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

Σχεδιασμός και Υλοποίηση Resource Manager για NUMA Υπολογιστικά Συστήματα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΘΕΟΔΩΡΟΣ Α. ΒΑΚΑΛΟΠΟΥΛΟΣ

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

Αθήνα, Ιούλιος 2020

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

Σχεδιασμός και Υλοποίηση Resource Manager για NUMA Υπολογιστικά Συστήματα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΘΕΟΔΩΡΟΣ Α. ΒΑΚΑΛΟΠΟΥΛΟΣ

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

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 16η Ιουλίου 2020.

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

Αθήνα, Ιούλιος 2020

Θεόδωρος Α. Βακαλόπουλος

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

Περίληψη

Σε αυτή τη διπλωματική εργασία παρουσιάζουμε μία πρακτική προσέγγιση για το πρόβλημα της δυναμικής τοποθέτησης της μνήμης των εφαρμογών σε ένα NUMA σύστημα. Η NUMA αρχιτεκτονική κυριαρχεί στα σύγχρονα πολυπεξεργαστικά συστήματα λόγω της κλιμακωσιμότητας της επεξεργαστικής ισχύος και του εύρους ζώνης της μνήμης. Αυτό αποτελεί σημαντικό πλεονέκτημα για την ταυτόχρονη εκτέλεση πολλών εφαρμογών αλλά εισάγει και σημαντικές καθυστερήσεις που εξαρτώνται από τον τρόπο που γίνεται η τοποθέτηση των εφαρμογών στο σύστημα. Ο αντίκτυπος στην απόδοση σχετίζεται με την αρχιτεκτονική του συστήματος και τα χαρακτηριστικά των εφαρμογών. Αρχικά, εκτελέσαμε μια μεγάλη ποικιλία από εφαρμογές από τη σουίτα SPEC 2017 κάτω από διαφορετικά σενάρια τοποθέτησης και λάβαμε μετρήσεις από τους hardware performance counters που είναι ενσωματωμένοι στο σύστημα. Στη συνέχεια, χρησιμοποιήσαμε τα πειραματικά δεδομένα για να εξάγουμε κάποια συμπεράσματα αναφορικά με τον τρόπο που τα χαρακτηριστικά των εφαρμογών επηρεάζουν την απόδοσή τους. Συνδυάζοντας αυτά τα αποτελέσματα με ιδέες από σχετικές ερευνητικές εργασίες αναπτύξαμε ένα μοντέλο για τη βέλτιστη επιλογή εφαρμογών για μεταφορά μνήμης μεταξύ των κόμβων ενός NUMA συστήματος. Κατόπιν, υλοποιήσαμε ένα πρόγραμμα resource manager επιπέδου χρήστη, που βασίζεται σε αυτό το μοντέλο και αναλαμβάνει να εκτελεί με βέλτιστο τρόπο εντολές μεταφοράς μνήμης. Η αξιολόγηση του resource manager και η σύγκριση του με άλλες προσεγγίσεις αποτυπώνει το πλεονέκτημα που προσφέρει η χρήση του αναφορικά με την αύξηση της απόδοσης των NUMA συστημάτων και τη βέλτιστη αξιοποίηση των διαθέσιμων πόρων τους.

Λέξεις κλειδιά

NUMA αρχιτεκτονικές, Διαχείριση πόρων, Επίδοση, Τοποθέτηση εφαρμογών, Δυναμική μεταφορά μνήμης, Μοντελοποίηση, SPEC 2017.

Abstract

In this diploma thesis we present a practical approach to the problem of application dynamic memory placement in a NUMA system. The NUMA architecture dominates in modern multiprocessor systems due to the scalability of processing power and memory bandwidth. This constitutes an important advantage for parallel execution of multiple applications, but it also introduces significant overheads that depend on the placement of applications on the system. The impact on performance is correlated with the system architecture and the characteristics of the applications. First of all, we executed a wide variety of applications from the SPEC 2017 suite under different placement scenarios and obtained measurements from the system's hardware performance counters. We then used the experimental data to draw some conclusions about the way the characteristics of the applications affect their performance. Combining these results with ideas from related research work we developed a model for optimal memory migration between the nodes of a NUMA system. Then we implemented a user-level resource manager program, which relies on this model and undertakes to execute memory migration commands in an optimal way. The evaluation of the resource manager and its comparison with other approaches reflects the advantage that its use offers in terms of improving the performance of NUMA systems and the utilization of their available resources.

Key words

NUMA architectures, Resource management, Performance, Application Placement, Dynamic memory migration, Modelling, SPEC 2017.

Ευχαριστίες

Η ολοκλήρωση της παρούσας διπλωματικής εργασίας σηματοδοτεί το πέρας των σπουδών μου στο Εθνικό Μετσόβιο Πολυτεχνείο. Στο σημείο αυτό θα ήθελα να ευχαριστήσω τα άτομα που με βοήθησαν όλα αυτά τα χρόνια να πετύχω τους στόχους μου και να διαμορφώσω την προσωπικότητά μου. Αρχικά, ευχαριστώ απο καρδιάς την οικογένειά μου που με υποστήριξε με κάθε τρόπο κατά τη διάρκεια των σπουδών μου και αξίζει κάθε έπαινο για ότι έχω καταφέρει στην ακαδημαϊκή, και όχι μόνο, πορεία μου. Επιπλέον, θα ήθελα να εκφράσω την ευγνωμωσύνη μου προς τους όλους τους καθηγητές της σχολής των Ηλεκτρολόγων Μηχανικών και Μηχανικών Η/Υ για την απρόσκοπτη συνεργασία και τις πολύτιμες γνώσεις που απλόχερα μου μεταλαμπάδευσαν. Ειδικότερα, αξίζει να αναφερθώ στον επιβλέποντα καθηγητή της παρούσας διπλωματικής κ. Γκούμα για την πολύτιμη βοήθεια και καθοδήγησή του, καθώς και στους κ. Κοζύρη και κ. Παπασπύρου, οι οποίοι συνέβαλαν ο καθένας με τον δικό του τρόπο στην καλλιέργεια του πνεύματός μου και στην μετάδοση του πάθους για ενασχόληση με την Επιστήμη των Υπολογιστών. Θα ήθελα, ακόμη, να ευχαριστήσω θερμά όλα τα μέλη του CSLab και ιδιαίτερα τον Βασίλη Καρακώστα που ήταν μέντορας μου καθόλη της διάρκεια της έρευνας και με τις εύστοχες παρατηρήσεις και επισημάνσεις του συνέβαλε καθοριστικά στο τελικό αποτέλεσμα. Τέλος, ένα μεγάλο ευχαριστώ στους φίλους μου που ήταν δίπλα μου κατά τη διάρκεια των σπουδών μου και με βοήθησαν να γίνω πιο ολοκληρωμένος άνθρωπος κάνοντας αξέχαστη την εμπειρία μου ως φοιτητής.

Θεόδωρος Α. Βακαλόπουλος,

Αθήνα, 16η Ιουλίου 2020

Περιεχόμενα

Κατάλογος σχημάτων

Κεφάλαιο 1

Εισαγωγή

1.1 Σκοπός της εργασίας

Τα σύγχρονα πολυεπεξεργαστικά συστήματα έχουν υιοθετηθεί ευρέως στις υποδομές υπολογισμού υψηλών επιδόσεων (High Performance Computing) λόγω των αυξημένων δυνατοτήτων που προσφέρουν για την ανάλυση και επεξεργασία μεγάλου όγκου δεδομένων και την ταυτόχρονη εξυπηρέτηση μεγάλου αριθμού χρηστών. Τα συστήματα αυτά χρησιμοποιούν σε μεγάλο βαθμό την τεχνική της εικονικοποίησης προκειμένου να μπορούν να υποστηρίξουν πολλές εικονικές μηχανές που τρέχουν στο ίδιο φυσικό μηχάνημα, καθεμιά από τις οποίες τρέχει τις δικές της εφαρμογές που ανήκουν σε ανεξάρτητους χρήστες. Προκύπτει, λοιπόν, μια ποικιλία εφαρμογών με διαφορετικά χαρακτηριστικά που πρέπει να τρέξουν στο ίδιο σύστημα όσο το δυνατόν πιο αποδοτικά.

Η NUMA (Non Uniform Memory Access) αρχιτεκτονική έχει κυριαρχήσει σε τέτοια πολυεπεξεργαστικά συστήματα λόγω της κλιμακωσιμότητας του εύρους ζώνης της μνήμης που προσφέρει. Ωστόσο, η NUMA αρχιτεκτονική εισάγει επιπρόσθετες επιβαρύνσεις, όπως είναι η καθυστέρηση πρόσβασης σε απομακρυσμένη μνήμη (remote access latency) και η συμφόρηση των διαύλων της μνήμης (memory traffic congestion) [Lame13]. Οι καθυστερήσεις αυτές μπορούν να υποβαθμίσουν σημαντικά την απόδοση των εφαρμογών και να οδηγήσουν σε ελλειπή αξιοποίση των διαθέσιμων πόρων [Gaud15]. Επιπλέον, δεδομένου ότι οι εφαρμογές τρέχουν γενικά σε εικονικοποιημένο περιβάλλον, δεν έχουν επίγνωση των NUMA χαρακτηριστικών του μηχανήματος που εκτελούνται με αποτέλεσμα να μην μπορεί να εγγυηθεί η απόδοσή τους χωρίς κατάλληλη εξωτερική διαχείριση που λαμβάνει υπόψιν τα συγκεκριμένα NUMA χαρακτηριστικά.

Αν και μέχρι σήμερα έχουν προταθεί αρκετές λύσεις που επιχειρούν να αντιμετωπίσουν τις καθυστερήσεις στα NUMA συστήματα, αυτές χαρακτηρίζονται από σημαντικούς περιορισμούς. Ορισμένες προσεγγίσεις [Arap18, McCo11, Wang16, Su12, Luo16] είναι στατικές και προσπαθούν να βελτιστοποιήσουν την απόδοση των εφαρμογών μέσω μιας αρχικής τοποθέτησής τους. Αυτό το επιτυγχάνουν κατασκευάζοντας μοντέλα για την πρόβλεψη των καθυστερήσεων που παρατηρούνται στα σύγχρονα NUMA συστήματα και κατόπιν εφαρμόζοντας αυτά τα μοντέλα για συγκεκριμένο workload και αρχιτεκτονικά χαρακτηριστικά. Έτσι, κάθε μοντέλο παράγει ένα στατικό αποτέλεσμα για να τοποθετηθούν τα νήματα και η μνήμη των εφαρμογών στο σύστημα πριν την εκτέλεσή τους όσο πιο αποδοτικά γίνεται. Αυτές οι προσεγγίσεις είναι αποδοτικές για εφαρμογές με ομοιόμορφα μοτίβα πρόσβασης στη μνήμη, αλλά λιγότερο αποτελεσματικές για τις εφαρμογές εκείνες που παρουσιάζουν ακανόνιστη συμπεριφορά αναφορικά με την πρόσβαση στα δεδομένα τους, όπως συμβαίνει στην πλειοψηφία των περιπτώσεων.

Ακόμη, άλλες προσεγγίσεις [Qian19, Dash13, DGur20, Lepe15, Funs18, Kotr17] προσπαθούν να ελέγχουν την κατανομή των πόρων στις εφαρμογές δυναμικά κατά τη διάρκεια της εκτέλεσής τους. Αυτό το επιτυγχάνουν συλλέγοντας διαρκώς δεδομένα από το σύστημα και τις εφαρμογές και στη συνέχεια εφαρμόζοντας ευρετικές μεθόδους και άλλες πιο πολύπλοκες τεχνικές για την αλλαγή της τοπολογίας των εφαρμογών όταν κρίνεται απαραίτητο ώστε να βελτιωθεί η απόδοση. Ωστόσο, κοινό χαρακτηριστικό όλων αυτών των ερευνών είναι ότι επικεντρώνονται να βελτιστοποιήσουν την απόδοση μιας εφαμοργής που τρέχει σε ένα NUMA περιβάλλον [DGur20] ή αντιμετωπίζουν το πρόβλημα σε επίπεδο πυρήνα [Dash13, Funs18, Kotr17, Lepe15] απαιτώντας παρεμβατικές τροποποιήσεις στο λειτουργικό σύστημα. Επιπρόσθετα, κάποιες προσεγγίσεις χρησιμοποιούν εξειδικευμένους μετρητές

που υπάρχουν στο υλικό κάποιων μηχανημάτων και επομένως εξαρτώνται άμεσα από την πλατφόρμα εκτέλεσης. Με άλλα λόγια αυτές οι λύσεις πάσχουν από το πρόβλημα της μεταφερσιμότητας διότι δεν μπορούν να εφαρμοστούν με ενιαίο τρόπο στην πλειονότητα των NUMA συστημάτων που υπάρχουν.

Στην παρούσα διπλωματική εργασία εστιάζουμε σε απλά αλλά πρακτικά μοντέλα και επικεντρωνόμαστε στην υλοποίηση μίας προσέγγισης σε επίπεδο χώρου χρήστη για την αντιμετώπιση του προβλήματος της κατανομής μνήμης των εφαρμογών με αποδοτικό τρόπο σε ένα NUMA σύστημα, η οποία δεν εμφανίζει τους περιορισμούς που προαναφέραμε. Η λύση που προτείνουμε είναι δυναμική δηλαδή τρέχει ταυτόχρονα με τις εφαρμογές και λαμβάνει αποφάσεις on-the-fly, καθώς και μεταφέρσιμη δεδομένου ότι χρησιμοποιεί κοινούς μετρητές υλικού (hardware performance counters) που υπάρχουν στα περισσότερα NUMA συστήματα. Πρόκειται ουσιαστικά για ένα πρόγραμμα που τρέχει σε επίπεδο χρήστη και αναλαμβάνει δυναμικά κατά τη διάρκεια της εκτέλεσης των εφαρμογών να ικανοποιεί αιτήματα μεταφοράς μνήμης μεταξύ των κόμβων του NUMA συστήματος. Τα αιτήματα αυτά θεωρούμε ότι προέρχονται από κάποο εξωτερικό παράγοντα, όπως για παράδειγμα από το χρήστη ή από κάποιο άλλο πρόγραμμα με κατάλληλα δικαιώματα, και εκτελούνται με τέτοιο τρόπο ώστε να μην υποβαθμίζεται η απόδοση των εφαρμογών. Στην εργασία αυτή ασχολούμαστε για απλότητα με native εκτελέσεις των εφαρμογών χωρίς να παρεμβάλλεται το στάδιο της εικονικοποίησης. Τα αποτελέσματα που παίρνουμε ωστόσο μπορούν εύκολα να γενικευτούν και στα σύχρονα NUMA συστήματα όπου οι εφαρμογές εκτελούνται σε εικονικό περιβάλλον. Η γενίκευση αυτή γίνεται εύκολα θεωρώντας ότι οι εφαρμογές που εκτελούμε στα πειράματα αντιστοιχούν με τις εικονικές μηχανές (VMs) που υπάρχουν σε ένα NUMA σύστημα.

Συνοψίζοντας, οι συνεισφορές της διπλωματικής εργασίας μπορούν να συνοψιστούν ως εξής:

- Μελέτη των benchmarks της σουίτας SPEC 2017 και εκτέλεσή τους σε διάφορα σενάρια και τοπολογίες.
- Δημιουργία ενός μοντέλου βελτίωσης της απόδοσης του συστήματος Sandman όταν τρέχουν ταυτόχρονα πολλές εφαρμογές.
- Υλοποίηση και αξιολόγηση ενός προγράμματος επιπέδου χρήστη, του resource manager, που βασίζεται σε αυτό το μοντέλο και εκτελεί δυναμικά εντολές μεταφοράς σελίδων μνήμης στο σύστημα.

1.2 Δομή της εργασίας

Η δομή των κεφαλαίων της διπλωματικής εργασίας είναι η ακόλουθη:

Κεφάλαιο 2: Θεωρητικό υπόβαθρο
Στο κεφάλαιο αυτό δίνουμε αρχικά ορισμένους βασικούς ορισμούς σχετικά με τα NUMA συστήματα και αναπτύσσουμε έννοιες που χρησιμοποιήσαμε στην πορεία της έρευνας. Περιγράφουμε ακόμη σχετικές ερευνητικές εργασίες γύρω από το πρόβλημα του NUMA placement τονίζοντας τους περιορισμούς της καθεμιάς και το κενό που πρόκειται να καλύψει η παρούσα διπλωματική εργασία.

Κεφάλαιο 3: Μεθοδολογία
Στο κεφάλαιο αυτό αναλύουμε την αρχιτεκτονική του μηχανήματος Sandman που χρησιμοποιήσαμε για την διεξαγωγή των πειραμάτων. Στη συνέχεια, περιγράφουμε τα εργαλεία που χρησιμοποιήσαμε σε διάφορα σημεία της έρευνάς μας.

Κεφάλαιο 4: Κίνητρο Εργασίας
Σε αυτό το κεφάλαιο περιγράφουμε τα benchmarks της σουίτας SPEC 2017 που χρησιμοποιήσαμε στην έρευνά μας και τα διάφορα πειράματα που εκτελέσαμε με αυτά. Στη συνέχεια σχολιάζουμε τα αποτελέσματα που λάβαμε και το κίνητρο που δημιουργήθηκε και μας οδήγησε προς την υλοποίηση του resource manager.

Κεφάλαιο 5: Υλοποίηση του Resource Manager
Στο κεφάλαιο αυτό περιγράφουμε το κύριο προϊόν αυτής της έρευνας, τον resource manager που κατασκευάσαμε. Αναλύουμε τα δομικά του στοιχεία, το μοντέλο που αναπτύξαμε και χρησιμοποιεί και τις βασικές λειτουργίες που επιτελεί.

Κεφάλαιο 6: Αξιολόγηση
Στο κεφάλαιο αυτό περιγράφουμε τα πειράματα που εκτελέσαμε για να ελέγξουμε την ορθότητα και την αποτελεσματικότητα του resource manager. Στη συνέχεια, αξιολογούμε την απόδοση του μοντέλου του σε σύγκριση με άλλες προσεγγίσεις ώστε να διαπιστώσουμε το πλεονέκτημα που προσφέρει η χρήση του στα σύγχρονα NUMA συστήματα.

Κεφάλαιο 7: Επίλογος
Στο κεφάλαιο αυτό ολοκληρώνεται η παρούσα διπλωματική εργασία. Παρουσιάζουμε ορισμένα γενικότερα συμπεράσματα που προέκυψαν από την έρευνα και αναφέρουμε πιθανές μελλοντικές επεκτάσεις που έχουν ενδιαφέρον.

Κεφάλαιο 2

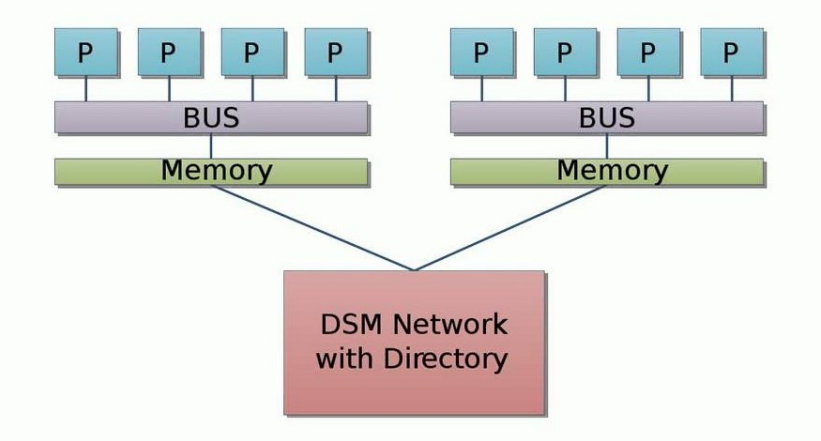
Θεωρητικό υπόβαθρο

Στο κεφάλαιο αυτό γίνεται μια σύντομη παρουσίαση των εννοιών που αφορούν στα Non-Uniform Memory Access (NUMA) συστήματα. Αρχικά δίνονται κάποιοι βασικοί ορισμοί που χρησιμποιούνται στην πορεία της έρευνας και στη συνέχεια αναλύεται η δομή και τα χαρακτηριστικά τους.

2.1 Βασικές έννοιες

Με τον όρο Non-Uniform Memory Access (NUMA) αναφερόμαστε σε ένα αρχιτεκτονικό μοντέλο μνήμης που χρησιμποιείται στην πολυεπεξεργασία κατά το οποίο ο χρόνος πρόσβασης στη μνήμη εξαρτάται από την σχετική θέση της μνήμης ως προς τον επεξεργαστή. Με άλλα λόγια, ένας επεξεργαστής μπορεί να έχει πρόσβαση στην τοπική του μνήμη γρηγορότερα σε σχέση με το να χρησιμοποιεί μια απομακρυσμένη (μη τοπική) μνήμη δεδομένου ότι η αυξανόμενη απόσταση μεταξύ επεξεργαστή και μνήμης δημιουργεί καθυστερήσεις που οφείλονται στον δίαυλο επικοινωνίας.

Τα NUMA συστήματα είναι μια ειδική κατηγορία πολυεπεξεργαστικών συστημάτων. Αποτελούνται συνήθως από αρκετούς πολυνηματικούς επεξεργαστές και πολλαπλές διακριτές μνήμες RAM που βρίσκονται διασκορπισμένες τοπολογικά σε όλη της έκτασή τους. Στο σχήμα 2.1 φαίνεται η τυπική αρχιτεκτονική ενός NUMA συστήματος. Τέτοια συστήματα συναντώνται συχνά σε εφαρμογές που απαιτούν κλιμακωσιμότητα και κατά συνέπεια έχουν μεγάλες απαιτήσεις από το υλικό όπως σε data centers και ερευνητικά κέντρα. Η πιο διαδεδομένη χρήση τους είναι σε διακομιστές (servers) καθώς προσφέρουν την αναγκαία υποδομή για την υποστήριξη της ταυτόχρονης επεξεργασίας μεγάλου όγκου αιτημάτων και δεδομένων από πολλαπλούς χρήστες.



Σχήμα 2.1: Αρχιτεκτονική ενός NUMA συστήματος.

Τα κύρια στοιχεία από τα οποία αποτελείται ένα NUMA σύστημα είναι οι κόμβοι (nodes), η κύρια μνήμη (RAM) και το δίκτυο διασύνδεσης.

Κόμβοι Κόμβος ενός NUMA συστήματος είναι μια διακριτή ομάδα από επεξεργαστές οι οποίοι είναι συνδεδεμένοι πάνω στο ίδιο socket. Ο αριθμός των κόμβων αποτελεί ένα από τα πιο βασικά

χαρακτηριστικά ενός NUMA συστήματος και είναι μεγαλύτερος από 1.

Κύρια μνήμη (RAM) Η κύρια μνήμη στα NUMA συστήματα δεν είναι συνεχόμενη όπως συμβαίνει στα κλασικά υπολογιστικά συστήματα αλλά είναι κατανεμημένη συνήθως ισόποσα στους κόμβους του συστήματος. Αυτή η ιδιαιτερότητα ως προς τον τρόπο οργάνωσης της μνήμης, η οποία και ξεχωρίζει τα NUMA συστήματα από τα υπόλοιπα υπολογιστικά συστήματα, δημιουργεί μια ανομοιομορφία ως προς το χρόνο πρόσβασης στη μνήμη και το διαθέσιμο bandwidth.

Κάθε κόμβος έχει ένα κομμάτι μνήμης που είναι κοντά του και ονομάζεται τοπική μνήμη (local memory) ενώ κάθε άλλη μνήμη που υπάρχει σε κάποιο άλλο κόμβο του συστήματος ονομάζεται απομακρυσμένη μνήμη (remote memory). Ο χαρακτηρισμός μιας μνήμης ως τοπική ή απομακρυσμένη δεν είναι μονοσήμαντος αλλά εξαρτάται από τη σχετική θέση της μνήμης και του κόμβου αναφοράς. Κάθε κόμβος διαθέτει επιπλέον έναν ελεγκτή για την κύρια μνήμη (memory controller), ο οποίος είναι υπεύθυνος για τη σωστή οργάνωση και λειτουργία της όσον αφορά στις προσβάσεις των εφαρμογών.

Η διάκριση των μνημών με βάση τη σχετική θέση τους ως προς τους κόμβους του συστήματος σχετίζεται άμεσα με το χρόνο πρόσβασης σε αυτές. Ένας επεξεργαστής χρειάζεται πάντα μικρότερο χρόνο για να κάνει μία πρόσβαση στην τοπική του μνήμη σε σχέση με τον χρόνο που χρειάζεται για την ίδια πρόσβαση σε μία απομαρκυσμένη. Ακόμη, ο χρόνος προσπέλασης μιας απομακρυσμένης μνήμης δεν είναι σταθερός αλλά ποικίλει ανάλογα με την απόσταση της μνήμης από τον κόμβο αναφοράς.

Δίκτυο διασύνδεσης Το δίκτυο διασύνδεσης αποτελεί ένα χαρακτηριστικό των NUMA συστημάτων εξίσου σημαντικό με τους κόμβους, καθώς καθορίζει τον τρόπο με τον οποίο είναι διασυνδεδεμένοι μεταξύ τους μέσα στο σύστημα. Κάθε κόμβος του συστήματος πρέπει να είναι σε θέση να επικοινωνήσει με οποιονδήποτε άλλο κόμβο είτε απευθείας είτε μέσω άλλων κόμβων. Τα δίκτυα διασύνδεσης μπορεί να διαφέρουν από σύστημα σε σύστημα δημιουργώντας πολλούς συνδυασμούς καθώς εξαρτώνται από τον αριθμό τον κόμβων και τον τρόπο επικοινωνίας τους.

Η ειδοποιός διαφορά των NUMA συστημάτων με τα υπόλοιπα πολυεπεξεργαστικά συστήματα είναι ο ανοιμοιόμορφος χρόνος πρόσβασης στη μνήμη από τους κόμβους. Για την πρόσβαση σε μία απομακρυσμένη μνήμη απαιτείται η επικοινωνία μεταξύ δύο ή περισσότερων κόμβων μέσω του δικτύου διασύνδεσης. Απόσταση μεταξύ δύο κόμβων ορίζεται ο χρόνος που χρειάζεται για να επικοινωνήσουν μέσω της πιο σύντομης διαδρομής του δικτύου διασύνδεσης. Η απόσταση μεταξύ δύο κόμβων έχει κατεύθυνση και επομένως δεν ορίζεται μονοσήμαντα. Για παράδειγμα, ένας επεξεργαστής που βρίσκεται στον κόμβο Α είναι δυνατόν να "βλέπει" σε διαφορετική απόσταση μία απομακρυσμένη ως προς αυτόν μνήμη που βρίσκεται στον κόμβο Β σε σχέση με την απόσταση που "βλέπει" ένας επεξεργαστής στον κόμβο Β μία μνήμη του κόμβου Α. Σε περίπτωση που δύο κόμβοι "βλέπουν" την ίδια απόσταση ο ένας για την μνήμη που βρίσκεται στον άλλο τότε λέμε ότι αυτή η απόσταση είναι συμμετρική. Σε αντίθετη περίπτωση η απόσταση ονομάζεται ασύμμετρη. Επομένως, με πιο αυστηρό συμβολισμό λέμε ότι μια απόσταση είναι συμμετρική αν και μόνο αν

$$\forall A, B \in Nodes, Distance(Processor(A), Memory(B)) = Distance(Processor(B), Memory(A))$$

.

Τα NUMA συστήματα μπορούν να διαχωρίστουν σε δύο βασικές κατηγορίες, τα συμμετρικά και τα ασύμμετρα. Ένα NUMA σύστημα χαρακτηρίζεται ως συμμετρικό αν όλες οι αποστάσεις μεταξύ των κόμβων είναι συμμετρικές. Αν υπάρχει έστω και μία ασύμμετρη απόσταση τότε το σύστημα χαρακτηρίζεται ως ασύμμετρο. Στην παρούσα διπλωματική εργασία όλα τα πειράματα πραγματοποιήθηκαν σε συμμετρικό μηχάνημα ώστε να ελαχιστοποιηθεί η πολυπλοκότητα που εισάγεται από αυτό τον παράγοντα.

2.2 Σχετική έρευνα

Η NUMA αρχιτεκτονική έχει κυριαρχήσει στα πολυεπεξεργαστικά συστήματα λόγω της κλιμακωσιμότητας που προσφέρει ως προς το εύρος ζώνης της μνήμης (memory bandwidth). Η χρήση των

NUMA συστημάτων σε εφαρμογές με μεγάλο όγκο δεδομένων και υψηλές υπολογιστικές απαιτήσεις οφείλεται στην ικανότητα τους να παρέχουν μεγάλες ποσότητες μνήμης στις εφαρμογές μέσω ενός δικτύου φυσικά κατανεμημένων κόμβων. Ωστόσο, οι πολλαπλές μονάδες μνήμης εισάγουν μία ποικιλία ως προς τους χρόνους πρόσβασης στη μνήμη (access latency) και το διαθέσιμο εύρος ζώνης (bandwidth) της κάθε εφαρμογής. Αυτή η ποικιλία, η οποία οφείλεται στους πολλούς πιθανούς συνδυασμούς με τους οποίους μπορεί να τρέξει μια δεδομένη εφαρμογή σε ένα NUMA σύστημα, δημιουργεί επιπρόσθετη πολυπλοκότητα και ορίζει ένα πρόβλημα συνδυαστικής βελτιστοποίησης με στόχο την αποδοτική εκμετάλλευση των διαθέσιμων πόρων του συστήματος. Επομένως, το πρόβλημα της κατανομής των νημάτων και της μνήμης των εφαρμογών στους κόμβους ενός NUMA μηχανήματος αποβαίνει κρίσιμο για την απόδοσή τους.

Σε αυτή την κατεύθυνση έχουν κινηθεί πολλές ερευνητικές εργασίες με στόχο την κατανόηση των παραμέτρων που επηρεάζουν την απόδοση των εφαρμογών σε ένα NUMA σύστημα και εν συνεχεία τη βέλτιστη τοποθέτησή τους. Με μια πρώτη ματιά εύκολα συμπεραίνει κανείς ότι η μνήμη των εφαρμογών θα πρέπει να δεσμεύεται τοπικά, δηλαδή κάθε νήμα εφαρμογής και η μνήμη του να ανατίθενται στον ίδιο κόμβο. Αυτή η τακτική βέβαια υπόκειται σε περιορισμούς που αφορούν στο διαθέσιμο bandwidth της μνήμης του κόμβου και του φαινομένου memory contention (περίπτωση όπου δύο διαφορετικά νήματα προσπαθούν να διαβάσουν ταυτόχρονα από το ίδιο block της μνήμης). Επιπλέον, αυτή η απλουστευμένη προσέγγιση δε λαμβάνει υπόψιν την περίπτωση που στο σύστημα τρέχουν πολλές εφαρμογές ταυτόχρονα οπότε κάποιες θα πρέπει να τρέξουν απομακρυσμένα λόγω έλλειψης πόρων, δηλαδή η μνήμη τους να βρίσκεται σε άλλο κόμβο από αυτόν που είναι τα νήματα τους. Σε ένα τέτοιο σενάριο η απόφαση για το ποιες εφαρμογές θα τρέξουν απομακρυσμένα είναι κρίσιμη για τη συνολική απόδοση του συστήματος και αποτελεί το αντικείμενο μελέτης των μελετών που θα αναφερθούν παρακάτω.

2.2.1 Μοντελοποίηση Επίδοσης Εφραμογών σε NUMA Συστήματα

Σημαντική έρευνα στη μοντελοποίηση της απόδοσης των NUMA συστημάτων έχει γίνει από τον Majo και τους συνεργάτες του [Majo11] που έχουν αναπτύξει ένα μοντέλο για να χαρακτηρίσουν το χωρισμό της ροής της μνήμης σε τοπική και απομακρυσμένη. Παράλληλα, ο McCormick και οι συνεργάτες του [McCo11] έχουν προτείνει ένα μοντέλο πρόσβασης στη μνήμη (memory-access model), το οποίο επιδιώκει να χρονοδρομολογεί τις διεργασίες κοντά στα δεδομένα τους με στόχο τη βελτιστοποίηση των αποφάσεων ενός task-scheduling χρονοδρομολογητή. Το μοντέλο αυτό, όμως, εξαρτάται μόνο από τα αρχιτεκτονικά χαρακτηριστικά του συστήματος και δε λαμβάνει υπόψιν τα χαρακτηριστικά των εφαρμογών. Ο Wang και οι συνεργάτες του [Wang16] έχουν προτείνει ένα μοντέλο που μπορεί να προβλέπει τη ροή της μνήμης και τη βέλτιστη τοποθέτησή της σε πολυνηματικές εφαρμογές σε ένα NUMA σύστημα. Ο Su και οι συνεργάτες του [Su12] έχουν χρησιμοποιήσει τεχνικές μηχανικές μάθησης για να προβλέψουν το walltime από πολυνηματικές εφαρμογές σε NUMA συστήματα για διάφορα σενάρια τοποθέτησης. Η προσέγγισή τους είναι κάθετη σε σχέση με τις προηγούμενες με την έννοια ότι η πρόβλεψη για την απόδοση σχετίζεται με την τοποθέτηση των νημάτων αντί της μνήμης. Τέλος, ο Luo και οι συνεργάτες του [Luo16] έχουν προτείνει ένα συνθετικό μοντέλο για την πλήρη τοποθέτηση των εφαρμογών στα NUMA συστήματα, δηλαδή τη βέλτιστη ανάθεση νημάτων και μνήμης, έτσι ώστε να επιτυγχάνεται αποδοτική αξιοποίηση των πόρων τους. Ωστόσο, το μοντέλο που προτείνουν επικεντρώνεται κυρίως σε πολυνηματικές εφαρμογές και χρησιμοποιεί τεχνικές profiling, οι οποίες μπορεί να επηρεάσουν αρνητικά την απόδοση των εφαρμογών.

Τέλος, ο F. Arapidis και οι συνεργάτες του [Arap18] πρότειναν μία προσέγγιση βασισμένη στη μηχανική μάθηση για να προβλέψουν το αντίκτυπο της κατανομής νημάτων και μνήμης στην απόδοση των εφαρμογών στα NUMA συστήματα. Ειδικότερα, στην έρευνα αυτή αναπτύσσονται διάφορα μοντέλα, τα οποία χρησιμοποιούν κάποιες χαρακτηριστικές μετρικές (performance counters) και προσπαθούν να προβλέψουν το overhead στην απόδοση μιας εφαρμογής στην περίπτωση που αυτή είναι αδύνατον να τρέξει τοπικά και πρέπει να χρησιμοποιηθεί ένας απομακρυσμένος κόμβος του συστήματος. Χρησιμοποιώντας τα μοντέλα αυτά, είναι δυνατόν να έχουμε μια στατική εκτίμηση όσον αφορά τον τρόπο που πρέπει να αναθέσουμε τις εφαρμογές στους κόμβους ενός NUMA συστήματος ού-

τως ώστε αυτές να επηρεάζονται λιγότερο από την απόσταση μεταξύ CPU και μνήμης να τρέχουν σε απομακρυσμένους κόμβους χωρίς να υποβαθμίζεται σημαντικά η απόδοσή τους. Αυτή η διαδικασία μπορεί να αποτελέσει το πρώτο βήμα για την λήψη αποφάσεων από ένα resource manager που διαχειρίζεται τις εφαρμογές σε ένα NUMA μηχάνημα και αποτελεί πιθανή επέκταση της συγκεκριμένης έρευνας.

Σε αυτή τη διπλωματική εργασία εστιάζουμε σε απλά αλλά πρακτικά μοντέλα και επικεντρωνόμαστε περισσότερο στην δημιουργία και αξιολόγηση ένος πρακτικού resource manager που προσπαθεί να βελτιώσει την συνολική επίδοση των εφαρμογών που τρέχουν ταυτόχρονα σε ένα NUMA σύστημα.

2.2.2 NUMA Resource Management

Ο J. Qian και οι συνεργάτες του [Qian19] προτείνουν το vDARM, έναν δυναμικό και προσαρμοστικό resource manager ο οποίος δεν κατασκευάζει μόνο μια αρχική κατανομή των εφαρμογών στο NUMA μηχάνημα βασιζόμενος σε στατικά δεδομένα, αλλά αντίθετα παρεμβαίνει δυναμικά προσπαθώντας να βελτιστποιήσει την απόδοση της κάθε εφαρμογής ανάλογα με τα χαρακτηριστικά της. Πιο αναλυτικά, σύμφωνα με το πρωτότυπο vDARM αρχικά γίνεται κατάταξη των εφαρμογών σε τρεις κλάσεις ανάλογα με την ευαισθησία που επιδεικνύουν ως προς δύο βασικά NUMA overheads, την καθυστέρηση λόγω απομακρυσμένης μνήμης (remote access latency) και τη συμφόρηση του διαύλου (traffic congestion).

Η κατάταξη αυτή γίνεται λαμβάνοντας υπόψιν το CPI των εφαρμογών και τις απαιτήσεις τους σε bandwidth και καθορίζει τις αποφάσεις που θα λάβει ο resource manager στη συνέχεια. Έτσι, αν η εφαρμογή είναι ευαίσθητη στο remote access latency επιδιώκεται η συγκέντρωση όλων των σελίδων της μνήμης της σε ένα κόμβο ώστε να τρέξει τοπικά και να έχει βέλτιστη απόδοση. Αντίθετα, αν η εφαρμογή παρουσιάζει αυξημένες ανάγκες για bandwidth ο αλγόριθμος μοιράζει ισόποσα τις σελίδες της μνήμης της στους διαθέσιμους κόμβους ώστε να μεγιστοποιηθεί η παράμετρος αυτή. Η εφαρμογή της συγκεκριμένης πολιτικής σε πραγματικά συστήματα οδηγεί σύμφωνα με την έρευνα σε βελτίωση της απόδοσης της τάξης του 40% κατά μέσο όρο, καταδεικνύοντας την καθοριστική σημασία της κατανομής των εφαρμογών στα σύγχρονα πολυπεξεργαστικά συστήματα τύπου NUMA. Ωστόσο, δεν είναι προφανής η επέκτασή της για την περίπτωση συνεκτέλεσης πολλών εφαρμογών καθώς οι συγκεκριμένες μεθοδολογίες εφαρμόζονται τοπικά για κάθε εφαρμογή χωρίς να λαμβάνουν υπόψιν την κατάσταση του συστήματος.

Ο D. Gureya και οι συνεργάτες του [DGur20] προσπαθώντας να προσεγγίσουν το πρόβλημα αυτό με ένα πιο γενικό τρόπο, προτείνουν το μηχανισμό BWAP που αναλμβάνει την κατανομή των σελίδων της μνήμης μιας εφαρμογής σε κόμβους ασύμμετρα με βάρη ώστε να μεγιστοπιείται η απόδοση της. Αφετηρία για αυτή την έρευνα αποτέλεσε η διαπίστωση ότι η τεχνική του ισομοιρασμού των σελίδων στους κόμβους που χρησιμποιείται από το vDARM αποτυγχάνει να μεγιστοποιήσει το memory throughput στα σύγχρονα NUMA συστήματα που χαρακτηρίζονται από ασύμμετρα bandwidths και καθυστερήσεις (latencies) και είναι ευαίσθητα στα φαινόμενα του memory contention και συμφόρησης του διαύλου (congestion).

