον οποίο στις πρώτες δύο εκτελέσεις δεν φαίνεται να υπάρχει ιδιαίτερη διαφορά στην απόδοση είναι πως ο μικρότερος αριθμός των χρηστών, δεν κατάφερε να αξιοποιήσει πλήρως τους πόρους του

συστήματος, με αποτέλεσμα να μη διαφέρουν οι αποδόσεις μεταξύ των εκτελέσεων. Αυτό σημαίνει ότι εάν εκτελούσαμε ξανά το benchmark με περισσότερους από 200 clients αυτή τη φορά στο κάθε setup, αναμένουμε τουλάχιστον την ίδια αν όχι μεγαλύτερη διαφορά στην απόδοση μεταξύ των διαφορετικών σεναρίων. Βέλτιστο σενάριο και σε αυτή την περίπτωση είναι αυτό του VM1, κατά το οποίο παρατηρήθηκε συνολική αύξηση στην απόδοση που φτάνει το 20% σε σχέση με αυτή του VM3, και περίπου 7% σε σχέση με αυτή του VM2.

4.2.8 Web Serving

Το Web Serving benchmark χρησιμοποιεί το Elgg εργαλείο για την προσομοίωση μιας εφαρμογής κοινωνικού δικτύου, καθώς και memcached server, βάση δεδομένων MySQL, ενώ παρέχει και λειτουργικότητα δημιουργίας χρηστών οι οποίοι χρησιμοποιούν τις λειτουργίες της εφαρμογής, όπως η δημιουργία προφίλ, αποστολή αιτήματος φιλίας σε άλλο χρήστη, ή προσθήκη νέου post στον προσωπικό «τοίχο». Το αποτέλεσμα που παίρνουμε στην έξοδο του προγράμματος είναι οι χρόνοι που απαιτήθηκαν προκειμένου να πραγματοποιηθούν οι διαφορετικές ενέργειες που πραγματοποιούν οι χρήστες (μέσος όρος), αλλά και ο συνολικός χρόνος εκτέλεσης του μετροπρογράμματος.

Επιπλέον, δίνεται η δυνατότητα στον χρήστη να επιλέξει τον αριθμό των χρηστών που θα δημιουργήσει η προσομοίωση, οι οποίοι ταυτόχρονα θα χρησιμοποιούν τις λειτουργίες της εφαρμογής. Στα πλαίσια της παρούσας άσκησης, η προσομοίωση εκτελέστηκε τρεις φορές σε κάθε σενάριο εκτέλεσης, για 10, 100 και 1000 χρήστες κάθε φορά. Στα αποτελέσματα που λάβαμε, βρίσκεται ο συνολικός χρόνος εκτέλεσης, ο αριθμός των operations/second που εκτελέστηκαν, καθώς και οι χρόνοι απόκρισης στις λειτουργίες προσπέλασης της εφαρμογής (Browse), στη δημιουργία νέου Post στον προσωπικό «τοίχο» του χρήστη (PostSelfWall), στην αποστολή αιτήματος φιλίας σε έναν χρήστη (AddFriend), αλλά και στην αποσύνδεση απ΄ την εφαρμογή (Logout).

Στον παρακάτω πίνακα 4.8 παρουσιάζονται τα αποτελέσματα που λάβαμε μετά το πέρας της εκτέλεσης του benchmark. Παρατηρώντας τα αποτελέσματα και συγκρίνοντάς τα ανά setup, είναι δύσκολο να αποφανθεί κανείς πού παρουσιάζονται σημαντικές διαφορές στις επιδόσεις, και ποιο σενάριο ανταποκρίθηκε καλύτερα στο benchmark. Και αυτό διότι δεν υπάρχουν ιδιαίτερες διαφορές ή μοτίβα που μπορούν να ερμηνευτούν και να οδηγήσουν σε συμπεράσματα για αυξημένη ή μειωμένη απόδοση μεταξύ των εκτελέσεων. Τόσο οι χρόνοι ολοκλήρωσης, όσο και τα ops/sec, κυμαίνονται σχεδόν στις ίδιες τιμές, ενώ δε για τους χρόνους απόκρισης στις διαφορετικές λειτουργίες, τα ποσά είναι τόσο μικρά (υποεκαντοταπλάσια του ms, με εξαίρεση αυτό του «Browse»), που και κάποιες διαφορές που υπάρχουν, δεν έχουν συνέπεια στις εκτελέσεις, αλλά ούτε και σημασία στην επίδοση ή στην εμπειρία του χρήστη κατά την χρήση της εφαρμογής. Ελέγχοντας τους πόρους των εικονικών μηχανών κατά τη διάρκεια της εκτέλεσης, παρατηρήθηκε μικρή έως μεσαία χρήση των CPU, και αρκετή δεσμευμένη μνήμη (κυρίως από το memcached), περιμένοντας έτσι να δούμε μεγαλύτερες αποκλίσεις στην απόδοση, κάτι το οποίο δεν επαληθεύτηκε. Δεν μπορούμε να

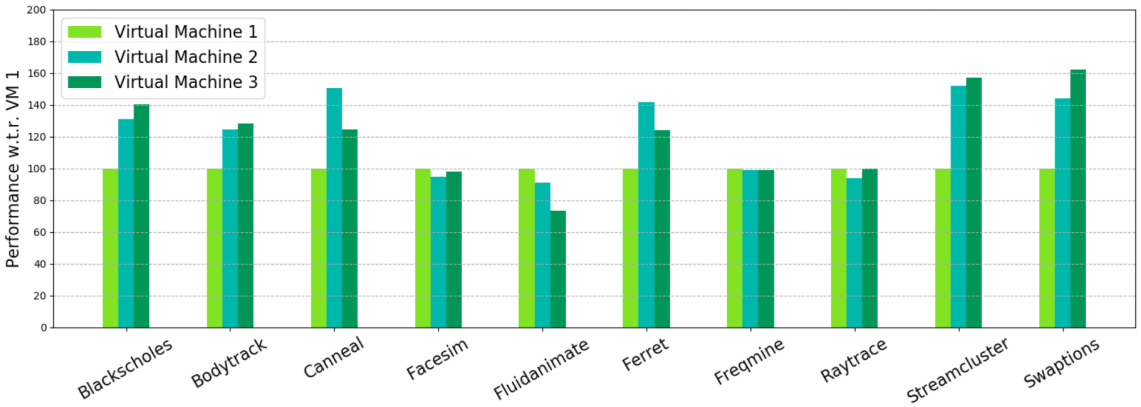
αποφανθούμε λοιπόν στο benchmark αυτό για το βέλτιστο σενάριο εκτέλεσης, ούτε για τη συμπεριφορά που παρουσιάζει καθώς μεταβάλλουμε την αποδοτικότητα της μνήμης.

		VM1	**VM2**	**VM3**
10 Users	*Total Time*	1m21sec	1m23sec	1m24sec
	ops / sec	6,267	5,967	5,933
	Browse (ms)	106,873	122,402	111,011
	PostSelfWall (ms)	0,011	0,010	0,016
	AddFriend (ms)	0,010	0,011	0,012
	Logout (ms)	0,014	0,013	0,011
100 Users	*Total Time*	2m49sec	2m51sec	2m54sec
	ops / sec	59,333	59,267	59,433
	Browse (ms)	95,782	111,515	107,568
	PostSelfWall (ms)	0,009	0,008	0,009
	AddFriend (ms)	0,010	0,007	0,010
	Logout (ms)	0,011	0,009	0,013
1000 Users	*Total Time*	17m50 sec	17m48sec	17m58sec
	ops / sec	594,333	595,333	599,167
	Browse (ms)	252,939	241,103	264,975
	PostSelfWall (ms)	0,017	0,012	0,035
	AddFriend (ms)	0,020	0,015	0,017
	Logout (ms)	0,022	0,008	0,015

Πίνακας 4.8: Αποτελέσματα εκτέλεσης Web Serving benchmark

4.3 Αποτελέσματα Parsec

Στην υποενότητα αυτή παρουσιάζονται τα αποτελέσματα που λάβαμε μετά την εκτέλεση των Parsec Benchmarks.