Συγκεκριμένα, το BWAP χωρίζει τους κόμβους του συστήματος σε δύο σύνολα -worker και non-worker nodes- και στη συνέχεια με ευριστικό τρόπο χρησιμοποιώντας μία πρότυπη εφαρμογή υπολογίζει ”βάρη” για τους κόμβους που μεγιστοποιούν το διαθέσιμο bandwidth της εφαρμογής. Όσο μεγαλύτερο βάρος έχει ένας κόμβος τόσο μεγαλύτερο ποσοστό των σελίδων της μνήμης της εφαρμογής ανατίθεται σε αυτό τον κόμβο. Ο υπολογισμός αυτών των ”βαρών” εξαρτάται από το NUMA σύστημα που χρησιμοποιείται καθώς επηρεάζεται από τα ιδιαίτερα χαρακτηριστικά του (πλήθος κόμβος, ασύμμετρες αποστάσεις) και επομένως πρέπει να γίνεται από την αρχή κάθε φορά που αλλάζει το υπό μελέτη μηχάνημα. Στη συνέχεια, όταν πρόκειται να τρέξει μια εφαρμογή η κατανομή των σελίδων της ακολουθεί αρχικά τα προϋπολογισμένα βάρη και με χρήση της μεθόδου hill climbing και ενός παράγοντα ευαισθησίας που παίρνει τιμές στο διάστημα [0,1] βρίσκει την ιδανική κατανομή μνήμης για τη συγκεκριμένη εφαρμογή. Όταν ο παράγοντας αυτός είναι ίσος με μηδέν (αρχική τιμή) η κατανομή της μνήμης είναι τέτοια που μεγιστοποιεί το bandwidth στο σύστημα και συνεπώς κατάλληλη για bandwidth-sensitive εφαρμογές. Όσο τείνει προς τη μονάδα οι σελίδες της μνήμης μαζεύονται στους

worker nodes και βελτιστοποιείται η απόδοση των latency-sensitive εφαρμογών που απαιτούν τοπική εκτέλεση. Επομένως, αυτή η έρευνα αν και προσανατολίζεται στη βελτιστοποίηση του bandwidth μπορεί να εφαρμοστεί εξίσου αποτελεσματικά για εφαρμογές με διαφορετικά χαρακτηριστικά και απαιτήσεις ως προς τη μνήμη προσεγγίζοντας το πρόβλημα του NUMA placement με έναν ευριστικό τρόπο.

Ο M. Dashti και οι συνεργάτες του [Dash13] παρατήρησαν ότι η κύρια αιτία υποβάθμισης της απόδοσης στα σύγχρονα NUMA συστήματα δεν είναι το κόστος πρόσβασης σε απομακρυσμένη μνήμη (remote access latency) αλλά η συμφόρηση στους ελεγκτές μνήμης και στους διαύλους επικοινωνίας που προκαλείται από τις data-intensive εφαρμογές. Στην έρευνά τους προτείνουν τον αλγόριθμο Carrefour, έναν ολιστικό αλγόριθμο που λαμβάνει αποφάσεις βασιζόμενος σε παρατηρήσεις της συμφόρησης μνήμης σε ολόκληρο το NUMA σύστημα. Ο αλγόριθμος αυτός χρησιμοποιεί γνωστούς μηχανισμούς, όπως μετακίνηση και αντιγραφή σελίδων μνήμης, αλλά είναι καινοτόμος διότι δεν προσπαθεί απλά να βελτιστοποιήσει την τοπική εκτέλεση των εφαρμογών, αλλά προσπαθεί να επιτύχει μια συνολικά βέλτιστη λύση συνδυάζοντας με αποδοτικό τρόπο αρκετές τεχνικές NUMA τοποθέτησης.

Ο B. Lepers και οι συνεργάτες του [Lepe15] μελέτησαν την επίδραση της ασυμμετρίας που χαρακτηρίζει το δίκτυο διασύνδεσης των περισσότερων NUMA συστημάτων. Παρατήρησαν ότι η απόδοση των εφαρμογών εξαρτάται όχι μόνο από την τοπολογία τους, δηλαδή από την κατανομή των νημάτων και της μνήμης τους στους πόρους του συστήματος, αλλά και από το τρόπο που συνδέονται οι κόμβοι μεταξύ τους. Στη συνέχεια, υλοποίησαν ένα δυναμικό αλγόριθμο τοποθέτησης των εφαρμογών που μεγιστοποιεί το συνολικό bandwidth του συστήματος επιτυγχάνοντας εφάμιλλη ή καλύτερη επίδοση σε σχέση με όλες τις στατικές προσεγγίσεις.

Ο J. Funston και οι συνεργάτες του [Funs18] ασχολούνται με το πρόβλημα της βέλτιστης τοποθέτησης των νημάτων των εφαρμογών στους πυρήνες ενός NUMA συστήματος. Δεδομένου ότι οι διαφορετικές τοπολογίες οδηγούν σε διαφορετικό βαθμό contention των διαμοιραζόμενων πόρων, όπως για παράδειγμα της cache, η συνεισφορά της συγκεκριμένης ερευνητικής προσπάθειας είναι η υλοποίηση ενός γενικού framework για τη βέλτιστη εκτέλεση ενός workload στους πόρους αυτούς. Με άλλα λόγια, το συγκεκριμένο framework δημιουργεί ένα μοντέλο απόδοσης προσαρμοσμένο στα ιδιαίτερα χαρακτηριστικά του NUMA μηχανήματος και ιδιαίτερα στο είδος των διαμοιραζόμενων πόρων που διαθέτει με σκοπό την ελαχιστοποίηση των κόμβων που διατίθενται για την εκτέλεση μιας εφαρμογής.

Τέλος, ο J. Kotra και οι συνεργάτες του [Kotr17] έχουν προτείνει ένα δυναμικό μηχανισμό εξέτασης (probing) της καθυστέρησης πρόσβασης στη μνήμη στα NUMA συστήματα. Ο μηχανισμός έχει σχεδιαστεί με τέτοιο τρόπο ώστε να εντοπίζει τα σημεία όπου υπάρχει συμφόρηση μνήμης στο σύστημα. Με τη χρήση του συγκεκριμένου δυναμικού μηχανισμού probing έχουν προτείνει μηχανισμούς για τη δέσμευση μνήμης (congestion-aware memory allocation) και τη μετακίνηση μνήμης (congestion-aware memory migration) βασιζόμενες στη συμφόρηση που παρατηρείται στο σύστημα με στόχο τη βελτιστοποίηση της απόδοσης των εφαρμογών κατά τη διάρκεια της εκτέλεσής τους.

Όλες οι παραπάνω προσεγγίσεις προσπαθούν να επιλύσουν το ίδιο πρόβλημα, αυτό της αποδοτικότερης κατανομής της μνήμης μια εφαρμογής σε ένα NUMA σύστημα. Καθεμία από αυτές προσφέρει μια διαφορετική οπτική γωνία και χρησιμοποιεί διαφορετικές μεθοδολογίες (machine learning, κατανομή εφαρμογών σε κλάσεις, iterative search). Ωστόσο, κοινό χαρακτηριστικό όλων αυτών των ερευνών είναι ότι επικεντρώνονται να βελτιστοποιήσουν την απόδοση μιας εφαμοργής που τρέχει σε ένα NUMA περιβάλλον [DGur20] ή αντιμετωπίζουν το πρόβλημα σε επίπεδο πυρήνα [Dash13, Funs18, Kotr17, Lepe15] απαιτώντας παρεμβατικές τροποποιήσεις στο λειτουργικό σύστημα.

Σε αυτή τη διπλωματική εργασία, λοιπόν, παίρνοντας ιδέες από τις προαναφερθείσες ερευνητικές δουλειές προσπαθούμε να προσεγγίσουμε αυτό το πρόβλημα σε επίπεδο χώρου χρήστη με ένα πιο γενικό και συστημικό τρόπο που ανταποκρίνεται στις απαιτήσεις των σύγχρονων πολυεπεξεργαστικών συστημάτων.

Κεφάλαιο 3

Μεθοδολογία

Σε αυτό το κεφάλαιο θα αναλύσουμε την αρχιτεκτονική του συστήματος που χρησιμοποιήσαμε για την διεξαγωγή των πειραμάτων καθώς και όλα τα βασικά εργαλεία και μεθοδολογίες που βοήθησαν σε διάφορα σημεία της παρούσας έρευνας.

3.1 Πειραματικό Μηχάνημα

Όλα τα πειράματα εκτελέστηκαν στο μηχάνημα ”Sandman” το οποίο αποτελεί ένα τυπικό πολυεπεξεργαστικό σύστημα βασισμένο σε NUMA αρχιτεκτονική. Το μηχάνημα Sandman αποτελείται από τέσσερις (4) κόμβους (NUMA nodes) συνολικά. Το μοντέλο επεξεργαστή του κάθε κόμβου είναι το Intel Xeon E5-4620, τέταρτης γενιάς, με συχνότητα χρονισμού στα 2.2 GHz. Ο κάθε κόμβος του συστήματος αποτελείται από 8 πυρήνες που ο καθένας μπορεί να τρέξει 2 νήματα.

Ο κάθε πυρήνας του συστήματος έχει δύο επίπεδα κρυφής μνήμης, την L1-cache και την L2-cache. Οι δύο κρυφές μνήμες είναι διαμοιραζόμενες και στα δύο νήματα του πυρήνα. Επιπλέον, κάθε κόμβος διαθέτει και ένα τρίτο επίπεδο κρυφής μνήμης, την L3-cache ή όπως αναφέρεται συχνότερα Last Level Cache (LLC), η οποία είναι διαμοιραζόμενη σε όλους τους πυρήνες του κόμβου. Στον Πίνακα 3.1 βλέπουμε το μέγεθος της κάθε κρυφής μνήμης.

Cache	Size	Shared
L1	32KB	Per core
L2	256KB	Per core
L3 (LLC)	16MB	Per NUMA node

Πίνακας 3.1: Στοιχεία κρυφών μνημών (caches) του μηχανήματος Sandman.

Ο κάθε κόμβος του συστήματος είναι συνδεδεμένος με τη δικιά του μνήμη. Το μέγεθος της μνήμης που διαθέτει ο κάθε κόμβος είναι 64GB, και το σύστημα έχει συνολικά 256GB κύριας μνήμης.

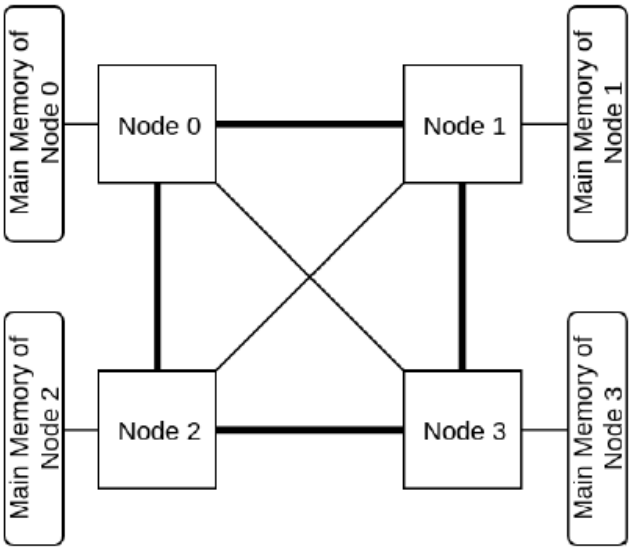
Οι ακμές του δικτύου διασύνδεσης στο Sandman, δηλαδή ο τρόπος με τον οποίο είναι συνδεδεμένος ο κάθε κόμβος με όλους τους άλλους κόμβους, είναι τύπου Intel QuickPath Interconnect. Οι αποστάσεις μεταξύ των κόμβων όπως λαμβάνονται με χρήση του εργαλείου numactl (με παράμετρο –hardware) φαίνονται στον πίνακα 3.2.

	Node 0	Node 1	Node 2	Node 3
Node 0	10	21	21	30
Node 1	21	10	30	21
Node 2	21	30	10	21
Node 3	30	21	21	10

Πίνακας 3.2: Αποστάσεις κόμβων του μηχανήματος Sandman.

Όπως παρατηρούμε ο πίνακας των αποστάσεων είναι συμμετρικός, επομένως το μηχάνημα Sandman είναι συμμετρικό. Στο Σχήμα 3.1 βλέπουμε τη γραφική αναπαράσταση του μηχανήματος Sandman.

Οι πιο παχιές γραμμές του σχήματος δηλώνουν τις ακμές του δικτύου διασύνδεσης που προσφέρουν την ταχύτερη επικοινωνία μεταξύ των εμπλεκόμενων κόμβων, όπως προκύπτει από τον πίνακα των αποστάσεων.



Σχήμα 3.1: Γραφική αναπαράσταση μηχανήματος Sandman.

3.2 Εργαλεία

Στο υποκεφάλαιο αυτό θα προχωρήσουμε σε μία αναλυτική περιγραφή των εργαλείων που χρησιμοποιήσαμε καθόλη τη διάρκεια της έρευνας για την υποστήριξη των πειραμάτων και τη διεξαγωγή των μετρήσεων. Θα παρουσιάσουμε το σκοπό του κάθε εργαλείου, τον τρόπο χρήσης του, τη λειτουργία του καθώς και πιθανά προβλήματα που προέκυψαν κατά τη χρήση του.

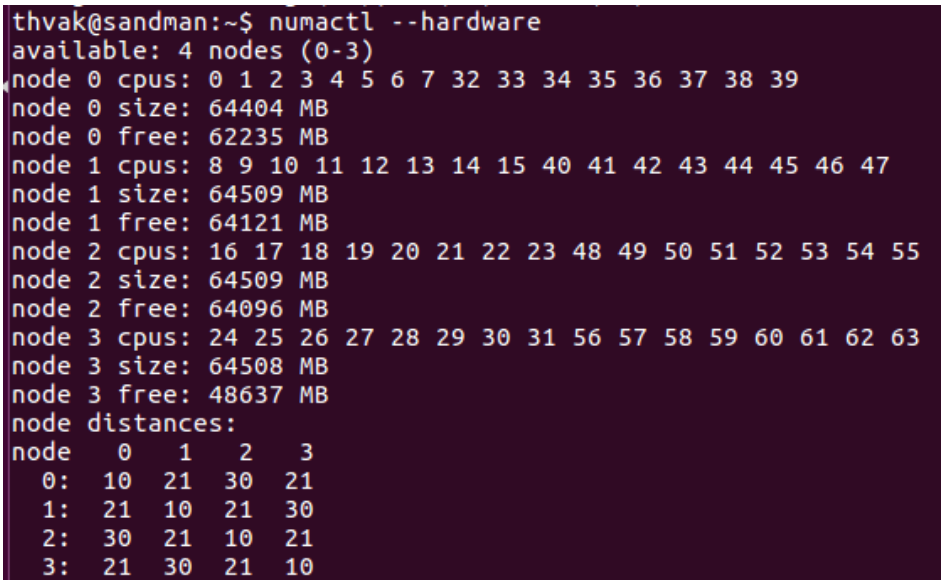
3.2.1 numactl

Το numactl [numa] παίζει κυρίαρχο ρόλο στην υποστήριξη των πειραμάτων στα NUMA υπολογιστικά συστήματα καθώς παρέχει αρκετές δυνατότητες, όπως:

- Παροχή πληροφοριών για το σύστημα
- Τοποθέτηση νημάτων των εφαρμογών στους επεξεργαστές του συστήματος
- Δέσμευση μνήμης για τις εφαρμογές σε συγκεκριμένους κόμβους του συστήματος

Ας δούμε πιο αναλυτικά πως εφαρμόσαμε κάθε δυνατότητα του εργαλείου numactl στην έρευνα μας.

Παροχή πληροφοριών για το σύστημα
Η πρώτη λειτουργία του εργαλείου numactl που χρησιμοποιήσαμε αποσκοπεί στο να μας δώσει κάποιες βασικές πληροφορίες για την αρχιτεκτονική του συστήματος που μελετάμε. Αυτό επιτυγχάνεται μέσω της κλήσης numactl –hardware (ή numactl -H) το αποτέλεσμα της οποίας φαίνεται στην εικόνα 3.2.



Σχήμα 3.2: Αποτέλεσμα της εντολής numactl –hardware για το μηχάνημα Sandman.

Το πρώτο κομμάτι πληροφοριών που μας παρέχει αφορά στη θέση των επεξεργαστών του συστήματος σε σχέση με τους κόμβους. Τα νήματα των επεξεργαστών είναι αριθμημένα από 0-63 και ανατίθενται στους κόμβους με προκαθορισμένο τρόπο. Συνολικά υπάρχουν 64 νήματα διαθέσιμα αφού στο Sandman κάθε κόμβος έχει 8 πυρήνες και κάθε πυρήνας μπορεί να τρέξει 2 νήματα οπότε εφόσον υπάρχουν τέσσερεις κόμβοι έχουμε 4*8 = 32 πυρήνες οι οποίοι υποστηρίζουν 32*2 = 64 νήματα.

Το επόμενο κομμάτι δεδομένων που μας παρέχει αφορά στο μέγεθος της μνήμης που είναι συνδεδεμένη σε κάθε κόμβο, καθώς και στο πόση μνήμη είναι ελεύθερη τη συγκεκριμένη χρονική στιγμή που εκτελέστηκε η εντολή. Τέλος, υπάρχει ο πίνακας των αποστάσεων των κόμβων δηλαδή ο πίνακας 3.2.

Τοποθέτηση νημάτων των εφαρμογών στους επεξεργαστές του συστήματος

Το numactl μας παρέχει τη δυνατότητα να τοποθετήσουμε τα νήματα ενός benchmark σε συγκεκριμένους πυρήνες (CPUs). Αυτό επιτυγχάνεται με δύο τρόπους οι οποίοι διαφέρουν ως προς την ακρίβεια με την οποία πρσδιορίζονται οι πυρήνες που θα τρέξει η εφαρμογή. Ειδικότερα, ο πρώτος τρόπος είναι μέσω της εντολής numactl –cpunodebind=<nodes> (ή numactl -N <nodes>) η οποία αναλαμβάνει να εκτελέσει την εφαρμογή σε πυρήνα που ανήκει στους κόμβους που ορίζουμε στη λίστα <nodes> χωρίς να προσδιορίζεται ακριβώς σε ποιο επεξεργαστή θα τρέξει. Ο δεύτερος τρόπος είναι μέσω της εντολής numactl –physcpubind=<cpus> (ή numactl -C <cpus>), η οποία δέχεται ως όρισμα μια λίστα από επεξεργαστές <cpus> και αναλαμβάνει να τρέξει την εφαρμογή χρησιμοποιώντας νήματα μόνο από αυτούς.

Η συγκεκριμένη λειτουργία του εργαλείου numactl χρησιμοποιείται ευρέως σε όλα μας τα πειράματα ώστε να ελέγχουμε σε ποιους πυρήνες τρέχουν οι εφαρμογές κάτι που στη συνέχεια καθορίζει το χαρακτηρισμό των μνημών του συστήματος ως προς αυτή την εφαρμογή δηλαδή ποια είναι η τοπική μνήμη και ποιες οι απομακρυσμένες.

Δέσμευση μνήμης για τις εφαρμογές σε συγκεκριμένους κόμβους του συστήματος

Το numactl μας δίνει τη δυνατότητα να ορίσουμε εμείς από ποιους κόμβους θα χρησιμοποιήσει μνήμη ένα benchmark κατά τη διάρκεια της εκτέλεσής του. Αυτό επιτυγχάνεται με τρεις εναλλακτικές εντολές οι οποίες διαφέρουν ως προς τον τρόπο με τον οποίο δεσμεύεται η μνήμη και είναι οι ακόλουθες:

1. numactl –membind=<nodes> (ή numactl -m <nodes>)
2. numactl –interleave=<nodes> (ή numactl -i <nodes>)

3. numactl –preferred=<node>

1. numactl -m <nodes>

Η κλήση αυτή δέχεται ως όρισμα μία λίστα από κόμβους του συστήματος που θέλουμε να μελετήσουμε και αναλαμβάνει να δεσμεύσει μνήμη για την εφαρμογή μόνο από τους συγκεκριμένους κόμβους. Εάν δεν υπάρχει διαθέσιμη ελεύθερη μνήμη στους δοθέντες κόμβους η κλήση αποτυγχάνει. Στην έρευνά μας χρησιμοποιούμε την εντολή αυτή για να δεσμεύουμε μνήμη από ένα συγκεκριμένο κόμβο κάθε φορά ανάλογα με το σενάριο υπό μελέτη.

Όταν θέλουμε να μελετήσουμε την απόδοση μιας εφαρμογής σε τοπική εκτέλεση (local), δηλαδή όταν η μνήμη της βρίσκεται στον ίδιο κόμβο με τα νήματά της, χρησιμοποιούμε τη συγκεκριμένη εντολή δίνοντας ως όρισμα τον κόμβο αυτό. Ακόμη, όταν θέλουμε να μελετήσουμε την απόδοση μιας εφαρμογής σε απομακρυσμένη εκτέλεση (remote), δηλαδή όταν η μνήμη της βρίσκεται σε άλλο κόμβο από αυτόν που βρίσκονται τα νήματά της, τότε πάλι χρησιμοποιούμε την εντολή αυτή δίνοντας αυτή τη φορά ως όρισμα τον απομακρυσμένο κόμβο από τον οποίο επιθυμούμε να δεσμεύσουμε τη μνήμη.

2. numactl -i <nodes>

Η κλήση αυτή δέχεται ως όρισμα μια λίστα από κόμβους του συστήματος που θέλουμε να μελετήσουμε και αναλαμβάνει να δεσμεύσει μνήμη εναλλάξ από αυτούς με κυκλικό (round robin) τρόπο ετσί ώστε να υπάρχει ίσος αριθμός από σελίδες μνήμης σε όλους τους κόμβους. Εάν δεν υπάρχει διαθέσιμη μνήμη σε ένα κόμβο τότε η δέσμευση μνήμης συνεχίζεται κυκλικά στους εναπομείναντες κόμβους που έχουν δοθεί.

Στην έρευνα μας χρησιμοποιούμε τη συγκεκριμένη κλήση για να μελετήσουμε την απόδοση των εφαρμογών μας σε interleave εκτέλεση όπως λέμε, δηλαδή στην περίπτωση που επιθυμούμε η μνήμη να δεσμεύεται ισόποσα σε περισσότερους από έναν κόμβους. Έτσι, μπορούμε να μοιράζουμε τις σελίδες και αντίστοιχα τη μνήμη των εφαρμογών σε πολλούς κόμβους χωρίς να απαιτείται εξωτερική παρέμβαση. Συνήθως επιλέγουμε να δίνουμε δύο κόμβους στη λίστα του ορίσματος, τον τοπικό της εφαρμογής και έναν απομακρυσμένο, οπότε καταφέρνουμε να έχουμε το 50% της μνήμης της στον τοπικό κόμβο και το υπόλοιπο 50% στον απομακρυσμένο.

3. numactl –preferred=<node>

Η κλήση αυτή δέχεται ως όρισμα ένα κόμβο του συστήματος που θέλουμε να μελετήσουμε και αναλαμβάνει να δεσμεύσει μνήμη από αυτόν τον κόμβο κατά προτίμηση εφόσον είναι εφικτό. Αν η μνήμη του κόμβου που δίνουμε ως όρισμα δεν επαρκεί για την εφαρμογή μας τότε το εργαλείο αναλαμβάνει να δεσμεύσει μνήμη από τους υπόλοιπους κόμβους του συστήματος.

Στην έρευνά μας η συγκεκριμένη κλήση χρησιμοποιείται μόνο στα πειράματα με τον resource manager όπου θέλουμε να ”ζορίσουμε” το σύστημά μας τρέχοντας παράλληλα αρκετές εφαρμογές που καταναλώνουν όλη τη μνήμη ενός κόμβου. Έτσι, επειδή δε θέλουμε να αποτυγχάνει η δέσμευση μνήμης χρησιμοποιούμε αυτή την εντολή ώστε αν δεν υπάρχει αρεκτή ελεύθερη μνήμη να χρησιμοποιείται κάποιος εναλλακτικός κόμβος του συστήματος.

3.2.2 perf

Το εργαλείο perf [perf] επιτρέπει τη συλλογή μετρήσεων κατά τη διάρκεια εκτέλεσης των εφαρμογών μας. Οι μετρήσεις αφορούν διάφορες παραμέτρους των benchmarks και γίνονται με τη χρήση performance counters.

Οι performance counters είναι μετρητές, οι οποίοι χρησιμοποιούνται για να μετράνε το πλήθος από συγκεκριμένα γεγονότα που συμβαίνουν στο σύστημά μας καθολικά ή σε μία μεμονωμένη εφαρμογή που τρέχουμε. Οι performance counters είναι απευθείας συνδεδεμένοι με το υλικό και χωρίζονται σε δύο βασικές κατηγορίες, τους *hardware performance counters* και τους *software performance*

counters. Οι hardware performance counters είναι μετρητές, οι οποίοι μετράνε γεγονότα που συμβαίνουν στο υλικό του συστήματός μας, όπως στη CPU, τη PMU (Performance Monitoring Unit) καθώς και σε άλλα κομμάτια του υλικού. Οι software performance counters μετράνε γεγονότα που συμβαίνουν σε επίπεδο λογισμικού του συστήματός μας. Ένα τυπικό παράδειγμα τέτοιων μετρητών είναι οι performance counters που μετράνε τα page faults που συμβαίνουν στο σύστημά. Στην εικόνα 3.3 βλέπουμε μία επισκόπηση των performance counters που υποστηρίζει η έκδοση 3.16 που χρησιμοποιούμε στα πειράματα. Όπως θα δούμε στη συνέχεια, στην έρευνά μας χρησιμοποιήσαμε αποκλειστικά hardware performance counters.

```
perf list

List of pre-defined events (to be used in -e):

 cpu-cycles OR cycles                       [Hardware event]
 instructions                               [Hardware event]
 cache-references                           [Hardware event]
 cache-misses                               [Hardware event]
 branch-instructions OR branches            [Hardware event]
 branch-misses                              [Hardware event]
 bus-cycles                                 [Hardware event]

 cpu-clock                                  [Software event]
 task-clock                                 [Software event]
 page-faults OR faults                      [Software event]
 minor-faults                               [Software event]
 major-faults                               [Software event]
 context-switches OR cs                     [Software event]
 cpu-migrations OR migrations               [Software event]
 alignment-faults                           [Software event]
 emulation-faults                           [Software event]

 L1-dcache-loads                            [Hardware cache event]
 L1-dcache-load-misses                      [Hardware cache event]
 L1-dcache-stores                           [Hardware cache event]
 L1-dcache-store-misses                     [Hardware cache event]
 L1-dcache-prefetches                       [Hardware cache event]
 L1-dcache-prefetch-misses                  [Hardware cache event]
 L1-icache-loads                            [Hardware cache event]
 L1-icache-load-misses                      [Hardware cache event]
 L1-icache-prefetches                       [Hardware cache event]
 L1-icache-prefetch-misses                  [Hardware cache event]
 LLC-loads                                  [Hardware cache event]
 LLC-load-misses                            [Hardware cache event]
 LLC-stores                                 [Hardware cache event]
 LLC-store-misses                           [Hardware cache event]

 LLC-prefetch-misses                        [Hardware cache event]
 dTLB-loads                                 [Hardware cache event]
 dTLB-load-misses                           [Hardware cache event]
 dTLB-stores                                [Hardware cache event]
 dTLB-store-misses                          [Hardware cache event]
 dTLB-prefetches                            [Hardware cache event]
 dTLB-prefetch-misses                       [Hardware cache event]
 iTLB-loads                                 [Hardware cache event]
 iTLB-load-misses                           [Hardware cache event]
 branch-loads                               [Hardware cache event]
 branch-load-misses                         [Hardware cache event]
```

Σχήμα 3.3: Performance counters που υποστηρίζει το εργαλείο perf (έκδοση 3.16).

Αξίζει να σημειωθεί ότι οι performance counters δεν είναι κοινοί σε όλα τα μηχανήματα. Ένα μηχάνημα δηλαδή μπορεί να διαθέτει κάποιους εξειδικευμένους performance counters που μετράνε κάποια γεγονότα ενώ ένα άλλο μπορεί να μη διαθέτει αυτούς τους performance counters ή να διαθέτει άλλους που μετράνε διαφορετικές παραμέτρους. Αυτή η διαπίστωση είναι αρκετά σημαντική καθώς μπορεί να δημιουργήσει περιορισμούς ως προς το εύρος εφαρμογής της έρευνας μας. Ας δώσουμε ένα παράδειγμα για να γίνει πιο κατανοητός ο προβληματισμός μας. Έστω ότι αποφασίζουμε να χρησιμοποιήσουμε στα πειράματά μας ή στον resource manager που κατασκευάσαμε τον performance counter P, ο οποίος υπάρχει στο μηχάνημα Sandman αλλά δεν υπάρχει σε ένα άλλο NUMA μηχάνημα, έστω X. Εδώ δεν έχει καμία σημασία τι ακριβώς μετράει ο P αλλά υποθέτουμε ότι είναι σημαντικό για την έρευνά μας. Σε αυτή την περίπτωση δεν μπορούμε να πάρουμε αντίστοιχες μετρήσεις στο μηχάνημα X για τον P, αφού δεν υποστηρίζεται, ούτε να πιστοποιήσουμε αν οι συσχετίσεις που κάνουμε με τους άλλους counters είναι έγκυρες. Επιπλέον, ο resource manager που θα κατασκευάσουμε με χρήση του P έχει περιορισμένο εύρος εφαρμογών. Αυτό συμβαίνει, διότι αν δεν υπάρχει ο P δεν μπορεί να λει-

τουργήσει σωστά η λογική του περιλαμβάνει ο resource manager οπότε δε θα μπορεί να λάβει σωστές αποφάσεις για τη μνήμη των εφαρμογών.

Τέτοια και άλλα παρόμοια προβλήματα προκύπτουν αν χρησιμοποιήσουμε performance counters με ιδιαίτερα εξειδικευμένη λειτουργία ή που δεν υπάρχουν ευρέως στα NUMA συστήματα. Για να τα αντιμετωπίσουμε αποφασίσαμε να χρησιμοποιήσουμε βασικούς performance counters που βρίσκονται ευρέως σε όλα τα NUMA μηχανήματα. Στον πίνακα 3.3 βλέπουμε όλους τους performance counters που χρησιμοποιήθηκαν στην έρευνά μας. Στην πρώτη γραμμή υπάρχουν οι performance counters που χρησιμοποιήθηκαν για την εξαγωγή του IPC (Instruction Per Cycle) και του MPKI (Misses Per Kilo Instructions) των εφαρμογών, ενώ στη δεύτερη υπάρχουν αυτοί που χρησιμοποιήθηκαν για την εξαγωγή του TLB MPKI.

Result	**Performance Counters**			
MPKI, IPC	instructions	cpu-cycles	LLC-load-misses	LLC-store-misses
TLB MPKI	instructions	dTLB-load-misses	iTLB-load-misses	dTLB-store-misses

Πίνακας 3.3: Performance counters που χρησιμοποιήθηκαν για κάθε επιθυμητή μέτρηση.

Ένας από τους κυριότερους περιορισμούς που έχουν οι performance counters και κατά συνέπεια και το εργαλείο perf είναι ότι δεν μπορούμε να μετρήσουμε πολλές παραμέτρους ταυτόχρονα. Αυτό προκύπτει από το γεγονός ότι αν ζητήσουμε από το perf να μετρήσει πολλούς performance counters ταυτόχρονα τότε κάνει δειγματοληψία μεταξύ τους, με αποτέλεσμα να μην μας δίνει την πραγματική τιμή της παραμέτρου που θέλουμε να μετρήσει, αλλά ένα ποσοστό και τη μέτρηση που πήρε μέσα σε αυτό το ποσοστό.

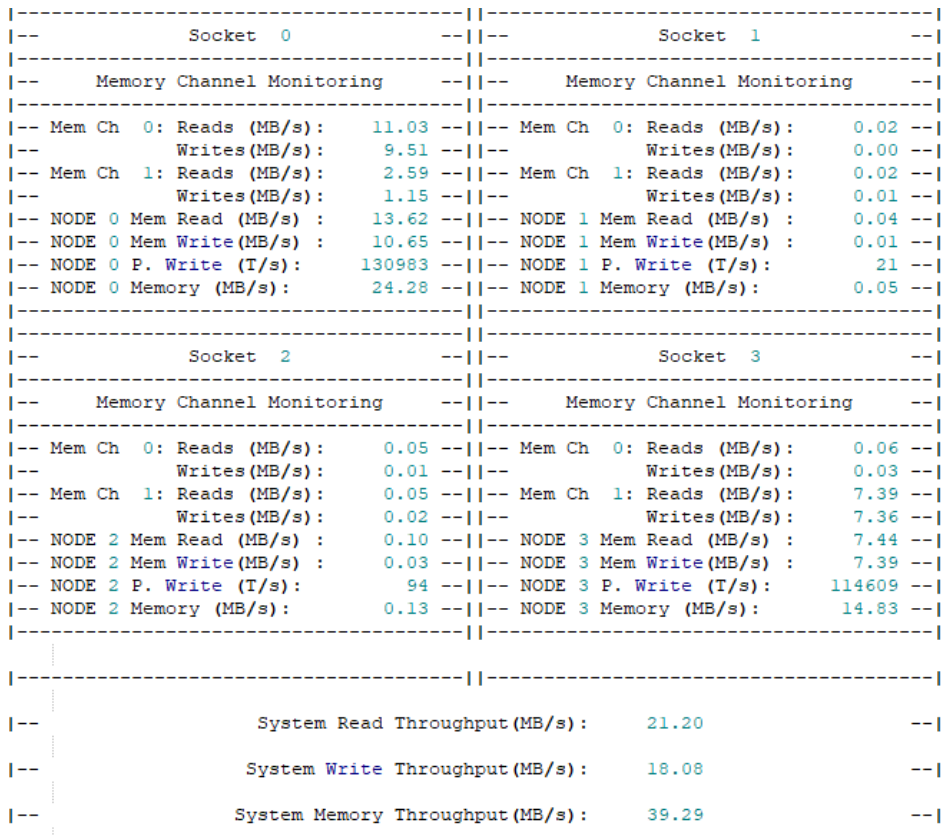
Ύστερα από πειράματα που πραγματοποιήσαμε διαπιστώσαμε ότι ο μέγιστος αριθμός από performance counters που μπορούν να μετρηθούν ταυτόχρονα χωρίς να χρειαστεί δειγματοληψία είναι τέσσερις. Με δεδομένο αυτόν τον περιορισμό, αναγκαστήκαμε να μετρήσουμε δύο φορές την εκτέλεση ενός benchmark για να πάρουμε όλες τις επιθυμητές μετρήσεις των παραμέτρων που επιθυμούσαμε. Η επιλογή των παραμέτρων προς μέτρηση σε κάθε εκτέλεση έγινε με τέτοιο τρόπο ώστε να συγχωνεύσουμε όσο το δυνατόν περισσότερη πληροφορία γίνεται. Έτσι, στην πρώτη εκτέλεση λάβαμε τις συνολικές εντολές και κύκλους για τον υπολογισμό του IPC, καθώς και τα συνολικά misses στην LLC cache για τον υπολογισμό του MPKI, ενώ στη δεύτερη γραμμή μετρήσαμε πάλι τις συνολικές εντολές και τα misses στο TLB για τον προσδιορισμό του TLB MPKI.

3.2.3 pcm-memory.x

Το εργαλείο pcm-memory.x [inte] χρησιμοποιήθηκε για τη μέτρηση της χρήσης του εύρου ζώνης της μνήμης (bandwidth) ως συμπληρωματικό του εργαλείου perf που αναλύσαμε στο προηγούμενο υποκεφάλαιο. Το pcm-memory.x προέρχεται από τη σουίτα προγραμμάτων Intel Performance Counter Monitor και εμείς χρησιμοποιήσαμε την έκδοση 2.11 της σουίτας. Τα αποτελέσματα μπορούν να τυπώνονται σε αρχείο είτε σε μορφή .csv, η οποία είναι εύκολα διαχειρίσιμη από τον υπολογιστή και χρησιμοποιείται από τον resource manager όπως θα δούμε στη συνέχεια, είτε σε μορφή .out, η οποία είναι πιο παραστατική για την παρουσίαση των αποτελεσμάτων.

Στην εικόνα 3.4 βλέπουμε τις πληροφορίες από τις οποίες αποτελείται ένα τέτοιο αρχείο. Ουσιαστικά μας παρέχει πληροφορίες για την ροή της μνήμης σε κάθε ελεγκτή του συστήματος καθώς και συνολικά τη ροή για ολόκληρο το σύστημα. Η ροή της μνήμης μετριέται σε MB/s και διαχωρίζεται σε τρεις κατηγορίες:

- Mem Read (MB/s)
- Mem Write (MB/s)
- Memory (MB/s)



Σχήμα 3.4: Αρχείο εξόδου του εργαλείου pcm-memory.x.

Η πρώτη κατηγορία αφορά στη ροή που προέρχεται από διάβασμα που συμβαίνει στη μνήμη, ενώ η δεύτερη κατηγορία αφορά στη ροή που προέρχεται από εγγραφές που συμβαίνουν στη μνήμη. Η τρίτη κατηγορία είναι το άθροισμα των δύο προηγούμενων κατηγοριών. Οι δύο πρώτες κατηγορίες χωρίζονται σε δύο υποκατηγορίες αλλά δεν μας απασχόλησαν στην έρευνά μας. Τέλος, χρησιμοποιήσαμε αποκλειστικά την τρίτη κατηγορία, δηλαδή τη συνολική ροή μνήμης σε κάθε ελεγκτή, αφού δεν μας ενδιέφερε να τη διαχωρίσουμε σε ροή ανάγνωσης και εγγραφής.

3.2.4 migratepages

Η κλήση migratepages [migr] που προσφέρει ο πυρήνας του Linux μας δίνει τη δυνατότητα να μεταφέρουμε τη μνήμη μιας εφαρμογής μεταξύ των κόμβων του NUMA συστήματος στο οποίο τρέχει κατά τη διάρκεια της εκτέλεσής της. Αυτή η λειτουργία μας φάνηκε πολύ χρήσιμη κατά τη δημιουργία του resource manager όπως θα δούμε στο επόμενο κεφάλαιο ώστε να μπορεί να αλλάζει δυναμικά την κατανομή των εφαρμογών στις μνήμες του συστήματος ανάλογα με τις παραμέτρους που διαβάζει από τα άλλα εργαλεία. Η μετακίνηση των σελίδων των εφαρμογών μας επιτρέπει να αλλάζουμε τη σχετική απόσταση μεταξύ μνήμης και CPU με στόχο την επίτευξη καλύτερης απόδοσης.

Ας δούμε πιο αναλυτικά τον τρόπο εκτέλεσης της κλήσης migratepages. Για να τη χρησιμοποιήσουμε δίνουμε ως όρισμα εισόδου το pid της διεργασίας της οποίας την μνήμη επιθυμούμε να μεταφέρουμε καθώς και τους κόμβους αναχώρησης και προορισμού. Η σύνταξη, λοιπόν, για τη συγκεκριμένη κλήση έχει τη μορφή *migratepages* <*pid*> <*from-nodes*> <*to-nodes*>, όπου τα ορίσματα <from-nodes> και <to-nodes> είναι λίστες από κόμβους του συστήματός μας.

Η λειτουργία της migratepages είναι να μετακινεί τη φυσική θέση των σελίδων της μνήμης μια εφαρμογής που τρέχει σε NUMA μηχάνημα χωρίς να προκαλεί αλλαγές στο χώρο των εικονικών της διευθύνσεων. Αν ορίσουμε πολλαπλούς κόμβους για τα ορίσματα <from-nodes> και <to-nodes> τότε το εργαλείο προσπαθεί να διατηρήσει τη σχετική θέση κάθε σελίδας στο κάθε σύνολο κόμβων. Για παράδειγμα, αν ορίσουμε ως κόμβους αναχώρησης τους κόμβους 2-5 και ως κόμβους προορισμού

τους κόμβους 7, 9, 12-13 τότε η επιθυμητή λειτουργία της εντολής θα είναι η μεταφορά των σελίδων μνήμης από τον κόμβο 2 στον κόμβο 7, από τον 3 στον 9, από τον 4 στον 12 και από τον 5 στον 13 αντίστοιχα.

3.2.5 taskset

Η κλήση taskset [task] που προσφέρει ο πυρήνας του Linux μας δίνει τη δυνατότητα να θέσουμε ή να ανακτήσουμε το cpu affinity μιας διεργασίας που εκτελείται βάση το pid της. Με τον όρο *cpu affinity* αναφερόμαστε σε μια ιδιότητα χρονοδρομολόγησης που σχετίζεται με τη συσχέτιση μιας διεργασίας με ένα συγκεκριμένο υποσύνολο από cpus του μηχανήματος μας. Έτσι, ο χρονοδρομολογητής του Linux θα λάβει υπόψιν την δοθείσα τιμή για την παράμετρο cpu affinity και η εφαρμογή μας δε θα τρέξει σε άλλες cpus εκτός από αυτές που έχουμε προσδιορίσει ρητά. Η παράμετρος cpu affinity αναπαριστάται σαν μια μάσκα από bits, με το LSB να αντιστοιχεί στην πρώτη λογική cpu και το MSB στην τελευταία λογική cpu.