Σχήμα 4.4: Κανονικοποιημένα αποτελέσματα Parsec

Στην παραπάνω εικόνα 4.4 βλέπουμε μια κανονικοποιημένη συνολική απόδοση όλων των διαφορετικών μετροπρογραμμάτων της Parsec σουίτας. Η κανονικοποίηση, όπως και προηγουμένως, έγινε ως προς την επίδοση του VM1, θεωρώντας ότι είναι το βέλτιστο σενάριο, του ανατέθηκε η τιμή 100 σε κάθε περίπτωση, ενώ στα υπόλοιπα δύο σενάρια, υπολογίστηκε η ποσοστιαία απόκλιση από την απόδοση του VM1, και προστέθηκε στο 100, εάν πρόκειται για χειρότερη επίδοση, ή αφαιρέθηκε από το 100 εάν πρόκειται για καλύτερη. Έτσι, όσο πιο πάνω απ΄ το 100 βρίσκεται μια προσομοίωση, τόσο χειρότερα απέδωσε (σε ποσοστό επί τοις εκατό). Η Parsec σουίτα, δίνει την επιλογή στο χρήστη να διαλέξει το μέγεθος του workload, καθώς και αν θέλει να μεταγλωττίσει τα μετροπρογράμματα ενεργοποιώντας λειτουργίες παραλληλοποίησης ώστε να αξιοποιηθούν περισσότεροι πόροι της CPU όπου είναι δυνατό. Για το λόγο αυτό, επιλέχθηκε το μεγαλύτερο δυνατό workload (native), το οποίο προσομοιώνει ρεαλιστικό σενάριο χρήσης της εκάστοτε εφαρμογής, ενώ οι εφαρμογές εκτελέστηκαν με ενεργοποιημένη την επιλογή για παραλληλία, όπου αυτό είναι διαθέσιμο, αφού τα σενάρια μνήμης τα οποία θέλουμε να μελετήσουμε έχουν προαπαιτούμενο την αξιοποίηση πολλαπλών πόρων της CPU.

Παρατηρώντας τα κανονικοποιμένα αποτελέσματα, αρχικά αντιλαμβάνεται κανείς πως υπήρξε ουσιαστική διαφορά στην απόδοση σε λιγότερα benchmarks σε σχέση με αυτά της προηγούμενης σουίτας. Συγκεκριμένα, με βάση το παραπάνω σχήμα, σε έξι από τα δέκα benchmarks παρατηρείται μείωση της απόδοσης, ενώ στα υπόλοιπα τέσσερα, η διαφορά είναι αρκετά μικρή, με αποτέλεσμα να μην μας επιτρέπει να την συνδέσουμε με τη μεταβολή της επίδοσης της μνήμης μεταξύ των setups. Τα έξι benchmarks που έχουν «επηρεαστεί» από την αλλαγή της NUMA τοπολογίας, είναι τα Blackscholes, Bodytrack, Canneal, Ferret, Streamcluster και Swaptions, με σημαντικές μειώσεις στην απόδοση, που φτάνει έως και το 60% ανά περιπτώσεις.

Στο σημείο αυτό θα πρέπει να αναφερθεί ότι σουίτα δεν αντιπροσωπεύει τις cloud εφαρμογές, και πιο δύσκολα θα τις συναντήσουμε σε cloud infrastructures. Οι cloud εφαρμογές απαιτούν συνεχή διαθεσιμότητα, και ανεκτικότητα σε μεγάλο όγκο χρηστών που προσπαθούν να προσπελάσουν δεδομένα, καθιστώντας έτσι, τις περισσότερες φορές, τη μνήμη του υπολογιστή το βασικότερο γρανάζι στη συνολική λειτουργία τους. Αντίθετα, οι εφαρμογές που συναντώνται στην σουίτα Parsec χαρακτηρίζονται κυρίως ως hpc εφαρμογές, δηλαδή συνήθως CPU heavy, που βρίσκουν χρήση σε προσωπικούς υπολογιστές και εργαστήρια, στα οποία μας ενδιαφέρει η επίλυση δύσκολων υπολογιστικά προβλημάτων. Παρ΄ όλ΄ αυτά, θεωρήθηκε χρήσιμο να συμπεριληφθούν στην πειραματική διαδικασία, αφού μερικές απ΄ τις προσομοιώσεις μπορούν να βρεθούν και σε cloud infrastructures, αλλά και για μια πιο ολοκληρωμένη προσέγγιση στην επίδραση που έχει η αποδοτικότητα της μνήμης στα σύγχρονα συστήματα.

Στη συνέχεια ακολουθούν αναλυτικά τα αποτελέσματα για κάθε εφαρμογή ξεχωριστά, κατ΄ αντιστοιχία με την προηγούμενη υποενότητα.

4.3.1 Blackscholes

Το Blackscholes benchmark χρησιμοποιείται για τον υπολογισμό και την αξιολόγηση επενδυτικών επιλογών χρησιμοποιώντας την διαφορική εξίσωση Black-Scholes. Πρόκειται για ένα υπολογιστικά intensive benchmark, στο οποίο απαιτούνται πολλά εκατομμύρια πράξεις. Με βάση το documentation της εφαρμογής [31][34], βλέπουμε πως έχουμε coarse-granular parallelism, δηλαδή χωρίζεται το input των 10.000.000 επιλογών σε κομμάτια, ανάλογα με τον αριθμό των διαθέσιμωνCPUs, και ανατίθενται ένα ανά CPU, ή αλλιώς static load-balancing. Επιπλέον, η επικοινωνία ανάμεσα στα CPUs είναι αμελητέα, και απαιτείται αμελητέα cache για τη βέλτιστη λειτουργία της εφαρμογής.

Στον παρακάτω πίνακα 4.9 παρουσιάζονται τα αποτελέσματα της εκτέλεσης του benchmark. Τόσο σε αυτό το benchmark, όσο και στα υπόλοιπα της σουίτας, το αποτέλεσμα και μετρική που συγκρίνεται είναι ο χρόνος εκτέλεσης.

	VM1	**VM2**	**VM3**
Execution Time	5m39.271s	7m25.012s	7m56.020s

Πίνακας 4.9: Αποτελέσματα εκτέλεσης Blackscholes benchmark

Ελέγχοντας τα αποτελέσματα του παραπάνω πίνακα, παρατηρεί κανείς πως η απόδοση της εφαρμογής μειώθηκε στα δύο τελευταία σενάρια εκτέλεσης σε σχέση με το πρώτο. Το αποτέλεσμα αυτό οφείλεται στο γεγονός πως πρόκειται για μια υπολογιστική εφαρμογή, η οποία αποθηκεύει τις δομές δεδομένων που χρησιμοποιεί στη μνήμη, όταν επαρκεί, και έτσι, χρειάζεται να την προσπελάσει όταν πρέπει να διαβάσει ή να γράψει δεδομένα. Η βελτίωση που παρατηρήθηκε φαίνεται μεγαλύτερη απ΄ αυτή που περιμέναμε να δούμε, καθώς με βάση τη

θεωρία πίσω απ΄ την προσομοίωση, δεν υπάρχει μεγάλη εξάρτηση απ΄ τη μνήμη. Η ελάχιστη intercore επικοινωνία επίσης βοήθησε να φανεί ακόμα περισσότερο η διαφορά στη μνήμη στο τελικό αποτέλεσμα. Βέλτιστο σενάριο είναι αυτό του VM1.

4.3.2 Bodytrack

Το Bodytrack benchmark είναι ένα computer vision application, το οποίο αναγνωρίζει και υποδεικνύει έναν άνθρωπο και τα μέλη του, με είσοδο ένα video. Και εδώ πρόκειται για υπολογιστικά intesive εφαρμογή, η οποία εξαντλεί σε μεγάλο βαθμό τους πόρους του συστήματος. Το benchmark παρουσιάζει medium-granular parallelism, και δυναμικό load-balancing. Η επικοινωνία μεταξύ των CPUs κατά την εκτέλεση είναι μικρή, ενώ χρειάζεται περίπου 8 MB μνήμη cache (αποθηκεύονται κυρίως τα input frames) για να πετύχει miss rate που αγγίζει το 0%. Στον παρακάτω πίνακα 4.10 παρουσιάζονται τα αποτελέσματα εκτέλεσης του benchmark.

	VM1	**VM2**	**VM3**
Execution Time	5m9.136s	6m25.524s	6m36.820s

Πίνακας 4.10: Αποτελέσματα εκτέλεσης Bodytrack benchmark

Παρατηρώντας τα αποτελέσματα του παραπάνω πίνακα, βλέπουμε πως η απόδοση μειώθηκε καθώς το setup άλλαζε. Όπως βλέπουμε και από το documentation, η εφαρμογή φορτώνει δεδομένα στη μνήμη (και αντίστοιχα στην cache), όπως τα frames προς επεξεργασία, προκειμένου να βελτιστοποιήσει την πρόσβαση σε αυτά. Η διαφορά στην μνήμη cache ανάμεσα στα σενάρια εκτέλεσης δεν ήταν αρκετή για να μεταβάλλει τις ισορροπίες, και δεν επηρεάστηκε η εκτέλεση σε μεγάλο βαθμό από την επικοινωνία μεταξύ των CPUs. Το αποτέλεσμα αυτό οφείλεται στη χρήση της μνήμης, σε έναν ικανοποιητικό βαθμό από την προσομοίωση, και γι΄ αυτό το βέλτιστο σενάριο εκτέλεσης είναι αυτό του VM1.

4.3.3 Canneal

Το Canneal benchmark είναι ένα optimization application για Electronic Design Automation (EDA). Το input του προγράμματος είναι τάξης μεγέθους εκατομμυρίων στοιχείων προς επεξεργασία. Το πρόγραμμα χαρακτηρίζεται από fine-granular parallelism, δηλαδή τα τμήματα που μπορούν να επεξεργαστούν παράλληλα χωρίζονται σε πολλά μικρά κομμάτια, τα οποία διαμοιράζονται στα διαθέσιμα CPUs. Επίσης, απαιτείται σχεδόν συνεχής επικοινωνία μεταξύ των CPUs για συγχρονισμό (communication intensive), ενώ περιορίζεται έως ένα βαθμό όσο μεγαλύτερο το μέγεθος της cache, η οποία πρέπει να είναι απλησίαστα μεγάλη (πάνω από 250 MB) ώστε να εξαλειφθεί το miss rate. Γενικώς, πρόκειται για ένα απ΄ τα απαιτητικότερα σε μνήμη benchmarks της σουίτας.

Στον παρακάτω πίνακα ακολουθούν τα αποτελέσματα που συλλέξαμε μετά την εκτέλεση.

	VM1	**VM2**	**VM3**
Execution Time	6m1.280s	9m4.508s	7m29.664s

Πίνακας 4.11: Αποτελέσματα εκτέλεσης Canneal benchmark

Παρατηρώντας τα αποτελέσματα του πίνακα, βλέπουμε πως στο VM2 η απόδοση είναι χειρότερη απ΄ όλες τις εκτελέσεις, ενώ και η απόδοση του VM3 είναι χειρότερη από του VM1. Το φαινόμενο αυτό, όπως είχε ξανασυμβεί και στο Data Caching benchmark της σουίτας Cloudsuite, οφείλεται στην επικοινωνία που απαιτείται για το συγχρονισμό της LLC μεταξύ των διαφορετικών σεναρίων εκτέλεσης. Όπως είχαμε ήδη αναφέρει, στο VM2, λόγω της χρήσης πόρων CPU από δύο NUMA nodes, διπλασιάζεται η LLC, η οποία διαμοιράζεται μεταξύ των CPUs που ανήκουν στο ίδιο node. Έτσι, παρ΄ όλο που η απόδοση της μνήμης είναι μικρότερη στο VM3, η ανάγκη για συνεχή συγχρονισμό δεδομένων μεταξύ των cache μνημών της CPU - αφού όπως είπαμε και διαβάσαμε στο documentation της εφαρμογής απαιτείται τα CPUs να επικοινωνούν μεταξύ τους - οδηγεί σενάρια όπως αυτό του VM2 στην περίπτωσή μας, να συμπεριφέρονται χειρότερα. Βέβαια, το συμπέρασμα αυτό οφείλεται αποκλειστικά στην επικοινωνία ή coherency των μνημών, και στη γενική περίπτωση, ο διπλασιασμός της μνήμης cache, δεν δικαιολογεί μείωση της απόδοσης, αν μη τι άλλο, ιδιαίτερα στην εφαρμογή αυτή όπου γνωρίζουμε τις μεγάλες απαιτήσεις σε cache. Αυτό σημαίνει ότι εάν εξετάζαμε ένα σενάριο στο οποίο η LLC είχε τόσο μέγεθος όσο αυτή στα VM1, VM3 αλλά υπήρχε ανάγκη για συγχρονισμό σε CPUs σε διαφορετικά NUMA nodes, θα περιμέναμε να δούμε ακόμα χειρότερη επίδοση απ΄ αυτή που κατέγραψε το σενάριο του VM2. Το σενάριο του VM1 είναι και σε αυτή την περίπτωση το βέλτιστο.

4.3.4 Facesim

Το Facesim benchmark προσομοιώνει τις κινήσεις ενός ανθρώπινου προσώπου που συνήθως χρησιμοποιείται σε video games και ταινίες. Πρόκειται για μια HPC εφαρμογή, με μεγάλα working sets, coarsed-granular parallelism, και μικρή επικοινωνία μεταξύ των CPUs κατά την εκτέλεση και ελάχιστο data sharing.

Τα αποτελέσματα της εκτέλεσης παρουσιάζονται στον παρακάτω πίνακα.