3.2.6 numa_maps

Ένα άλλο εργαλείο που χρησιμοποιήσαμε είναι το numa_maps [**?**] που παρέχει πληροφορίες για την κατάσταση της μνήμης των εφαρμογών μας. Κατά τη διάρκεια της εκτέλεσης μιας εφαρμογής αλλά και μετά το πέρας της θέλουμε να ξέρουμε πόση μνήμη δεσμεύτηκε και σε ποιούς κόμβους, κάτι που γίνεται εφικτό μέσω του συγκεκριμένου εργαλείου.

Ας δούμε ένα απλό παράδειγμα για να καταλάβουμε τη λειτουργία του. Έστω ότι επιθυμούμε να δούμε την κατάσταση της μνήμης της εφαρμογής X που τρέχει στο σύστημά μας. Αρχικά, πρέπει να γνωρίζουμε το pid της εφαρμογής αυτής, έστω PID. Στη συνέχεια, αυτό που χρειάζεται να κάνουμε είναι να ανοίξουμε το αρχείο /proc/<PID>/numa_maps, το οποίο έχει δημιουργηθεί αυτόματα κατά την έναρξη του εκτέλεσης της εφαμρογής X και περιέχει τη στιγμαία κατάσταση στην οποία βρίσκεται η μνήμη της τη χρονική στιγμή που ανοίχθηκε το αρχείο. Η δομή αυτού του αρχείου φαίνεται στην εικόνα 3.5 και όπως μπορούμε να δούμε είναι αρκετά πλήρης παρέχοντας όλες τις απαραίτητες πληροφορίες για την κατάσταση της μνήμης της εφαρμογής καθόλη τη διάρκεια της εκτέλεσής της.

```
thvak@sandman:~$ cat /proc/36350/numa_maps
00400000 bind:0 file=/various/diplom/thvak/spec-2017/508.namd_r/ref/namd_r_base.dunnington-x86-m64 mapped=88 N1=88 kernelpagesize_kB=4
006e8000 bind:0 file=/various/diplom/thvak/spec-2017/508.namd_r/ref/namd_r_base.dunnington-x86-m64 anon=1 dirty=1 N0=1 kernelpagesize_kB=4
00dff000 bind:0 heap anon=28044 dirty=28044 N0=28044 kernelpagesize_kB=4
7f1d0a1a6000 bind:0 anon=79 dirty=79 N0=79 kernelpagesize_kB=4
7f1d0a22b000 bind:0 anon=11079 dirty=11079 N0=11079 kernelpagesize_kB=4
7f1d0d127000 bind:0 file=/lib/x86_64-linux-gnu/libc-2.19.so mapped=281 mapmax=61 N1=1 N2=280 kernelpagesize_kB=4
7f1d0d2c8000 bind:0 file=/lib/x86_64-linux-gnu/libc-2.19.so
7f1d0d4c8000 bind:0 file=/lib/x86_64-linux-gnu/libc-2.19.so anon=4 dirty=4 N0=4 kernelpagesize_kB=4
7f1d0d4cc000 bind:0 file=/lib/x86_64-linux-gnu/libc-2.19.so anon=2 dirty=2 N0=2 kernelpagesize_kB=4
7f1d0d4ce000 bind:0 anon=2 dirty=2 N0=2 kernelpagesize_kB=4
7f1d0d4d2000 bind:0 file=/lib/x86_64-linux-gnu/libgcc_s.so.1 mapped=21 mapmax=5 N3=21 kernelpagesize_kB=4
7f1d0d4e8000 bind:0 file=/lib/x86_64-linux-gnu/libgcc_s.so.1
7f1d0d6e7000 bind:0 file=/lib/x86_64-linux-gnu/libgcc_s.so.1 anon=1 dirty=1 N0=1 kernelpagesize_kB=4
7f1d0d6e8000 bind:0 file=/lib/x86_64-linux-gnu/libm-2.19.so mapped=152 mapmax=19 N2=88 N3=64 kernelpagesize_kB=4
7f1d0d7e8000 bind:0 file=/lib/x86_64-linux-gnu/libm-2.19.so
7f1d0d9e7000 bind:0 file=/lib/x86_64-linux-gnu/libm-2.19.so anon=1 dirty=1 N0=1 kernelpagesize_kB=4
7f1d0d9e8000 bind:0 file=/lib/x86_64-linux-gnu/libm-2.19.so anon=1 dirty=1 N0=1 kernelpagesize_kB=4
7f1d0d9e9000 bind:0 file=/usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.20 mapped=215 mapmax=4 N2=215 kernelpagesize_kB=4
7f1d0dad5000 bind:0 file=/usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.20
7f1d0dcd5000 bind:0 file=/usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.20 anon=8 dirty=8 N0=8 kernelpagesize_kB=4
7f1d0dcdd000 bind:0 file=/usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.20 anon=2 dirty=2 N0=2 kernelpagesize_kB=4
7f1d0dcdf000 bind:0 anon=4 dirty=4 N0=4 kernelpagesize_kB=4
7f1d0dcf4000 bind:0 file=/lib/x86_64-linux-gnu/ld-2.19.so mapped=33 mapmax=60 N2=33 kernelpagesize_kB=4
7f1d0dd2d000 bind:0 anon=60 dirty=60 N0=60 kernelpagesize_kB=4
7f1d0dd85000 bind:0 anon=306 dirty=306 N0=306 kernelpagesize_kB=4
7f1d0df12000 bind:0 anon=2 dirty=2 N0=2 kernelpagesize_kB=4
7f1d0df14000 bind:0 file=/lib/x86_64-linux-gnu/ld-2.19.so anon=1 dirty=1 N0=1 kernelpagesize_kB=4
7f1d0df15000 bind:0 file=/lib/x86_64-linux-gnu/ld-2.19.so anon=1 dirty=1 N0=1 kernelpagesize_kB=4
7f1d0df16000 bind:0 anon=1 dirty=1 N0=1 kernelpagesize_kB=4
7ffe0254f000 bind:0 stack anon=4 dirty=4 N0=4 kernelpagesize_kB=4
7ffe0259c000 bind:0
7ffe0259e000 bind:0
```

Σχήμα 3.5: Δομή των αρχείων numa_maps.

Συγκεκριμένα, το αρχείο /proc/<PID>/numa_maps μας παρέχει πληροφορίες για:

- Τη θέση της μνήμης σε σχέση με τον σωρό (heap) και τη στοίβα (stack)
- Την εικονική διεύθυνση των σελίδων

- Την τοποθεσία τους σε σχέση με τους κόμβους του συστήματος
- Ποιες σελίδες είναι καθαρές και ποιες όχι
- Το μέγεθος της κάθε σελίδας
- Τις ανώνυμες σελίδες που έχει η εφαρμογή
- Τα ανοιχτά αρχεία που έχει η εφαρμογή και το μέγεθός τους

Η κυριότερη χρήση που είχε το συγκεκριμένο εργαλείο στην έρευνά μας είναι η πληροφόρηση μας για την κατανομή της μνήμης κάθε εφαρμογής στους κόμβους του συστήματος. Ειδικότερα, θέλουμε να ξέρουμε πόσες σελίδες της μνήμης της έχουν δεσμευτεί σε κάθε κόμβο ώστε να μπορούμε να εξακριβώσουμε αν μια μεταφορά μνήμης με χρήση της κλήσης migratepages λειτουργεί σωστά. Δηλαδή αν θεωρήσουμε ότι η εφαρμογή X έχει αρχικά ισομοιρασμένες τις σελίδες της στους κόμβους 0 και 1 και εκτελέσουμε την εντολή *migratepages <PID> 0 1* θα πρέπει αμέσως μετά όλες οι σελίδες της να βρίσκονται στον κόμβο 1 κάτι που μπορούμε να επιβεβαιώσουμε με επισκόπηση του αρχείου /proc/<PID>/numa_maps.

3.3 Μετροπρογράμματα

Σε αυτό το υποκεφάλαιο θα αναλύσουμε τις εφαρμογές που χρησιμοποιήσαμε στα πειράματά μας. Αποφασίσαμε να χρησιμοποιήσουμε τη σουίτα SPEC CPU 2017 [Buce18] που περιέχει 43 benchmarks οργανωμένα σε τέσσερεις κατηγορίες εφαρμογών. Πιο αναλυτικά, αυτές είναι:

- Η κατηγορία SPECrate 2017 Integer
- Η κατηγορία SPECrate 2017 Floating Point
- Η κατηγορία SPECspeed 2017 Integer
- Η κατηγορία SPECspeed 2017 Floating Point

Στους Πίνακες 3.4 και 3.5 μπορούμε να δούμε αναλυτικά τα benchmarks που περιέχει η κάθε σουίτα καθώς και πληροφορίες για αυτά, όπως η γλώσσα που έχουν υλοποιηθεί και ο σκοπός που επιτελούν.

SPECrate 2017 Integer	SPECspeed 2017 Integer	Γλώσσα	Εφαρμογή
500.perlbench_r	600.perlbench_s	C	Perl interpreter
502.gcc_r	602.gcc_s	C	GNU C compiler
505.mcf_r	605.mcf_s	C	Route planning
520.omnetpp_r	620.omnetpp_s	C++	Discrete Event simulation - computer network
523.xalancbmk_r	623.xalancbmk_s	C++	XML to HTML conversion via XSLT
525.x264_r	625.x264_s	C	Video compression
531.deepsjeng_r	631.deepsjeng_s	C++	Artificial Intelligence: alpha-beta tree search (Chess)
541.leela_r	641.leela_s	C++	Artificial Intelligence: Monte Carlo tree search (Go)
548.exchange2_r	648.exchange2_s	Fortran	Artificial Intelligence: recursive solution generator (Sudoku)
557.xz_r	657.xz_s	C	General data compression

Πίνακας 3.4: Οι σουίτες SPECrate 2017 Integer και SPECspeed 2017 Integer

Στη μελέτη μας χρησιμοποιήσαμε όλα τα benchmarks της σουίτας SPEC 2017 εκτός από το 511.povray_r, το οποίο έτρεχε σε πολύ σύντομο χρόνο με αποτέλεσμα να μη μας δίνει αξιοποιήσιμη πληροφορία, και το 627.cam4_s που δεν μπορούσε να ολοκληρωθεί η εκτέλεσή του εξαιτίας κάποιου σφάλματος εκτέλεσης (segmentation fault). Για κάθε benchmark υπάρχουν τρεις διαφορετικές κατηγορίες από αρχεία εισόδου, οι οποίες είναι test, train και ref με σειρά αυξανόμενου χρόνου εκτέλεσης. Με άλλα λόγια, η κατηγορία test περιέχει inputs για τα οποία η εκτέλεση των benchmarks

SPECrate 2017 Floating Point	SPECspeed 2017 Floating Point	Γλώσσα	Εφαρμογή
503.bwaves_r	603.bwaves_s	Fortran	Explosion modeling
507.cactuBSSN_r	607.cactuBSSN_s	C++, C, Fortran	Physics: relativity
508.namd_r	-	C++	Molecular dynamics
510.parest_r	-	C++	Biomedical imaging
511.povray_r	-	C++, C	Ray tracing
519.lbm_r	619.lbm_s	C	Fluid dynamics
521.wrf_r	621.wrf_s	Fortran, C	Weather forecasting
526.blender_r	-	C++, C	3D rendering and animation
527.cam4_r	627.cam4_s	Fortran, C	Atmosphere modeling
-	628.pop2_s	Fortran, C	Wide-scale ocean modeling (climate level)
538.imagick_r	638.imagick_s	C	Image manipulation
544.nab_r	644.nab_s	C	Molecular dynamics
549.fotonik3d_r	649.fotonik3d_s	Fortran	Computational Electromagnetics
554.roms_r	654.roms_s	Fortran	Regional ocean modeling

Πίνακας 3.5: Οι σουίτες SPECrate 2017 Floating Point και SPECspeed 2017 Floating Point

διαρκεί σύντομο χρονικό διάστημα (τάξη μεγέθους 10^0 δευτερόλεπτα), η train περιέχει inputs για τα οποία η εκτέλεση απαιτεί μια τάξη μεγέθους παράπανω χρόνο (10^1 δευτερόλεπτα) και τέλος η κατηγορία ref περιέχει inputs που απαιτούν μεγαλύτερο χρονικό διάστημα εκτέλεσης (τάξη μεγέθους 10^2 δευτερόλεπτα). Εμείς στα πειράματά μας τρέξαμα τα benchmarks με όλες τις παραπάνω κατηγορίες αρχείων εισόδου. Ακόμη, κάποα από τα benchmarks της σουίτας μπορούν να τρέξουν με πολλαπλά αρχεία εισόδου για κάθε κατηγορία από τις test, train και ref, οπότε τρέξαμε τα benchmarks όσες φορές χρειάζονταν για όλα τα εναλλακτικά αρχεία που υπάρχουν. Στον Πίνακα 3.6 φαίνονται όλα τα benchmarks της σουίτας SPEC 2017 που χρησιμοποιήσαμε στα πειράματά μας μαζί με τον αριθμό των διαφορετικών αρχείων εισόδου του καθενός για την κατηγορία ref, η οποία αποτελεί την πιο σημαντική για την εξαγωγή αξιόπιστων συμπερασμάτων. Σημειώνεται ότι οι εφαρμογές της μορφής 5xx είναι μονονηματικές ενώ οι εφαρμογές της μορφής 6xx είναι πολυνηματικές.

500s series		600s series	
Benchmark	**of inputs**	**Benchmark**	**of inputs**
500.perlbench_r	3	600.perlbench_s	3
502.gcc_r	5	602.gcc_s	3
503.bwaves_r	4	603.bwaves_s	2
505.mcf_r	1	605.mcf_s	1
507.cactuBSSN_r	1	607.cactuBSSN_s	1
508.namd_r	1	619.lbm_s	1
510.parest_r	1	620.omnetpp_s	1
519.lbm_r	1	621.wrf_s	1
520.omnetpp_r	1	623.xalancbmk_s	1
521.wrf_r	1	625.x264_s	3
523.xalancbmk_r	1	628.pop2_s	1
525.x264_r	3	631.deepsjeng_s	1
526.blender_r	1	638.imagick_s	1
527.cam4_r	1	641.leela_s	1
531.deepsjeng_r	1	644.nab_s	1
538.imagick_r	1	648.exchange2_s	1
541.leela_r	1	649.fotonik3d_s	1
544.nab_r	1	654.roms_s	1
548.exchange2_r	1	657.xz_s	2
549.fotonik3d_r	1	-	-
554.roms_r	1	-	-
557.xz_r	3	-	-

Πίνακας 3.6: Τα benchmarks που χρησιμοποιήσαμε στην εργασία.

Κεφάλαιο 4

Κίνητρο Εργασίας

Με αυτό το κεφάλαιο ξεκινάμε ουσιαστικά το κυρίως κομμάτι της έρευνάς μας. Αρχικά, ορίζουμε τις μετρικές που χαρακτηρίζουν τις εφαρμογές και τον τρόπο με τον οποίο υπολογίζονται από τους hardware performance counters του συστήματος. Στη συνέχεια, εκτελούμε πειράματα για τις εφαρμογές της σουίτας SPEC 2017 προκειμένου να προσδιορίσουμε τη συμπεριφορά τους και τον τρόπο που επηρεάζονται από τις διάφορες τοπολογίες. Τέλος, προσπαθούμε να θεμελιώσουμε τις συσχετίσεις που υπάρχουν μεταξύ των μετρικών των εφαρμογών και της απόδοσής τους και καταλήγουμε σε κάποια συμπεράσματα που εισάγουμε στο μοντέλο του resource manager του επόμενου κεφαλαίου.

4.1 Τοπολογίες

Εδώ ορίζουμε τις τοπολογίες για το μηχάνημα που χρησιμοποιήσαμε στα πειράματά μας. Με τον όρο τοπολογία αναφερόμαστε στον τρόπο που χρησιμοποιεί τους πόρους του συστήματος μια εφαρμογή, δηλαδή ποιους πυρήνες και ποιες μνήμες δεσμεύει για την εκτέλεσή της. Ορίζουμε τις παρακάτω τοπολογίες για το Sandman:

- **Τοπολογία local**: Τοπική εκτέλεση του benchmark σε ένα κόμβο του συστήματος, όπως για παράδειγμα στον κόμβο 0 (Σχήμα 4.1a).

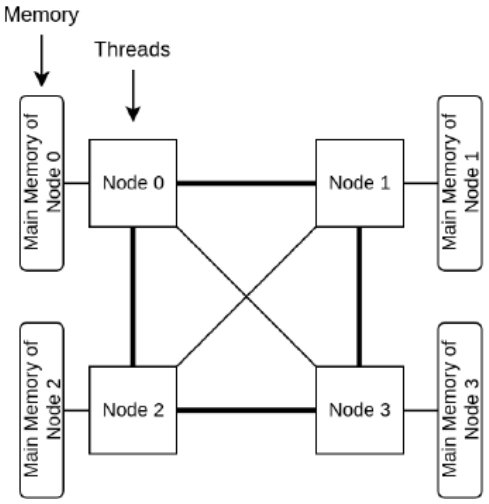
- **Τοπολογία remote near**: Εκτέλεση του benchmark μεταξύ δύο κόμβων που απέχουν απόσταση 21, όπως για παράδειγμα μεταξύ των κόμβων 0 και 1. Συγκεκριμένα, το benchmark χρησιμοποιεί πυρήνα που ανήκει στον ένα κόμβο και μνήμη που ανήκει στον άλλο κόμβο. Οι αποστάσεις μεταξύ των κόμβων λαμβάνονται σύμφωνα με τον πίνακα αποστάσεων (Σχήμα 4.1b).

- **Τοπολογία interleave near**: Εκτέλεση του benchmark μεταξύ δύο κόμβων που απέχουν απόσταση 21. Συγκεκριμένα, το benchmark χρησιμοποιεί πυρήνα που ανήκει στον ένα κόμβο και μνήμη σε configuration interleave, δηλαδή εναλλάξ από τους δύο κόμβους (Σχήμα 4.1c).

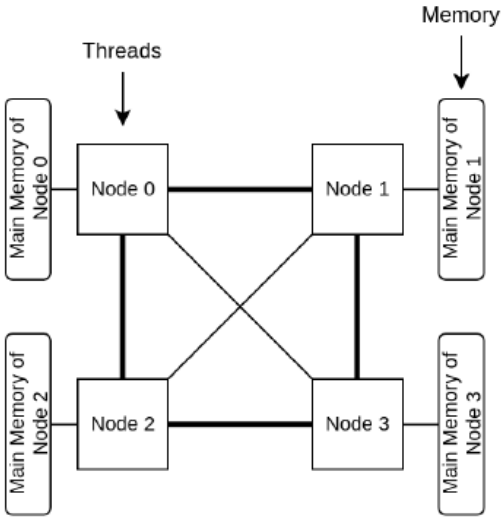
- **Τοπολογία remote far**: Εκτέλεση του benchmark μεταξύ δύο κόμβων που απέχουν απόσταση 30, όπως για παράδειγμα μεταξύ των κόμβων 0 και 3. Αντίστοιχη με την τοπολογία 1 (Σχήμα 4.1d).

- **Τοπολογία interleave far**: Εκτέλεση του benchmark μεταξύ δύο κόμβων που απέχουν απόσταση 30. Αντίστοιχη με την τοπολογία 2 (Σχήμα 4.1e).

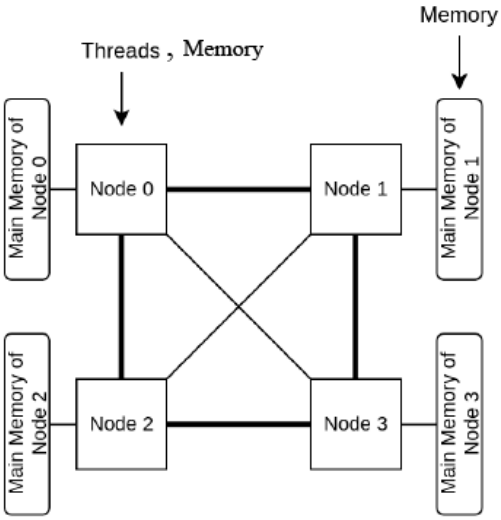
Από αυτές τις τοπολογίες στην έρευνά μας χρησιμοποιήσαμε εκτεταμένα τις local, remote (near) και interleave (near). Αυτό συμβαίνει, διότι θέλαμε να παρατηρήσουμε γενικές συμπεριφορές των benchmarks στις διάφορες τοπολογίες που υπάρχουν στο Sandman. Επειδή οι τοπολογίες remote (near) και remote (far) μοιάζουν αρκετά μεταξύ τους (με μόνη διαφορά της απόσταση πυρήνα-μνήμης), όπως επίσης και το ζεύγος interleave (near) και interleave (far), αποφασίσαμε να συμπεριλάβουμε μία από το καθένα ώστε να πάρουμε τις απαραίτητες πληροφορίες για την πορεία της έρευνάς μας χωρίς να απαιτούνται πολλά πειράματα που απαιτούν αρκετό χρόνο.



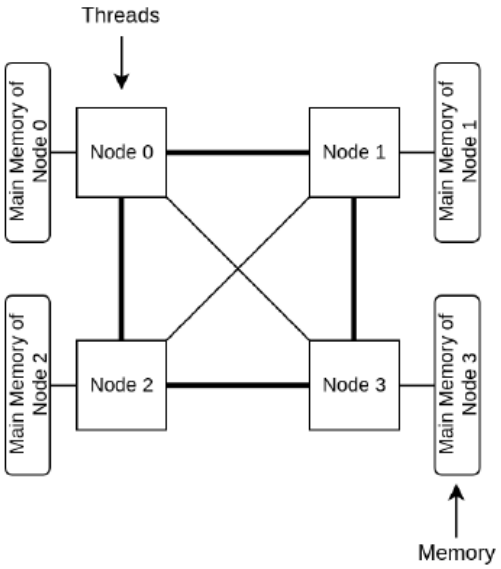
(a) Τοπολογία local



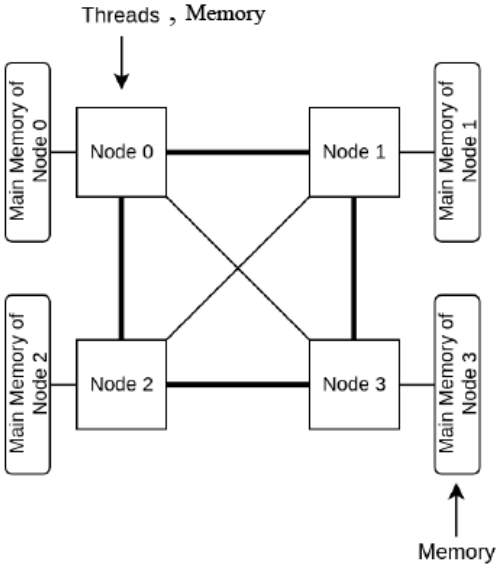
(b) Τοπολογία remote near



(c) Τοπολογία interleave near



(d) Τοπολογία remote far



(e) Τοπολογία interleave far

Σχήμα 4.1: NUMA τοπολογίες στο μηχάνημα Sandman.

4.2 Παράμετροι

Σε αυτή την ενότητα περιγράφουμε τις παραμέτρους που μετράμε για κάθε benchmark και τις οποίες χρησιμοποιούμε για να κατανοήσουμε τον αντίκτυπο της NUMA αρχιτεκτονικής στην επίδοση της εφαρμογής. Οι παράματροι αυτες είναι το IPC, το LLC-MPKI και το TLB-MPKI με το εργαλείο perf και η χρήση του εύρου ζώνης της μνήμης (bandwidth) με το εργαλείο pcm-memory.x.

IPC
Η παράμετρος Instructions Per Cycle (IPC) μας δείχνει τον αριθμό των εντολών που εκτελούνται (γίνονται commited) κατά μέσο όρο σε κάθε κύκλο λειτουργίας του επεξεργαστή. Το IPC είναι πάρα πολύ διαδεδομένη μετρική, καθώς χρησιμοποιείται ως δείκτης μέτρησης της απόδοσης ενός προγράμματος. Συγκεκριμένα, όσο μεγαλύτερο είναι το IPC τόσο περισσότερες εντολές εκτελούνται σε κάθε κύκλο λειτουργίας και η εφαρμογή μας τρέχει πιο αποδοτικά σε συντομότερο χρόνο. Ο τύπος για τον υπολογισμό του από τους performance counters είναι ο παρακάτω:

$$IPC = \frac{instructions}{cpu\_cycles} \tag{4.1}$$

Ο αντίστροφος της παραμέτρου IPC είναι το CPI (Cycles Per Instruction) που μας δείχνει τον αριθμό των κύκλων του επεξεργαστή που απαιτούνται κατά μέσο όρο για την εκτέλεση μιας εντολής. Το CPI χρησιμοποιείται εξίσου συχνά με το IPC και ορίζεται ως:

$$CPI = \frac{cpu\_cycles}{instructions} \tag{4.2}$$

MPKI
Η παράμετρος Last-Level Cache Misses Per Kilo Instructions (LLC MPKI ή χάρην συντομίας MPKI) μας δείχνει τον αριθμό των αποτυχιών (misses) που συμβαίνουν κατά μέσο όρο στην cache που βρίσκεται υψηλότερα στην ιεραρχία του συστήματός μας (LLC) κάθε χίλιες εντολές του προγράμματός μας. Το MPKI είναι μια ευρέως χρησιμοποιούμενη μετρική των προγραμμάτων, που σχετίζεται άμεσα με την εφαρμογή που μελετάμε και με την αρχιτεκτονική του μηχανήματός μας. Όσο μικρότερη είναι η τιμή του MPKI τόσο λιγότερες αποτυχίες συμβαίνουν στην LLC κατά την εκτέλεση του προγράμματος και επομένως επηρεάζεται λιγότερο από τη μνήμη, διότι κάθε αποτυχία στην LLC οδηγεί αναγκαστικά σε προσπέλαση της κύριας μνήμης. Ο τύπος για τον υπολογισμό του MPKI από τους performance counters είναι ο παρακάτω:

$$MPKI = \frac{LLC\_load\_misses + LLC\_store\_misses}{instructions} \cdot 1000 \tag{4.3}$$

TLB MPKI
Η παράμετρος Translation Lookaside Buffer Misses Per Kilo Instructions (TLB MPKI) μας δείχνει τον αριθμό των αποτυχιών (misses) που συμβαίνονυ κατά μέσο όρο στο TLB κάθε χίλιες εντολές του προγράμματός μας. Το TLB MPKI χρησιμοποιείται ως μετρική για την αξιολόγηση της σχέσης μιας εφαρμογής με τη μνήμη της. Όσο μεγαλύτερη τιμή έχει η παράμετρος αυτή τόσο περισσότερο εξαρτημένη από τη μνήμη (memory intensive) είναι η εφαρμογή μας, ενώ όσο χαμηλότερη είναι η εξάρτηση τόσο χαμηλότηερη είναι και η τιμή της. Ο τύπος για τον υπολογισμό του TLB MPKI από τους performance counters είναι ο παρακάτω:

$$TLB\ MPKI = \frac{dTLB\_load\_misses + dTLB\_store\_misses + iTLB\_load\_misses}{instructions} \cdot 1000 \tag{4.4}$$

Bandwidth
Η παράμετρος Bandwidth (BW) μας δείχνει το μέσο όρο τη ροή της μνήμης που διέρχεται από το σύστημά μας κατά τη διάρκεια εκτέλεσης της εφαρμογής. Όσο μεγαλύτερη είναι η τιμή της παραμέτου αυτής τόσο περισσότερη ροή διέρχεται από το σύστυμα. Μέσω του εργαλείου pcm-memory.x όπως είδαμε μπορούμε να πάρουμε αναλυτικά τη ροή για κάθε κόμβο του συστήματός μας καθώς και τη συνολική ροή. Επίσης, καθεμιά από τις ροές αυτές διαχωρίζεται σε δύο κύριες ροές που αντιστοιχούν σε προσπελάσεις ανάγνωσης και εγγραφής της μνήμης αντίστοιχα. Σε αυτή τη φάση της συλλογής μετρήσεων για τα benchmarks αρκεστήκαμε στη συνολική ροή που διέρχεται από το σύστημα η οποία ουσιαστικά ταυτίζεται με τη ροή που χρησιμοποιεί κάθε benchmark αφού τρέχει μόνο του στο σύστημα σε κάθε πείραμα που εκτελούμε. Ένας εναλλακτικός τρόπος προσδιορισμού του bandwidth που χρειάζεται ένα benchmark ανά πάσα στιγμή στο σύστημα και χρησιμοποιείται όπως θα δούμε στη συνέχεια από τον resource manager είναι μέσω της μέτρησης των read και write requests που συμβαίνουν στον ελεκτή της μνήμης (memory controller) σε ένα συγκεκριμένο χρονικό διάστημα. Ο τύπος υπολογισμού του bandwidth με αυτό τον τρόπο είναι:

$$Bandwidth = \frac{NumReads + NumWrites}{Profiling\ Interval} \cdot BlockSize \tag{4.5}$$

όπου BlockSize είναι το μέγεθος του block για κάθε λειτουργία ανάγνωσης ή εγγραφής στη μνήμη και ProfilingInterval το χρόνικο διάστημα στο οποίο λαμβάνουμε τις μετρήσεις.

Αποτελέσματα
Στους Πίνακες 4.1 και 4.2 βλέπουμε τις τιμές των παραπάνω παραμέτρων για όλα τα benchmarks που πήραμε μετρήσεις. Οι τιμές αυτές αντιστοιχούν στην τοπολογία local όπου κάθε benchmark τρέχει μόνο του στο σύστημα σε ένα κόμβο παίρνοντας CPU και μνήμη από αυτόν και επομένως αποτελούν τιμές αναφοράς που χαρακτηρίζουν τις εφαρμογές μας. Στην αριστερή στήλη φαίνεται το όνομα κάθε benchmark σε επεκτεταμένη μορφή ώστε να μπορούμε να ξεχωρίζουμε τις διαφορετικές εκτελέσεις των benchmarks που περιέχουν πολλαπλά αρχεία εισόδου μεταξύ τους. Η επεκτεταμένη μορφή περιέχει πληροφορία για το όνομα του benchmark και το αρχείο εισόδου του και κατασκευάζεται σύμφωνα με το πρότυπο *"benchmark_name"_"number of input file"*. Οι επόμενες στήλες του πίνακα περιέχουν κατά σειρά το συνολικό χρόνο εκτέλεσης του benchmark σε δευτερόλεπτα (TT), το IPC, το MPKI, το TLB MPKI και το BW όπως τα λάβαμε με χρήση των εργαλείων perf και pcm-memory.x.

4.3 Απόδοση των εφαρμογών στα πειράματα

Αφού πήραμε τις παραμέτρους για την τοπολογία local για όλες τις εφαρμογές μας θέλαμε να δούμε πως μεταβάλλεται η απόδοσή τους σε σχέση με την απόσταση της μνήμης που χρησιμοποιούν. Για το σκοπό αυτό, λοιπόν, εκτελέσαμε πειράματα για τις τοπολογίες remote και interleave όπως αυτές ορίστηκαν στα σχήματα 4.1b και 4.1c αντίστοιχα. Στα πειράματα αυτά μας ενδιέφερε μόνο ο χρόνος εκτέλεσης και η σχέση που έχει με το αντίστοιχο baseline σενάριο, δηλαδή την εκτέλεση του ίδιου benchmark στην τοπολογία local.

Για να γίνουν κατανοητά τα αποτελέσματα των πειραμάτων πρέπει αρχικά να ορίσουμε την έννοια της απόδοσης μιας εφαρμογής όταν αυτή τρέχει σε μια συγκεκριμένη τοπολογία. Ορίζουμε, επομένως, την απόδοση μιας εφαρμογής ως εξής:

$$Performance(i) = \frac{Total\ Time_i}{Total\ Time_{local}} \cdot 100\% \equiv \frac{IPC_{local}}{IPC_i} \cdot 100\% \tag{4.6}$$

όπου το i μας δείχνει την τοπολογία. Με αυτόν τον ορισμό βρίσκουμε την απόδοση για κάθε τοπολογία θεωρώντας ως baseline απόδοση το 100%. Αν θέλουμε να υπολογίσουμε τη μεταβολή της απόδοσης από την τοπολογία 0 στην τοπολογία i μπορούμε να χρησιμοποιήσουμε τη σχέση

$$\triangle Performance(0 \rightarrow i) = (\frac{Total\ Time_i}{Total\ Time_{local}} - 1) \cdot 100\% \tag{4.7}$$

Benchmark	TT	IPC	MPKI	TLB MPKI	BW
500.perlbench_r_0	249.8213	1.9008	0.01169	0.2764	39.29
500.perlbench_r_1	208.8784	1.5596	0.09784	0.1684	87.31
500.perlbench_r_2	185.1351	1.6494	0.5457	0.5624	214.44
502.gcc_r_0	66.1357	1.1542	0.9962	1.7483	364.14
502.gcc_r_1	92.4269	1.1462	1.3808	1.9366	469.79
502.gcc_r_2	87.6188	1.2271	1.7351	1.855	661.72
502.gcc_r_3	84.1235	1.01259	6.0823	1.6033	1759.75
502.gcc_r_4	126.74034	0.9427	4.4988	1.9079	1338.38
503.bwaves_r_0	258.1106	1.7582	4.9787	0.1680	2630.73
503.bwaves_r_1	435.6441	1.6175	5.8042	0.1991	2903.11
503.bwaves_r_2	357.58	1.6836	5.3436	0.179	2742.73
503.bwaves_r_3	442.5118	1.7334	5.2662	0.172	2722.29
505.mcf_r	506.0017	0.7285	14.3880	4.9070	2305.63
507.cactuBSSN_r	478.8759	1.2607	3.0138	0.2862	997.57
508.namd_r	458.9689	2.2271	0.04931	0.01046	149.98
510.parest_r	672.1901	2.1731	0.2423	0.3285	186.24
519.lbm_r	498.8904	1.3622	3.2309	0.3652	3394.22
520.omnetpp_r	621.3706	0.6524	7.7763	8.1619	1189.5
521.wrf_r	1595.2488	1.1249	0.6862	0.1103	480.07
523.xalancbmk_r	559.6631	0.882	0.5993	10.1146	199.88
525.x264_r_0	75.6822	2.6295	0.2669	0.0368	345.34
525.x264_r_1	321.5565	7.6912e-05	100.6736	0.0848	119.12
525.x264_r_2	283.8836	2.7558	0.09933	0.073	118.74
526.blender_r	491.3456	1.5063	0.6131	0.9138	222.14
527.cam4_r	859.5855	1.3417	0.6309	0.3329	353.98
531.deepsjeng_r	467.2693	1.5629	0.4262	0.2024	186.47
538.imagick_r	700.5715	2.5949	0.006	0.0159	37.09
541.leela_r	694.0687	1.1956	0.0013	0.0538	33.68
544.nab_r	697.8799	1.233	0.0402	0.0109	114.38
548.exchange2_r	834.7775	1.9941	0.0002	0.0678	31.88
549.fotonik3d_r	642.3222	1.4470	15.0958	1.1732	7457.43
554.roms_r	701.7977	1.6694	3.831	0.537	2122.71
557.xz_r_0	171.4317	0.9316	1.9513	5.1471	478.14
557.xz_r_1	218.7761	1.9571	0.1874	0.9244	135.27
557.xz_r_2	163.7431	1.4025	0.6693	1.8282	265.24

Πίνακας 4.1: Αποτελέσματα των benchmarks της σειράς 500 της σουίτας SPEC 2017 για το μηχάνημα Sandman και την τοπολογία local.

Το πρόσημο στην περίπτωση αυτή έχει φυσική σημασία και αν είναι θετικό μας δείχνει ότι έχουμε μείωση της απόδοσης (επιβράδυνση της εφαρμογής) ενώ σε αντίθετη περίπτωση έχουμε αύξηση της απόδοσης (επιτάχυνση της εφαρμογής).

4.3.1 Scenario 1 - Alone Execution

Το συγκεκριμένο σετ από πειράματα που αντιστοιχεί στις μεμονωμένες εκτελέσεις των benchmarks δηλαδή χωρίς να τρέχουν άλλες εφαρμογές ταυτόχρονα στο σύστημα το ονομάσαμε Scenario 1 ώστε να το διαχωρίζουμε από τα υπόλοιπα που θα ακολουθήσουν. Στον πίνακα Α.2 βλέπουμε τα αποτελέσματα σχετικά με τη μεταβολή της απόδοσης των εφαρμογών μας όταν αυτές τρέχουν στις τοπολογίες

Benchmark	TT	IPC	MPKI	TLB MPKI	BW
600.perlbench_s_0	250.1376	1.8985	0.0126	0.2805	39.25
600.perlbench_s_1	180.8754	1.5492	0.0983	0.2199	88.98
600.perlbench_s_2	155.8813	1.6684	0.5922	0.5393	235.52
602.gcc_s_0	402.166	1.1882	11.0803	1.4746	2703.03
602.gcc_s_1	182.0229	1.1352	0.5939	1.8546	223.33
602.gcc_s_2	171.3677	1.1697	0.7893	1.8471	288.19
603.bwaves_s_0	1408.247	0.8553	2.67	0.0866	8540.69
603.bwaves_s_1	1399.4506	0.8882	2.5135	0.0799	8088.03
605.mcf_s	1045.9709	0.6392	15.9323	11.0101	2179.3
607.cactuBSSN_s	794.4153	0.4799	5.45	12.7984	7845.52
619.lbm_s	1391.0048	0.1042	25.2749	0.8031	15577.67
620.omnetpp_s	632.5717	0.6679	7.6155	8.0008	1179.47
621.wrf_s	1425.6028	0.6137	0.8214	0.189	3874.71
623.xalancbmk_s	571.402	0.8807	0.6113	10.0048	204.68
625.x264_s_0	76.8868	2.6268	0.2667	0.0355	345.91
625.x264_s_1	278.0524	2.746	0.1104	0.0829	118.44
625.x264_s_2	283.75	2.7434	0.0978	0.0753	119.66
628.pop2_s	1837.0487	1.5087	1.5655	0.1337	3685.31
631.deepsjeng_s	579.1807	1.501	1.0404	0.2174	445.38
638.imagick_s	1649.0669	1.2927	0.0819	0.1013	612.82
641.leela_s	702.2522	1.1949	0.0021	0.0542	32.68
644.nab_s	791.4798	0.8075	0.0472	0.0413	922.71
648.exchange2_s	866.1795	1.9517	0.0002	0.0678	32.51
649.fotonik3d_s	1022.9178	0.2138	15.2425	0.988	16521.07
654.roms_s	2483.7832	0.313	10.8173	2.3574	15309.58
657.xz_s_0	372.2493	0.8002	2.4508	2.6748	3547.17
657.xz_s_1	329.8812	0.4184	9.5861	8.0505	8631.14

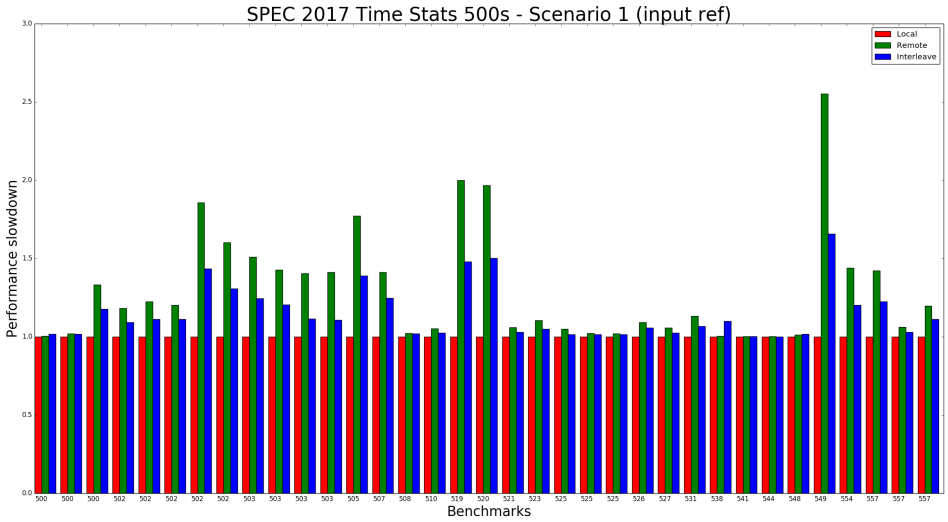
Πίνακας 4.2: Αποτελέσματα των benchmarks της σειράς 600 της σουίτας SPEC 2017 για το μηχάνημα Sandman και την τοπολογία local.

remote και interleave. Για να γίνουν καλύτερα κατανοητά τα αποτελέσματα όσον αφορά στην απόδοση των benchmark παραθέτουμε και τις αντίστοιχες γραφικές παραστάσεις 4.2 και 4.3.