	VM1	**VM2**	**VM3**
Execution Time	16m18.387s	15m27.045s	16m0.651s

Πίνακας 4.12: Αποτελέσματα εκτέλεσης Facesim benchmark

Βλέποντας τα αποτελέσματα του παραπάνω πίνακα, παρατηρούμε ότι η απόδοση της εφαρμογής φαίνεται περίπου ίδια κατά την εκτέλεση στο VM1 και VM3, ενώ αντίθετα παρατηρούμε ότι η εκτέλεση στο VM2 είναι η καλύτερη απ΄ τις 3, με διαφορά περίπου 6%. Για

να εξηγήσουμε το αποτέλεσμα αυτό, το οποίο μοιάζει ξένο, καθώς δεν έχουμε βρει άλλο μετροπρόγραμμα στο οποίο η απόδοση εκτέλεσης σε κάποιο εκ των VM2, VM3 να είναι καλύτερη απ΄ αυτή του VM1, έπρεπε να ανατρέξουμε στο documentation. Εκεί θα δούμε ότι η εφαρμογή όσο μεγαλύτερη μνήμη cache διαθέτει, μειώνει τον αριθμό των cache misses, καθώς το fase mesh έχει μεγάλο μέγεθος, και ιδανικά θέλουμε να χωρέσει όλο στην cache ώστε να μην χρειαστεί να εμπλακεί η μνήμη περισσότερες φορές. Επιπλέον, ξέρουμε ότι το intercore communication και το intercore data sharing δεν επηρεάζουν την εκτέλεση της εφαρμογής καθώς είναι μικρά, οπότε η εφαρμογή έχει το περιθώριο να εκμεταλλευτεί την παραπάνω cache που προσφέρει η εκτέλεση στο VM2, χωρίς τα μειονεκτήματα που την χαρακτηρίζουν. Έτσι, το benchmark αυτό μας έδειξε κάτι που περιμέναμε αλλά δεν είχαμε συναντήσει μέχρι στιγμής, δηλαδή ότι βέλτιστο σενάριο εκτέλεσης είναι αυτό του VM2 και όχι του VM1.

4.3.5 Fluidanimate

Το Fluidanimate benchmark, προσομοιώνει κινήσεις υγρών στοιχείων, τα οποία συναντώνται σε video-games και ταινίες, όπως και προηγουμένως. Και αυτή η εφαρμογή χαρακτηρίζεται από coarsed-granular parallelism, στατικό load balancing, και μικρή επικοινωνία μεταξύ των CPUs κατά την παράλληλη εκτέλεση.

Τα αποτελέσματα της εκτέλεσης παρουσιάζονται στον παρακάτω πίνακα.

	VM1	**VM2**	**VM3**
Execution Time	18m13.840s	16m35.524s	13m25.280s

Πίνακας 4.13: Αποτελέσματα εκτέλεσης Fluidanimate benchmark

Και σε αυτή την περίπτωση, έχουμε ένα HPC oriented benchmark, στο οποίο η μνήμη δεν είχε μεγάλη επίδραση. Οι διαφορές στην απόδοση που παρατηρούνται ανά σενάριο εκτέλεσης στον παραπάνω πίνακα, είναι αποτέλεσμα του scheduler, και των υπόλοιπων running processes που υπήρχαν κατά τη διάρκεια της εκτέλεσης. Το benchmark, όπως διαβάζουμε και στο documentation, δεν έχει μεγάλη εξάρτηση απ΄ τη μνήμη του συστήματος, οπότε και στην περίπτωση αυτή δεν μπορούμε να αποφανθούμε για βέλτιστο σενάριο εκτέλεσης, καθώς όλα είναι ισάξια.

4.3.6 Ferret

Το Ferret benchmark πραγματοποιεί αναζήτηση σε μια βάση δεδομένων που αποτελείται από εικόνες, στην οποία και αναζητά εικόνες παρόμοιες με την ζητούμενη, αναλύοντας με υπολογιστικά δύσκολο τρόπο τα περιεχόμενά τους. Πρόκειται για ένα server application, δηλαδή μια εφαρμογή που θα μπορούσε να βρεθεί σε cloud infrastructures και να εξυπηρετεί τις ανάγκες των εκάστοτε χρηστών. Το benchmark χαρακτηρίζεται από pipeline parallelism

με πολλαπλά thread pools που δουλεύουν ταυτόχρονα, πολύ μεγάλα working sets και συνεχή επικοινωνία μεταξύ των CPUs.

Στον παρακάτω πίνακα παρουσιάζονται τα αποτελέσματα εκτέλεσης της προσομοίωσης.

	VM1	**VM2**	**VM3**
Execution Time	13m25.972s	19m1.012s	16m40.260s

Πίνακας 4.14: Αποτελέσματα εκτέλεσης Ferret benchmark

Παρατηρώντας τα αποτελέσματα, παρατηρούμε πως βέλτιστη απόδοση ήταν αυτή του VM1, ενώ χείριστη αυτή του VM2. Το φαινόμενο αυτό, όπως έχουμε ήδη ξανασυναντήσει σε αντίστοιχες περιπτώσεις, οφείλεται στο ότι η εφαρμογή είναι communication intensive όσον αφορά τα CPUs, διότι χρειάζεται συνεχώς να συγχρονίζουν τα δεδομένα μεταξύ τους. Έτσι, επειδή η intercore επικοινωνία έχει μεγαλύτερο latency στο δεύτερο σενάριο εκτέλεσης (μόνο για την LLC), είναι προτιμότερο η εφαρμογή να εκτελείται με όλα τα CPUs στο ίδιο NUMA node και τη μνήμη πιο μακριά, σε σχέση με το ανάποδο σενάριο. Βέλτιστο σενάριο εκτέλεσης είναι αυτό του VM1, με απόδοση βελτιωμένη κατά 43% και 24% σε σχέση με την δεύτερη και τρίτη εκτέλεση αντίστοιχα.

4.3.7 Freqmine

Το Freqmine benchmark είναι μια data mining εφαρμογή, στην οποία πραγματοποιείται αναζήτηση μοτίβων σε μία βάση δεδομένων με πολλές διασυνδεδεμένες (χρησιμοποιώντας FP-trees) HTML σελίδες. Η εφαρμογή χαρακτηρίζεται από medium-granular parallelism, μεγάλα working sets, και απαιτείται μικρή επικοινωνία μεταξύ των CPUs για συγχρονισμό δεδομένων.

Τα αποτελέσματα εκτέλεσης της προσομοίωσης στα διαφορετικά σενάρια φαίνονται στον παρακάτω πίνακα.

	VM1	**VM2**	**VM3**
Execution Time	11m27.385s	11m19.008s	11m20.128s

Πίνακας 4.15: Αποτελέσματα εκτέλεσης Freqmine benchmark

Παρατηρώντας τα αποτελέσματα του πίνακα, βλέπουμε πως η εφαρμογή δεν παρουσίασε διαφορά στο χρόνο εκτέλεσης σε κανένα απ΄ τα σενάρια που χρησιμοποιήθηκαν. Το συγκεκριμένο benchmark ήταν CPU oriented, χωρίς να χρειάζεται να προσπελάσει τη μνήμη συχνά, ενώ και η intercore επικοινωνία είναι περιορισμένη, αλλά και με σχετικά μικρή μνήμη cache το ποσοστό του miss rate είναι σχεδόν 0%. Έτσι το πρόγραμμα δεν επηρεάστηκε από τη διαφορά στην απόδοση της μνήμης ή του μεγέθους της LLC και της επικοινωνίας, γι΄ αυτό και δεν μπορούμε να αποφανθούμε για βέλτιστο σενάριο εκτέλεσης, καθώς είναι όλα ισάξια.

4.3.8 Raytrace

Το Raytrace benchmark προσομοιώνει τη λειτουργία του ray-tracing, το οποίο είναι μια rendering τεχνική που χρησιμοποιείται σε video games και ταινίες, για πιο ρεαλιστικές εικόνες, σε ό,τι έχει να κάνει με το φως και τις σκιές. Η εφαρμογή χαρακτηρίζεται από fine-granular parallelism, δυναμικό load balancing, σχετικά μεγάλα working sets και μικρή intercore επικοινωνία. Επιπλέον, διαβάζουμε πως η εφαρμογή αξιοποιεί το memory bandwidth για να βελτιώσει την απόδοσή της.

Στον παρακάτω πίνακα παρουσιάζονται τα αποτελέσματα που λάβαμε μετά την εκτέλεση στα διαφορετικά σενάρια.