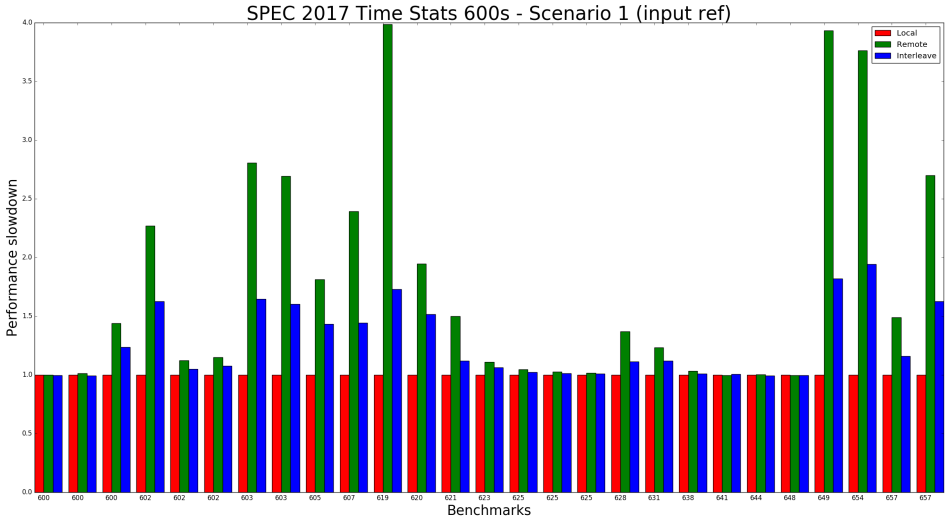
Αποτελέσματα και παρατηρήσεις

Αρχικά, παρατηρούμε ότι η απόδοση των benchmarks δεν επηρεάζεται με τον ίδιο τρόπο όταν αλλάζει η τοπολογία. Αυτό το γεγονός δεν μας φαίνεται περίεργο αφού έτσι κι αλλιώς αναμέναμε τέτοια συμπεριφορά. Είναι αξιοσημείωτο, ωστόσο, ότι πολλά benchmarks επηρεάζονται ελάχιστα από την τοπολογία, όπως τα 541.leela_r και 544.nab_r. Τα συγκεκριμένα benchmarks παρουσιάζουν από τις χαμηλότερες τιμές όσον αφορά τις παραμέτρους MPKI, TLB MPKI και BW όπως φαίνεται με επισκόπηση του πίνακα 4.1. Υπάρχουν επίσης ορισμένα benchmarks, όπως τα 641.leela_r και 648.exchange_r, των οποίων η απόδοση στις τοπολογίες remote και interleave φαίνεται να είναι οριακά μικρότερη από 100% δηλαδή φαίνεται πως οριακά επιταχύνονται σε αυτές τις τοπολογίες. Προφανώς κάτι τέτοιο δε συμβαίνει στην πράξη αφού η τοπική τοπολογία είναι πάντα η αποδοτικότερη στις μεμονωμένες εκτελέσεις και οφείλεται σε σφάλματα μετρήσεων που θεωρούμε ότι είναι μικρότερα από 2%.

Από την άλλη βλέπουμε ότι η απόδοση κάποιων benchmarks μεταβάλλεται σημαντικά από την



Σχήμα 4.2: Αποτελέσματα Scenario 1 για τη σειρά 500s.



Σχήμα 4.3: Αποτελέσματα Scenario 1 για τη σειρά 600s.

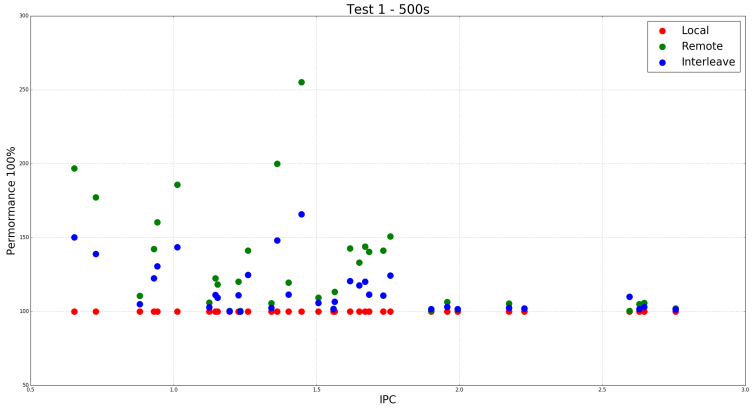
αλλαγή τοπολογίας. Χαρακτηριστικό παράδειγμα είναι το 619.lbm_s για το οποίο ισχύει

$$\triangle Performance(local \rightarrow remote) = (\frac{Total\ Time_{remote}}{Total\ Time_{local}} - 1) \cdot 100\% = 298\%$$

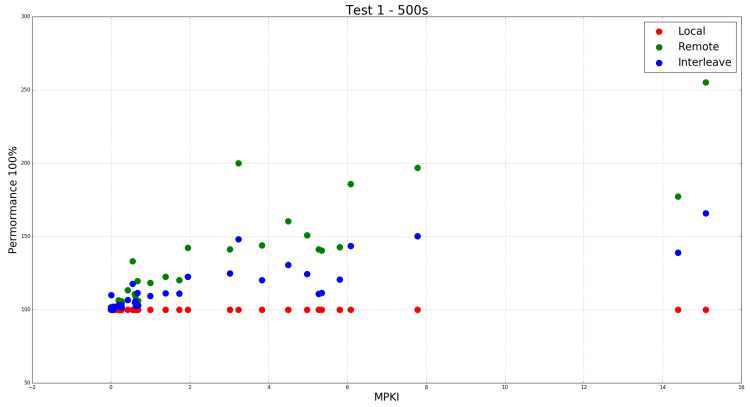
και

$$\triangle Performance(local \rightarrow interleave) = (\frac{Total\ Time_{interleave}}{Total\ Time_{local}} - 1) \cdot 100\% = 73\%$$

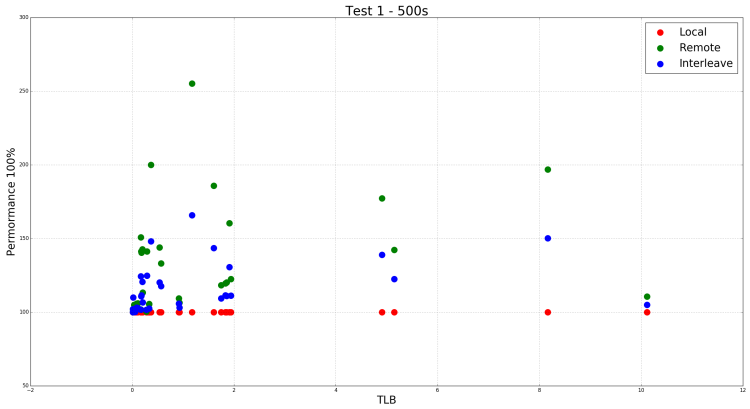
δηλαδή επιβραδύνεται κατά 3 φορές στη remote τοπολογία σε σχέση με τη local και 0.73 φορές στην interleave τοπολογία. Το συγκεκριμένο benchmark παρουσιάζει από τις υψηλότερες τιμές των παραμέτρων MPKI και BW. Αυτές οι παρατηρήσεις σχετικά με τη συσχέτιση που μπορεί να έχουν οι τιμές των παραμέτρων των εφαρμογών με την απόδοσή τους μας κίνησαν το ενδιαφέρον και προχωρήσαμε στην κατασκευή των γραφικών παραστάσεων 4.4 και 4.5 για την καλύτερη οπτική αποτύπωσή τους.



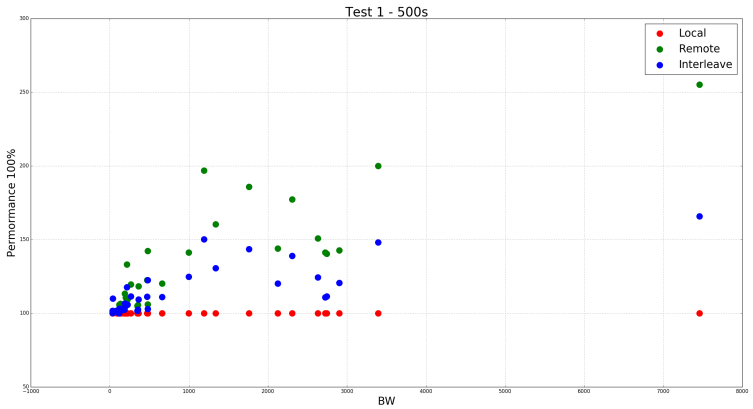
(a) Συσχέτιση Performance - IPC



(b) Συσχέτιση Performance - MPKI

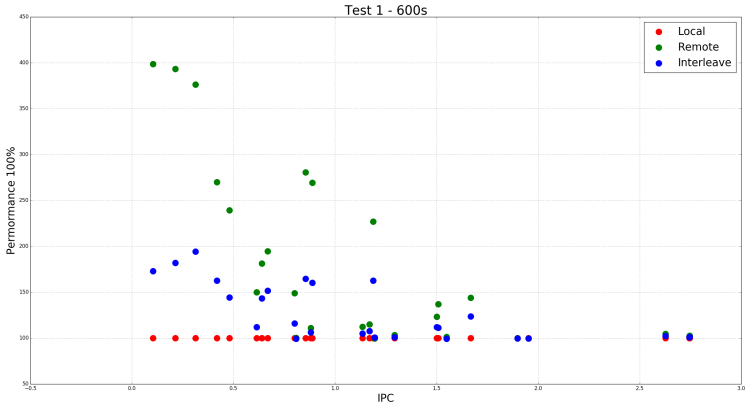


(c) Συσχέτιση Performance - TLB MPKI

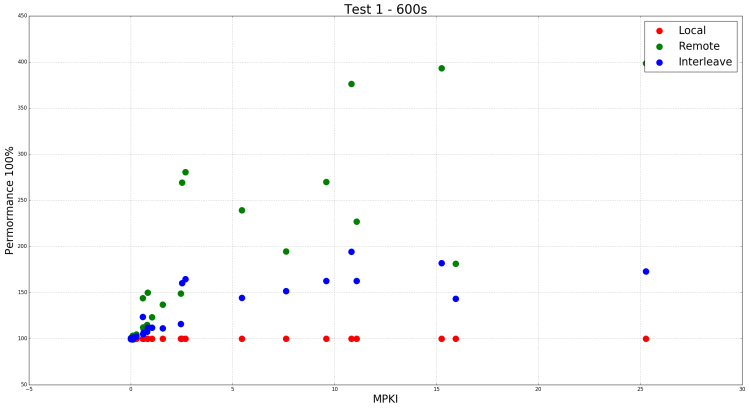


(d) Συσχέτιση Performance - BW

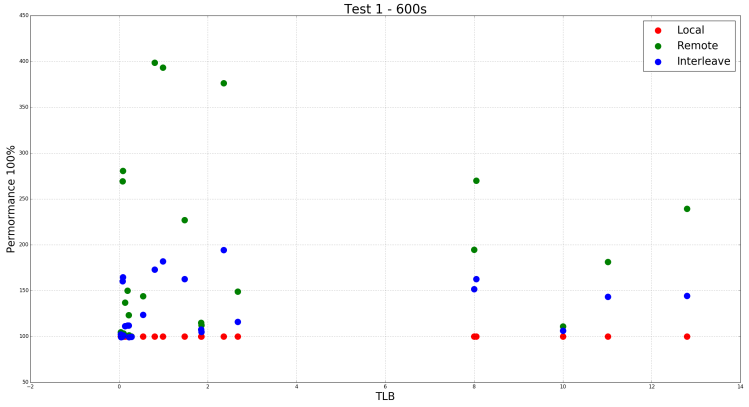
Σχήμα 4.4: Συσχετίσεις μεταξύ των παραμέτρων και της απόδοσης των εφαρμογών για τη σειρά 500s.



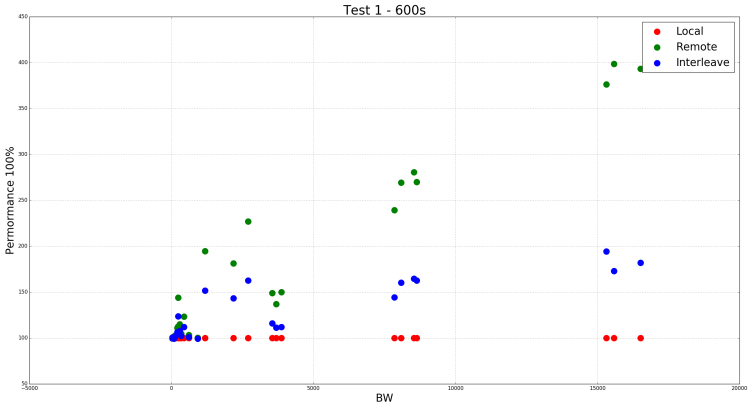
(a) Συσχέτιση Performance - IPC



(b) Συσχέτιση Performance - MPKI



(c) Συσχέτιση Performance - TLB MPKI



(d) Συσχέτιση Performance - BW

Σχήμα 4.5: Συσχετίσεις μεταξύ των παραμέτρων και της απόδοσης των εφαρμογών για τη σειρά 600s.

Με μια πρώτη επισκόπιση των γραφικών παραστάσεων επιβεβαιώνουμε τις αρχικές παρατηρήσεις μας αναφορικά με τη σχέση της απόδοσης των εφαρμογών και των παραμέτρων τους. Ειδικότερα, από τις γραφικές παραστάσεις 4.4a και 4.5a φαίνεται ότι όσο η παράμετρος IPC αυξάνεται, τα benhcmarks τρέχουν πιο αποδοτικά και επηρεάζονται λιγότερο από την αλλαγή της τοπολογίας. Αυτό προκύπτει από το γεγονός ότι όσο κινούμαστε προς τα δεξιά στον οριζόντιο άξονα τόσο ελαττώνεται το χάσμα μεταξύ των αποδόσεων στις τοπολογίες local, remote και interleave. Ακόμη, για τις παραμέτρους MPKI και BW παρατηρούμε ότι όσο αυξάνονται τόσο περισσότερο επηρεάζονται τα benchmarks από τη μεταβολή της τοπολογίας και μεγαλώνει το χάσμα της απόδοσης των εφαρμογών μεταξύ τους. Με άλλα λόγια, μια εφαρμογή που χαρακτηρίζεται από υψηλές τιμές MPKI ή/και BW επιβραδύνεται πολύ περισσότερο όταν εκτελεστεί σε μη τοπική τοπολογία σε σχέση με μια άλλη που χαρακτηρίζεται από χαμηλότερες τιμές. Γενικά, θα μπορούσαμε να πούμε προσεγγιστικά ότι η αύξηση της απόδοσης είναι λογαριθμική ως προς την αύξηση αυτών των παραμέτρων τόσο για την τοπολογία remote όσο και για την τοπολογία interleave.

Μοντέλο πρόβλεψης της απόδοσης

Ύστερα από την εξαγωγή των αποτελεσμάτων από τα πειράματα αυτά και τις παρατηρήσεις που κάναμε αποφασίσαμε σε πρώτη φάση να δημιουργήσουμε ένα μοντέλο αντίστοιχο με αυτό που υλοποιήθηκε στην ερευνητική εργασία [Arap18] με στόχο να μπορούμε να προβλέπουμε την απόδοση των εφαρμογών που τρέχουν στο σύστημα σε μη τοπικές τοπολογίες με βάση τις παραμέτρους τους. Η μεθοδολογία που ακολουθήσαμε περιελάμβανε την κατασκευή μιας οικογένειας συναρτήσεων αποτελούμενη απο κύριους και δευτερεύοντες όρους με μεταβλητούς συντελεστές και εκθέτες ώστε να εξετάσουμε όσο το δυνατόν περισσότερα μοντέλα ως προς την καταλληλότητά τους στην πρόβλεψη της απόδοσης. Το συνολικό πλήθος των συναρτήσεων που προέκυψαν ήταν της τάξης του 10^7 με αποτέλεσμα η διαδικασία εκπαίδευσής τους στα δεδομένα μας να απαιτεί αρκετές ώρες. Για κάθε συνάρτηση χρησιμοποιήσαμε τη μέθοδο του cross validation για να την εκπαιδεύσουμε και στη συνέχεια την αξιολογήσαμε με μετρικές όπως το R^2 και το MEA (Mean Absolute Error). Τα αποτελέσματα, όμως, αυτής της διαδικασίας δεν ήταν τα αναμενόμενα αφού η καλύτερη συνάρτηση που προέκυψε ήταν η

$$y = a_0 \cdot log(x_2+1)^2 + a_1 \cdot x1^{1.5} + a_2 \cdot x_2^{-2} + a_3 \cdot x_3^{-1.5} + a_4 \cdot x_4 + a_5 \cdot x_1^{1.5} \cdot x_2^{-2} + a_6 \cdot x_1^{1.5} \cdot x_3^{-1.5} + a_7 \cdot x_1^{1.5} \cdot x_4$$

με $R^2 = 0.7279$ που είναι αρκετά χαμηλό για να μπορεί να χαρακτηριστεί ακριβής στις προβλέψεις της.

Αυτό πιθανόν συνέβη διότι η σουίτα SPEC 2017 περιέχει benchmarks με αρκετά διαφορετικά χαρακτηριστικά σε σχέση με τις σουίτες SPEC 2006 και Parsec στις οποίες είχε βασιστεί η εκπαίδευση και αξιολόγηση του συγκεκριμένου μοντέλου [Arap18]. Έτσι, ενδεχομένως να χρειάζονταν κάποια επέκταση προκειμένου να μπορεί να συμπεριφερθεί με τον αναμενόμενο τρόπο παράγοντας ακριβείς προβλέψεις. Αυτή η επέκταση θα απαιτούσε την εξέταση περισσότερων συναρτήσεων και με διαφορετικά χαρακτηριστικά, όπως για παράδειγμα να διαθέτουν κάποιον άλλο κυρίαρχο όρο καθώς και περισσότερους (μη πολυωνυμικούς) δευτερεύοντες όρους. Ακόμη, είναι πιθανό να απαιτούνταν περισσότερες από τέσσερεις μεταβλητές στις συναρτήσεις που θα κατασκευάζαμε κάτι που με τη σειρά του συνεπάγεται συλλογή μετρήσεων από περισσότερους hardware performance counters. Ωστόσο, κάτι τέτοιο αντιτίθεται με τον αρχικό σκοπό μας που ορίζει ότι το μοντέλο που θα αναπτύξουμε θα είναι μεταφέρσιμο στην πλειονότητα των NUMA συστημάτων, ενώ απαιτείται και πολύς χρόνος για την κατασκευή και εκπαίδευση αυτού του όγκου των συναρτήσεων.

4.3.2 Scenario 2 - Co-execution with one instance of stress-ng on the remote node

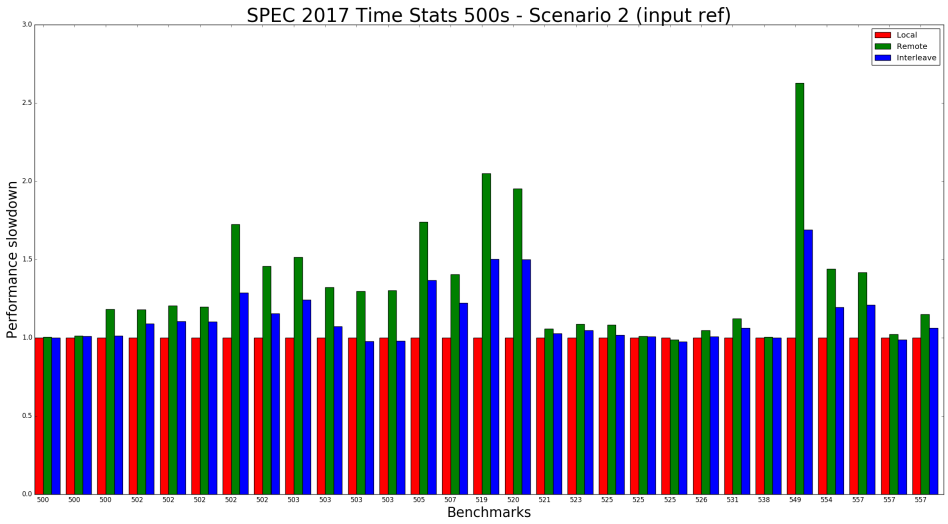
Σε αυτό το σημείο αποφασίσαμε να διαφοροποιηθούμε από τη συγκεκριμένη ερευνητική δουλειά [Arap18] και να ακολουθήσουμε μια περισσότερο πρακτική προσέγγιση στο θέμα του NUMA placement. Συγκεκριμένα, αναπτύχθηκε η ιδέα για την υλοποίηση ενός resource manager, δηλαδή

ενός user-level πρόγραμμα το οποίο θα μπορεί δυναμικά να επεμβαίνει κατά τη διάρκεια της εκτέλεσης των εφαρμογών σε ένα NUMA σύστημα και να ορίζει την τοπολογία τους με στόχο την επίτευξη υψηλότερης απόδοσης για το σύστημα συνολικά.

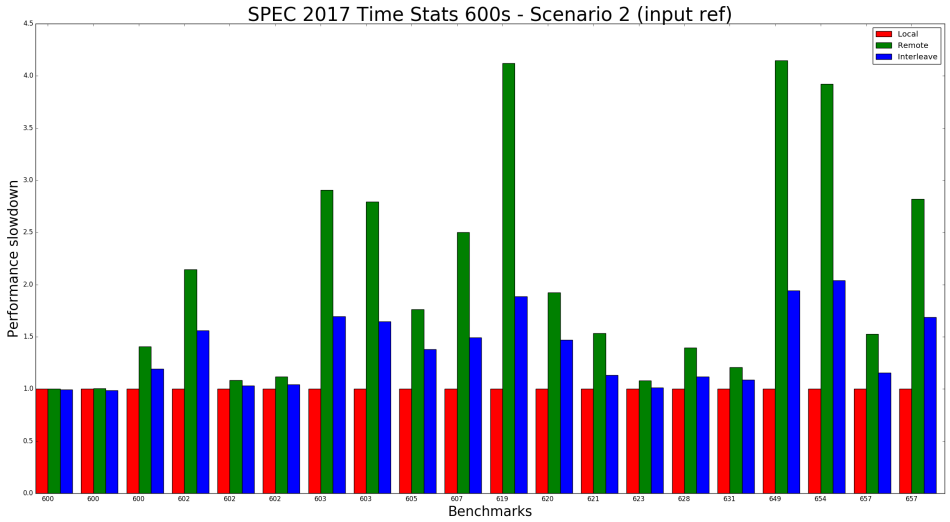
Για να καταστεί εφικτή αυτή η προσπάθεια και να αναπτυχθεί μια λογική-μοντέλο που θα επιτυγχάνει βελτίωση της απόδοσης κρίθηκε αναγκαία η εκτέλεση περισσότερων πειραμάτων που θα μας αποκάλυπταν περισσότερες πτυχές για τα benchmarks και τον τρόπο που συμπεριφέρονται όταν στο σύστημα τρέχουν και άλλες εφαρμογές που καταναλώνουν πόρους. Πιο αναλυτικά, χρησιμοποιήσαμε το εργαλείο stress-ng [stre] το οποίο έχει σχεδιαστεί με τέτοιο τρόπο ώστε να ”στρεσάρει” ένα υπολογιστικό σύστημα με διάφορους τρόπους. Εμάς μας ενδιέφερε το θέμα της μνήμης οπότε χρησιμοποιήσαμε το stress-ng για να καταναλώνει bandwidth από συγκεκριμένους κόμβους και να δούμε στη συνέχεια πως θα αλλάξει η συμπεριφορά και η απόδοση της κύριας εφαρμογής που τρέχουμε.

Πιο αναλυτικά, στο επόμενο σετ από πειράματα (Scenario 2) επιλέξαμε ένα υποσύνολο των benchmarks του πίνακα 3.6 με κριτήριο αν επηρεάζονται αισθητά από την αλλαγή τοπολογίας. Καθένα από τα benchmarks που επιλέξαμε το εκτελέσαμε στις τοπολογίες remote και interleave με τη διαφορά ότι τώρα στον απομακρυσμένο κόμβο έτρεχε σε τοπολογία local ένα instance του stress-ng που καταναλώνε bandwidth από τη μνήμη. Για να γίνει πιο ξεκάθαρη η τοπολογία σε αυτό το σετ από πειράματα θεωρούμε χωρίς βλάβη της γενικότητας ότι οι κόμβοι που έλαβαν μέρος σε αυτά ήταν οι κόμβοι 0 και 1. Τότε, το στιγμιότυπο του εργαλείου stress-ng έτρεχε με cpu και μνήμη από τον κόμβο 1, ενώ κάθε benchmark έτρεχε με cpu από τον κόμβο 0 και χρησιμοποιούσε είτε μόνο τη μνήμη του κόμβου 1 για την τοπολογία remote είτε εναλλάξ τις μνήμες των κόμβων 0 και 1 για την τοπολογία interleave. Δεν χρειάστηκε να ξανατρέξουμε τα benchmarks για την τοπολογία local διότι θα λαμβάναμε τις ίδιες μετρήσεις με αυτές των προηγούμενων πειραμάτων. Αυτό συμβαίνει, διότι το stress-ng τρέχει αποκλειστικά στον απομακρυσμένο κόμβο οπότε δε θα επηρέαζε πρακτικά τη συμπεριφορά του benchmark όταν αυτό χρησιμοποιούσε την τοπική του μνήμη μόνο. Στον πίνακα Α.3 βλέπουμε τα αποτελέσματα που λάβαμε για αυτό το σετ πειραμάτων.

Για να γίνουν καλύτερα κατανοητά τα αποτελέσματα για το σετ πειραμάτων Scenario 2 όσον αφορά στην απόδοση των benchmark παραθέτουμε και τις αντίστοιχες γραφικές παραστάσεις 4.6 και 4.7.



Σχήμα 4.6: Αποτελέσματα Scenario 2 για τη σειρά 500s.



Σχήμα 4.7: Αποτελέσματα Scenario 2 για τη σειρά 600s.

Αποτελέσματα και παρατηρήσεις

Παρατηρώντας τα αποτελέσματα που λάβαμε και συγκρίνοντας τα με αυτά του πίνακα Α.2 βλέπουμε ότι δεν υπάρχουν σημαντικές διαφορές αναφορικά με την απόδοση των benchmarks. Κάποια από αυτά φαίνονται να επηρεάζονται λίγο από το stress-ng που τρέχει στον απομακρυσμένο κόμβο με αποτέλεσμα να επιβραδύνονται. Για παράδειγμα, το benchmark 657.xz_s_1 παρουσιάζει μεταβολή της απόδοσης από το Scenario 1 στο Scenario 2 για την τοπολογία remote ίση με

$$\triangle Performance(Scenario1 \rightarrow Scenario2|\, remote) = (\frac{Total\ Time_{Scenario\ 2}}{Total\ Time_{Scenario\ 1}} - 1) \cdot 100\% \approx 5\%$$

ενώ για την τοπολογία interleave ίση με

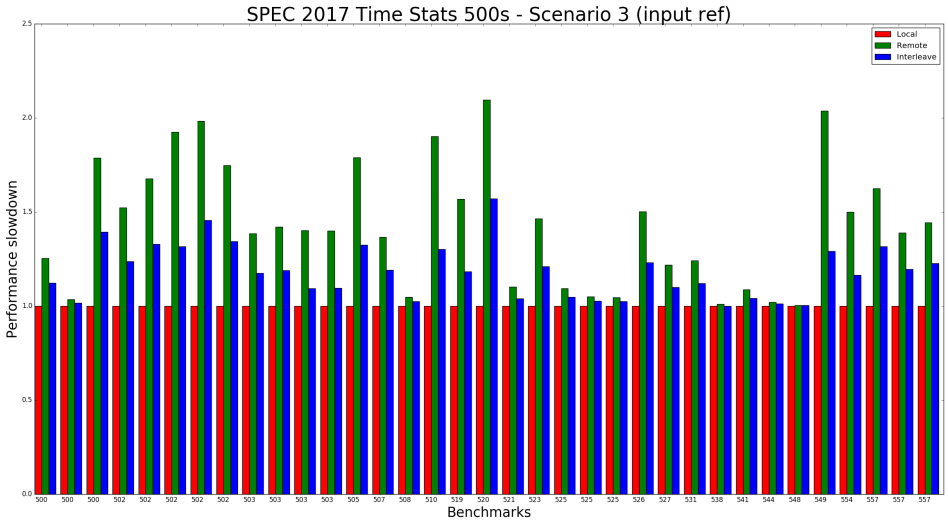
$$\triangle Performance(Scenario1 \rightarrow Scenario2|\, interleave) = (\frac{Total\ Time_{Scenario\ 2}}{Total\ Time_{Scenario\ 1}} - 1) \cdot 100\% \approx 4\%.$$

Άλλα benchmark, όπως το 500.perlbench_r_0, δεν παρουσιάζουν καμία διαφορά στην απόδοσή τους, ενώ σε κάποιες περιπτώσεις οι τιμές που παίρνουμε για το Scenario 2 είναι ελαφρώς μικρότερες από τις αντίστοιχες του Scenario 1 γεγονός που οφείλεται στο σφάλμα 2% στις μετρήσεις που λαμβάνουμε όπως έχει ήδη αναφερθεί.

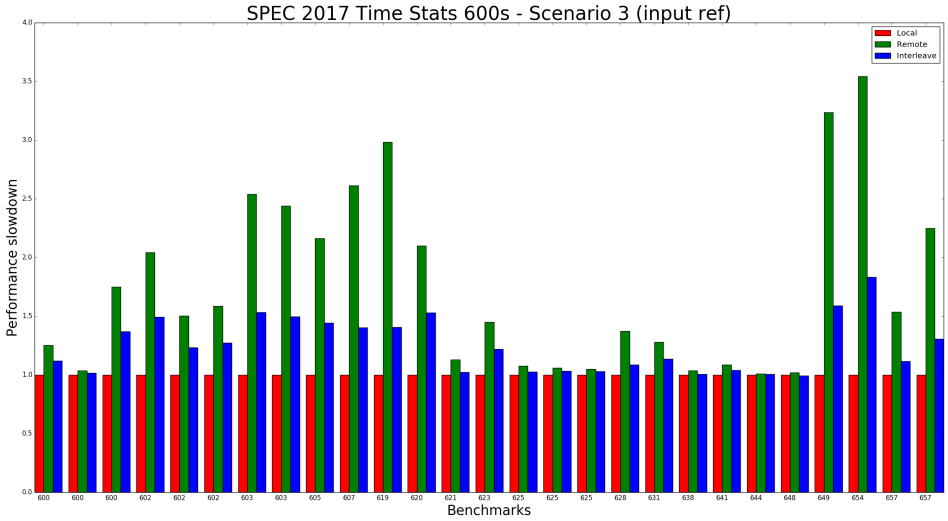
4.3.3 Scenario 3 - Co-execution with one instance of stress-ng on the local node

Για τη συνέχεια της έρευνας και την καλύτερη κατανόηση της συμπεριφοράς των benchmarks όταν εκτελούνται ταυτόχρονα με άλλες εφαρμογές στο σύστημα που καταναλώνουν πόρους - στην περίπτωσή μας το stress-ng - αποφασίσαμε να εκτελέσουμε ένα νέο σετ από πειράματα (Scenario 3). Σε αυτά τα πειράματα εκτελέσαμε όλα τα benchmarks που χρησιμοποιούμε στην έρευνα σε όλες τις τοπολογίες, δηλαδή σε local, remote και interleave, ενώ ταυτόχρονα τρέχει στον ίδιο κόμβο ένα στιγμιότυπο του εργαλείου stress-ng που καταναλώνει bandwidth από την τοπική μνήμη. Με αυτή την τοπολογία στο μηχάνημα Sandman μπορούμε να μελετήσουμε την επίδραση στην απόδοση των εφαρμογών όταν ο τοπικός κόμβος που τρέχουν χρησιμοποιείται και από άλλες διεργασίες. Εδώ έχει νόημα η εκτέλεση των εφαρμογών μας και στην τοπολογία local διότι αυτή δε θα ταυτίζεται με τα αποτελέσματα της baseline εκτέλεσης λόγω της ύπαρξης του stressor στον τοπικό κόμβο. Στους πίνακες Α.4 και Α.5 βλέπουμε τα αποτελέσματα που λάβαμε για αυτό το σετ πειραμάτων.

Πιο αναλυτικά, στον πίνακα Α.4 παρουσιάζουμε την απόδοση για κάθε benchmark στις τοπολογίες remote και interleave θεωρώντας ως βάση τα αποτελέσματα για την τοπολογία local του Scenario 3. Αντίθετα, στον πίνακα Α.5 παρουσιάζουμε την απόδοση για κάθε benchmark στις τοπολογίες local, remote, και interleave θεωρώντας ως βάση αναφοράς τα αποτελέσματα για την τοπολογία local του Scenario 1. Για την καλύτερη κατανόηση των αποτελεσμάτων και την εξαγωγή συμπερασμάτων κατασκευάσαμε και τις αντίστοιχες γραφικές παραστάσεις (4.8-4.11).



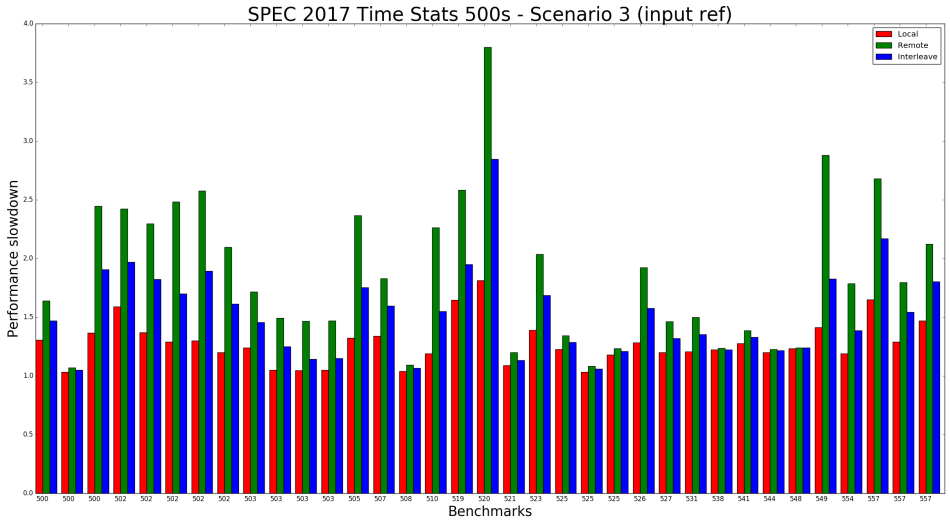
Σχήμα 4.8: Αποτελέσματα Scenario 3 για τη σειρά 500s.



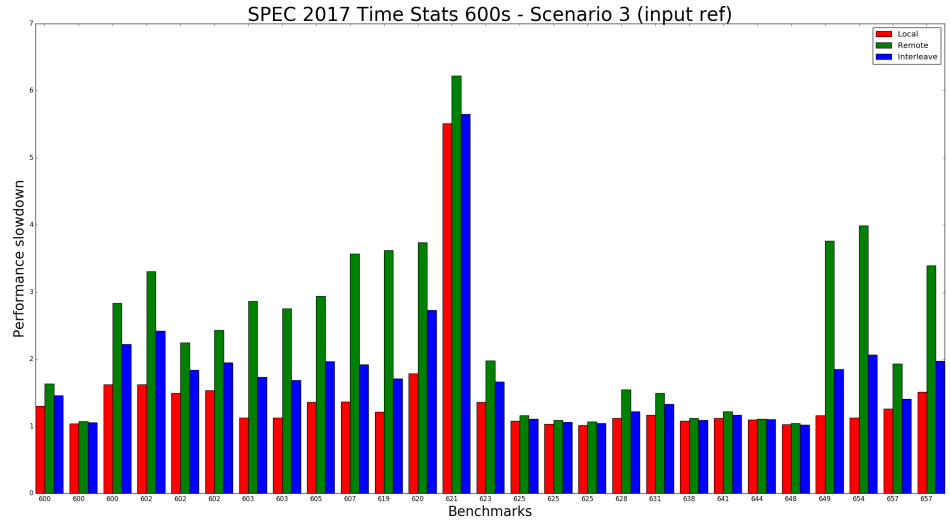
Σχήμα 4.9: Αποτελέσματα Scenario 3 για τη σειρά 600s.

Αποτελέσματα και παρατηρήσεις

Αρχικά, παρατηρώντας τις γραφικές παραστάσεις 4.10 και 4.11 είναι προφανές ότι υπάρχει αύξηση στο χρόνο εκτέλεσης των benchmarks στην τοπολογία local που απεικονίζεται με κόκκινο χρώμα σε σύγκριση με τα αποτελέσματα του Scenario 1. Αυτό είναι αναμενόμενο, καθώς στο συγκεκριμένο σετ από πειράματα όπως είπαμε παράλληλα με κάθε benchmark τρέχει ένα στιγμιότυπο του stress-ng



Σχήμα 4.10: Αποτελέσματα Scenario 3 (κανονικοποιημένα ως προς baseline) για τη σειρά 500s.



Σχήμα 4.11: Αποτελέσματα Scenario 3 (κανονικοποιημένα ως προς baseline) για τη σειρά 600s.