	VM1	**VM2**	**VM3**
Execution Time	5m50.628s	5m29.236s	5m49.690s

Πίνακας 4.16: Αποτελέσματα εκτέλεσης Raytrace benchmark

Παρατηρώντας τα αποτελέσματα του πίνακα, βλέπουμε πως δεν υπάρχει μεγάλη διαφορά ανάμεσα στις εκτελέσεις στα VMs. Αν και δεν μοιάζει περίεργο εξ αρχής, αφού έχουμε να κάνουμε με HPC εφαρμογή, που δεν θα συναντήσουμε σε cloud infrastructures, αλλά σε μια κάρτα γραφικών σε προσωπικούς υπολογιστές, θα περιμέναμε να δούμε διαφορά στην εκτέλεση και πιο συγκεκριμένα, θα έπρεπε η απόδοση της εφαρμογής όταν αυτή εκτελείται στο VM1 να είναι καλύτερη απ΄ τις υπόλοιπες, αφού γνωρίζουμε ότι η εφαρμογή εκμεταλλεύεται το memory bandwidth, το οποίο είναι βέλτιστο στην περίπτωση του VM1. Αυτό δεν συνέβη στην περίπτωσή μας, και ο λόγος που πιθανώς έγινε αυτό είναι διότι το bandwidth που προσφέρει το σύστημά μας επαρκούσε και στις τρεις περιπτώσεις, και δεν υπήρξε congestion των δεδομένων στα links. Έτσι, δεν μπορούμε να αποφανθούμε για βέλτιστο σενάριο εκτέλεσης με βάση τις μετρήσεις μας. Όμως, είναι σημαντικό να αναφέρουμε ότι στη γενική περίπτωση το σενάριο του VM1 θα ήταν το καλύτερο απ΄ τα τρία.

4.3.9 Streamcluster

Το Streamcluster benchmark είναι ένα Machine Learning application, το οποίο υπολογίζει μια εκτίμηση του βέλτιστου clustering σε ένα stream από data points. Το πρόγραμμα χαρακτηρίζεται από coarse-granular parallelism, στατικό load balancing, ενώ έχει μεγάλα working sets στην είσοδο, και δεν απαιτείται συχνή intercore επικοινωνία. Από το documentation της εφαρμογής διαβάζουμε πως το benchmark είναι memory bound αλλά και computationaly intensive σε μεγάλα working sets όπου τα data points έχουν πολλές διαστάσεις, οπότε, βλέποντας το ζήτημα από θεωρητική σκοπιά, θα πρέπει να δούμε διαφορά στην επίδοση μεταξύ των διαφορετικών εκτελέσεων.

Στον παρακάτω πίνακα ακολουθούν τα αποτελέσματα των εκτελέσεων.

	VM1	**VM2**	**VM3**
Execution Time	11m21.670s	17m16.711s	17m51.223s

Πίνακας 4.17: Αποτελέσματα εκτέλεσης Streamcluster benchmark

Παρατηρώντας τα παραπάνω αποτελέσματα, πράγματι γίνεται αντιληπτό πως η απόδοση της εφαρμογής μειώθηκε δραστικά στα δύο τελευταία σενάρια σε σχέση με το πρώτο. Το πρόγραμμα χρειάζεται να φέρνει συνεχώς νέα data points απ΄ τη μνήμη, ώστε να υπολογίζει πιθανά κέντρα, και να τα αποθηκεύει εκ νέου πίσω στη μνήμη, ώστε να ξαναχρησιμοποιηθούν στην πορεία της εκτέλεσης. Καταλαβαίνει κανείς λοιπόν πόσο συχνά χρειάζεται το πρόγραμμα να έχει δοσοληψίες με τη μνήμη. Αυτός είναι και ο λόγος που η απόδοση της μνήμης είναι πολύ σημαντική στο συγκεκριμένο benchmark. Βέλτιστη περίπτωση εκτέλεσης είναι αυτή του VM1 ενώ και το σενάριο του VM2 φαίνεται να ήταν καλύτερο απ΄ αυτό του VM3, με μικρή διαφορά.

4.3.10 Swaptions

Το Swaptions benchmark είναι ένα computational finance πρόγραμμα, σκοπός του οποίου είναι η επενδυτική αξιολόγηση των δοσμένων ως input, swap options, ενώ για τον υπολογισμό χρειάζεται να επιλυθούν διαφορικές εξισώσεις. Όπως γίνεται αντιληπτό, το πρόγραμμα μοιάζει με το Blackscholes benchmark, το οποίο ήταν και αυτό computational finance benchmark που απαιτούσε επίλυση διαφορικών εξισώσεων. Το Swaptions παρουσιάζει στατικό load balancing, μικρά working sets και αμελητέα intercore επικοινωνία.

Ακολουθεί ο πίνακας με τα αποτελέσματα της εκτέλεσης στα τρία διαφορετικά σενάρια.

	VM1	**VM2**	**VM3**
Execution Time	9m24.797s	13m33.615s	15m16.992s

Πίνακας 4.18: Αποτελέσματα εκτέλεσης Swaptions benchmark

Παρατηρώντας τα αποτελέσματα του παραπάνω πίνακα, όπως και τα αντίστοιχα προηγούμενα μετροπρογράμματα, η απόδοση της εφαρμογής μειώθηκε περίπου 45% στο σενάριο του VM2, και 62% στο σενάριο του VM3 σε σχέση με το αρχικό του VM1. Συμπεραίνουμε ότι η εφαρμογή χρειάστηκε να φέρει πολλές φορές δεδομένα απ΄ τη μνήμη, τα οποία κυρίως έχουν να κάνουν με τις δομές δεδομένων που αποθηκεύονται τα swap options, ενώ λόγω του επαναληπτικού χαρακτήρα που παρουσιάζει η εφαρμογή (αφού χρησιμοποιεί το μοντέλο Μόντε-Κάρλο), οι προσβάσεις στη μνήμη επαναλαμβάνονται συνεχώς. Έτσι, η απόδοση της εφαρμογής εξαρτάται άμεσα από το latency που παρουσιάζει η πρόσβαση στη μνήμη, ενώ δεν φαίνεται να εξαρτάται από το μέγεθος της cache ή την ενδοεπικοινωνία μεταξύ των CPUs για συγχρονισμό των δεδομένων. Βέλτιστο σενάριο είναι αυτό του VM1.

Κεφάλαιο 5

Συμπεράσματα

Όπως έχει ήδη αναφερθεί και σε προηγούμενες ενότητες, σκοπός της παρούσας εργασίας, είναι η μελέτη του νέφους και της απόδοσης των διαφόρων εφαρμογών, όταν αυτές εκτελούνται σε εικονικές μηχανές με διαφορετική αντιστοίχιση εικονικών πόρων σε φυσικούς, σε ό,τι έχει να κάνει με τη μνήμη και την «απόστασή» της από τους επεξεργαστές. Μια τέτοια μελέτη είναι χρήσιμη, και γι αυτό επικεντρωθήκαμε κυρίως σε cloud infrastructures, διότι στα data centers που φιλοξενούν τους servers που χρησιμοποιούνται για το σκοπό αυτό, βρίσκονται μηχανήματα με πολλά CPUs, μνήμη και γενικώς υπολογιστικούς πόρους, τα οποία λόγω της φύσης του νέφους διαμοιράζουν τους πόρους αυτούς σε εικονικές μηχανές που τους ζητούν. Έτσι, γνωρίζοντας ότι η διασύνδεση των επεξεργαστών με τη μνήμη στα σύγχρονα πολυπύρηνα συστήματα ακολουθεί NUMA αρχιτεκτονική, μπορεί κανείς να συμπεράνει πως όλα τα σενάρια που κατασκευάσαμε και εκτελέσαμε, αφορούν άμεσα ρεαλιστικά σενάρια που θα μπορούσαν και μπορούν να συμβούν σε τέτοιες συνθήκες διαμοιρασμού πόρων.

Μετά την ολοκλήρωση όλων των προσομοιώσεων, στα διαφορετικά σενάρια εκτέλεσης που κατασκευάσαμε, σκοπός ήταν να έχουμε μια εικόνα για το πόσο σημαντικό είναι το latency πρόσβασης στη μνήμη στις διάφορες εφαρμογές που εξετάσαμε, καθώς και να εξετάσουμε τα σενάρια μεταξύ τους, ώστε να βρούμε ποιο σενάριο εκτέλεσης είναι το βέλτιστο. Στο σημείο αυτό, υπενθυμίζουμε εν τάχει τα τρία διαφορετικά σενάρια που χρησιμοποιήθηκαν για την εκτέλεση των μετροπρογραμμάτων:

- **VM1**: Τα CPUs και η μνήμη της εικονικής μηχανής αντιστοιχίζονται σε φυσικά του ίδιου NUMA node
- **VM2**: Τα CPUs αντιστοιχίζονται σε φυσικά από δύο διαφορετικά NUMA nodes και η μνήμη αντιστοιχίζεται ολόκληρη σε ένα απ΄ τα δύο.
- **VM3**: Τα CPUs αντιστοιχίζονται στο ίδιο NUMA node και η μνήμη σε διαφορετικό.

Στο σημείο αυτό αξίζει να δει κανείς τη συνολική εικόνα των αποτελεσμάτων. Συνολικά εκτελέστηκαν και στις τρεις εικονικές μηχανές 18 benchmarks, από τα οποία, τα 12 (δηλαδή τα 2/3), εμφάνισαν βελτιωμένη απόδοση όταν εκτελέστηκαν υπό τις συνθήκες του VM1, το

οποίο είναι και ένα αποτέλεσμα που έμοιαζε εξ αρχής λογικό. Παίρνοντας τους μέσους όρους βελτίωσης για κάθε μία απ΄ τις 12 εφαρμογές, βρίσκουμε ότι όταν η εφαρμογή εκτελείται υπό τις συνθήκες του VM1, σε σχέση με τις συνθήκες των άλλων δύο περιπτώσεων, τότε παρουσιάζει βελτίωση στην απόδοση 29% κατά μέσο όρο. Το συμπέρασμα αυτό μπορεί να γραφεί ως εξής: **Τα 2/3 των εφαρμογών που εκτελούνται με CPUs και memory στο ίδιο NUMA node, σε σχέση με τις υπόλοιπες επιλογές, παρουσιάζουν βελτιωμένη απόδοση κατά 29%.**

Το επόμενο συμπέρασμα που αξίζει να σημειωθεί με βάση τη συνολική εικόνα που έχουμε απ΄ την εκτέλεση των benchmarks, αφορά την εκτέλεση στο VM2. Ελέγχοντας τα αποτελέσματα για κάθε προσομοίωση ξεχωριστά, βλέπουμε ότι υπήρξαν 3 προσομοιώσεις στις οποίες η απόδοση της εκτέλεσης ήταν χειρότερη απ΄ αυτή του VM3 ενώ γνωρίζουμε ότι η πρόσβαση στη μνήμη στο VM3 είναι χειρότερη τόσο σε άποψη latency όσο και σε bandwidth. Όπως ήδη έχουμε εξηγήσει, το συμπέρασμα οφείλεται στον συγχρονισμό των δεδομένων στις caches του επεξεργαστή, όπου απαιτεί περισσότερο χρόνο στο VM2. Επομένως: **Μια πολυπύρηνη communication intensive εφαρμογή, μπορεί να είναι προτιμότερο να τρέξει σε σενάριο όπου οι πυρήνες βρίσκονται στο ίδιο NUMA node ώστε να μην χρησιμοποιηθούν interconnect links για intercore communication, από το να έχει μικρότερο latency στην πρόσβαση στη μνήμη**.

Ένα ακόμη συμπέρασμα που θα πρέπει να σημειώσουμε, προκύπτει ξανά από την εκτέλεση στο VM2 και αφορά μία προσομοίωση όπου η απόδοση της εκτέλεσης στο VM2 είναι καλύτερη απ΄ όλες τις υπόλοιπες. Όπως εξηγήσαμε, χωρίζοντας τα CPUs σε διαφορετικά NUMA nodes, αυξάνεται η LLC η οποία διαμοιράζεται στα cores που ανήκουν στο ίδιο NUMA node. Επομένως διαλέγοντας cores που ανήκουν σε διαφορετικά nodes, η διαθέσιμη cache θα είναι μεγαλύτερη. Μπορούμε λοιπόν να πούμε: **Μία εφαρμογή που μπορεί να αξιοποιήσει μεγάλα μεγέθη cache μνήμης, για να μειωθούν τα memory accesses, είναι προτιμότερο να εκτελεστεί σε σενάριο όπου τα cores βρίσκονται σε διαφορετικά NUMA nodes, παρά το χειρότερο memory latency.**

Τέλος, αν και δεν εξετάστηκε στην παρούσα εργασία στο πειραματικό κομμάτι, είναι σημαντικό να αναφερθεί ότι σε περίπτωση που η εφαρμογή λειτουργεί καλύτερα και πιο αποδοτικά όσο μεγαλύτερο είναι το memory bandwidth, τότε το βέλτιστο σενάριο εκτέλεσης δεν είναι καμία απ΄ τις τρεις εικονικές μηχανές που κατασκευάσαμε, αλλά μια εικονική μηχανή που έχει τα cores μοιρασμένα μεταξύ των NUMA nodes, αλλά και τη μνήμη μοιρασμένη στα nodes που χρησιμοποιούνται. Έτσι το bandwidth πολλαπλασιάζεται ανάλογα με τα νέα memory DIMMs που χρησιμοποιούνται, με το overhead βέβαια του latency και cache coherency που προστίθεται. Άρα: **Όταν μια εφαρμογή απαιτεί για τη βέλτιστη λειτουργία της μεγαλύτερο memory bandwidth, τότε είναι προτιμότερο να εκτελείται με τα CPUs και τη μνήμη διαμοιρασμένα μεταξύ των NUMA nodes**.

Αφού εξήγαμε όλα τα συμπεράσματα που προέκυψαν απ΄ την εκτέλεση των προσομοιώσεων στα διαφορετικά σενάρια εκτέλεσης, θα πρέπει να μπορούμε να απαντήσουμε σε ένα ερώτημα που εκκρεμεί. Αυτό είναι εάν τελικά υπάρχει, και αν ναι, ποια είναι η καλύτερη δυνατή διάταξη ή αντιστοίχιση των εικονικών και φυσικών πόρων ανάμεσα στις εικονικές μηχανές, που θα πρέπει να φροντίσω να έχω στην υποδομή μου· Η απάντηση στο ερώτημα εξαρτάται από το πόση βελτιστοποίηση θέλω να πετύχω. Η απλούστερη λύση είναι ότι θα ήθελα όλες οι εικονικές μηχανές να αντιστοιχίζονται σε πόρους που βρίσκονται στο ίδιο node. Με αυτόν τον τρόπο, μπορώ να είμαι σίγουρος ότι θα έχω κατά μέσο όρο 29% βελτίωση σε σχέση με πιθανά άλλα σενάρια σε πολλές απ΄ τις εφαρμογές που φιλοξενούνται στις εικονικές μηχανές της υποδομής μου. Όμως, το βέλτιστο δυνατό σενάριο, είναι να μπορώ να μελετήσω ξεχωριστά την κάθε εφαρμογή που εκτελείται σε κάθε εικονική μηχανή, ώστε να γνωρίζω τα χαρακτηριστικά της, και ποια είναι η ιδανική περίπτωση αντιστοίχισης πόρων ανάμεσα στην εικονική μηχανή και το φυσικό μηχάνημά μου. Κάτι τέτοιο σε ρεαλιστικές συνθήκες έχει μεγάλη δυσκολία στην επίλυση, καθώς ο κάθε χρήστης εικονικής μηχανής μπορεί να εκτελεί οποιαδήποτε εφαρμογή επιθυμεί οποτεδήποτε θέλει, χωρίς να εμπλέκεται ο cloud provider. Όμως ιδανικά, θα πρέπει ο εκάστοτε hypervisor και το cloud software που χρησιμοποιώ να λαμβάνει υπόψιν του τα δεδομένα αυτά, και να διαχωρίζει τους πόρους του με βάση τη βέλτιστη αξιοποίηση όλων των εφαρμογών.