στον τοπικό κόμβο το οποίο καταναλώνει πόρους από αυτόν και κυρίως bandwidth. Έτσι, απομένει λιγότερο bandwidth για το benchmark που τρέχει με αποτέλεσμα να αυξάνεται η απόδοσή του όπως την ορίσαμε (επιβράδυνση). Είναι αξιοσημείωτο, ωστόσο ότι η μεταβολή της απόδοσης δεν είναι η ίδια για όλα τα benchmarks αλλά εμφανίζει μεγάλη διακύμανση. Χαρακτηριστικό παράδειγμα αποτελούν οι εφαρμογές 621.wrf και 648.exchange2. Η πρώτη εμφανίζει μεταβολή της απόδοσης από την τοπική τοπολογία του Scenario 1 στην αντίστοιχη του Scenario 3 ίση με

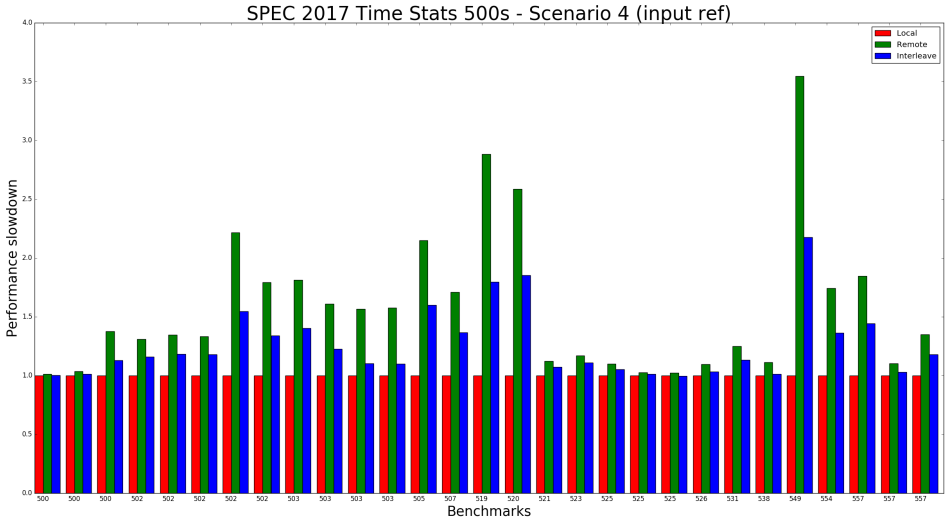
$$\triangle Performance(Scenario1 \rightarrow Scenario3|\, local) = (\frac{Total\ Time_{Scenario\ 3}}{Total\ Time_{Scenario\ 1}} - 1) \cdot 100\% \approx 450\%$$

ενώ η δεύτερη εμφανίζει μεταβολή

$$\triangle Performance(Scenario1 \rightarrow Scenario3|\, local) = (\frac{Total\ Time_{Scenario\ 3}}{Total\ Time_{Scenario\ 1}} - 1) \cdot 100\% \approx 103\%.$$

4.3.4 Scenario 4 - Co-execution with multiple instances of stress-ng on the remote node

Τα αποτελέσματα που λάβαμε από το Scenario 2 σχετικά με την μεταβολή της απόδοσης των benchmarks όταν στον απομακρυσμένο κόμβο τρέχει μία εφαρμογή που καταναλώνει πόρους κατέδειξαν μία τάση η οποία ωστόσο δεν ήταν ιδιαίτερα αισθητή για να βγουν ασφαλή συμπεράσματα. Για το λόγο αυτό αποφασίσαμε να εκτελέσουμε ένα νέο σετ από πειράματα (Scenario 4) αντίστοιχα με αυτά του Scenario 2 με τη διαφορά ότι η κατανάλωση πόρων στον απομαρκυσμένο κόμβο θα ήταν σημαντικά μεγαλύτερη ώστε να εξετάσουμε αν θα υπάρξουν μεγαλύτερες αποκλίσεις στη συμπεριφορά των εφαρμογών μας. Ουσιαστικά, δηλαδή, εκτελέσαμε ξανά τα ίδια benchmarks με αυτά του Scenario 2 στις τοπολογίες remote και interleave και παράλληλα σε κάθε εκτέλεση έτρεχαν 10 στιγμιότυπα του εργαλείου stress-ng στον απομακρυσμένο κόμβο που δημιουργούσαν 10πλάσια ζήτηση για bandwidth στη μνήμη. Οι μετρήσεις που λάβαμε από αυτά τα πειράματα φαίνονται στον πίνακα A.6 και οι γραφικές παραστάσεις για τις σειρές 500s και 600s στις εικόνες 4.12 και 4.13.



Σχήμα 4.12: Αποτελέσματα Scenario 4 για τη σειρά 500s.

Αποτελέσματα και παρατηρήσεις

Παρατηρώντας τις γραφικές παραστάσεις επιβεβαιώνουμε τις προβλέψεις μας ότι η ταυτόχρονη εκτέλεση των 10 στιγμιοτύπων του stress-ng στον απομακρυσμένο κόμβο επιβραδύνει τις εφαρμογές μας στις τοπολογίες remote και interleave. Για παράδειγμα, το benchmark 657.lbm_s που παρουσίαζε στο Scenario 2 μεταβολή της απόδοσης του $\triangle Performance(Scenario1 \rightarrow Scenario2|\, remote) \approx 3.5\%$ και $\triangle Performance(Scenario1 \rightarrow Scenario2|\, interleave) \approx 8.7\%$ στο Scenario 4 εμφάνισε μεταβολή της απόδοσής του

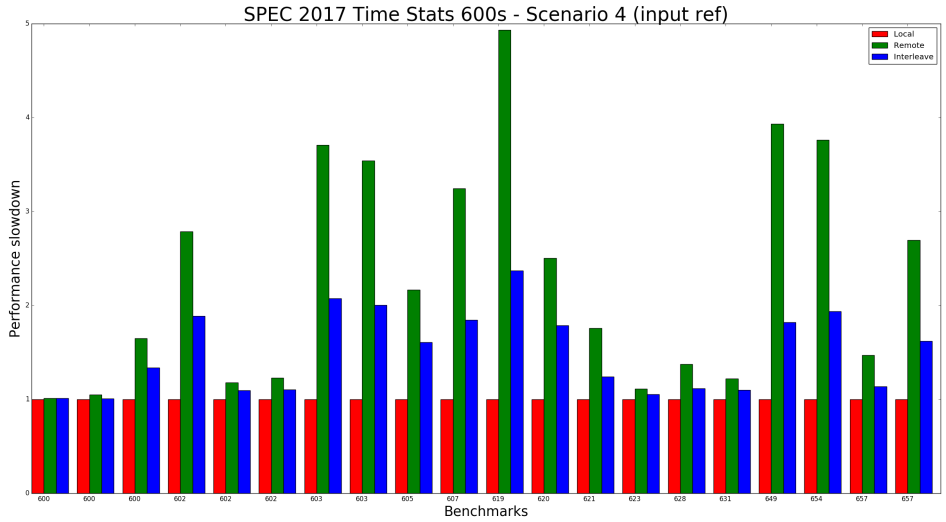
$$\triangle Performance(Scenario1 \rightarrow Scenario4|\, remote) = (\frac{Total\ Time_{Scenario\ 4}}{Total\ Time_{Scenario\ 1}} - 1) \cdot 100\% \approx 24\%$$

και

$$\triangle Performance(Scenario1 \rightarrow Scenario4|\, interleave) = (\frac{Total\ Time_{Scenario\ 4}}{Total\ Time_{Scenario\ 1}} - 1) \cdot 100\% \approx 37\%$$

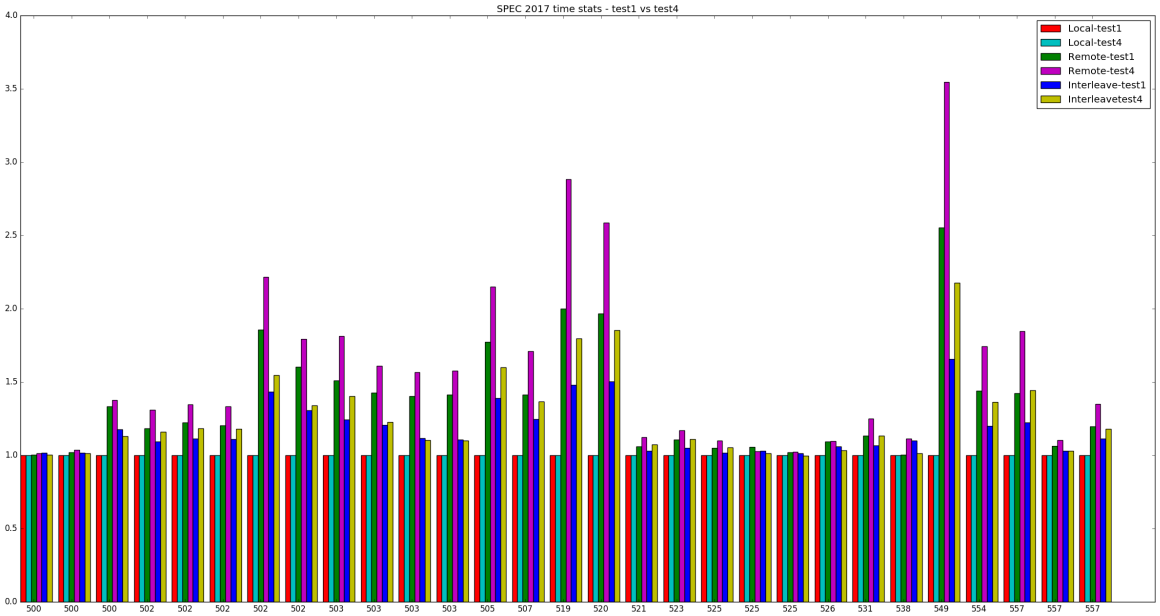
αντίστοιχα.

Για την καλύτερη σύγκριση των αποτελεσμάτων των Scenario 1 και Scenario 4 κατσκευάσαμε τις γραφικές παραστάσεις 4.14 και 4.15. Από αυτές διαπιστώνουμε εύκολα ότι για όλα τα benchmarks



Σχήμα 4.13: Αποτελέσματα Scenario 4 για τη σειρά 600s.

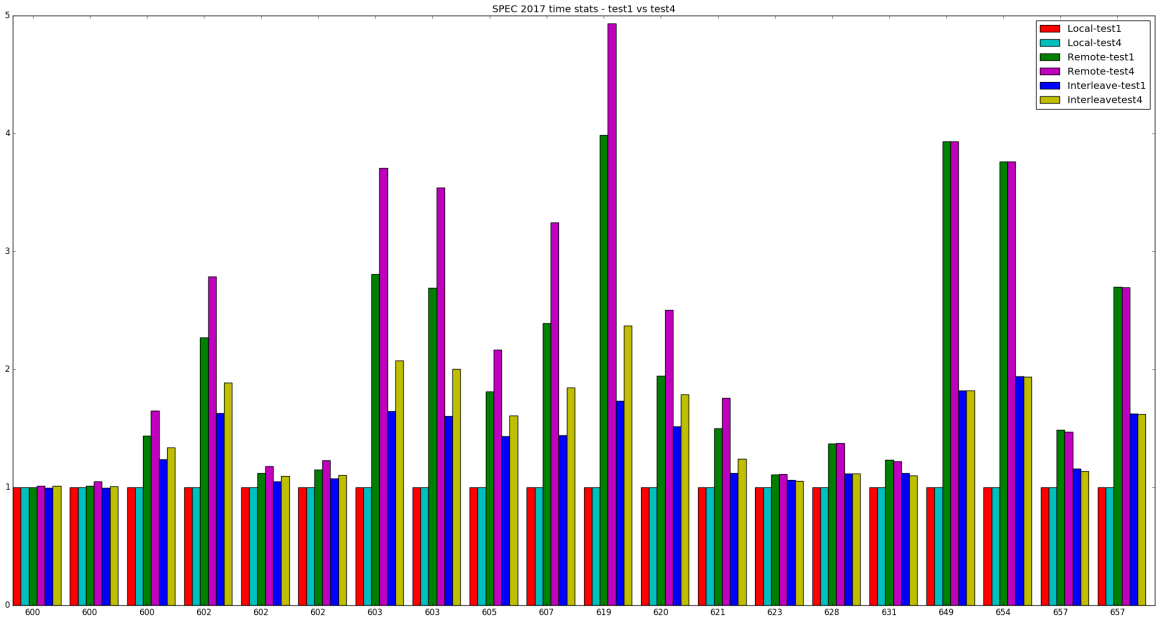
υπάρχει επιβράδυνση στις τοπολογίες remote και interleave, αφού οι μωβ μπάρες (remote Scenario 4) βρίσκονται πάντα υψηλότερα από τις πράσινες (remote Scenario 1) και οι κίτρινες (interleave Scenario 4) υψηλότερα από τις μπλε (interleave Scenario 1). Οι μεταβολές στην απόδοση δεν είναι ούτε εδώ ίδιες σε όλα τα benchmarks αλλά εμφανίζουν μεγάλη διακύμανση λόγω των διαφορετικών χαρακτηριστικών τους. Τέλος, η τοπολογία local εμφανίζει εφάμιλλη επίδοση και στα δύο σετ από πειράματα, κάτι που είναι αναμενόμενο αφού τα στιγμιότυπα του stress-ng τρέχουν αποκλειστικά στον απομακρυσμένο κόμβο οπότε δεν επηρεάζουν με κανένα τρόπο τον τοπικό.



Σχήμα 4.14: Σύγκριση των αποτελεσμάτων των Scenario 1 και Scenario 4 για τη σειρά 500s.

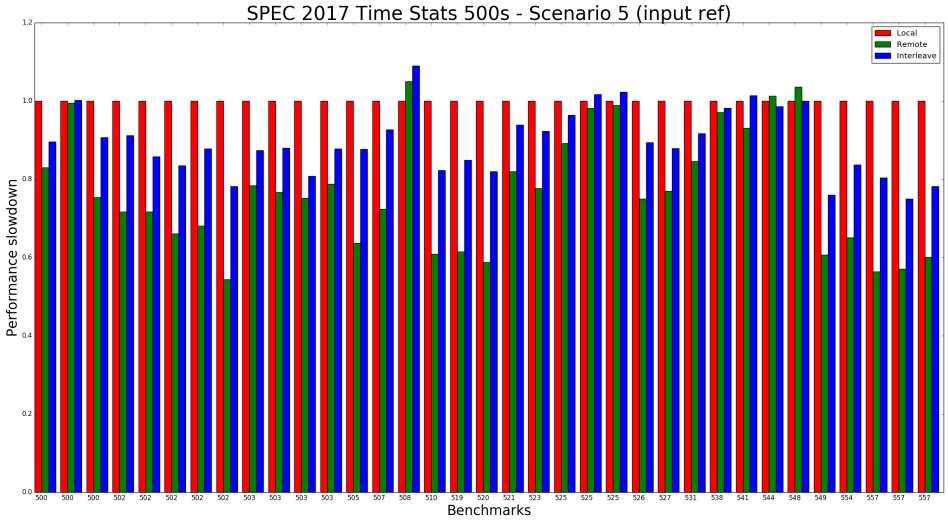
4.3.5 Scenario 5 - Co-execution with multiple instances of stress-ng on the local node

Κατ' αναλογία με το ζεύγος Scenario 2 - Scenario 4, επιθυμήσαμε να δούμε τη μεταβολή της απόδοσης των benchmarks όταν ο τοπικός κόμβος που εκτελούνται επιβαρύνεται από την εκτέλεση

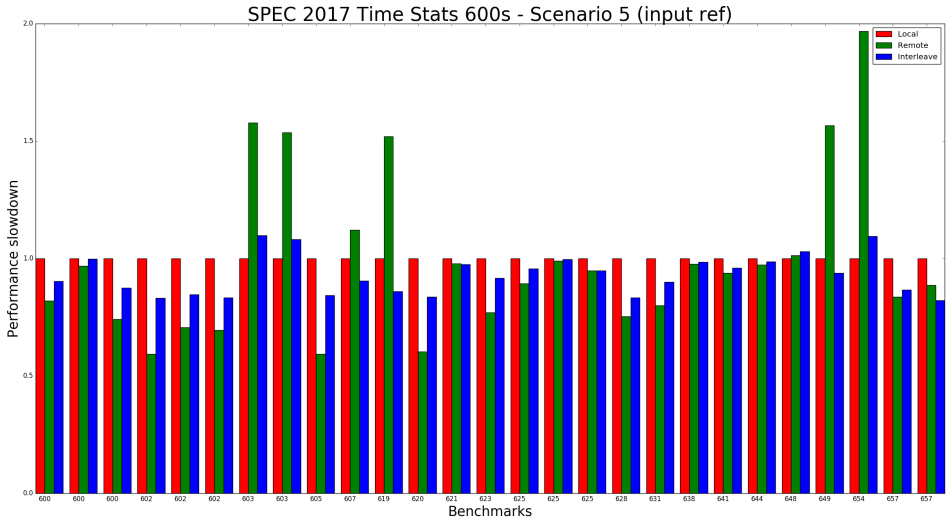


Σχήμα 4.15: Σύγκριση των αποτελεσμάτων των Scenario 1 και Scenario 4 για τη σειρά 600s.

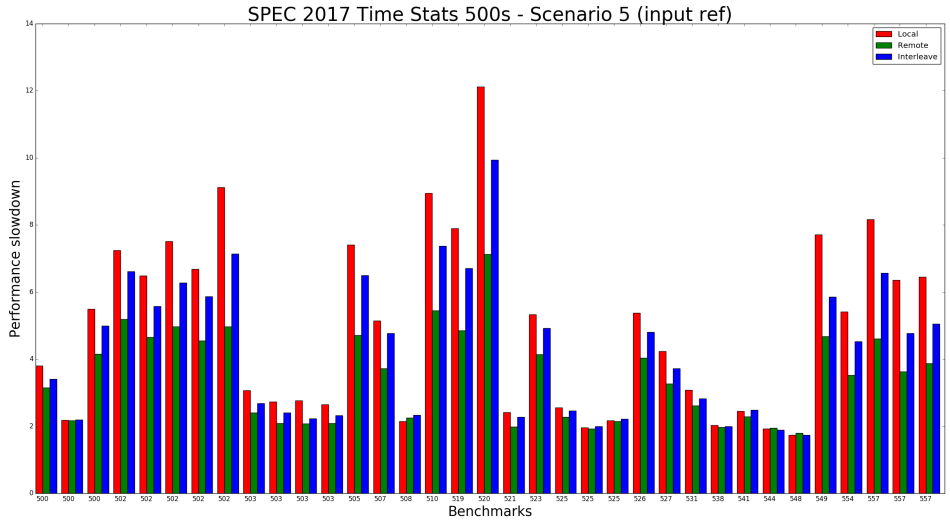
πολλαπλών εφαρμογών που καταναλώνουν πόρους. Έτσι, αποφασίσαμε να εκτελέσουμε ένα νεό σετ από πειράματα (Scenario 5) με παρόμοια τοπολογία με το Scenario 3, δηλαδή τα benchmarks να τρέχουν στις τοπολογίες local, remote και interleave όπως τις έχουμε ορίσει, με τη διαφορά ότι τώρα στον τοπικό κόμβο τρέχουν παράλληλα με κάθε εφαρμογή 10 στιγμιότυπα του εργαλείου stress-ng. Με αυτό τον τρόπο η ζήτηση για bandwidth στον τοπικό κόμβο από εξωτερικές εφαρμογές γίνεται δεκαπλάσια σε σχέση με αυτή που είχαμε στο Scenario 3 με αποτέλεσμα να αναμένουμε μια πιο έντονη μεταβολή στην απόδοση των benchmarks. Οι μετρήσεις που λάβαμε από αυτά τα πειράματα φαίνονται στους πίνακες A.7 και A.8. Για την καλύτερη οπτικοποίησή τους και την εξαγωγή χρήσιμων συμπερασμάτων από αυτά κατασκευάσαμε και τις αντίστοιχες γραφικές παραστάσεις (4.16 - 4.19).



Σχήμα 4.16: Αποτελέσματα Scenario 5 για τη σειρά 500s.



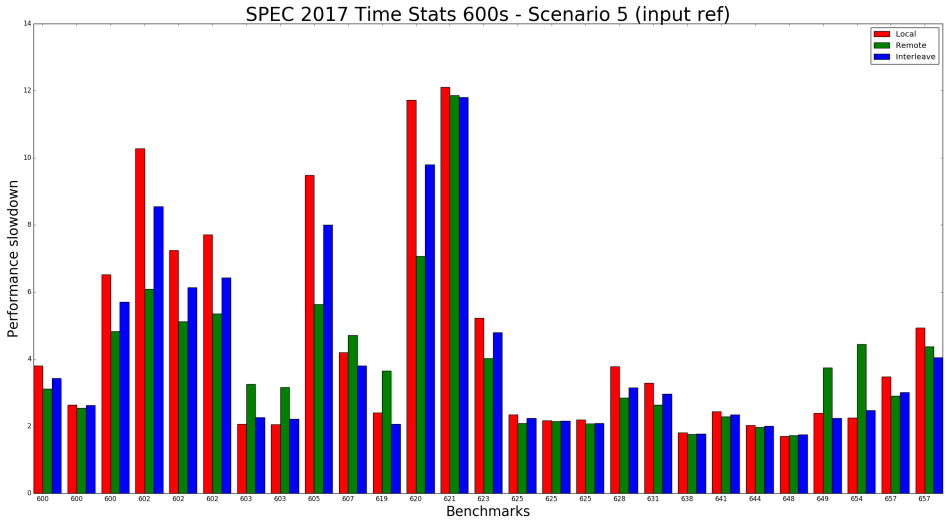
Σχήμα 4.17: Αποτελέσματα Scenario 5 για τη σειρά 600s.



Σχήμα 4.18: Αποτελέσματα Scenario 5 (κανονικοποιημένα ως προς baseline) για τη σειρά 500s.

Αποτελέσματα και παρατηρήσεις

Παρατηρώντας τις γραφικές παραστάσεις καταλήγουμε σε ένα πολύ ενδιαφέρον συμπέρασμα. Αυτό είναι ότι η τοπολογία local εμφανίζει τη μεγαλύτερη επιβράδυνση σε σχέση με τις άλλες στη συντριπτική πλειοψηφία των benchmarks. Με άλλα λόγια, βλέπουμε ότι στην περίπτωση που ο τοπικός κόμβος επιβαρύνεται λόγω της ταυτόχρονης εκτέλεσης άλλων εφαρμογών, η local τοπολογία καταλήγει να γίνεται η πιο αργή σε σύγκριση με τις remote και interleave. Αυτό το συμπέρασμα, σε πρώτη ανάγνωση φαίνεται αρκετά αντιφατικό αφού θεωρούμε ότι η τοπική μνήμη έχει πάντοτε τη μικρότερη απόσταση από τον επεξεργαστή οπότε και το μικρότερο χρόνο προσπέλασης, με αποτέλεσμα όλες οι μεθοδολογίες που έχουν αναπτυχθεί μέχρι τώρα σχετικά με το πρόβλημα του NUMA placement να στοχεύουν στην εκτέλεση των εφαρμογών με χρήση της τοπικής τους μνήμης. Ωστόσο, μπορεί να εξηγηθεί αν αναλογιστούμε ότι η ζήτηση για bandwidth που δημιουργούν οι εξωτερικές εφαρμογές στην τοπική μνήμη δημιουργεί πολύ μεγαλύτερες καθυστερήσεις όσον αφορά στην προσπέλασή της σε σχέση με τον επιπλέον χρόνο που χρειάζεται η εφαρμογή για να προσπελάσει μια



Σχήμα 4.19: Αποτελέσματα Scenario 5 (κανονικοποιημένα ως προς baseline) για τη σειρά 600s.

απομακρυσμένη μνήμη που διαθέτει όμως πολύ περισσότερο ελεύθερο bandwidth.

Ας δούμε κάποια ποσοτικά παραδείγματα για να κατανοήσουμε τη σημασία της παραπάνω παρατήρησης. Ας πάρουμε το benchmark 520.omnetpp_r το οποίο κατά το Scenario 3 εμφάνισε μεταβολή στην απόδοσή του για την τοπολογία 0 $\triangle Performance(Scenario1 \rightarrow Scenario3|\, local) \approx 81\%$. Η αντίστοιχη τιμή για το Scenario 5 είναι

$$\triangle Performance(Scenario1 \rightarrow Scenario5|\, local) = (\frac{Total\ Time_{Scenario\ 5}}{Total\ Time_{Scenario\ 1}} - 1) \cdot 100\% \approx 1112\%$$

Με άλλα λόγια το συγκεκριμένο benchmark επιβραδύνθηκε περισσότερες από 11 φορές στο Scenario 5 σε σχέση με το Scenario 1 για εκτέλεση στην τοπολογία 0. Αυτό είναι αρκετά εντυπωσιακό αλλά το κλειδί της παραπάνω παρατήρησης βρίσκεται στη σχέση που έχει η απόδοση της local τοπολογίας σε σχέση με τις remote και interleave. Έτσι, για το παράδειγμα που μελετάμε, ενώ στο Scenario 1 ίσχυε ότι $Performance(remote) \approx 197\%$ και $Performance(interleave) \approx 150\%$, στο Scenario 5 έχουμε ότι

$$Performance(remote) = \frac{Total\ Time_{remote}}{Total\ Time_{local}} \cdot 100\% \approx 59\%$$

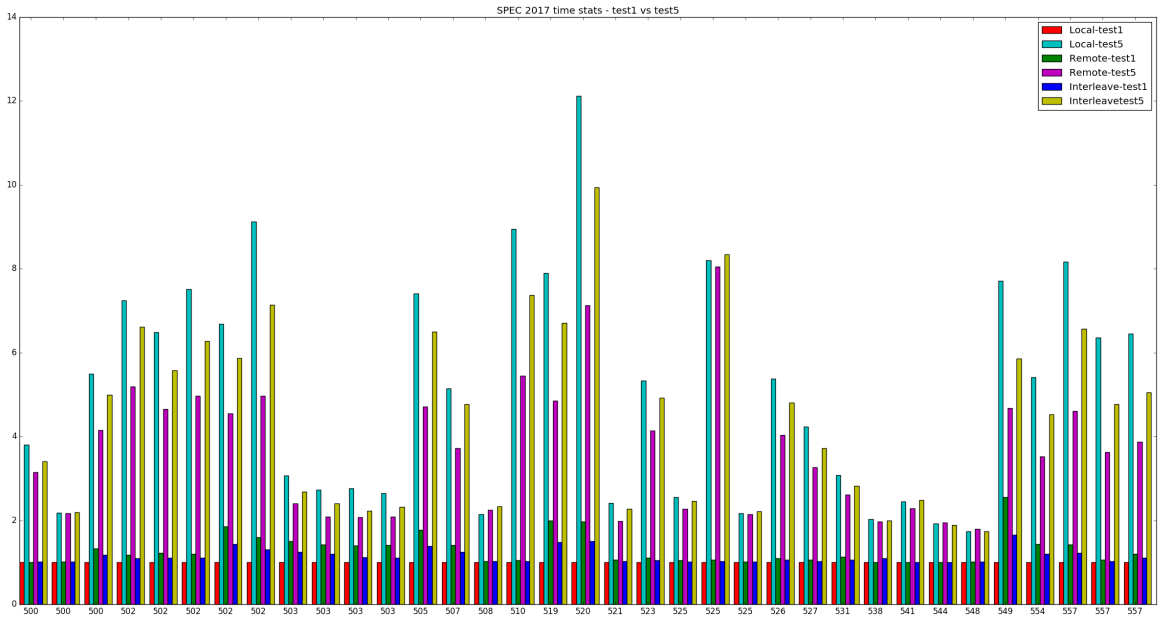
και

$$Performance(interleave) = \frac{Total\ Time_{interleave}}{Total\ Time_{local}} \cdot 100\% \approx 82\%$$

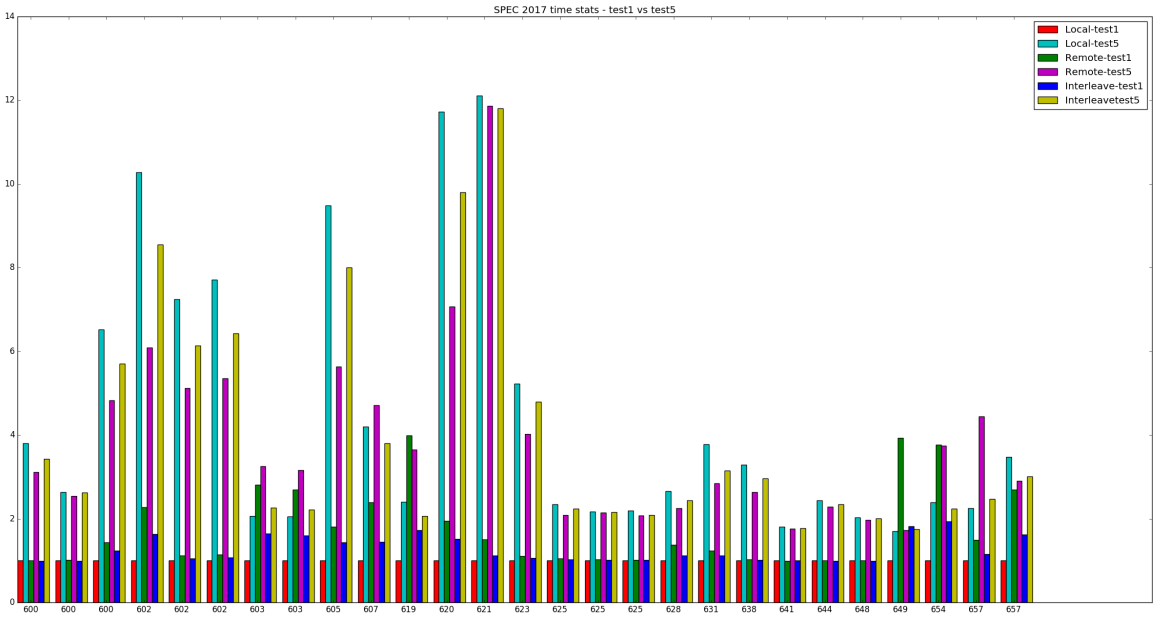
. Αυτό σημαίνει ότι ενώ αρχικά η local τοπολογία ήταν η ταχύτερη, αφού ο χρόνος εκτέλεσης του benchmark στη remote τοπολογία ήταν πρακτικά διπλάσιας και στην interleave 1.5 φορές μεγαλύτερος, όταν ο τοπικός κόμβος επιβαρύνθηκε με την εκτέλεση και άλλων εφαρμογών η local τοπολογία έγινε η πιο αργή, με τη remote τοπολογία να απαιτεί σχεδόν το μισό χρόνο εκτέλεσης και την interleave σχεδόν το 80% του χρόνου της. Για την καλύτερη σύγκριση των αποδόσεων των benchmarks μεταξύ των δύο σετ πειραμάτων, Scenario 1 και Scenario 5, κατασκευάσαμε τις γραφικές παραστάσεις 4.20 και 4.21.

Stress effect

Το φαινόμενο που παρατηρήσαμε και σχολιάσαμε παραπάνω το οποίο αφορά στην αλλαγή της συμπεριφοράς των benchmarks όταν στον τοπικό κόμβο που εκτελούνται τρέχουν παράλληλα άλλες εφαρμογές που καταναλώνουν σημαντικό ποσοστό του διαθέσιμου bandwidth το ονομάσαμε stress



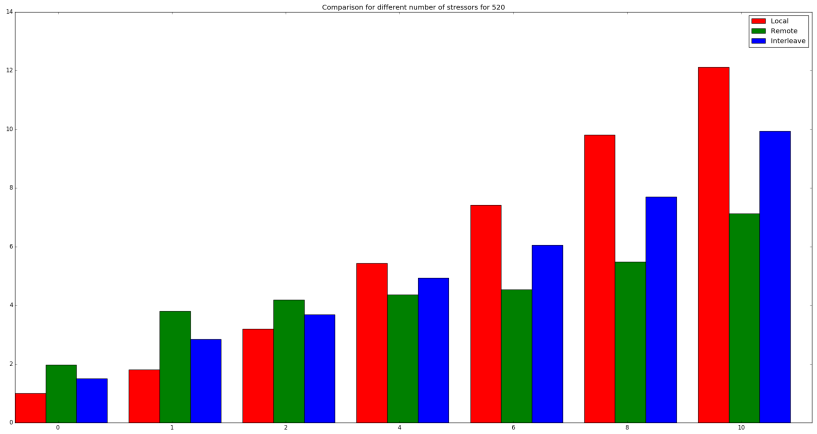
Σχήμα 4.20: Σύγκριση των αποτελεσμάτων των Scenario 1 και Scenario 5 για τη σειρά 500s.



Σχήμα 4.21: Σύγκριση των αποτελεσμάτων των Scenario 1 και Scenario 5 για τη σειρά 600s.

effect. Πρόκειται ουσιαστικά για τη μεταβολή των αποδόσεων των τοπολογιών remote και interleave ως προς την τοπολογία local κάθε φορά με τέτοιο τρόπο που αυτή να καθίσταται η πιο αργή παρά το πλεονέκτημα που διαθέτει λόγω της κοντινότερης απόστασης από τη μνήμη.

Αποφασίσαμε να μελετήσουμε λίγο παραπάνω αυτό το φαινόμενο για να διαπιστώσουμε πως μεταβάλλεται σταδιακά η απόδοση καθώς ο τοπικός κόμβος που τρέχουν οι εφαρμογές επιβαρύνεται παραπάνω. Για το σκοπό αυτό εκτελέσαμε κάποια επιπρόσθετα πειράματα με τοπολογία αντίστοιχη του Scenario 5. Επιλέξαμε το benchmark 520.omnetpp_r για τα πειράματα καθώς όπως είδαμε επηρεάζεται περισσότερο από το stress effect σε σχέση με τα άλλα. Σε κάθε πείραμα αυτού του σετ εκτελέσαμε το συγκεκριμένο benchmark στις τοπολογίες local, remote και interleave, ενώ ταυτόχρονα έτρεχαν στον τοπικό κόμβο N στιγμιότυπα του εργαλείου stress-ng όπου N=0, 1, 2, 4, 6, 8, 10. Η περίπτωση N=0 δηλαδή όταν δεν τρέχει κανένα στιγμιότυπο του stress-ng ταυτίζεται με το Scenario



Σχήμα 4.22: Αποτελέσματα για το stress effect και το benchmark 520.omnetpp_r.

1. Ακόμη, όταν N=1 έχουμε τα αποτελέσματα του Scenario 3 και όταν N=10 τα αποτελέσματα του Scenario 5. Οι μετρήσεις που λάβαμε απεικονίζονται στη γραφική παράσταση 4.22.

Παρατηρούμε ότι η κρίσιμη τιμή για τον αριθμο των στιγμιοτύπων του stress-ng που αλλάζουν οι συσχετίσεις μεταξύ των αποδόσεων για τις τοπολογίες local, remote και interleave είναι N=4. Έτσι, από την τιμή αυτή και πάνω βλέπουμε ότι η local τοπολογία (κόκκινο χρώμα) είναι η πιο αργή, η interleave τοπολογία (μπλε χρώμα) η επόμενη πιο αργή και η remote τοπολογία (πράσινο χρώμα) η πιο γρήγορη. Δηλαδή, η ύπαρξη των στιγμιοτύπων του stress-ng στον τοπικό κόμβο αντιστρέφει εντελώς την κατάταξη των τοπολογιών ως προς την ταχύτητα εκτέλεσης του benchmark.

4.4 Συμπεράσματα

Ολοκληρώνοντας τα πειράματα που αφορούν τις εκτελέσεις των εφαρμογών μας στο σύστημα, στην ενότητα αυτή θα συνοψίσουμε τις παρατηρήσεις μας και τα συμπεράσματα που εξάγαμε και μας οδήγησαν προς την κατασκευή του μοντέλου του resource manager.

- Αρχικά, από το Scenario 1 έγινε ξεκάθαρο σε τι βαθμό επηρεάζονται οι εφαρμογές μας επηρεάζονται από την αλλαγή της τοπολογίας. Έτσι, είδαμε ότι κάποιες εφαρμογές δεν επηρεάζονται σχεδόν καθόλου όταν τρέχουν απομακρυσμένα, ενώ άλλες είναι περισσότερο ευαίσθητες και επιβραδύνονται έως και 3 φορές. Αυτό σημαίνει ότι το μοντέλο μας πρέπει να κάνει ”σωστές” επιλογές αναφορικά με τη μνήμη των εφαρμογών ώστε να διασφαλίζει συνολικά όσο το δυνατόν καλύτερη απόδοση. Με άλλα λόγια πρέπει να φροντίζει όσες εφαρμογές επηρεάζονται περισσότερο από την τοπολογία να τρέχουν τοπικά, ενώ όσες επηρεάζονται λιγότερο μπορούν να τρέχουν και απομακρυσμένα χωρίς να μειώνεται σημαντικά η ταχύτητά τους.

- Στη συνέχεια συσχετίσαμε αυτές τις συμπεριφορές με τα χαρακτηριστικά των εφαρμογών όπως αυτά ποσοτικοποιήθηκαν με χρήση των hardware performance counters IPC, MPKI, TLB MPKI και BW. Τα διαγράμματα 4.4 και 4.5 ήταν πολύ βοηθητικά στην ανακάλυψη αυτών των συσχετίσεων. Έτσι διαπιστώσαμε ότι η εξάρτηση από την τοπολογία μεγαλώνει καθώς αυξάνεται η χρήση του εύρους ζώνης της μνήμης της εφαρμογής, κάτι που αποτελεί το θεμέλιο λίθο του μοντέλου μας.

- Ακόμη, μελετήσαμε κάποα σενάρια συνεκτέλεσης των εφαρμογών μας με το stress-ng ώστε να διαπιστώσουμε πως αλλάζει η συμπεριφορά τους όταν αυξάνεται η χρήση του εύρου ζώνης της μνήμης του κόμβου που εκτελούνται. Συγκεκριμένα, παρατηρήσαμε ότι όλες οι εφαρμογές μας επιβραδύνθηκαν λόγω της συνεκτέλεσης αλλά σε διαφορετικό βαθμό, με τις ευαίσθητες στις μη τοπικές τοπολογίες να σημείωνουν σημειώνουν τις μεγαλύτερες μεταβολές στο χρόνο εκτέλεσης. Αυτό η διαπίστωση καταδεικνύει ότι στα πραγματικά NUMA συστήματα το πρόβλημα

της τοποθέτησης των εφαρμογών (NUMA placement) είναι ακόμα πιο έντονο και η χρήση ενός μοντέλου για τη διασφάλιση της απόδοσης καθίσταται αναγκαία.

- Τέλος, σε ορισμένα σενάρια της συνεκτέλεσης διαπιστώσαμε ένα αρκετά ενδιαφέρον φαινόμενο, το stress effect όπως το ονομάσαμε. Πρόκειται για την κατάσταση όπου η τοπική τοπολογία γίνεται η χειρότερη από θέμα απόδοσης σε σχέση με τις μη τοπικές. Αυτό ανατρέπει πολλές προσεγγίσεις που έχουν προταθεί μέχρι τώρα και στοχεύουν στην τοποθέτηση των εφαρμογών με τέτοιο τρόπο ώστε να τρέχουν τοπικά εφόσον είναι εφικτό. Ο τρόπος που αναπτύξαμε τον resource manager δε λαμβάνει υπόψιν άμεσα αυτό το φαινόμενο, αλλά το μοντέλο μας μπορεί εύκολα να επεκταθεί και να το συμπεριλάβει.

Κεφάλαιο 5

Υλοποίηση του Resource Manager

Στο κεφάλαιο αυτό θα περιγράψουμε αναλυτικά τον τρόπο υλοποίησης του resource manager που κατασκευάσαμε στα πλαίσια της παρούσας έρευνας, τα δομικά στοιχεία από τα οποία αποτελείται, τη λογική που εμπεριέχεται στο μοντέλο του και τις λειτουργίες που επιτελεί.