Συνοψίζοντας, η εργασία που υλοποιήσαμε, αποσκοπούσε στο να δώσει μια εικόνα για την επίδραση που έχει η NUMA αρχιτεκτονική στην λειτουργία ενός νέφους, το πώς και πόσο επηρεάζει η διαφορετική κατανομή των πόρων την απόδοση των εφαρμογών που συναντώνται σε τέτοια περιβάλλοντα. Αυτό που πρέπει να κρατήσει κάποιος διαβάζοντας την παρούσα εργασία, είναι ότι η λειτουργία των εφαρμογών νέφους επηρεάζεται σε πολύ μεγάλο βαθμό από την NUMA τοπολογία, και ενώ δεν υπάρχει εύκολη λύση που μπορεί να εφαρμοστεί σε cloud infrastructures προκειμένου να επιτύχουμε τα βέλτιστα δυνατά αποτελέσματα, υπάρχει τρόπος να αυξηθεί σημαντικά η απόδοση που αντιλαμβάνεται ο τελικός χρήστης στην εφαρμογή του, με κατάλληλες αναδιατάξεις των πόρων των εικονικών μηχανών.

Παράρτημα Α΄

Αρκτικόλεξο

ΛΣ Λειτουργικό σύστημα
B Byte
KB Kilo Byte
MB Mega Byte
GB Giga Byte
MS Micro Second
SEC Second
VM Virtual Machine
CPU Central Proccesing Unit
API Application Programming Interface
KVM Kernel-based Virtual Machine
QEMU Quick Emulator
OPS Operations
LLC Last Level Cache

Παράρτημα Β΄

LibVirt Configuration Files

Για την κατασκευή της εικονικής μηχανής το LibVirt αντλεί τα χαρακτηριστικά που πρέπει να ικανοποιούνται (π.χ. αριθμός επεξεργαστών, μέγεθος μνήμης, δίσκου κλπ) μέσα από ένα XML configuration αρχείο. Το αρχείο αυτό είτε αρχικοποιείται από τον χρήστη, είτε κατασκευάζεται αυτόματα δίνοντας κατάλληλες παραμέτρους σε εντολή (virt-install) που χρησιμοποιείται για την δημιουργία νέας εικονικής μηχανής.

Ακολουθεί το XML αρχείο που αφορά την πρώτη εικονική μηχανή, κατά την οποία οι επεξεργαστές και η μνήμη αντιστοιχίζονται σε φυσικούς κάτω απ΄ το ίδιο NUMA node.

```
<domain type='kvm'>
  <name>gdimitrakop-vm</name>
  <memory unit='GiB'>64</memory>
  <vcpu placement='static'>4</vcpu>
  <cputune>
    <vcpupin vcpu='0' cpuset='0'/>
    <vcpupin vcpu='1' cpuset='1'/>
    <vcpupin vcpu='2' cpuset='2'/>
    <vcpupin vcpu='3' cpuset='3'/>
  </cputune>
  <numatune>
    <memory mode="strict" nodeset="0"/>
  </numatune>
  <os>
    <type arch='x86_64'>hvm</type>
    <boot dev='cdrom'/>
  </os>
  <clock offset='utc'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2'/>
      <source file='/local/gdimitrakop/vm/root.qcow2.img'/>
```

```
26        <target dev='vda' bus='virtio'/>
27      </disk>
28      <interface type='bridge'>
29        <mac address='00:00:00:33:22:10'/>
30        <source bridge='virbr0'/>
31        <model type='virtio'/>
32      </interface>
33      <serial type='pty'>
34        <target port='0'/>
35      </serial>
36      <console type='pty'>
37        <target type='serial' port='0'/>
38      </console>
39      <memballoon model='virtio'>
40        <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
41      </memballoon>
42      <graphics type='vnc' port='-1' autoport='yes' keymap='en-us'/>
43    </devices>
44  </domain>
```

Στο παραπάνω αρχείο οι γραμμές που αξίζουν προσοχή είναι οι 5-13. Εκεί ορίσαμε την αντιστοίχηση εικονικών - φυσικών επεξεργαστών, και κατόπιν ορίσαμε τη μνήμη να βρίσκεται «αυστηρά» στο ίδιο node με τους επεξεργαστές (node 0).

Ακολουθούν το αντίστοιχο αρχείο για το δεύτερο VM που υλοποιήθηκε για την παρούσα εργασία, και τέλος το ίδιο και για το τρίτο. Η λογική είναι η ίδια που περιγράφηκε και παραπάνω, και οι μόνες διαφορές των αρχείων εντοπίζονται στις γραμμές 5-13.

```
1   <domain type='kvm'>
2     <name>gdimitrakop-vm</name>
3     <memory unit='GiB'>64</memory>
4     <vcpu placement='static'>4</vcpu>
5     <cputune>
6       <vcpupin vcpu='0' cpuset='0'/>
7       <vcpupin vcpu='1' cpuset='1'/>
8       <vcpupin vcpu='2' cpuset='24'/>
9       <vcpupin vcpu='3' cpuset='25'/>
10    </cputune>
11    <numatune>
12      <memory mode="strict" nodeset="3,2"/>
13    </numatune>
14    <os>
15      <type arch='x86_64'>hvm</type>
16      <boot dev='cdrom'/>
17    </os>
18    <clock offset='utc'/>
19    <on_poweroff>destroy</on_poweroff>
20    <on_reboot>restart</on_reboot>
21    <on_crash>restart</on_crash>
```

```
  <devices>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2'/>
      <source file='/local/gdimitrakop/vm/root.qcow2.img'/>
      <target dev='vda' bus='virtio'/>
    </disk>
    <interface type='bridge'>
      <mac address='00:00:00:33:22:10'/>
      <source bridge='virbr0'/>
      <model type='virtio'/>
    </interface>
    <serial type='pty'>
      <target port='0'/>
    </serial>
    <console type='pty'>
      <target type='serial' port='0'/>
    </console>
    <memballoon model='virtio'>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
    </memballoon>
    <graphics type='vnc' port='-1' autoport='yes' keymap='en-us'/>
  </devices>
</domain>
```

```
<domain type='kvm'>
  <name>gdimitrakop-vm</name>
  <memory unit='GiB'>64</memory>
  <vcpu placement='static'>4</vcpu>
  <cputune>
    <vcpupin vcpu='0' cpuset='0'/>
    <vcpupin vcpu='1' cpuset='1'/>
    <vcpupin vcpu='2' cpuset='2'/>
    <vcpupin vcpu='3' cpuset='3'/>
  </cputune>
  <numatune>
    <memory mode="strict" nodeset="3,2"/>
  </numatune>
  <os>
    <type arch='x86_64'>hvm</type>
    <boot dev='cdrom'/>
  </os>
  <clock offset='utc'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2'/>
      <source file='/local/gdimitrakop/vm/root.qcow2.img'/>
```

```
26        <target dev='vda' bus='virtio'/>
27      </disk>
28      <interface type='bridge'>
29        <mac address='00:00:00:33:22:10'/>
30        <source bridge='virbr0'/>
31        <model type='virtio'/>
32      </interface>
33      <serial type='pty'>
34        <target port='0'/>
35      </serial>
36      <console type='pty'>
37        <target type='serial' port='0'/>
38      </console>
39      <memballoon model='virtio'>
40        <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
41      </memballoon>
42      <graphics type='vnc' port='-1' autoport='yes' keymap='en-us'/>
43    </devices>
44  </domain>
```

Βιβλιογραφία

[1] *Public cloud providers: Amazon's ec2, microsoft azure, google's compute engine.* https://aws.amazon.com/ec2/, https://azure.microsoft.com/, https://cloud.google.com/compute/.