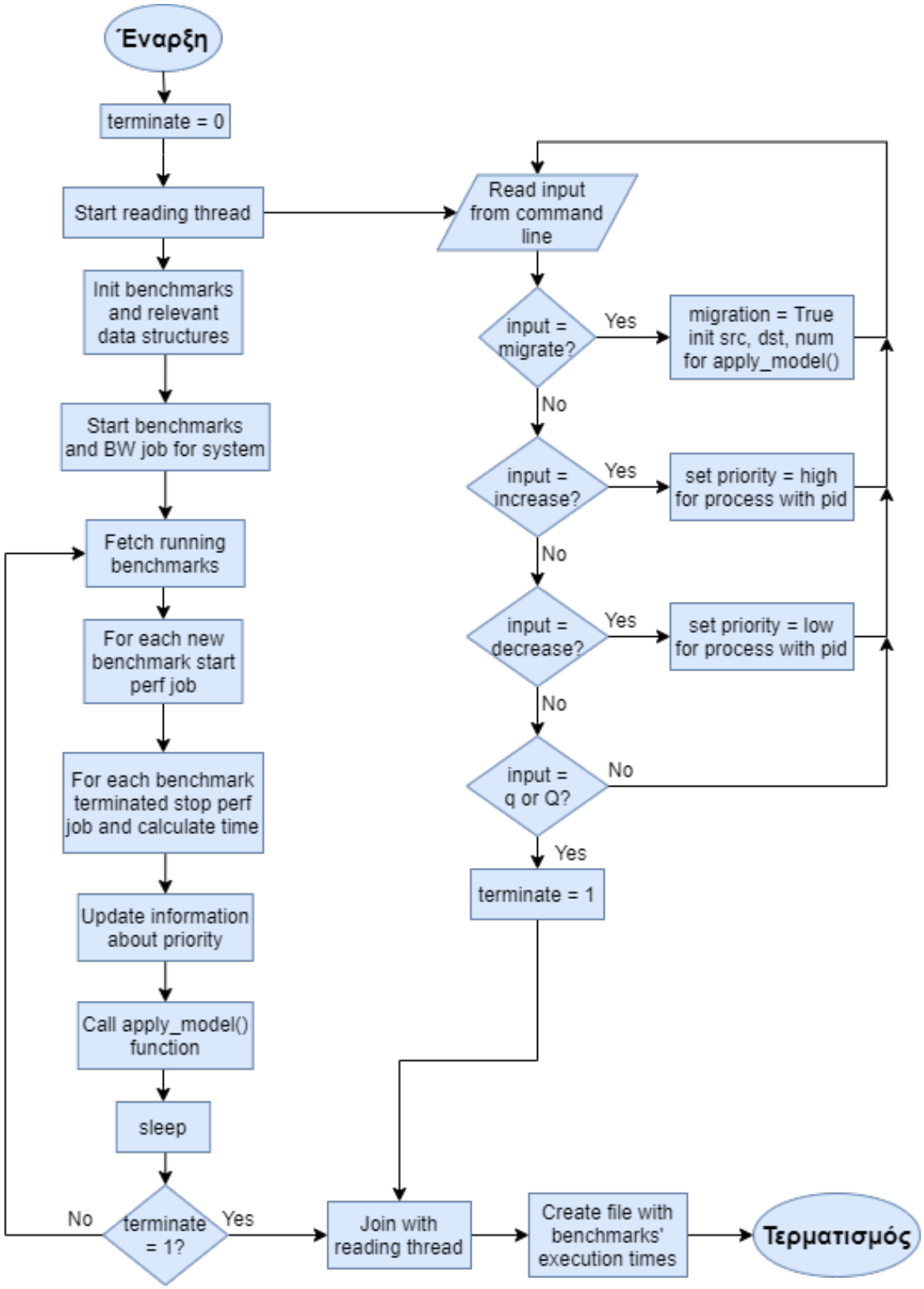
5.1 Εισαγωγή

Ο resource manager που κατασκευάσαμε ως βασικό αποτέλεσμα της παρούσας έρευνας για τη διαχείριση της μνήμης του NUMA συστήματος Sandman είναι ένα user-level πρόγραμμα γραμμένο στη γλώσσα προγραμματισμού Python. Ο κώδικας του resource manager είναι διαθέσιμος στην ιστοσελίδα: `https://github.com/theovaka/Resource_Manager.git`. Αυτό σημαίνει, ότι δεν χρειάζεται κάποια τροποποίηση του πυρήνα του λειτουργικού συστήματος για τη λειτουργία του ούτε δικαιώματα διαχειριστή (root), αλλά αντίθετα χρησιμοποιεί έτοιμα εργαλεία που αναλαμβάνουν να συλλέξουν τα απαραίτητα δεδομένα από το σύστημα και να εκτελέσουν ενέργειες αν απαιτείται (βλ. υποκεφάλαιο 3.2). Το συγκεκριμένο χαρακτηριστικό αποτελεί σημαντικό πλεονέκτημα της λύσης που προτείνουμε καθώς προσδίδει στον resource manager μεταφερσιμότητα. Με άλλα λόγια, μπορεί εύκολα να χρησιμοποιηθεί χωρίς σημαντικές τροποποιήσεις σε οποιοδήποτε NUMA σύστημα και να επιτελέσει το σκοπό της αύξησης της απόδοσης των εφαρμογών που τρέχουν σε αυτό.

5.2 Δομή του resource manager

Ας αρχίσουμε με την περιγραφή των βασικών δομικών στοιχείων από τα οποία αποτελείται το πρόγραμμα αυτό. Ο resource manager αποτελείται από δύο κύρια νήματα (threads) τα οποία τρέχουν παράλληλα καθόλη τη διάρκεια της εκτέλεσής του. Το πρώτο αναλαμβάνει να τρέχει έναν ατέρμονα βρόχο για τη διάδραση του προγράμματος με το χρήστη και την εκτέλεση συγκεκριμένων εντολών. Το δεύτερο που είναι και το πιο σημαντικό αναλαμβάνει να τρέχει έναν ατέρμονα βρόχο κατά τη διάρκεια του οποίου συλλέγει όλες τις απαραίτητες πληροφορίες από το σύστημα, όπως για παράδειγμα ποιες εφάρμογες εκτελούνται και τις τιμές συγκεκριμένων performance counters για αυτά, και αναλαμβάνει να εκτελέσει ενέργειες όταν απαιτείται σύμφωνα με το μοντέλο που έχουμε αναπτύξει. Τα δύο νήματα αλληλεπιδρούν με την έννοια ότι η λειτουργία του resource manager τερματίζεται όταν δοθεί αντίστοιχη εντολή από το χρήστη (με το πάτημα των πλήκτρων q ή Q), η οποία προκαλεί τη διακοπή της εκτέλεσης του πρώτου βρόχου και αυτή με τη σειρά της διακόπτει την εκτέλεση και του δεύτερου βρόχου. Ακόμη, τα δύο νήματα ανταλλάσσουν πληροφορίες μέσω κάποιων καθολικών μεταβλητών που αφορούν στις εφαρμογές που εκτελούνται και τις ενέργειες που εισάγει εξωτερικά ο χρήστης. Στην εικόνα 5.1 φαίνεται το διάγραμμα ροής του resource manager που περιλαμβάνει όλα τα βασικά δομικά στοιχεία από τα οποία αποτελείται.

Πιο αναλυτικά, το νήμα εκτέλεσης που αναλαμβάνει την διάδραση με το χρήστη υλοποιήθηκε με χρήση της κλάσης Cmd που παρέχεται από τη βιβλιοθήκη της Python και δημιουργεί ένα απλό framework για την υποστήριξη ενός command line διερμηνέα. Οι εντολές που υποστηρίζει ο resource manager είναι οι παρακάτω:



Σχήμα 5.1: Διάγραμμα ροής με τα βασικά δομικά στοιχεία του resource manager.

- **migrate**: Η εντολή αυτή συντάσσεται ως εξής:

  migrate <source_node> <destination_node> <num_pages>

  και η λειτουργία της είναι να μεταφέρει <*num_pages*> σελίδες μνήμης από τον κόμβο <*source_node*>

στον κόμβο <*destination_node*>. Αν κάποια προηγούμενη λειτουργία μεταφοράς σελίδων μνήμης είναι σε αναμονή ή κάποιος από τους δοθέντες κόμβους δεν είναι έγκυρος η εντολή αποτυγχάνει. Κατά τη διάρκεια της μεταφοράς των σελίδων, όπως θα δούμε και στη συνέχεια, αν κάποια στιγμή δεν υπάρχουν άλλες διαθέσιμες σελίδες για μεταφορά η διαδικασία ολοκληρώνεται ενημερώνοντας τον χρήστη μέσω κατάλληλου μηνύματος (”Can not migrate any more pages”). Επειδή, οι σελίδες μεταφέρονται ανά εφαρμογή, ο αριθμός <*num_pages*> αποτελεί κάτω όριο των σελίδων που θα μεταφερθούν καθώς η διαδικάσια της μεταφοράς ολοκληρώνεται μόλις επιτευχθεί ή (συχνότερα) ξεπεραστεί η συγκεκριμένη τιμή.

- **increase**: Η εντολή αυτή συντάσσεται ως εξής:

  increase <pid>

  και η λειτουργία της είναι να αυξήσει την προτεραιότητα της εφαρμογής με PID *pid* από χαμηλή (low) σε υψηλή (high). Η λειτουργία των δύο επιπέδων προτεραιότητας των εφαρμογών έχει ως στόχο να διατηρεί αναλλοίωτη τη συμπεριφορά κάποιων ”σημαντικών” εξ αυτών με βάση εξωτερικά κριτήρια και θα γίνει κατανοητή στη συνέχεια όταν θα περιγράψουμε το ρόλο της στο μοντέλο του resource manager. Σε περίπτωση που το pid που δίνεται από το χρήστη δεν αντιστοιχεί σε εφαρμογή που τρέχει στο σύστημα, η εντολή αποτυγχάνει ενημερώνοντας το χρήστη μέσω κατάλληλου μηνύματος (”Invalid PID”).

- **decrease**: Η εντολή αυτή είναι αντίστοιχη με την εντολή increase που περιγράψαμε παραπάνω. Συντάσσεται ως εξής:

  decrease <pid>

  και η λειτουργία της είναι να μειώση την προτεραιότητα της εφαρμογής με PID *pid* από υψηλή (high) σε χαμηλή (low).

- **q ή Q**: Οι εντολές αυτές όπως αναφέρθηκε και προηγουμένως τερματίζουν τη λειτουργία του resource manager.

Το δεύτερο και πιο σημαντικό νήμα εκτέλεσης του resource manager αναλαμβάνει να εκτελέσει όλες τις απαραίτητες λειτουργίες προκειμένου να τρέχει ορθά και να επιτελεί το έργο για το οποίο σχεδιάστηκε, το οποίο σχετίζεται με την αύξηση της απόδοσης κάθε εφαρμογής και του συστήματος συνολικά. Αποτελείται ουσιαστικά από έναν κύριο βρόχο μέσα στον οποίο ελέγχει ποιες εφαρμογές εκτελούνται στο σύστημα, εκτελεί τις απαραίτητες ενέργειες κάθε φορά που μια νέα εφαρμογή ξεκινάει ή ολοκληρώνει την εκτέλεσή της και καλεί τη συνάρτηση *apply_model* (βλ. υποκεφάλαιο 5.3) για την εφαρμογή του μοντέλου που σχεδιάσαμε.

Κατά την έναρξη της εκτέλεσης του νήματος δημιουργούμε έναν πίνακα με τα ονόματα όλων των benchmarks που λαμβάνουμε υπόψιν στην έρευνά μας. Αρχικοποιούμε ένα αρχείο log που θα καταγράφει όλες τις απαραίτητες πληροφορίες που συλλέγει ο resource manager και τον τρόπο που λαμβάνει αποφάσεις καθώς και τις απαραίτητες δομές δεδομένων που θα χρησιμοποιήσουμε για να χειριζόμαστε τις πληροφορίες αυτές. Στην περίπτωση που θέλουμε να εκκινήσουμε αυτόματα κάποια benchmarks χρησιμοποιούμε ένα template κώδικα bash που έχουμε κατασκευάσει το οποίο παραμετροποιούμε με τις εφαρμογές που θέλουμε να τρέξουν και τα την τοπολογία καθεμιάς. Ακόμη, εκκινούμε στο background ένα job με χρήση της βιβλιοθήκης subprocess της python το οποίο αναλαμβάνει να τρέχει το εργαλείο pcm-memory.x στο σύστημα μας και να καταγράφει τις μετρήσεις σε ένα αρχείο ανά τακτά χρονικά διαστήματα.

Στον κύριο βρόχο αυτού του νήματος, που τρέχει μέχρι ο χρήστης να δηλώσει ότι επιθυμεί να τερματιστεί η λειτουργία του resource manager μέσω της γραμμής εντολών, αρχικά δημιουργούμε μία δομή με όλες τις εφαρμογές που τρέχουν στο σύστημά μας. Αυτό επιτυγχάνεται μέσω της εκτέλεσης

της κλήσης *pidof <exe_name>* στο σύστημα μας η οποία μας επιστρέφει όλα τα pids των στιγμιοτύπων με όνομα εκτελέσιμου <exe_name> που τρέχουν στο σύστημά μας. Στη συνέχεια, για κάθε νέα εφαρμογή που ξεκίνησε την εκτέλεσή της μεταξύ της προηγούμενης και της τρέχουσας επανάληψης ξεκινάμε ένα job, που αναλαμβάνει να εκτελεί το εργαλείο perf και να μας παρέχει μετρήσεις για τους performance counters που την αφορούν (βλ. 3.3) οι οποίες καταγράφονται σε αντίστοιχο αρχείο με σκοπό τη ανάκτησή τους κατά την εκτέλεση του μοντέλου μας. Επιπλέον, για κάθε εφαρμογή που ολοκληρώθηκε η εκτέλεσή της μεταξύ της προηγούμενης και της τρέχουσας επανάληψης γίνεται υπολογισμός του χρόνου που χρειάστηκε για την εκτέλεσή της, τερματίζεται η εκτέλεση του job που σχετίζεται με το perf και διαγράφονται τα αρχεία που την αφορούν. Στη συνέχεια, σε περίπτωση που ο χρήστης έχει δώσει εντολή για αλλαγή της προτεραιότητας μίας ή περισσότερων εφαρμογών, γίνεται η ενημέρωση των αντίστοιχων δομών δεδομένων που χειρίζονται αυτές τις πληροφορίες. Τέλος, ο βρόχος καλεί τη συνάρτηση *apply_model()*, η οποία εμπεριέχει όλη τη λογική του resource manager, και αναστέλλει την εκτέλεση της για ένα προκαθορισμένο χρονικό διάστημα (interval) μέσω κλήσης της συνάρτησης *sleep* της python. Αυτό είναι αρκετά σημαντικό ώστε ο resource manager να μην τρέχει συνέχεια επιβαρύνοντας το σύστημα και την απόδοση των εφαρμογών, αλλά να εκτελείται ανά τακτά χρονικά διαστήματα ως επόπτης παρεμβαίνοντας στο διαμοιρασμό της μνήμης όταν αυτό απαιτείται.

Μετά το πέρας της εκτέλεσης του παραπάνω βρόχου και ακριβώς πριν την ολοκλήρωση της εκτέλεσης του resource manager δημιουργείται ένα αρχείο που περιέχει τους χρόνους εκτέλεσης όλων των benchmarks αναλυτικά και με μία προκαθορισμένη σειρά ώστε να είναι εύκολη η οπτικοποίηση τους μέσω γραφικών παραστάσεων. Ακόμη, τερματίζεται το job που σχετίζεται με το εργαλείο pcm-memory.x και διαγράφεται το αρχείο εξόδου που χρησιμοποιεί από το σύστημα.

5.3 Μοντέλο

Σε αυτή την ενότητα θα περιγράψουμε την ”καρδιά” του resource manager, τη συνάρτηση *apply_model()* που αναλαμβάνει να συγκεντρώσει τις πληροφορίες από την εκτέλεση των εφαρμογών μας και τις εντολές του χρήστη και να τις εκτελέσει με έναν αποδοτικό τρόπο.

Αρχικά η συνάρτηση apply_model() συλλέγει τις μετρήσεις που αφορούν στη ζήτηση για bandwidth από κάθε κόμβο του συστήματος καθώς και για όλο το σύστημα συνολικά από το αρχείο που ενημερώνεται μέσω του εργαλείου pcm-memory.x. Στη συνέχεια, για κάθε εφαρμογή που τρέχει συλλέγει τους performance counters από το αντίστοιχο αρχείο που την αφορά και ενημερώνεται μέσω του εργαλείου perf και υπολογίζει τις παραμέτους CPI και Bandwidth για αυτή με βάση τους τύπους 4.2 και 4.5 αντίστοιχα. Για να υπολογίσει την κατανομή των σελίδων της μνήμης κάθε εφαρμογής στους κόμβους γίνεται χρήση των αρχείων numa_maps. Έτσι, για κάθε διεργασία υπολογίζονται οι σελίδες που είναι δεσμευμένες σε κάθε κόμβο και οι συνολικές σελίδες μνήμης την τρέχουσα χρονική στιγμή. Κατόπιν, η συνάρτηση με χρήση του εργαλείου taskset υπολογίζει το cpu affinity της κάθε εφαρμογής, δηλαδή σε ποιους πυρήνες εκτελείται, οπότε με βάση την αρίθμηση των επεξεργαστών για το μηχάνημα Sandman μπορούμε να προσδιορίσουμε σε ποιο κόμβο τρέχει.

Αφού ολοκληρωθεί η διαδικασία της συλλογής των απαραίτητων δεδομένων για το σύστημα και τις εφαρμογές που τρέχουν σε αυτό, η συνάρτηση apply_model() αρχίζει να θέτει σε εφαρμογή τις κατάλληλες ενέργειες ανάλογα με τις εντολές του χρήστη. Αρχικά, για κάθε εφαρμογή που εκτελείται γίνεται έλεγχος σχετικά με την προτεραιότητά της. Σε περίπτωση που πρόκειται για εφαρμογή υψηλής προτεραιότητας στόχος μας είναι να επιβάλλουμε τοπική εκτέλεση, δηλαδή ο πυρήνας που εκτελείται και η μνήμη που χρησιμοποιεί να βρίσκονται στον ίδιο κόμβο. Για το σκοπό αυτό, δεδομένου ότι δεν επιθυμούμε την αλλαγή της παραμέτρου cpu affinity για τις εφαρμογές μας ώστε να μην επηρεάζονται τα αποτελέσματά μας από το φαινόμενο memory contention, πρέπει να μεταφέρουμε όλη τη μνήμη της εφαρμογής στον κόμβο που βρίσκεται η cpu που χρησιμοποιεί. Πράγματι, ο resource manager με κατάλληλες κλήσεις της migratepages μεταφέρει τις σελίδες της μνήμης της εφαρμογής που βρίσκονται σε όλους τους κόμβους του συστήματος πλην του τοπικού στον τοπικό κόμβο ώστε αυτή να τρέχει σύμφωνα με την τοπολογία 0 και να διασφαλίζουμε ότι δεν υπάρχει υποβάθμιση της απόδοσής

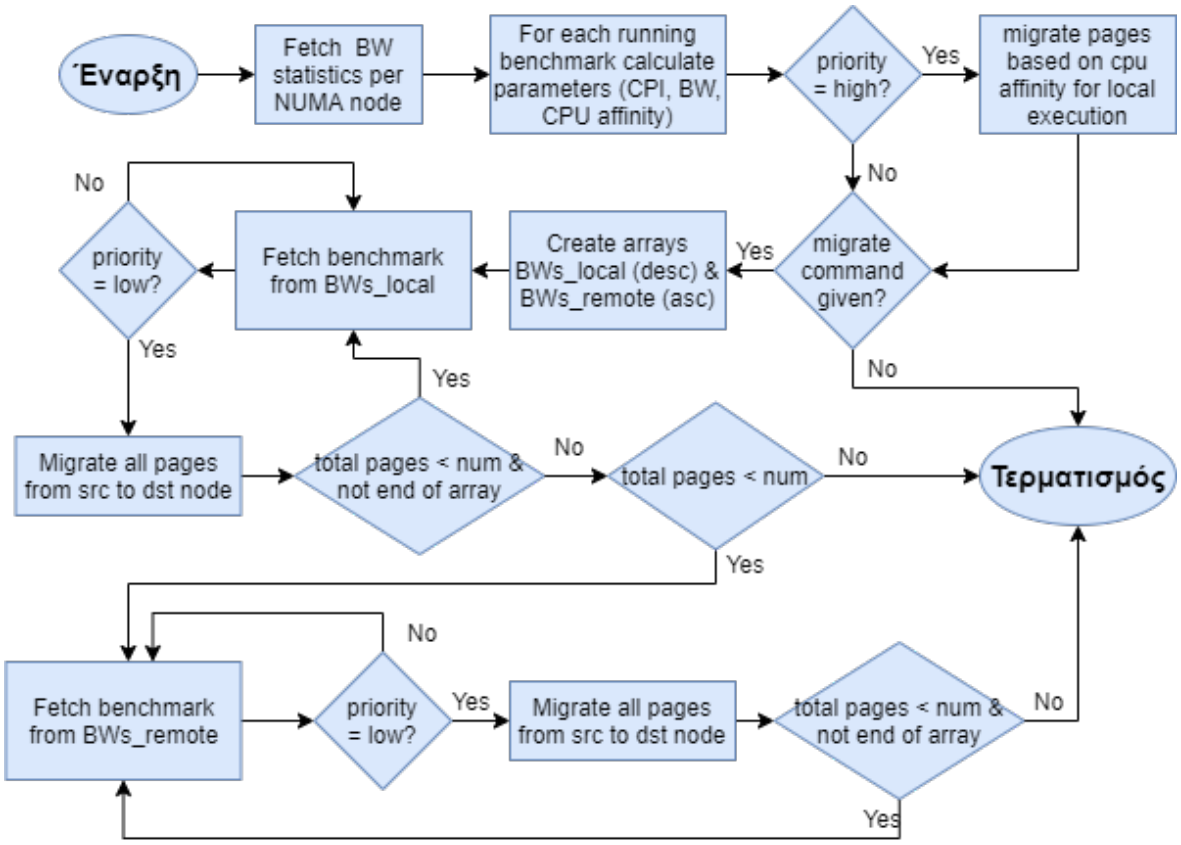
της.

Κατόπιν, η συνάρτηση apply_model ελέγχει αν έχει δοθεί από το χρήστη εντολή για μεταφορά σελίδων μνήμης μεταξύ των κόμβων του συστήματος. Σε περίπτωση που ισχύει κάτι τέτοιο, δηλαδή ο χρήστης έχει δώσει εντολή της μορφής *migrate* <*source_node*> <*destination_node*> <*num_pages*> το μοντέλο μας θα επιχειρήσει να μεταφέρει (τουλάχιστον) <num_pages> σελίδες μνήμης από τον κόμβο <source_node> στον κόμβο <destination_node> με όσο γίνεται πιο αποδοτικό τρόπο. Για το σκοπό αυτό κατασκευάζουμε δύο πίνακες, τον BWs_local που περιέχει τις τρέχουσες τιμές για τα bandwidths των εφαρμογών που τρέχουν χρησιμοποιώντας cpu από τον κόμβο <destination_node>, και τον BWs_remote που περιέχει τα bandwidths των εφαρμογών που τρέχουν χρησιμοποιώντας cpu από οποιονδήποτε κόμβο του συστήματος πλην του <destination_node>. Ταξινομούμε τον πίνακα BWs_local με φθίνουσα σειρά και τον πίνακα BWs_remote με αύξουσα σειρά.

Ας εξηγήσουμε σε αυτό το σημείο το συλλογισμό που μας οδήγησε σε αυτή την απόφαση. Με βάση τα πειράματα που εκτελέσαμε και τις μετρήσεις που πήραμε διαπιστώσαμε ότι η μεταβολή της απόδοσης από την τοπολογία local στις τοπολογίες remote και interleave είναι μεγαλύτερη όσο μεγαλύτερο είναι το bandwidth που καταναλώνει η κάθε εφαρμογή. Επομένως, για την ικανοποίηση ενός αιτήματος για μεταφορά σελίδων μνήμης στον κόμβο <destination_node> είναι λογικό να εργαστούμε με τέτοιο τρόπο ώστε η μεταφορά να ξεκινήσει από τις εφαρμογές που επηρεάζονται περισσότερο αν τρέχουν με cpu από αυτό τον κόμβο και έχουν σελίδες μνήμης στον κόμβο <source_node>. Έτσι, μετά τη μεταφορά οι εφαρμογές αυτές θα έχουν μεγαλύτερο ποσοστό των σελίδων της μνήμης τους τοπικά και συνεπώς θα τρέχουν πιο αποδοτικά. Ακόμη, για τις υπόλοιπες εφαρμογές, δηλαδή αυτές που τρέχουν με cpu από άλλο κόμβο πλην του <destination_node> είναι λογικό να αρχίσουμε τη μεταφορά σελίδων από αυτές που επηρεάζονται λιγότερο από την απόσταση της μνήμης. Αυτό συμβαίνει, διότι μετά τη μεταφορά αυτές οι εφαρμογές θα έχουν μεγαλύτερο ποσοστό των σελίδων της μνήμης τους σε μη τοπικό κόμβο και συνεπώς επιθυμούμε να επιβαρυνθούν όσο γίνεται λιγότερο από αυτή τη διαδικασία.

Ύστερα από την ταξινόμηση των πινάκων ξεκινάει η διαδικασία της μεταφοράς σελίδων μνήμης. Αρχίζουμε από τον πίνακα BWs_local τον οποίο διατρέχουμε (κατά φθίνουσα σειρά bandwidth) και για κάθε εφαρμογή που είναι χαμηλής προτεραιότητας μετακινούμε τις σελίδες της από τη μνήμη του κόμβου <source_node> στη μνήμη του κόμβου <destination_node>. Οι εφαρμογές υψηλής προτεραιότητας δεν επηρεάζονται από αυτή τη διαδικασία προκειμένου να παραμένει όσο γίνεται πιο σταθερή η απόδοσή τους. Αν μετά απο μία μεταφορά διαπιστώσουμε ότι ο αριθμός των σελίδων που έχει προσδιορίσει ο χρήστης ξεπεράστηκε, σταματάμε τη διαδικασία χωρίς να μεταφέρουμε άλλες σελίδες. Στην περίπτωση που μεταφέρουμε σελίδες για όλες τις εφαρμογές του πίνακα BWs_local και δεν έχουμε φτάσει τον αριθμό <num_pages> συνεχίζουμε με τον πίνακα BWs_remote για τον οποίο επαναλαμβάνουμε την ίδια διαδικασία μέχρι να συμπληρωθεί ο απαιτούμενος αριθμός σελίδων μνήμης. Αν και ύστερα από την εξέταση όλων των εφαρμογων του πίνακα BWs_remote δεν έχουν μεταφερθεί συνολικά <num_pages> σελίδες μνήμης από τον κόμβο <source_node> στον κόμβο <destination_node> σημαίνει ότι δεν υπάρχουν αρκετές σελίδες στη μνήμη για μεταφορά οπότε ενημερώνεται ο χρήστης με κατάλληλο μήνυμα και ολοκληρώνεται η διαδικασία. Στην εικόνα 5.2 φαίνεται το διάγραμμα ροής της συνάρτησης apply_model() που περιλαμβάνει όλες τις βασικές λειτουργίες που επιτελεί.

Το παραπάνω μοντέλο του resource manager (το ονομάζουμε ”Best”) υλοποιεί τη βασική μας ιδέα και ο κώδικάς του φαίνεται στο παράρτημα Β. Στη συνέχεια υλοποιήσαμε και μία βελτιωμένη έκδοσή του (Best+) που προκύπτει ουσιαστικά ως άμεση επέκταση. Στο βελτιωμένο μοντέλο μας χωρίζουμε τον πίνακα BWs_remote που χρησιμοποιήσαμε προηγουμένως για τις εφαρμογές που τρέχουν με cpu από κόμβο εκτός του <destination_node> σε δύο υποπίνακες. Ο ένας που ονομάσαμε BWs_other περιέχει τις εφαρμογές που εκτελούνται σε cpu εκτός των κόμβων <source_node> και <destination_node> και ο άλλος που ονομάσαμε BWs_remote περιέχει τώρα μόνο τις εφαρμογές που εκτελούνται σε cpu του κόμβου <source_node>. Ο διαχωρισμός αυτός έγινε ώστε κατά τη μεταφορά σελίδων να δοθεί προτεραιότητα στις εφαρμογές του πίνακα BWs_other έναντι του πίνακα BWs_remote, διότι στην πρώτη περίπτωση οι σελίδες βρίσκονται στον απομακρυσμένο κόμβο



Σχήμα 5.2: Το μοντέλο του resource manager.

<source_node> και μεταφέρονται στον άλλο απομακρυσμένο κόμβο <destination_node> ενώ στη δεύτερη περίπτωση οι σελίδες μεταφέρονται από τοπικό κόμβο σε απομακρυσμένο. Έτσι σε αυτό το μοντέλο η σειρά με την οποία εξετάζονται οι πίνακες των εφαρμογών για να γίνει η μετακίνηση των σελίδων είναι BWs_local->BWs_other->BWs_remote.

5.4 Σύγκριση με άλλες προσεγγίσεις

Η παραπάνω μεθολογία όπως περιγράφηκε αποτελεί τη βασική ιδέα στην οποία στηρίζεται η λειτουργία του resource manager που υλοποιήσαμε. Για την αξιολόγηση του μοντέλου και την καλύτερη κατανόηση του πλεονεκτήματος που προσφέρει όσον αφορά στη βελτίωση της απόδοσης του συστήματος συνολικά απαιτήθηκε η σύγκρισή του με άλλες προσεγγίσεις. Πιο αναλυτικά στην παρούσα έρευνα επικεντρωθήκαμε σε τρία εναλλακτικά σενάρια που σχετίζονται με τη σειρά επιλογής των benchmarks για τη μεταφορά των σελίδων της μνήμης τους.

Το πρώτο μοντέλο που θελήσαμε να εισαγάγουμε ως μέτρο σύγκρισης είναι αυτό της τυχαίας επιλογής (Random). Αυτό μας δίνει μια εκτίμηση για την απόδοση όταν σε ένα σύστημα πρέπει να γίνουν μεταφορές σελίδων μνήμης λόγω εξωτερικών παραγόντων και δεν υπάρχει κάποια λογική να εφαρμοστεί οπότε η επιλογή για το ποιες εφαρμογές θα αλλάξουν την κατανομή της μνήμης τους γίνεται με τυχαίο τρόπο. Για την υλοποίηση αυτής της προσέγγισης χρησιμοποιήσαμε έναν πίνακα με τα όλα τα benchmarks που εκτελούνται και κάθε φορά που έπρεπε να εκτελεστεί μια μεταφορά κάναμε αναδιάταξη των στοιχείων του με τυχαίο τρόπο ώστε η σειρά εξέτασης των εφαρμογών να μην ακολουθεί κάποια λογική.

Το δεύτερο μοντέλο που θεωρήσαμε κατάλληλο να υλοποιήσουμε προκειμένου να έχουμε μια εκτίμηση για τη χειρότερη απόδοση που μπορεί να έχει το σύστημα είναι ουσιαστικά το δυαδικό του μοντέλου που παρουσιάσαμε και το ονομάσαμε ”Worst”. Χρησιμοποιούμε και σε αυτή την περίπτωση

τους δύο πίνακες BWs_local και BWs_remote με τα bandwidths των εφαρμογών που εκτελούνται στο σύστημά μας. Ωστόσο, αυτή τη φορά ταξινομούμε τον πίνακα BWs_local κατά αύξουσα σειρά bandwidth και τον πίνακα BWs_remote κατά φθίνουσα σειρά και ξεκινάμε τις μετακινήσεις σελίδων μνήμης από τον δεύτερο. Με αυτόν τον τρόπο, μεταφέρουμε πρώτα στον κόμβο <destination_node> σελίδες από εφαρμογές που τρέχουν σε άλλους κόμβους και επηρεάζονται περισσότερο από τις μη τοπικές τοπολογίες. Αν δεν έχει συμπληρωθεί ο αριθμός των απαιτούμενων σελίδων προς μεταφορά διατρέχουμε τον πίνακα BWs_local και μεταφέρουμε σελίδες ξεκινώντας από τις εφαρμογές που επηρεάζονται λιγότερο από την απόσταση της μνήμης. Έτσι, παραμένουν μακριά οι σελίδες των εφαρμογών που δε θα εξεταστούν αν συμπληρωθεί νωρίτερα ο απαιτούμενος αριθμός <num_nodes> κάτι που οδηγεί σε πτώση της απόδοσης του συστήματος συνολικά.

Το τρίτο μοντέλο είναι ουσιαστικά το δυαδικό της βελτιωμένης έκδοσης που παρουσιάσαμε (”Worst+”). Έχουμε και εδώ τους τρεις πίνακες με τις εφαρμογές τους οποίους ταξινομούμε κατάλληλα κατ’ αντιστοιχία με το μοντέλο Worst, δηλαδή τον πίνακα BWs_local σε αύξουσα σειρά bandwidth και τους πίνακες BWs_other και BWs_remote σε φθίνουσα σειρά. Στη συνέχεια, η σειρά με την οποία εξετάζουμε τους πίνακες για τη μεταφορά σελίδων είναι η αντίθετη από αυτή που ακολουθήσαμε στο μοντέλο Best+, δηλαδή BWs_remote->BWs_other->BWs_local έτσι ώστε να αποκτήσουν απομακρυσμένη τοπολογία όσες περισσότερες εφαρμογές γίνεται

Στον πίνακα 5.1 φαίνονται συνοπτικά όλα τα μοντέλα που αναπτύξαμε και κατόπιν αξιολογήσαμε στα πειράματα του κεφαλαίου 6 μέσω της ενσωμάτωσής του στο πρόγραμμα του resource manager.

Μοντέλο	**Περιγραφή**
Best	Πίνακες BWs_local (φθίνουσα ταξινόμηση) και BWs_remote (αύξουσα ταξινόμηση) με σειρά εξέτασης BWs_local -> BWs_remote
Best+	Πίνακες BWs_local (φθίνουσα ταξινόμηση), BWs_other (αύξουσα ταξινόμηση) και BWs_remote (αύξουσα ταξινόμηση) με σειρά εξέτασης BWs_local -> BWs_other -> BWs_remote
Random	Ένας πίνακας BWs_random με τυχαία αναδιάταξη των στοιχείων του σε κάθε μεταφορά
Worst	Πίνακες BWs_local (αύξουσα ταξινόμηση) και BWs_remote (φθίνουσα ταξινόμηση) με σειρά εξέτασης BWs_remote -> BWs_local
Worst+	Πίνακες BWs_local (αύξουσα ταξινόμηση), BWs_other (φθίνουσα ταξινόμηση) και BWs_remote (φθίνουσα ταξινόμηση) με σειρά εξέτασης BWs_remote -> BWs_other -> BWs_local

Πίνακας 5.1: Τα μοντέλα του resource manager που αξιολογήσαμε

Κεφάλαιο 6

Αξιολόγηση

Στο κεφάλαιο αυτό θα προχωρήσουμε στην αξιολόγηση του resource manager που παρουσιάσαμε στο προηγούμενο κεφάλαιο. Θα περιγράψουμε τον τρόπο διεξαγωγής των πειραμάτων, θα παρουσιάσουμε τις μετρήσεις που λάβαμε, θα σχολιάσουμε τα αποτελέσματα και θα συγκρίνουμε το μοντέλο μας με τις άλλες προσεγγίσεις που εξετάσαμε προκειμένουν να γίνει εμφανές το συγκριτικό πλεονέκτημα που παρουσιάζει και να ποσοτικοποιηθεί η βελτίωση της απόδοσης του συστήματος που επιτυγχάνει.

6.1 Τρόπος διεξαγωγής των πειραμάτων

Τα πειράματα που διεξάγαμε μπορούν να χωριστούν σε δύο βασικές κατηγορίες, τα *στατικά* και τα *δυναμικά*. Η πρώτη κατηγορία περιλαμβάνει όλα τα πειράματα που εκτελέσαμε κατά τη διάρκεια της ανάπτυξης του resource manager προκειμένου να βγάλουμε κάποια συμπεράσματα σχετικά με την ταυτόχρονη εκτέλεση των εφαρμογών στο σύστημά μας σε διάφορους συνδυασμούς από τοπολογίες. Είναι στατικά διότι η τοπολογία των εφαρμογών παραμένει σταθερή καθόλη τη διάρκεια της εκτέλεσής τους και δεν πυροδοτούνται οι ενέργειες του μοντέλου. Η δεύτερη κατηγορία περιλαμβάνει τα πειράματα στα οποία συμβαίνουν μεταφορές σελίδων μνήμης κατά τη διάρκεια της εκτέλεσής τους με αποτέλεσμα να αλλάζει δυναμικά η τοπολογία των εφαρμογών. Σε αυτά τα πειράματα, βλέπουμε ουσιαστικά την απόδοση του συστήματος στην πράξη όταν εφαρμόζεται το μοντέλο του resource manager και μπορούμε να αξιολογήσουμε την αποτελεσματικότητά του.

6.2 Στατικά πειράματα

Σε αυτή την κατηγορία πειραμάτων επιλέξαμε δύο ομάδες benchmarks από αυτά του πίνακα 3.6 μία για κάθε σετ πειραμάτων που εκτελέσαμε. Για κάθε σετ εκτελέσαμε τα benchmarks σε διάφορες τοπολογίες προκειμένου να δούμε τις μεταβολές στην απόδοση των εφαρμογών και τον τρόπο που αλληλοεπηρεάζονται όταν εκτελούνται ταυτόχρονα.

Στο πρώτο σετ πειραμάτων επιλέξαμε να εκτελέσουμε τα benchmarks που φαίνονται στον πίνακα 6.1. Η επίλογη των εφαρμογών δεν έγινε τυχαία αλλά λάβαμε υπόψιν τα αποτελέσματα των πειραμάτων της ενότητας 4.3. Έτσι, τα πρώτα τέσσερα benchmarks επιλέχθηκαν διότι εμφανίζουν τη μικρότερη μεταβολή στην απόδοσή τους όταν η τοπολογία που τρέχουν αλλάξει, ενώ τα τελευταία τέσσερα εμφανίζουν τη μεγαλύτερη μεταβολή στην απόδοσή τους όταν αλλάζει η τοπολογία. Με αυτή την ομάδα εφαρμογών καλύπτουμε επομένως όλο το φάσμα των συμπεριφορών και αποτελούν καλή επιλογή για να βγάλουμε χρήσιμα συμπεράσματα που εισάγαμε στο μοντέλο του resource manager.

Το πρώτο σετ πειραμάτων περιλαμβάνει τα πειράματα που φαίνονται συνοπτικά στον πίνακα 6.2. Πιο αναλυτικά στο πρώτο πείραμα εκτελούμε τα benchmarks ταυτόχρονα στον κόμβο 0 για να δούμε πως συμπεριφέρονται και πόσο μεταβάλλεται η απόδοσή τους σε σχέση με τη μεμονωμένη εκτέλεση του καθενός. Στο επόμενο πείραμα, θεωρούμε το καλό σενάριο όπου για τη βελτίωση της απόδοσης του συστήματος εκτελούμε την πρώτη τετράδα των benchmarks στην απομακρυσμένη μνήμη του κόμβου 1 γνωρίζοντας ότι δεν επηρεάζεται η απόδοσή τους, ενώ η άλλη τετράδα που περιέχει τα benchmarks που είναι ευαίσθητα αναφορικά με την απόσταση της μνήμης τρέχουν τοπικά. Στο

Benchmarks	Ευαισθησία στην τοπολογία
500.perlbench_r	low
508.namd_r	low
541.leela_r	low
544.nab_r	low
520.omnetpp_r	high
505.mcf_r	high
519.lbm_r	high
549.fotonik3d_r	high

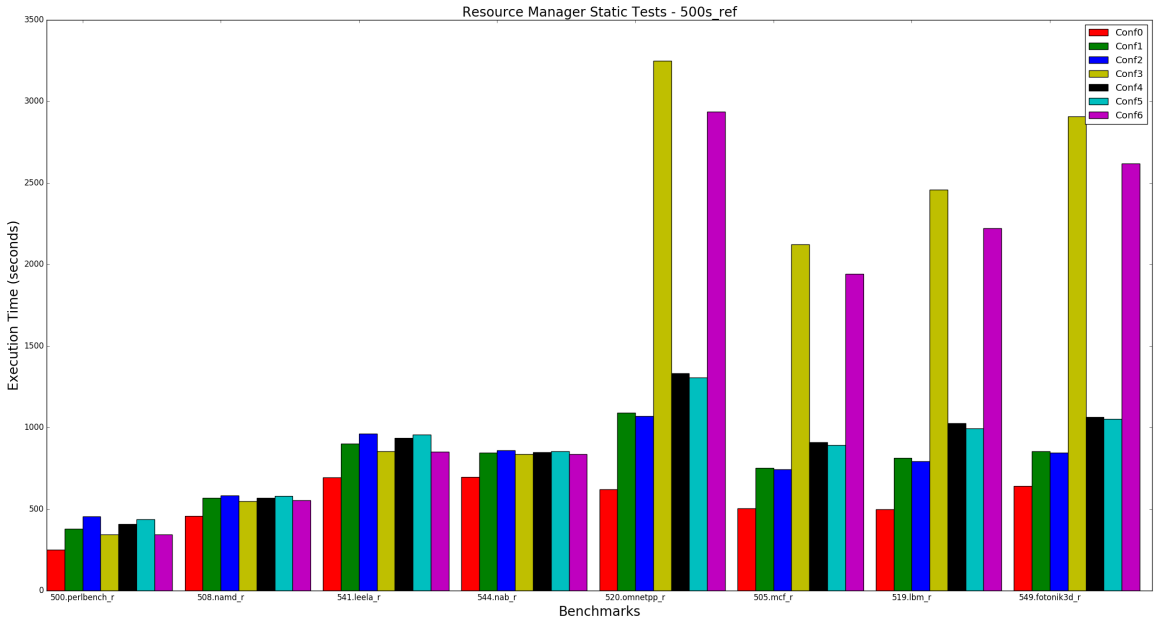
Πίνακας 6.1: Benchmarks που εκτελέσαμε στο πρώτο σετ στατικών πειραμάτων.