[2] *Private cloud frameworks: Vmware's esxi and openstack.* https://www.vmware.com/products/esxi-and-esx, https://www.openstack.org/.

[3] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, *Mesos: A platform for fine-grained resource sharing in the data center*, Boston, MA, 2011.

[4] *J hamilton. cost of power in large-scale data centers.* [Online]. Available: http://perspectives.mvdirona.com.

[5] L. A. Barroso, *Warehouse-scale computing: Entering the teenage decade*, 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=2019527.

[6] L. A. Barroso, J. Clidaras, and U. Hölzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second Edition*. 2013. [Online]. Available: http://dx.doi.org/10.2200/S00516ED2V01Y201306CAC024.

[7] I. Ali and N. Meghanathan, "Virtual machines and networks - installation, performance, study, advantages and virtualization options", *CoRR*, vol. abs/1105.0061, Jan. 2011. DOI: 10.5121/ijnsa.2011.3101.

[8] A. Rashid and A. Chaturvedi, *Virtualization and its role in cloud computing environment*, Apr. 2019. DOI: 10.26438/ijcse/v7i4.11311136.

[9] A. Qumranet, Y. Qumranet, D. Qumranet, U. Qumranet, and A. Liguori, "Kvm: The linux virtual machine monitor", *Proceedings Linux Symposium*, vol. 15, Jan. 2007.

[10] E. Z. Zhang, Y. Jiang, and X. Shen, "Does cache sharing on modern cmp matter to the performance of contemporary multithreaded programs?", *SIGPLAN Not.*, vol. 45, no. 5, pp. 203–212, Jan. 2010. DOI: 10.1145/1837853.1693482. [Online]. Available: https://doi.org/10.1145/1837853.1693482.

[11] H. Kang and J. L. Wong, "To hardware prefetch or not to prefetch? a virtualized environment study and core binding approach", *SIGARCH Comput. Archit. News*, vol. 41, no. 1, pp. 357–368, Mar. 2013. DOI: 10.1145/2490301.2451155. [Online]. Available: https://doi.org/10.1145/2490301.2451155.

[12] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious", *SIGARCH Comput. Archit. News*, vol. 23, no. 1, pp. 20–24, Mar. 1995. DOI: 10.1145/216585.216588. [Online]. Available: https://doi.org/10.1145/216585.216588.

[13] B. Veal and A. Foong, "Performance scalability of a multi-core web server", Jan. 2007, pp. 57–66. DOI: 10.1145/1323548.1323562.

[14] Z. Majo and T. Gross, "Memory management in numa multicore systems: Trapped between cache contention and interconnect overhead", vol. 46, Nov. 2011, pp. 11–20. DOI: 10.1145/2076022.1993481.

[15] M. Awasthi, D. W. Nellans, K. Sudan, R. Balasubramonian, and A. Davis, "Handling the problems and opportunities posed by multiple on-chip memory controllers", in *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '10, Vienna, Austria: Association for Computing Machinery, 2010, pp. 319–330. DOI: 10.1145/1854273.1854314. [Online]. Available: https://doi.org/10.1145/1854273.1854314.

[16] L. Tang, J. Mars, N. Vachharajani, R. Hundt, and M. L. Soffa, "The impact of memory subsystem resource sharing on datacenter applications", in *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2011, pp. 283–294.

[17] Dhruba Chandra, Fei Guo, Seongbeom Kim, and Yan Solihin, "Predicting inter-thread cache contention on a chip multi-processor architecture", in *11th International Symposium on High-Performance Computer Architecture*, Feb. 2005, pp. 340–351. DOI: 10.1109/HPCA.2005.27.

[18] Y. Cheng and W. Chen, "Evaluation of virtual machine performance on numa multicore systems", in *2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, Oct. 2013, pp. 136–143. DOI: 10.1109/3PGCIC.2013.27.

[19] S. Blagodurov, A. Fedorova, S. Zhuravlev, and A. Kamali, "A case for numa-aware contention management on multicore systems", in *2010 19th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Sep. 2010, pp. 557–558.

[20] B. Lepers, "Improving performance on numa systems", PhD thesis, Jan. 2014.

[21] M. Dashti, A. Fedorova, J. Funston, F. Gaud, R. Lachaize, B. Lepers, V. Quéma, and M. Roth, "Traffic management: A holistic approach to memory placement on numa systems", vol. 48, Apr. 2013, pp. 381–394. DOI: 10.1145/2499368.2451157.

[22] S. Novakovic, A. Daglis, B. R. Grot, E. Bugnion, and B. Falsafi, "Scale-out non-uniform memory access", 2015. [Online]. Available: http://infoscience.epfl.ch/record/228381.

[23] A. M. Devices, "Amd's hypertransport technology white paper", 2001. [Online]. Available: https://25867bd8-75a6-40d7-a851-331c4fcdbd99.filesusr.com/ugd/071cb6_5fe1f8d4c3b343799a9604568ca47c6d.pdf.

[24] F. Bellard, "Qemu, a fast and portable dynamic translator.", Jan. 2005, pp. 41–46.

[25] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the clouds: A study of emerging scale-out workloads on modern hardware", *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2012. [Online]. Available: http://infoscience.epfl.ch/record/173764.

[26] T. Palit, Y. Shen, and M. Ferdman, "Demystifying cloud benchmarking", Apr. 2016, pp. 122–132. DOI: 10.1109/ISPASS.2016.7482080.

[27] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters", *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. DOI: 10.1145/1327452.1327492. [Online]. Available: https://doi.org/10.1145/1327452.1327492.

[28] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb", in *Proceedings of the 1st ACM Symposium on Cloud Computing*, ser. SoCC '10, Indianapolis, Indiana, USA: Association for Computing Machinery, 2010, pp. 143–154. DOI: 10.1145/1807128.1807152. [Online]. Available: https://doi.org/10.1145/1807128.1807152.

[29] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web.", Stanford InfoLab, Technical Report 1999-66, Nov. 1999, Previous number = SIDL-WP-1999-0120. [Online]. Available: http://ilpubs.stanford.edu:8090/422/.

[30] C. Bienia, "Benchmarking modern multiprocessors", PhD thesis, Princeton University, Jan. 2011.

[31] C. Bienia and K. Li, "Parsec 2.0: A new benchmark suite for chip-multiprocessors", in *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, Jun. 2009.

[32] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications", in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, Oct. 2008.

[33] F. Black and M. Scholes, "The pricing of options and corporate liabilities", *Journal of political economy*, vol. 81, no. 3, p. 637, 1973.

[34] *Parsec benchmark suite documentation and help manual*, https://parsec.cs.princeton.edu/download/tutorial/2.0/parsec-2.0-tutorial.pdf.