πείραμα 2 θεωρούμε το κακό σενάριο όπου συμβαίνει η αντίθετη αντιστοίχιση των τετράδων με τις τοπολογίες 0 και 1, δηλαδή τα benchmarks των οποίων η απόδοση μεταβάλλεται περισσότερο όταν τρέχουν απομακρυσμένα τα εκτελούμε με τοπολογία 1 ενώ τα υπόλοιπα τρέχουν τοπικά. Τα επόμενα 3 πειράματα είναι αντίστοιχα ένα προς ένα με τα παραπάνω με τη διαφορά ότι ο κόμβος 0 επιβαρύνεται σημαντικά λόγω της αυξημένης ζήτησης σε bandwidth που δημιουργούν τα 8 στιγμιότυπα του stress-ng. Παρατηρούμε ότι οι πυρήνες που τρέχουν τα benchmarks μένουν σταθεροί σε όλα τα πειράματα ενώ και για το stress-ng χρησιμοποιούνται διαφορετικές cpu αφού όπως έχουμε αναφέρει θέλουμε να έχουμε ίδιες συνθήκες σε κάθε εκτέλεση ώστε τα αποτελέσματά μας να είναι αξιόπιστα και να μην επηρεάζονται απο το φαινόμενο memory contention. Στους πίνακες Α.9 και Α.10 έχουμε καταγράψει τις μετρήσεις που λάβαμε για αυτό το σετ πειραμάτων και στις γραφικές παραστάσεις 6.1 και 6.2 βλέπουμε τα αντίστοιχα αποτελέσματα.

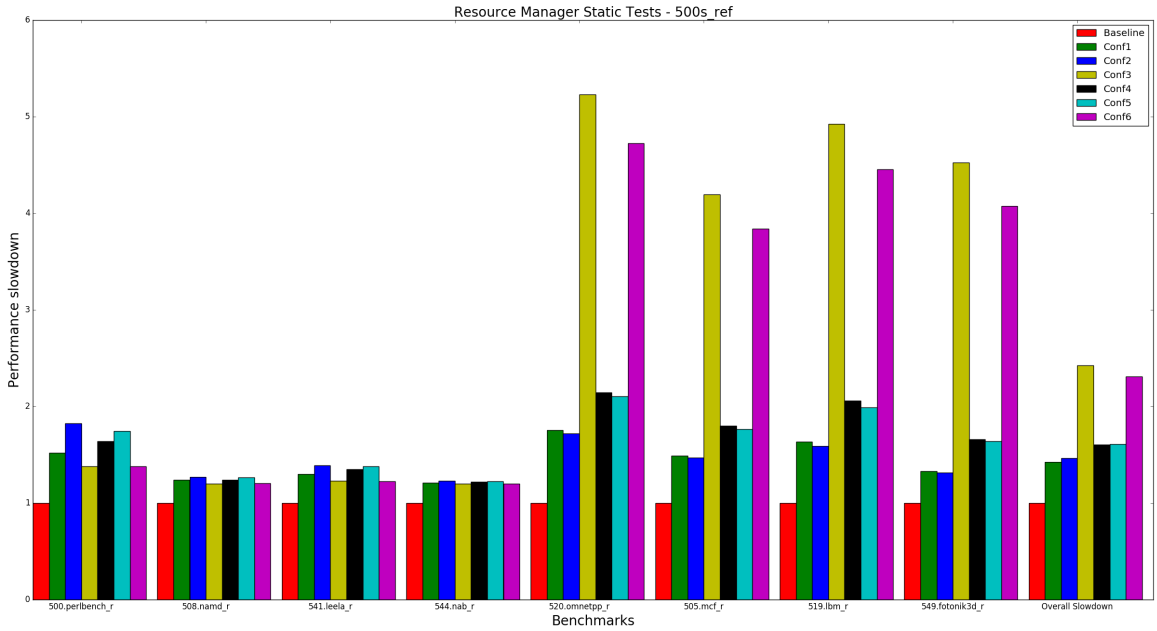
Πείραμα	Περιγραφή
1	Τα benchmarks τρέχουν όλα μαζί στον κόμβο 0 χρησιμοποιώντας την τοπική του μνήμη (τοπολογία 0)
2	Τα τέσσερα πρώτα benchmarks τρέχουν με cpu από τον κόμβο 0 και μνήμη από τον κόμβο 1 (τοπολογία 1) ενώ τα τέσσερα τελευταία benchmarks τρέχουν στον κόμβο 0 χρησιμοποιώντας την τοπική του μνήμη (τοπολογία 0)
3	Τα τέσσερα πρώτα benchmarks τρέχουν στον κόμβο 0 χρησιμοποιώντας την τοπική του μνήμη (τοπολογία 0) ενώ τα τέσσερα τελευταία benchmarks τρέχουν με cpu από τον κόμβο 0 και μνήμη από τον κόμβο 1 (τοπολογία 1)
4	Όπως στο πείραμα 1 αλλά ταυτόχρονα εκτελούνται 8 στιγμιότυπα του stress-ng με cpu από τον κόμβο 1 και μνήμη από τον κόμβο 0
5	Όπως στο πείραμα 2 αλλά ταυτόχρονα εκτελούνται 8 στιγμιότυπα του stress-ng με cpu από τον κόμβο 1 και μνήμη από τον κόμβο 0
6	Όπως στο πείραμα 3 αλλά ταυτόχρονα εκτελούνται 8 στιγμιότυπα του stress-ng με cpu από τον κόμβο 1 και μνήμη από τον κόμβο 0

Πίνακας 6.2: Τα πειράματα που περιλαμβάνει το πρώτο σετ στατικών πειραμάτων.

Η πρώτη γραφική παράσταση παρουσιάζει τους χρόνους εκτέλεσης όλων των benchmarks σε όλα τα πειράματα που αναφέραμε. Η πρώτη στήλη (κόκκινη) αντιστοιχεί στο χρόνο μεμονωμένης εκτέλεσης του κάθε benchmark στην τοπολογία local όπως έχει ληφθεί από τα αποτελέσματα των πειραμάτων της ενότητας 4.3. Η δεύτερη γραφική παράσταση παρουσιάζει τις ίδιες πληροφορίες κανονικοποιημένες για κάθε benchmark ως προς το χρόνο της μεμονωμένης εκτέλεσής του στην τοπολογία local προκειμένου να μπορούν να γίνουν άμεσες συγκρίσεις μεταξύ τους. Ακόμη, περιέχει ένα



Σχήμα 6.1: Αποτελέσματα πρώτου σετ στατικών πειραμάτων (μη κανονικοποιημένα).



Σχήμα 6.2: Αποτελέσματα πρώτου σετ στατικών πειραμάτων (κανονικοποιημένα).

επιπρόσθετο σετ αποτελεσμάτων στο τέλος που μας δείχνει την απόδοση του συστήματος συνολικά σε κάθε πείραμα. Η τιμή αυτή υπολογίζεται ως ο αρμονικός μέσος των αποδόσεων των benchmarks σε κάθε πείραμα σύμφωνα με τον παρακάτω τύπο:

$$Performance_{system}(i) = \sqrt[8]{\prod_{j \in benchmarks} Performance_j(i)} \tag{6.1}$$

όπου benchmarks είναι το σύνολο των εφαρμογών του πίνακα 6.1 και $i \in \{1, 2, 3, 4, 5, 6\}$ το id του πειράματος σύμφωνα με τον πίνακα 6.2.

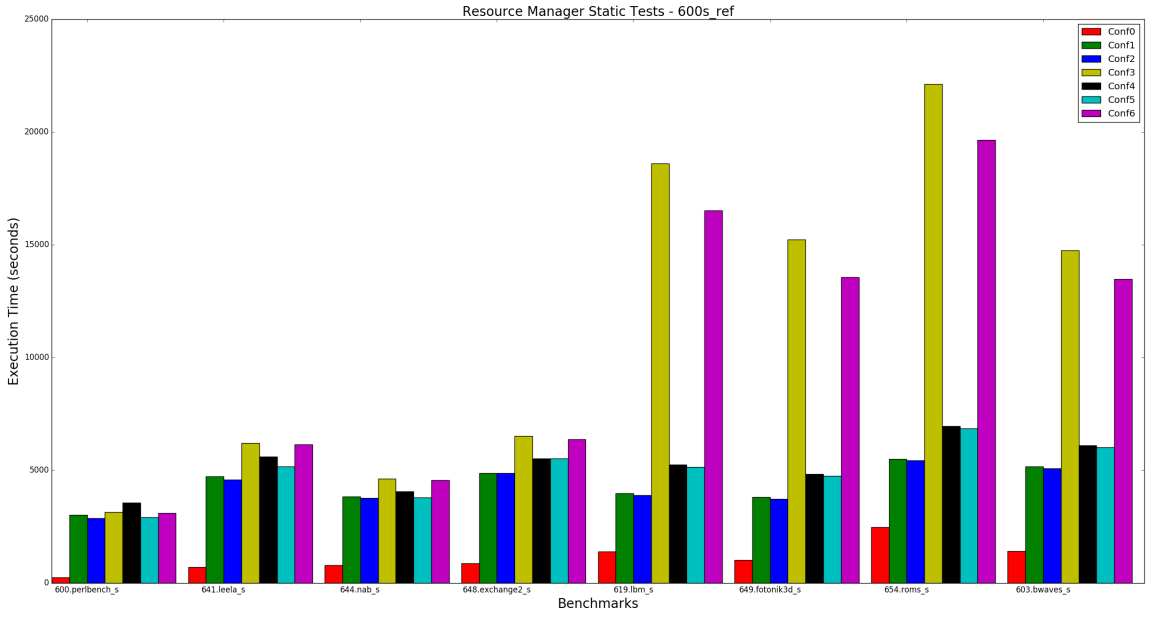
Αυτό που γίνεται εύκολα αντιληπτό παρατηρώντας τις γραφικές παραστάσεις είναι ότι η μεγαλύτερη μεταβολή της απόδοσης συμβαίνει στα πειράματα 3 και 6 και αφορά στα benchmarks που

επηρεάζονται αισθητά από την απόσταση που βρίσκεται η μνήμη τους. Έτσι, ενώ η μεταβολή της απόδοσης του συστήματος μεταξύ των πειραμάτων 1 και 2 όπως επίσης μεταξύ των πειραμάτων 4 και 5 είναι αμελητέα, παρατηρούμε επιβράδυνση ίση με 70% στο πείραμα 3 και 44% στο πείραμα 6 σε σχέση με τα πειράματα 1 και 4 αντίστοιχα. Αυτό σημαίνει ότι, αν θέλουμε να εξασφαλίσουμε ότι η απόδοση του συστήματος θα είναι κοντά στη βέλτιστη πρέπει οι εφαρμογές με μεγάλη ζήτηση σε bandwidth να τρέχουν τοπικά.

Το δεύτερο σετ πειραμάτων περιλαμβάνει ακριβώς τα ίδια πειράματα με το πρώτο όπως περιγράφονται αναλυτικά στον πίνακα 6.2. Αυτό που αλλάζει είναι οι εφαρμογές που χρησιμοποιούμε, οι οποίες σε αυτό το σετ είναι από τη σειρά 600s και φαίνονται στον πίνακα 6.3. Η επιλογή τους έγινε με την ίδια λογική που ακολουθήσαμε και στο προηγούμενο σετ, δηλαδή τα πρώτα τέσσερα είναι αυτά που επηρεάζονται λιγότερο από την απόσταση της μνήμης ενώ τα τελευταία τέσσερα επηρεάζονται περισσότερο. Στους πίνακες Α.11 και Α.12 έχουμε καταγράψει τις μετρήσεις που λάβαμε για αυτό το σετ πειραμάτων και στις γραφικές παραστάσεις 6.3 και 6.4 βλέπουμε τα αντίστοιχα αποτελέσματα.

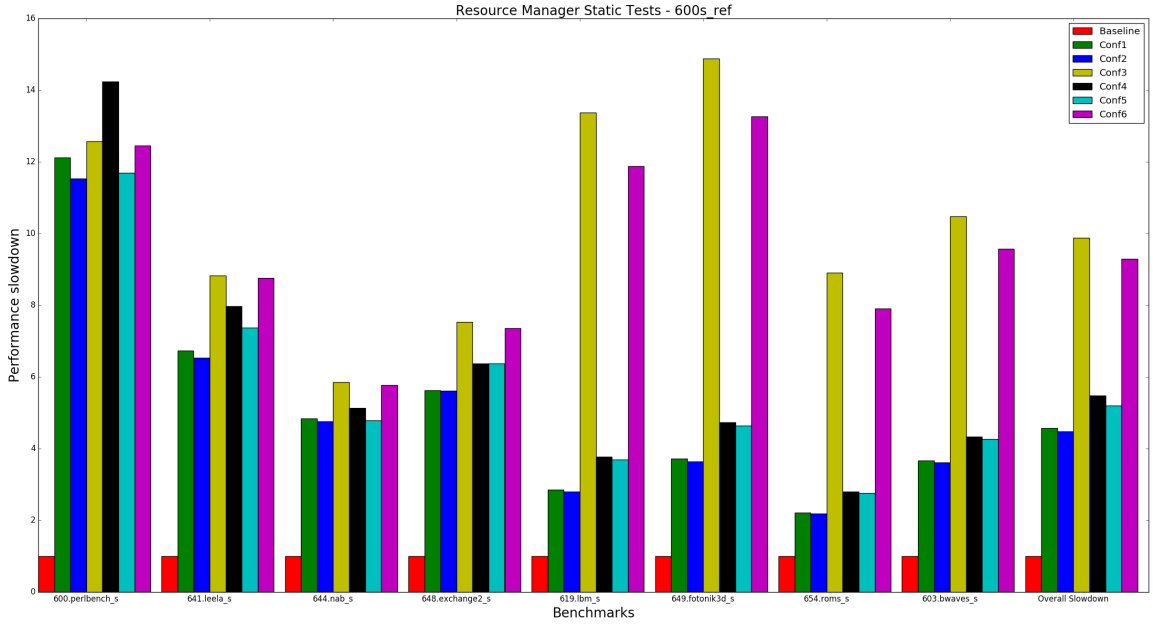
Benchmarks	**Ευαισθησία στην τοπολογία**
600.perlbench_s	low
641.leela_s	low
644.nab_s	low
648.exchange2_s	low
619.lbm_s	high
649.fotonik3d_s	high
654.roms_s	high
603.bwaves_s	high

Πίνακας 6.3: Benchmarks που εκτελέσαμε στο δεύτερο σετ στατικών πειραμάτων.



Σχήμα 6.3: Αποτελέσματα δεύτερου σετ στατικών πειραμάτων (μη κανονικοποιημένα).

Τα συμπεράσματα που εξάγουμε απο τις μετρήσεις του δεύτερου σετ πειραμάτων είναι αντίστοιχα με αυτά του πρώτου σετ. Με άλλα λόγια, παρατηρούμε ότι η μεγαλύτερη μεταβολή της απόδοσης εμφανίζεται στα πειράματα 3 και 6 όταν η δεύτερη τετράδα των εφαρμογών που επηρεάζονται από την απόσταση της μνήμης τρέχει απομακρυσμένα. Εδώ η μεταβολή της απόδοσης του συστήματος από το



Σχήμα 6.4: Αποτελέσματα δεύτερου σετ στατικών πειραμάτων (κανονικοποιημένα).

πείραμα 1 στο πείραμα 3 είναι 116% ενώ από το πείραμα 4 στο πείραμα 6 ίση με 70%. Επαληθεύουμε με αυτό τον τρόπο την διαπίστωσή μας ότι για την επίτευξη μιας (σχεδόν) βέλτιστης απόδοσης του συστήματος είναι σημαντικό οι εφαρμογές που επηρεάζονται να τρέχουν τοπικά. Η διαπίστωση αυτή αποτέλεσε τη θεμελιώδη ιδέα πάνω στην οποία στηρίχθηκε το μοντέλο του resource manager αφού ο στόχος του είναι να επιβάλλει μια σειρά διάταξης των εφαρμογών για τη μετακίνηση των σελίδων της μνήμης τους έτσι ώστε αν πρόκεται να οδηγηθούν στον τοπικό κόμβο που εκτελούνται να επιλέγονται πρώτα αυτές που επηρεάζονται περισσότερο ενώ αν πρόκειται να οδηγηθούν σε απομακρυσμένο να επιλέγονται πρώτα αυτές που επηρεάζονται λιγότερο.

6.3 Δυναμικά πειράματα

Σε αυτή την κατηγορία πειραμάτων δοκιμάσαμε στην πράξη τον resource manager και συγκρίναμε το μοντέλο του (Best) με αυτά της τυχαίας επιλογής (Random) και της χειρότερης περίπτωσης (Worst) προκειμένου να προσδιορίσουμε τη βελτίωση της απόδοσης που επιτυγχάνει.

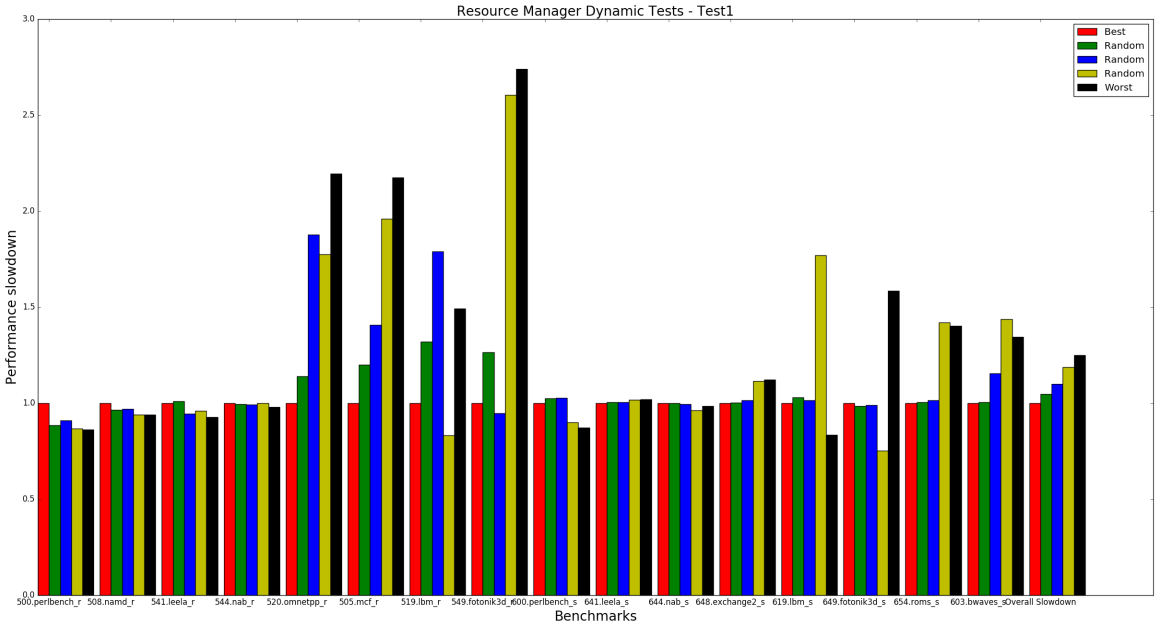
Κάθε σετ πειραμάτων χαρακτηρίζεται από τα benchmarks που περιλαμβάνει και την αρχική τοποθέτησή τους στο σύστημα καθώς και από μια αλληλουχία εντολών μεταφοράς μνήμης που εκτελούνται σε όλα τα πειράματα του σετ. Τα πειράματα σε κάθε σετ είναι ουσιαστικά πανομοιότυπες εκτελέσεις ως προς τις αρχικές συνθήκες και τις εντολές για μεταφορά σελίδων μνήμης που δίνονται όσο τρέχουν με τη διαφορά ότι αλλάζει το μοντέλο που χρησιμοποιεί ο resource manager και καθορίζει ποιες εφαρμογές θα επηρεαστούν από την κάθε εντολή μεταφοράς. Έτσι, εκτελούμε αρχικά το βέλτιστο μοντέλο του resource manager που κατασκευάσαμε για να το χρησιμοποιήσουμε ως αναφορά και στη συνέχεια εκτελούμε το μοντέλο της τυχαίας επιλογής πολλαπλές φορές για να προσδιορίσουμε το μέσο όρο της απόδοσης που το χαρακτηρίζει. Τέλος, εκτελούμε το μοντέλο Worst που επιτυγχάνει τη χειρότερη απόδοση για να έχουμε ένα άνω όριο και να είναι πιο σαφής και ολοκληρωμένη η αξιολόγηση.

Κατά την διεξαγωγή των πειραμάτων αξιολόγησης της απόδοσης του resource manager συνειδητοποιήσαμε ότι η χρήση της γραμμής εντολών δεν ήταν πρακτική. Αυτό που θέλαμε ήταν να εκτελούμε τα πειράματα με ένα όσο γίνεται περισσότερο αυτοματοποιημένο τρόπο και υπό τις ίδιες συνθήκες τροποποιώντας κάποιες παραμέτρους. Για το λόγο αυτό τροποποιήσαμε τον κώδικα του resource manager ώστε να μπορεί να δέχεται τις εντολές για μεταφορά μνήμης μέσω ενός αρχείου και να τις

εκτελεί σε προκαθορισμένο χρόνο, πανομοιότυπο για κάθε σετ πειραμάτων. Έτσι, σε κάθε σετ πειραμάτων υπάρχει ένα αρχείο με όνομα *automatic_migration.txt* που περιέχει τις εντολές μεταφοράς μνήμης οι οποίες εκτελούνται σειριακά ανά τακτά χρονικά διαστήματα και ένα αρχείο *info.txt* που περιέχει τα benchmarks που εκτελούνται σε αυτό το σετ με τις αρχικές τοποθετήσεις τους και το μοντέλο που χρησιμοποιούμε σε κάθε μεμονωμένο πείραμα.

6.3.1 Σενάριο 1

Στο πρώτο σετ από τα δυναμικά πειράματα ορίσαμε για την αρχική τοποθέτησή των εφαρμογών στο σύστημα την εκτέλεση των benchmarks του πίνακα 6.1 στον κόμβο 0 με την τοπολογία local και την εκτέλεση των benchmarks του πίνακα 6.3 στον κόμβο 1 πάλι με τοπολογία local. Κατά τη διάρκεια της εκτέλεσης των εφαρμογών αυτών δίνονται στο σύστημα 25 εντολές για τη μεταφορά σελίδων μνήμης μεταξύ των κόμβων όλου του συστήματος. Ο αριθμός των σελίδων που ορίζονται για μεταφορά σε κάθε εντολή είναι της τάξης του 10^4 το οποίο ισοδυναμεί με μεταφορές της τάξης των 40 MB δεδομένου ότι το μέγεθος της κάθε σελίδας είναι 4 KB. Στο πρώτο πείραμα χρησιμοποιήσαμε το βέλτιστο μοντέλο (Best) του resource manager που κατασκευάσαμε, στα επόμενα τρία το μοντέλο της τυχαίας επιλογής (Random) και στο τελευταίο το δυαδικό μοντέλο του βέλτιστου (Worst) που θεωρητικά προκαλεί τη χειρότερη απόδοση στο σύστημα. Σους πίνακες Α.13 και Α.14 βλέπουμε τις μετρήσεις που λάβαμε για αυτό το σετ πειραμάτων και στη γραφική παράσταση 6.5 οπτικοποιούνται τα αποτελέσματα ομαδοποιημένα ως προς τα στιγμιότυπα κάθε benchmark που τρέχουν στους τέσσερεις κόμβους του συστήματος.



Σχήμα 6.5: Αποτελέσματα 1ου σετ δυναμικών πειραμάτων.

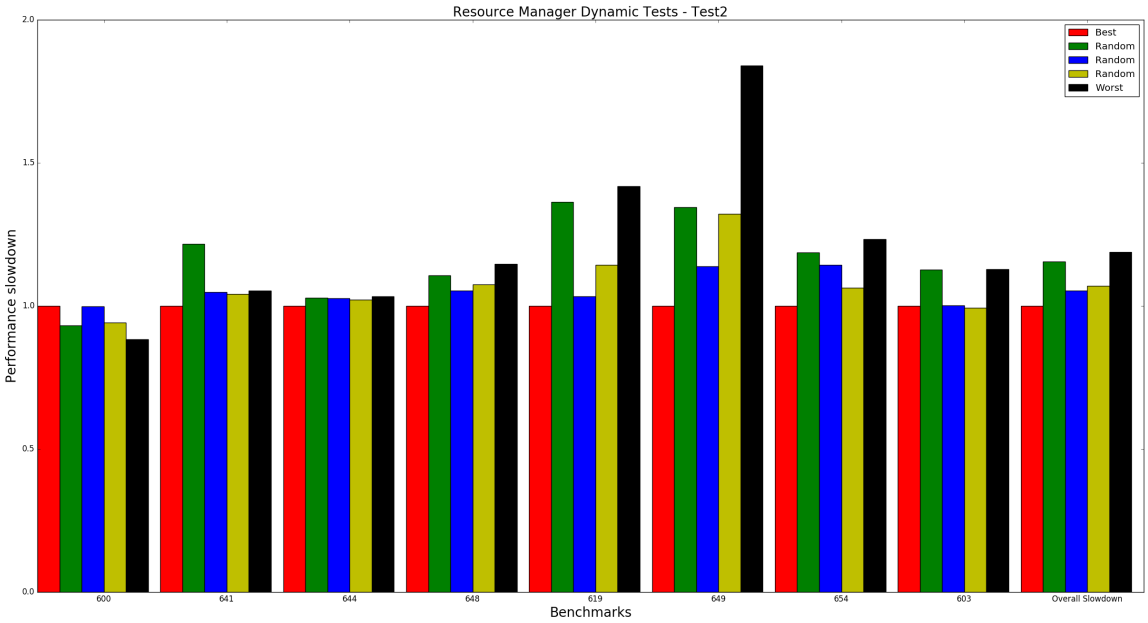
Η γραφική παράσταση απεικονίζει τη μεταβολή της απόδοσης των benchmarks για κάθε πείραμα που διεξάγαμε σε αυτό το σετ πειραμάτων κανονικοποιημένη ως προς το χρόνο εκτέλεσης με χρήση του μοντέλου Best. Καθεμιά από τις 8 πρώτες δέσμες στηλών αντιπροσωπεύει μια ομάδα από τα στιγμιότυπα ενός συγκεκριμένου benchmark και απεικονίζει το μέσο όρο της μεταβολής της απόδοσης αυτών σε κάθε σενάριο. Η τελευταία δέσμη στηλών στη γραφική παράσταση παρουσιάζει την απόδοη του συστήματος σε κάθε πείραμα κανονικοποιημένη πάλι ως προς την απόδοση του πρώτου πειράματος που υπολογίζεται σύμφωνα με τον τύπο 6.1. Όπως παρατηρούμε οι αποδόσεις των benchmarks σε κάθε πείραμα μεταβάλλονται με διαφορετικό τρόπο, δηλαδή ένα συγκεκριμένο μοντέλο μπορεί να επωφελήσει κάποια benchmarks και να υποβαθμίσει την απόδοση άλλων. Αυτό είναι λογικό αφού οι εντολές για μεταφορά μνήμης που δίνονται στον resource manager προκαλούν αναγκαστικά την

εκτέλεση κάποιων εφαρμογών απομακρυσμένα οπότε είναι πιθανόν να επιβραδύνονται. Ακόμη, ειδικά στην περίπτωση των πειραμάτων με χρήση του μοντέλου τυχαίας επιλογής, ο τρόπος επιλογής των benchmarks που θα μεταφερθεί η μνήμη τους σε κάθε εντολής δεν είναι προκαθορισμένος οπότε δεν μπορούμε να ξέρουμε εκ των προτέρων την απόδοση του καθενός.

Αυτό που μας ενδιαφέρει σε αυτά τα δυναμικά πειράματα είναι η απόδοση ολόκληρου του συστήματος σε κάθε πείραμα που εκτελούμε για να μπορούμε να αξιολογήσουμε το μοντέλο μας. Εδώ παρατηρούμε ότι η απόδοση των πειραμάτων 2, 3 και 4 που βασίζονται στο μοντέλο της τυχαίας επιλογής (Random) είναι 104.7%, 109.9% και 118.8% αντίστοιχα. Αυτό σημαίνει ότι η χρήση αυτού του μοντέλου οδηγεί σε επιβράδυνση του συστήματος κάτα 11% στη μέση περίπτωση σε σχέση με το βέλτιστο μοντέλο μας. Ακόμη, στο πείραμα 5 που εφαρμόζεται το μοντέλο της χείριστης επιλογής (Worst) η απόδοση είναι 125%, δηλαδή υπάρχει επιβράδυνση 25% σε σχέση με το βέλτιστο μοντέλο.

6.3.2 Σενάριο 2

Στο δεύτερο σετ από τα δυναμικά πειράματα επιλέξαμε να τρέξουμε περισσότερες εφαρμογές και σε όλους τους κόμβους του συστήματος Sandman ώστε να γίνει περισσότερο εμφανές το πλεονέκτημα που προσφέρει η χρήση του resource manager στην απόδοση. Πιο αναλυτικά, αποφασίσαμε να εκτελέσουμε την ομάδα των εφαρμογών του πίνακα 6.3 σε κάθε κόμβο του συστήματος, δηλαδή να εκτελούνται ταυτόχρονα τέσσερα στιγμιότυπα κάθε εφαρμογής στους τέσσερεις κόμβους του συστήματος. Ακόμη, για τις μεταφορές σελίδων μνήμης κατά τη διάρκεια αυτών των πειραμάτων τροποποιήσαμε το αρχείο automatic_migration.txt ώστε να συμπεριλαμβάνει 100 εντολές μεταφοράς που εκτελούνται ανά 30 δευτερόλεπτα με πλήθος σελίδων που κυμαίνεται από 1000 έως 400.000 (4MB-1.5GB). Σους πίνακες Α.15 και Α.16 βλέπουμε τις μετρήσεις που λάβαμε για αυτό το σετ πειραμάτων και στη γραφική παράσταση 6.6 οπτικοποιούνται τα αντίστοιχα αποτελέσματα.



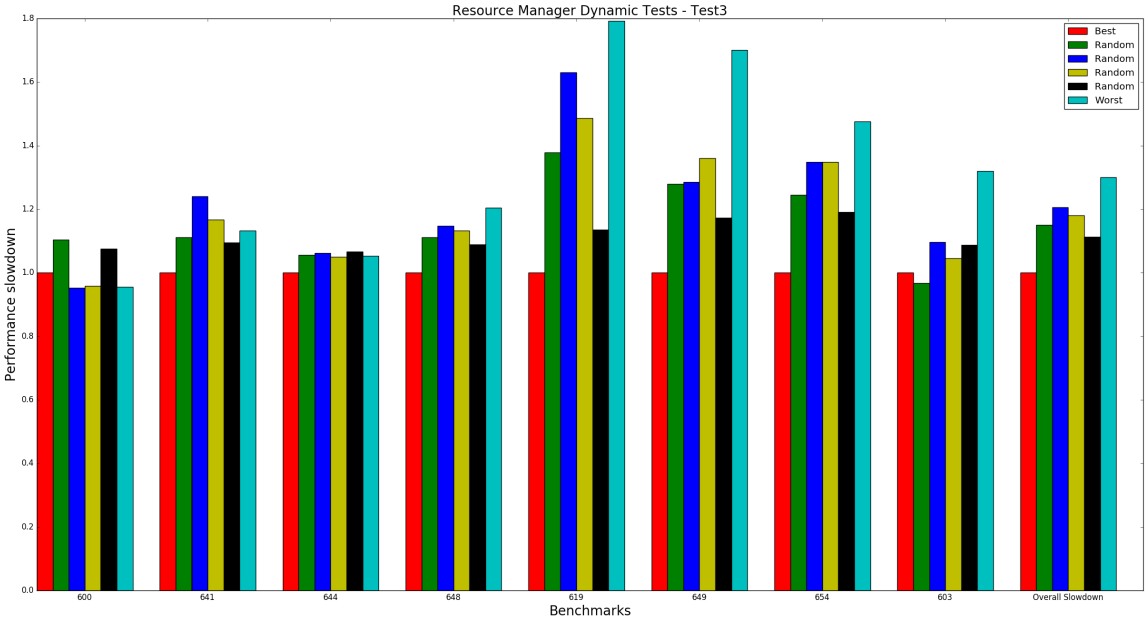
Σχήμα 6.6: Αποτελέσματα 2ου σετ δυναμικών πειραμάτων.

Παρατηρούμε και εδώ ποικιλία ως προς τη μεταβολή της απόδοσης των εφαρμογών σε κάθε πείραμα με την έννοια ότι κάποια επωφελούνται από τις μεταφορές σελίδων μνήμης ενώ άλλα επιβραδύνονται. Αυτό που μας ενδιαφέρει όπως είπαμε είναι η απόδοση ολόκληρου του συστήματος σε κάθε πείραμα, η οποία είναι άρρηκτα συνδεδεμένη με το μοντέλο που εφαρμόζεται. Εδώ παρατηρούμε ότι η απόδοση των πειραμάτων 2, 3 και 4 που βασίζονται στο μοντέλο της τυχαίας επιλογής (Random) είναι 113%, 104.5% και 105.3% αντίστοιχα. Αυτό σημαίνει ότι η χρήση του συγκεκριμένου μοντέλου συνεπάγεται επιβράδυνση του συστήματος περίπου 8% κατά μέσο όρο σε σχέση με το βέλτιστο

μοντέλο μας. Επιπλέον, στο πείραμα 5 που εφαρμόζεται το μοντέλο Worst που κατασκευάτηκε για να επιτυγχάνει τη χειρότερη απόδοση, η απόδοση είναι 116.3% δηλαδή υπάρχει επιβράδυνση 16.3% σε σχέση με το βέλτιστο.

6.3.3 Σενάριο 3

Στο τρίτο σετ των δυναμικών πειραμάτων μας εκτελέσαμε ακριβώς τις ίδιες εφαρμογές και με την ίδια αρχική τοπολογία όπως στο δεύτερο σετ. Αυτό που αλλάξαμε είναι οι εντολές για μεταφορά σελίδων μνήμης κατά της διάρκεια της εκτέλεσης τους. Συγκεκριμένα, το αντίστοιχο αρχείο automatic_migration.txt περιέχει 100 εντολές που εκτελούνται ανά 30 δευτερόλεπτα με πλήθος σελίδων από 4.000 έως 100.000 (16MB-390MB). Ωστόσο, αυτή τη φορά οι μεταφορές συμβαίνουν με τέτοιο τρόπο ώστε να αποκαθίσταται ισορροπία μετά από κάποιες διαδοχικές εντολές. Με άλλα λόγια, αν σε κάποια φάση μεταφερθούν 100 MB από τον κόμβο 0 στον κόμβο 1 και 50 MB από τον κόμβο 3 στον κόμβο 2 μετά από κάποιες εντολές δίνονται η μεταφορά 100 MB από τον κόμβο 1 στον κόμβο 0 και η μεταφορά 50 MB από τον κόμβο 2 στον κόμβο 3 αντίστοιχα. Έτσι, θεωρητικά θα διευρυνθεί το εύρος μεταξύ της καλύτερης και της χειρότερης περίπτωσης, καθώς το μοντέλο μας θα αποκαθιστά την αρχική τοποθέτηση των εφαρμογών στο σύστημα μετά απο κάποιο χρονικό διάστημα, ενώ το δυαδικό του θα επιδιώκει να τις τοποθετεί με όσο το δυνατόν λιγότερο αποδοτικό τρόπο. Στους πίνακες Α.17 και Α.18 βλέπουμε τις μετρήσεις που λάβαμε για αυτό το σετ πειραμάτων και στη γραφική παράσταση 6.6 οπτικοποιούνται τα αντίστοιχα αποτελέσματα.



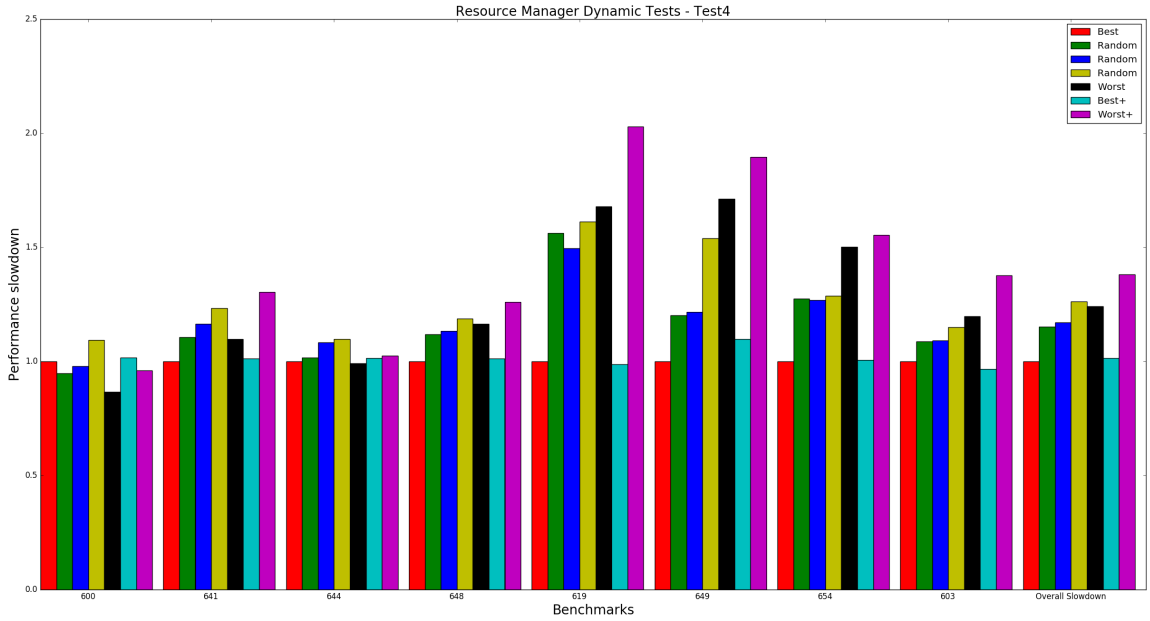
Σχήμα 6.7: Αποτελέσματα 3ου σετ δυναμικών πειραμάτων.

Η παρατήρηση των αποτελεσμάτων που λάβαμε επιβεβαιώνει τη θεωρητική πρόβλεψη που κάναμε σχετικά με το ευρός μεταξύ του βέλτιστου μοντέλου (Best) και του δυαδικού του (Worst). Συγκεκριμένα, το πείραμα 6 στο οποίο εφαρμόζεται το μοντέλο Worst έχει απόδοη 129.5% δηλαδή επιβραδύνει το σύστημα σχεδόν κατά 30% σε σχέση με το βέλτιστο . Όσον αφορά τα πειράματα 2, 3, 4 και 5 στα οποία εφαρμόζεται το μοντέλο της τυχαίας επιλογής (Random) μετρήσαμε αποδόσεις του συστήματος 113.3%, 118.7%, 115.4% και 110.7% αντίστοιχα, δηλαδή το συγκεκριμένο μοντέλο προκαλεί επιβράδυνση στο σύστημα 15% κατά μέσο όρο σε σε σχέση με το βέλτιστο.

6.3.4 Σενάριο 4

Στο τέταρτο σετ των δυναμικών πειραμάτων εκτελέσαμε ξανά τις ίδιες εφαρμογές και με την ίδια αρχική τοποθέτηση όπως και στα προηγούμενα δύο. Σχετικά με τις εντολές μεταφοράς σελίδων

μνήμης ακολουθήσαμε την ίδια λογική με το προηγούμενο σετ πειραμάτων ώστε να γίνει όσο πιο εμφανής γίνεται η διαφορά της απόδοσης μεταξύ των μοντέλων. Ωστόσο, σε αυτό το σετ αυξήσαμε το πλήθος των σελίδων που μεταφέρονται κάθε φορά ώστε να επηρεάζονται περισσότερες εφαρμογές κάθε φορά. Έτσι, τώρα το αρχείο automatic_migration.txt περιέχει πάλι 100 εντολές που εκτελούνται ανά 30 δευτερόλεπτα αλλά οι σελίδες που μεταφέρονται σε κάθε εντολή κυμαίνονται από 1.000.000 έως 3.000.000 (3.8GB-11.4GB). Ακόμη σε αυτό το σετ εκτελέσαμε πειράματα με τα μοντέλα Best+ και Worst+ που κατασκευάσαμε προκειμένου να εξετάσουμε πως συμπεριφέρονται στην πράξη. Στους πίνακες Α.19 και Α.20 βλέπουμε τις μετρήσεις που λάβαμε για αυτό το σετ πειραμάτων και στη γραφική παράσταση 6.8 οπτικοποιούνται τα αντίστοιχα αποτελέσματα.



Σχήμα 6.8: Αποτελέσματα 4ου σετ δυναμικών πειραμάτων.

Και εδώ υπάρχει ποικιλία στον τρόπο που επηρεάζονται οι εφαρμογές μας από κάθε μοντέλο με εξαίρεση το Worst+ (μωβ στήλες) το οποίο τις επιβραδύνει όλες ανεξαιρέτως σε διαφορετικό βαθμό βέβαια. Συγκεκριμένα, το μοντέλο της τυχαίας επιλογής (Random) που εφαρμόζεται στα πειράματα 2, 3 και 4 προκαλεί στο σύστημα αποδόσεις 113.5%, 115.9% και 124.8% αντίστοιχα, δηλαδή κατά μέσο όρο το επιβραδύνει κατά 18% σε σχέση με το βέλτιστο μοντέλο (Best). Όσον αφορά στο μοντέλο Worst που εφαρμόζεται στο πείραμα 5, μετρήσαμε απόδοση 122.6% για το σύστημα. Παρατηρούμε ότι αυτή η τιμή είναι μικρότερη κατά 2% σε σχέση με το πείραμα 4 που εφαρμόζεται το μοντέλο Random, δηλαδή σε αυτή την περίπτωση το τελευταίο φαίνεται να είναι ελάχιστα πιο αργά από το Worst, γεγονός που μπορεί να οφείλεται είτε σε σφάλματα στις μετρήσεις μας είτε σε κάποια επιλογή που έκανε το Random μοντέλο τυχαία και επηρέασε σημαντικά το σύστημα. Τέλος, αναφορικά με το βελτιωμένο μοντέλο μας (Best+) και το δυαδικό του (Worst+) οι αποδόσεις που προκαλούν είναι 101.1% και 138.1% αντίστοιχα. Αυτό σημαίνει ότι στο συγκεκριμένο σετ πειραμάτων το μοντέλο Best+ δεν επιτυγχάνει καλύτερη απόδοση και είναι εφάμιλλο με το Best, ενώ στην περίπτωση του Worst+ υπάρχει σημαντική διαφορά, καθώς επιβραδύνει το σύστημα κατά 38.1% δηλαδή προκαλεί επιπρόσθετη μείωση της απόδοσης της τάξης του 15.5% σε σχέση με το Worst.

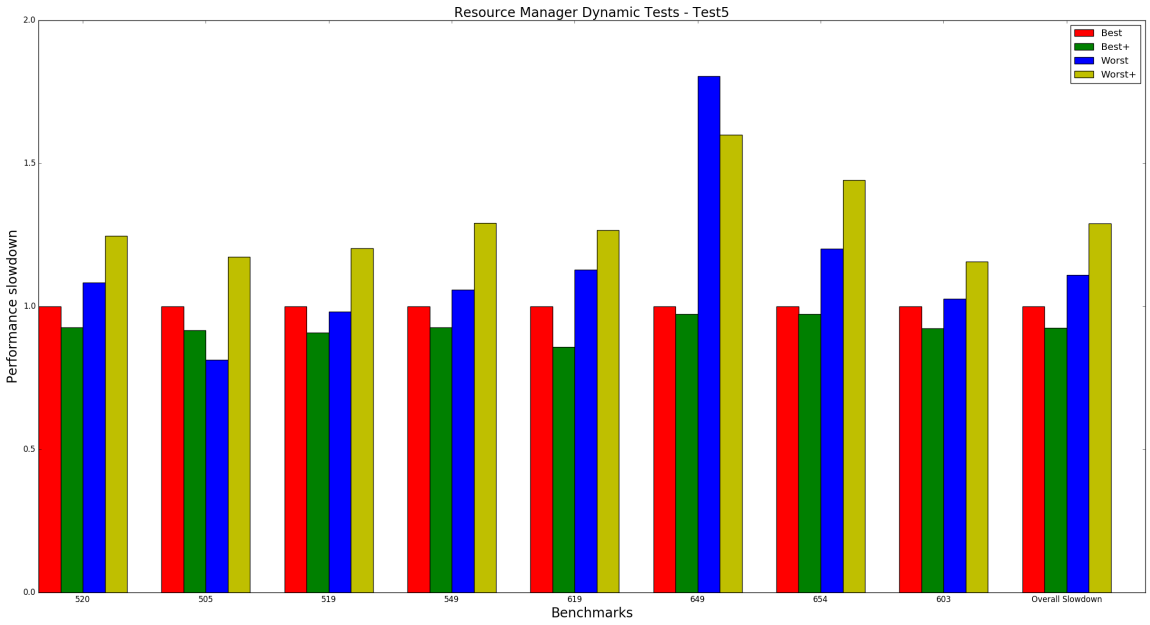
6.3.5 Σενάριο 5

Στο πέμπτο και τελευταίο σετ από τα δυναμικά πειράματα εκτελέσαμε τα benchmarks του πίνακα 6.4. Πρόκειται ουσιαστικά για τις δύο τετράδες των benchmarks που επηρεάζονται από την τοπολογία από τους πίνακες 6.1 και 6.3. Συγκεκριμένα, εκτελέσαμε τέσσερα στιγμιότυπα για κάθε benchmark που τρέχουν σε διαφορετικό κόμβο του συστήματος Sandman το καθένα. Το αρχείο automatic_migration.txt

έμεινε αμετάβλητο σε σχέση με το προηγούμενο σετ πειραμάτων, δηλαδή εκτελέσαμε τις ίδιες εντολές μεταφοράς σελίδων μνήμης με πριν. Αναφορικά με τα μοντέλα, στόχος μας εδώ ήταν αποκλειστικά να δούμε τη εύρος της απόδοσης μεταξύ των καλύτερων και χειρότερων μοντέλων οπότε παραλείψαμε το μοντέλο Random και εκτελέσαμε με τη σειρά πειράματα με χρήση των μοντέλων Best, Best+, Worst και Worst+ αντίστοιχα. Στους πίνακες Α.21 και Α.22 βλέπουμε τις μετρήσεις που λάβαμε για αυτό το σετ πειραμάτων και στη γραφική παράσταση 6.9 οπτικοποιούνται τα αντίστοιχα αποτελέσματα.

Benchmarks	**Ευαισθησία στην τοπολογία**
520.omnetpp_r	high
505.mcf_r	high
519.lbm_r	high
549.fotonik3d_r	high
619.lbm_s	high
649.fotonik3d_s	high
654.roms_s	high
603.bwaves_s	high

Πίνακας 6.4: Benchmarks που εκτελέσαμε στο πέμπτο σετ δυναμικών πειραμάτων



Σχήμα 6.9: Αποτελέσματα 5ου σετ δυναμικών πειραμάτων.

Με τα αποτελέσματα αποκτάμε μια καλύτερη εικόνα για τη συγκριτική απόδοση των τεσσάρων μοντέλων που χρησιμοποιούν κάποια λογική για να καθορίσουν τη σειρά εξέτασης των benchmarks. Πιο αναλυτικά, παρατηρούμε ότι το μοντέλο Best+ (πράσινες στήλες) επιτυγχάνει καλύτερη απόδοση για όλα τις εφαρμογές (με εξαίρεση δύο) σε σχέση με το μοντέλο Best, ενώ αντίστοιχα το μοντέλο Worst+ (κίτρινες στήλες) επιτυγχάνει χειρότερη απόδοση σε σχέση με το μοντέλο Worst σχεδόν για όλες τις εφαρμογές. Αυτό σημαίνει ότι η τροποποίηση που κάναμε σε στα μοντέλα Best+/Worst+ έχει αντίκτυπο στην πράξη και κρίνεται επιτυχημένη. Όσον αφορά στην απόδοση του συστήματος συνολικά, το μοντέλο Best+ που εφαρμόζεται στο πείραμα 2 επιταχύνει το σύστημα κατά 8% σε σχέση με το Best. Ακόμη, το μοντέλο Worst οδηγεί σε απόδοση 110% δηλαδή επιβραδύνει το σύστημα κατά 10% σε σχέση με το Best, ενώ το μοντέλο Worst+ προκαλεί απόδοση σχεδόν 128%. Η διαφορά, λοιπόν, μεταξύ καλύτερου και χειρότερου μοντέλου σε αυτό το σετ πειραμάτων είναι 36% η οποία

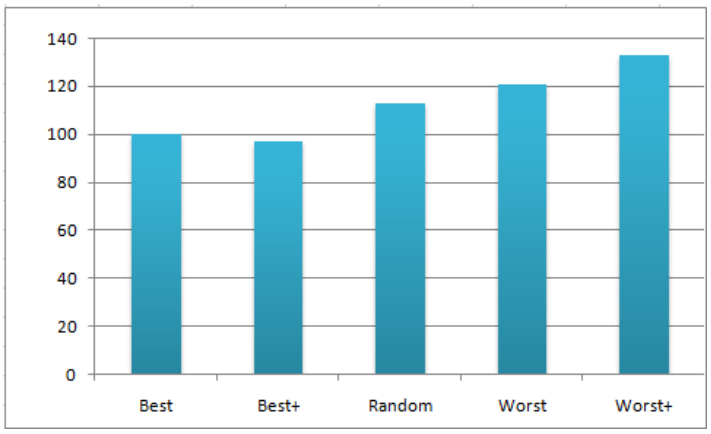
	Σετ πειραμάτων					
Μοντέλο	**1**	**2**	**3**	**4**	**5**	**Μ.Ο.**
Best	1	1	1	1	1	1
Random	1.11	1.08	1.15	1.18	-	1.13
Worst	1.25	1.16	1.3	1.23	1.1	1.21
Best+	-	-	-	1.01	0.92	0.97
Worst+	-	-	-	1.38	1.28	1.33
Range (%)	25	16	30	38	39	37

Πίνακας 6.5: Συγκεντρωτικά αποτελέσματα για τις αποδόσεις των μοντέλων.

κρίνεται πλήρως ικανοποιητική και δικαιολογεί τη χρήση του Resource Manager ως προς το σκοπό για τον οποίο σχεδιάστηκε.

6.4 Συμπεράσματα

Στον Πίνακα 6.5 φαίνονται συνοπτικά οι αποδόσεις των μοντέλων σε κάθε πείραμα που εκτελέσαμε. Όλες οι αποδόσεις είναι κανονικοποιημένες ως προς την απόδοση του μοντέλου Best στο αντίστοιχο πείραμα την οποία θεωρούμε ίση με τη μονάδα. Η απόδοση του μοντέλου Random για κάθε σετ πειραμάτων υπολογίζεται ως ο μέσος όρος των αποδόσεων από τα πειράματα που βασίστηκαν σε αυτό. Από τη δεξιά στήλη παρατηρούμε ότι το μοντέλο Best+ εμφανίζει κατά μέσο όρο την καλύτερη απόδοση που είναι 3% καλύτερη από αυτή του Best, ενώ το μοντέλο Worst+ εμφανίζει τη χειρότερη απόδοση επιβραδύνοντας το σύστημα 33% κατά μέσο όρο σε σχέση με το μοντέλο Best. Η απόδοση του μοντέλου Random βρίσκεται σχεδόν πάντα μεταξύ του εύρους που ορίζουν οι αποδόσεις των μοντέλων Best+ και Worst+ αντίστοιχα. Τέλος, στην τελευταία γραμμή καταγράφουμε για κάθε σετ πειραμάτων το εύρος μεταξύ καλύτερης και χειρότερης απόδοσης ως ποσοστό ποσοστό της καλύτερης. Η αναπαράσταση των αποτελεσμάτων αυτών φαίνεται στο σχήμα 6.10. Κατά μέσο όρο βλέπουμε ότι το καλύτερο μοντέλο μας βελτιώνει την απόδοση του συστήματος κατά 16% στη μέση περίπτωση και κατά 37% στη χειρότερη περίπτωση. Επομένως, η χρήση του Resource Manager με το μοντέλο Best+ είναι ιδιαίτερα σημαντική και ενδείκνυται για τη διαφάλιση της αποδοτικής αξιοποίησης των πόρων των NUMA συστημάτων, καθώς επιταχύνει την εκτέλεση των εφαρμογών χωρίς να εισάγει σημαντική πολυπλοκότητα.



Σχήμα 6.10: Συγκριτική αξιολόγηση των μοντέλων του Resource Manager.

Κεφάλαιο 7

Επίλογος

Με το κεφάλαιο αυτό ολοκληρώνεται η παρούσα διπλωματική εργασία. Θα παρουσιάσουμε κάποια πιο γενικά συμπεράσματα στα οποία καταλήξαμε μέσα από τη μελέτη μας και θα αναφερθούμε σε πιθανές μελλοντικές επεκτάσεις που παρουσιάζουν ενδιαφέρον.

7.1 Συμπεράσματα

Στο πρώτο κεφάλαιο είχαμε αναφερθεί στους στόχους της διπλωματικής. Ας τους θυμηθούμε πάλι και ας εξετάσαμε το βαθμό στον οποίο τους επιτύχαμε.

- Μελέτη των benchmarks της σουίτας SPEC 2017 και εκτέλεσή τους σε διάφορα σενάρια και τοπολογίες

  Στο κεφάλαιο 4 ορίσαμε τις βασικές μετρικές για τη μελέτη των εφαρμογών (IPC, MPKI, TLB MPKI, BW) και στη συνέχεια τις εκτελέσαμε και συγκεντρώσαμε μετρήσεις από αυτά με χρήση των εργαλείων της ενότητας 3.2. Κατόπιν, προσπαθήσαμε να αναζητήσουμε συσχετίσεις μεταξύ των τιμών των παραμέτρων και της απόδοσης των εφαρμογών στις διάφορες τοπολογίες (βλ. διαγράμματα 4.4 και 4.5) και καταλήξαμε σε κάποια συμπεράσματα.

- Δημιουργία ενός μοντέλου βελτίωσης της απόδοσης του συστήματος Sandman όταν τρέχουν ταυτόχρονα πολλές εφαρμογές

  Στο κεφάλαιο 5 παρουσιάσαμε ένα μοντέλο που αναπτύξαμε με βάση τα συμπεράσματα από την διεξαγωγή των πειραμάτων. Το μοντέλο αυτό στοχεύει στη βελτίωση της απόδοσης ενός NUMA συστήματος όταν σε αυτό τρέχουν πολλές εφαρμογές ταυτόχρονα και πρέπει για εξωτερικούς λόγους να εκτελεστούν μεταφορές σελίδων μνήμης μεταξύ των κόμβων.

- Υλοποίηση και αξιολόγηση ενός προγράμματος επιπέδου χρήστη, του resource manager, που επιτελεί το σκοπό αυτό

  Στο κεφάλαιο 6 αξιολογούμε τον resource manager, ένα πρόγραμμα επιπέδου χρήστη που υλοποιεί το μοντέλο μας. Συγκρίναμε την απόδοσή του με άλλα μοντέλα και διαπιστώσαμε το συγκριτικό πλεονέκτημα που παρέχει η χρήση του στην καλύτερη αξιοποίηση των πόρων του συστήματος.

7.2 Μελλοντικές επεκτάσεις

Ολοκληρόνωντας τη διπλωματική εργασία ας επισημάνουμε ορισμένες μελλοντικές επεκτάσεις που μας κίνησαν το ενδιαφέρον:

- Μελέτη του φαινομένου **stress effect** που περιγράψαμε αναλυτικά στην υποενότητα 4.3.5 και σχετίζεται με τη μεταβολή της απόδοσης ορισμένων εφαρμογών σε απομακρυσμένες τοπολογίες σε τέτοιο βαθμό ώστε να τρέχουν πιο γρήγορα σε σχέση αντίστοιχη τοπική τοπολογία.

Αυτή η παρατήρηση ανατρέπει την καθιερωμένη αντίληψη ότι η τοπική εκτέλεση των εφαρμογών είναι πάντα η ταχύτερη και αποδοτικότερη και μπορεί να αξιοποιηθεί για τη δημιουργία ενός πιο εξελιγμένου δυναμικού μοντέλου.

- Βελτίωση του μοντέλου που υλοποιήσαμε ώστε να λαμβάνει υπόψιν περισσότερους performance counters και να χρησιμοποιεί πιο εξελιγμένες τεχνικές για την επιλογή των εφαρμογών των οποίων οι σελίδες θα μεταφερθούν ώστε να ικανοποιηθεί ένα αντίστοιχο αίτημα μεταφοράς μνήμης μεταξύ των κόμβων ενός NUMA συστήματος.
- Αξιοποίηση του resource manager και ένταξή του σε ένα πιο ολοκληρωμένο περιβάλλον διαχείρισης των εφαρμογών που τρέχουν σε ένα NUMA σύστημα το οποίο θα υπολογίζει τη βέλτιστη αρχική τοποθέτησή τους και θα πυροδοτεί ενέργειες δυναμικά ώστε να επιτυγχάνεται όσο το δυνατόν καλύτερη απόδοση.

Βιβλιογραφία

[Arap18] Fanourios Arapidis, Vasileios Karakostas, Nikela Papadopoulou, Konstantinos Nikas, Georgios I. Goumas and Nectarios Koziris, "Performance Prediction of NUMA Placement: A Machine-Learning Approach", in *2018 IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2018, Nicosia, Cyprus, December 10-13, 2018*, pp. 296–301, 2018.

[Buce18] James Bucek, Klaus-Dieter Lange and Jóakim v. Kistowski, "SPEC CPU2017: Next-Generation Compute Benchmark", in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ICPE '18, p. 41–42, New York, NY, USA, 2018, Association for Computing Machinery.

[Dash13] Mohammad Dashti, Alexandra Fedorova, Justin R. Funston, Fabien Gaud, Renaud Lachaize, Baptiste Lepers, Vivien Quéma and Mark Roth, "Traffic management: a holistic approach to memory placement on NUMA systems", in Vivek Sarkar and Rastislav Bodík, editors, *Architectural Support for Programming Languages and Operating Systems, ASPLOS '13, Houston, TX, USA - March 16 - 20, 2013*, pp. 381–394, ACM, 2013.

[DGur20] R. Karimi J. Barreto P. Bhatotia V. Quema R. Rodrigues P. Romano D. Gureya, J. Neto and V. Vlassov, "Bandwidth-Aware Page Placement in NUMA Systems", in *Proceedings of 34th IEEE International Parallel Distributed Processing Symposium*, IPDPS, 2020.

[Funs18] Justin R. Funston, Maxime Lorrillere, Alexandra Fedorova, Baptiste Lepers, David Vengerov, Jean-Pierre Lozi and Vivien Quéma, "Placement of Virtual Containers on NUMA systems: A Practical and Comprehensive Model", in Haryadi S. Gunawi and Benjamin Reed, editors, *2018 USENIX Annual Technical Conference, USENIX ATC 2018, Boston, MA, USA, July 11-13, 2018*, pp. 281–294, USENIX Association, 2018.

[Gaud15] Fabien Gaud, Baptiste Lepers, Justin R. Funston, Mohammad Dashti, Alexandra Fedorova, Vivien Quéma, Renaud Lachaize and Mark Roth, "Challenges of memory management on modern NUMA systems", *Commun. ACM*, vol. 58, no. 12, pp. 59–66, 2015.

[inte] "Intel Performance Counter Monitor - A better way to measure CPU utilization. https://software.intel.com/en-us/articles/intel-performance-counter-monitor".

[Kotr17] Jagadish B. Kotra, Seongbeom Kim, Kamesh Madduri and Mahmut T. Kandemir, "Congestion-aware memory management on NUMA platforms: A VMware ESXi case study.", in *IISWC*, pp. 146–155, IEEE Computer Society, 2017.

[Lame13] Christoph Lameter, "An Overview of Non-Uniform Memory Access", *Commun. ACM*, vol. 56, no. 9, p. 59–54, September 2013.

[Lepe15] Baptiste Lepers, Vivien Quéma and Alexandra Fedorova, "Thread and Memory Placement on NUMA Systems: Asymmetry Matters", in *Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '15, p. 277–289, USA, 2015, USENIX Association.

[Luo16] H. Luo, J. Brock, Pengcheng Li, C. Ding and Chencheng Ye, "Compositional model of coherence and NUMA effects for optimizing thread and data placement", in *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 151–152, 2016.

[Majo11] Zoltan Majo and Thomas R. Gross, "Memory System Performance in a NUMA Multicore Multiprocessor", in *Proceedings of the 4th Annual International Conference on Systems and Storage*, SYSTOR '11, New York, NY, USA, 2011, Association for Computing Machinery.

[McCo11] Patrick S McCormick, Ryan Karl Braithwaite and Wu-chun Feng, "Empirical Memory-Access Cost Models in Multicore NUMA Architectures", 1 2011.

[migr] "migratepages - Migrate the physical location a processes pages. https://linux.die.net/man/8/migratepages".

[numa] "numactl - Control NUMA policy for processes or shared memory. https://linux.die.net/man/8/numactl".

[perf] "perf: Linux profiling with performance counters. https://perf.wiki.kernel.org/index.php/MainPage".

[Qian19] Jianmin Qian, Jian Li, Ruhui Ma and Haibing Guan, "vDARM: Dynamic Adaptive Resource Management for Virtualized Multiprocessor Systems", in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2019, Florence, Italy, March 25-29, 2019*, pp. 658–661, 2019.

[stre] "stress-ng - a tool to load and stress a computer system. https://manpages.ubuntu.com/manpages/artful/man1/stress-ng.1.html".

[Su12] ChunYi Su, Dong Li, Dimitrios S. Nikolopoulos, Kirk W. Cameron, Bronis R. de Supinski and Edgar A. Leon, "Model-Based, Memory-Centric Performance and Power Optimization on NUMA Multiprocessors", in *Proceedings of the 2012 IEEE International Symposium on Workload Characterization (IISWC)*, IISWC '12, p. 164–173, USA, 2012, IEEE Computer Society.

[task] "taskset - set or retrieve a process's CPU affinity. https://linux.die.net/man/1/taskset".

[Wang16] W. Wang, J. W. Davidson and M. L. Soffa, "Predicting the memory bandwidth and optimal core allocations for multi-threaded applications on large-scale NUMA machines", in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 419–431, 2016.

Παράρτημα Α

Πίνακες με τα πειραματικά αποτελέσματα

Σε αυτό το παράρτημα της εργασίας θα παρουσιάσουμε τα αποτελέσματα που λάβαμε από τις διάφορες εκτελέσεις των benchmarks στο μηχάνημα Sandman όπως περιγράφονται αναλυτικά στην ενότητα 4.3. Συγκεντρωτικά τα σετ από πειράματα που εκτελέσαμε στο στάδιο αυτό είναι πέντε και φαίνονται στον πίνακα Α.1.

Όνομα Test	**Αποτελέσματα**	**Περιγραφή**
Scenario 1	Α.2	Τα benchmarks τρέχουν με cpu από τον κόμβο 0 στις τοπολογίες 0, 1 και 2
Scenario 2	Α.3	Τα benchmarks που επηρεάζεται η απόδοσή τους στις τοπολογίες 1 και 2 σε σχέση με την τοπολογία 0 τρέχουν με cpu από τον κόμβο 0 στις τοπολογίες 1 και 2 ενώ ταυτόχρονα στον κόμβο 1 τρέχει τοπικά ένα στιγμιότυπο του εργαλείου stress-ng
Scenario 3	Α.4 και Α.5	Τα benchmarks τρέχουν με cpu από τον κόμβο 0 στις τοπολογίες 0, 1 και 2 ενώ ταυτόχρονα στον κόμβο 0 τρέχει τοπικά ένα στιγμιότυπο του εργαλείου stress-ng
Scenario 4	Α.6	Όπως το Scenario 2 με τη διαφορά ότι στον κόμβο 1 τρέχουν τοπικά 10 στιγμιότυπα του εργαλείου stress-ng
Scenario 5	Α.7 και Α.8	Όπως το Scenario 3 με τη διαφορά ότι στον κόμβο 0 τρέχουν τοπικά 10 στιγμιότυπα του εργαλείου stress-ng

Πίνακας Α.1: Σύνοψη πειραμάτων που εκτελέσαμε στα benchmarks της σουίτας SPEC 2017

Ακόμη, θα παρουσιάσουμε τις μετρήσεις μας από τα στατικά και δυναμικά πειράματα που εκτελέσαμε στα στάδια της υλοποίησης και αξιολόγησης του resource manager όπως αυτά περιγράφονται στο κεφάλαιο 6.

Performance %					
500s series			600s series		
Benchmark	**Topology 1**	**Topology 2**	**Benchmark**	**Topology 1**	**Topology 2**
500.perlbench_r_0	100.5	101.7	600.perlbench_s_0	100.1	99.7
500.perlbench_r_1	102.0	101.7	600.perlbench_s_1	101.4	99.4
500.perlbench_r_2	133.2	117.6	600.perlbench_s_2	143.9	123.8
502.gcc_r_0	118.4	109.3	602.gcc_s_0	227.0	162.8
502.gcc_r_1	122.5	111.2	602.gcc_s_1	112.3	105.1
502.gcc_r_2	120.2	111.1	602.gcc_s_2	114.9	107.7
502.gcc_r_3	185.8	143.4	603.bwaves_s_0	280.7	164.8
502.gcc_r_4	160.4	130.7	603.bwaves_s_1	269.2	160.3
503.bwaves_r_0	150.9	124.4	605.mcf_s	181.3	143.2
503.bwaves_r_1	142.8	120.6	607.cactuBSSN_s	239.3	144.3
503.bwaves_r_2	140.4	111.5	619.lbm_s	398.8	173.2
503.bwaves_r_3	141.3	110.8	620.omnetpp_s	194.6	151.8
505.mcf_r	177.2	138.9	621.wrf_s	150.0	112.0
507.cactuBSSN_r	141.3	124.8	623.xalancbmk_s	111.0	106.3
508.namd_r	102.1	102.1	625.x264_s_0	104.8	102.3
510.parest_r	105.3	102.4	625.x264_s_1	102.7	101.4
519.lbm_r	199.9	148.1	625.x264_s_2	101.5	100.9
520.omnetpp_r	196.8	150.2	628.pop2_s	137.1	111.5
521.wrf_r	106.0	103.0	631.deepsjeng_s	123.3	112.1
523.xalancbmk_r	110.6	105.1	638.imagick_s	103.2	101.0
525.x264_r_0	105.0	101.6	641.leela_s	99.7	100.7
525.x264_r_1	102.3	101.5	644.nab_s	100.2	99.4
525.x264$_r$_2	102.0	101.4	648.exchange2_s	99.8	99.7
526.blender_r	109.3	105.9	649.fotonik3d_s	393.3	182.1
527.cam4_r	105.7	102.4	654.roms_s	376.3	194.2
531.deepsjeng_r	113.3	106.7	657.xz_s_0	148.8	115.9
538.imagick_r	100.5	110.0	657.xz_s_1	270.0	162.6
541.leela_r	100.3	100.3			
544.nab_r	100.4	100.1			
548.exchange2_r	101.4	101.7			
549.fotonik3d_r	255.3	165.8			
554.roms_r	144.0	120.2			
557.xz_r_0	142.3	122.4			
557.xz_r_1	106.4	103.1			
557.xz_r_2	119.7	111.4			

Πίνακας Α.2: Scenario 1: Αποδόσεις των benchmarks στις τοπολογίες remote και interleave σε σχέση με τη baseline

Performance %					
500s series			600s series		
Benchmark	**Topology 1**	**Topology 2**	**Benchmark**	**Topology 1**	**Topology 2**
500.perlbench_r_0	100.4	99.9	600.perlbench_s_0	100.2	99.3
500.perlbench_r_1	101.2	100.9	600.perlbench_s_1	100.4	98.8
500.perlbench_r_2	118.2	101.2	600.perlbench_s_2	140.5	119.3
502.gcc_r_0	118.0	109.0	602.gcc_s_0	214.5	155.9
502.gcc_r_1	120.5	110.4	602.gcc_s_1	108.4	103.1
502.gcc_r_2	119.7	110.3	602.gcc_s_2	111.7	104.4
502.gcc_r_3	172.5	128.8	603.bwaves_s_0	290.6	169.4
502.gcc_r_4	145.7	115.5	603.bwaves_s_1	279.2	164.5
503.bwaves_r_0	151.4	124.3	605.mcf_s	176.3	138.1
503.bwaves_r_1	132.2	107.4	607.cactuBSSN_s	250.3	149.3
503.bwaves_r_2	129.7	97.7	619.lbm_s	412.2	188.5
503.bwaves_r_3	130.2	97.9	620.omnetpp_s	192.2	146.9
505.mcf_r	173.9	136.8	621.wrf_s	153.5	113.1
507.cactuBSSN_r	140.5	122.2	623.xalancbmk_s	108.0	101.2
519.lbm_r	205.1	150.2	628.pop2_s	139.7	111.9
520.omnetpp_r	195.2	150.0	631.deepsjeng_s	120.8	108.7
521.wrf_r	105.7	102.7	649.fotonik3d_s	414.8	194.3
523.xalancbmk_r	108.8	104.6	654.roms_s	392.3	203.9
525.x264_r_0	108.4	101.6	657.xz_s_0	152.5	115.5
525.x264_r_1	101.0	100.8	657.xz_s_1	282.0	168.7
525.x264_r_2	98.9	97.6			
526.blender_r	104.8	100.7			
531.deepsjeng_r	112.2	106.1			
538.imagick_r	100.5	100.0			
549.fotonik3d_r	262.8	169.1			
554.roms_r	144.0	119.5			
557.xz_r_0	141.7	121.1			
557.xz_r_1	102.3	98.7			
557.xz_r_2	115.1	106.2			

Πίνακας Α.3: Scenario 2: Αποδόσεις των benchmarks στις τοπολογίες remote και interleave σε σχέση με τη baseline

Performance %					
500s series			600s series		
Benchmark	**Topology 1**	**Topology 2**	**Benchmark**	**Topology 1**	**Topology 2**
500.perlbench_r_0	125.4	112.4	600.perlbench_s_0	125.4	112.0
500.perlbench_r_1	103.6	101.7	600.perlbench_s_1	103.8	101.6
500.perlbench_r_2	178.9	139.4	600.perlbench_s_2	174.9	137.0
502.gcc_r_0	152.3	123.8	602.gcc_s_0	204.2	149.2
502.gcc_r_1	167.6	133.0	602.gcc_s_1	150.4	123.2
502.gcc_r_2	192.6	131.7	602.gcc_s_2	158.6	127.2
502.gcc_r_3	198.2	145.5	603.bwaves_s_0	253.9	153.5
502.gcc_r_4	174.7	134.4	603.bwaves_s_1	244.2	149.6
503.bwaves_r_0	138.4	117.5	605.mcf_s	216.2	144.4
503.bwaves_r_1	142.2	118.9	607.cactuBSSN_s	261.3	140.4
503.bwaves_r_2	140.3	109.3	619.lbm_s	298.4	140.7
503.bwaves_r_3	140.0	109.6	620.omnetpp_s	209.8	153.0
505.mcf_r	178.9	132.6	621.wrf_s	112.9	102.5
507.cactuBSSN_r	136.8	119.1	623.xalancbmk_s	145.0	122.1
508.namd_r	104.9	102.4	625.x264_s_0	107.7	102.6
510.parest_r	190.3	130.2	625.x264_s_1	105.9	103.2
519.lbm_r	156.9	118.4	625.x264_s_2	105.1	102.9
520.omnetpp_r	209.6	157.0	628.pop2_s	137.5	108.7
521.wrf_r	110.1	104.0	631.deepsjeng_s	127.9	113.7
523.xalancbmk_r	146.5	121.1	638.imagick_s	103.8	100.8
525.x264_r_0	109.3	104.9	641.leela_s	108.8	104.2
525.x264_r_1	105.0	102.8	644.nab_s	101.1	100.6
525.x264$_r$_2	104.6	102.5	648.exchange2_s	102.0	99.4
526.blender_r	150.1	123.1	649.fotonik3d_s	323.8	159.1
527.cam4_r	121.9	109.9	654.roms_s	354.2	183.2
531.deepsjeng_r	124.2	112.0	657.xz_s_0	153.6	111.6
538.imagick_r	101.1	100.0	657.xz_s_1	225.0	130.6
541.leela_r	108.7	104.2			
544.nab_r	102.1	101.2			
548.exchange2_r	100.5	100.5			
549.fotonik3d_r	203.8	129.2			
554.roms_r	150.0	116.5			
557.xz_r_0	162.5	131.6			
557.xz_r_1	139.0	119.5			
557.xz_r_2	144.4	122.6			

Πίνακας Α.4: Scenario 3: Αποδόσεις των benchmarks στις τοπολογίες remote και interleave σε σχέση με τη local

Performance %							
500s series				600s series			
Benchmark	**Topology 0**	**Topology 1**	**Topology 2**	**Benchmark**	**Topology 0**	**Topology 1**	**Topology 2**
500.perlbench_r_0	130.7	163.9	146.8	600.perlbench_s_0	130.3	163.4	145.9
500.perlbench_r_1	103.3	107.0	105.1	600.perlbench_s_1	103.7	107.6	105.3
500.perlbench_r_2	136.8	244.6	190.6	600.perlbench_s_2	162.0	283.4	222.0
502.gcc_r_0	159.2	242.4	197.1	602.gcc_s_0	162.1	331.0	241.9
502.gcc_r_1	137.0	229.7	182.2	602.gcc_s_1	149.3	224.7	184.0
502.gcc_r_2	128.9	248.3	169.9	602.gcc_s_2	153.3	243.1	195.0
502.gcc_r_3	130.0	257.7	189.2	603.bwaves_s_0	112.7	286.3	173.0
502.gcc_r_4	120.0	209.6	161.3	603.bwaves_s_1	112.7	275.1	168.6
503.bwaves_r_0	123.9	171.5	145.6	605.mcf_s	136.1	294.3	196.5
503.bwaves_r_1	105.0	149.3	124.9	607.cactuBSSN_s	136.6	357.0	191.9
503.bwaves_r_2	104.6	146.7	114.3	619.lbm_s	121.4	362.4	170.9
503.bwaves_r_3	105.1	147.1	115.2	620.omnetpp_s	178.3	374.1	272.7
505.mcf_r	132.3	236.7	175.4	621.wrf_s	551.5	622.6	565.2
507.cactuBSSN_r	133.9	183.1	159.5	623.xalancbmk_s	136.2	197.5	166.3
508.namd_r	104.1	109.2	106.6	625.x264_s_0	107.9	116.2	110.8
510.parest_r	119.0	226.4	155.0	625.x264_s_1	103.0	109.0	106.3
519.lbm_r	164.7	258.3	195.0	625.x264_s_2	101.7	106.9	104.6
520.omnetpp_r	181.3	379.9	284.6	628.pop2_s	112.2	154.3	122.0
521.wrf_r	109.0	120.0	113.3	631.deepsjeng_s	116.9	149.5	133.0
523.xalancbmk_r	139.1	203.7	168.5	638.imagick_s	108.0	112.1	108.8
525.x264_r_0	122.8	134.2	128.8	641.leela_s	112.1	121.9	116.7
525.x264_r_1	103.2	108.4	106.1	644.nab_s	109.8	111.0	110.4
525.x264$_r$_2	118.1	123.5	121.0	648.exchange2_s	102.6	104.6	102.0
526.blender_r	128.2	192.5	157.8	649.fotonik3d_s	116.2	376.3	185.0
527.cam4_r	120.1	146.3	132.0	654.roms_s	112.6	398.7	206.2
531.deepsjeng_r	120.8	150.1	135.3	657.xz_s_0	125.8	193.2	140.4
538.imagick_r	122.3	123.7	122.3	657.xz_s_1	150.8	339.3	197.0
541.leela_r	127.5	138.6	132.9				
544.nab_r	120.1	122.6	121.5				
548.exchange2_r	123.5	124.1	124.1				
549.fotonik3d_r	141.3	288.0	182.6				
554.roms_r	119.1	178.6	138.8				
557.xz_r_0	165.0	268.2	217.2				
557.xz_r_1	129.2	179.6	154.4				
557.xz_r_2	147.0	212.4	180.3				

Πίνακας A.5: Scenario 3: Αποδόσεις των benchmarks στις τοπολογίες local, remote και interleave σε σχέση με τη baseline

Performance %					
500s series			600s series		
Benchmark	**Topology 1**	**Topology 2**	**Benchmark**	**Topology 1**	**Topology 2**
500.perlbench_r_0	101.2	100.2	600.perlbench_s_0	101.2	101.1
500.perlbench_r_1	103.6	101.4	600.perlbench_s_1	104.9	100.7
500.perlbench_r_2	137.8	112.9	600.perlbench_s_2	165.2	133.9
502.gcc_r_0	131.0	116.1	602.gcc_s_0	278.8	188.7
502.gcc_r_1	134.7	118.4	602.gcc_s_1	118.1	109.4
502.gcc_r_2	133.3	118.0	602.gcc_s_2	123.0	110.5
502.gcc_r_3	221.7	154.7	603.bwaves_s_0	370.7	207.6
502.gcc_r_4	179.3	133.9	603.bwaves_s_1	354.4	200.4
503.bwaves_r_0	181.2	140.3	605.mcf_s	216.5	160.8
503.bwaves_r_1	160.9	122.7	607.cactuBSSN_s	324.5	184.5
503.bwaves_r_2	156.6	110.3	619.lbm_s	493.4	237.3
503.bwaves_r_3	157.8	110.0	620.omnetpp_s	250.4	178.8
505.mcf_r	214.9	160.2	621.wrf_s	175.9	124.1
507.cactuBSSN_r	170.9	136.7	623.xalancbmk_s	111.4	105.3
519.lbm_r	288.3	179.6	628.pop2_s	137.3	111.7
520.omnetpp_r	258.6	185.5	631.deepsjeng_s	122.0	110.0
521.wrf_r	112.2	107.3	649.fotonik3d_s	393.4	182.0
523.xalancbmk_r	117.0	111.0	654.roms_s	376.2	193.7
525.x264_r_0	110.0	105.2	657.xz_s_0	147.2	113.9
525.x264_r_1	102.5	101.3	657.xz_s_1	269.6	162.0
525.x264_r_2	102.4	99.7			
526.blender_r	109.7	103.2			
531.deepsjeng_r	124.9	113.4			
538.imagick_r	111.2	101.3			
549.fotonik3d_r	354.8	217.8			
554.roms_r	174.4	136.2			
557.xz_r_0	184.5	144.3			
557.xz_r_1	110.4	103.0			
557.xz_r_2	135.1	118.1			

Πίνακας Α.6: Scenario 4: Αποδόσεις των benchmarks στις τοπολογίες remote και interleave σε σχέση με τη baseline

Performance %					
500s series			600s series		
Benchmark	**Topology 1**	**Topology 2**	**Benchmark**	**Topology 1**	**Topology 2**
500.perlbench_r_0	83.0	89.6	600.perlbench_s_0	82.1	90.3
500.perlbench_r_1	99.5	100.2	600.perlbench_s_1	96.8	99.8
500.perlbench_r_2	75.4	90.7	600.perlbench_s_2	74.1	87.5
502.gcc_r_0	71.7	91.2	602.gcc_s_0	59.3	83.1
502.gcc_r_1	71.7	85.8	602.gcc_s_1	70.7	84.7
502.gcc_r_2	66.1	83.5	602.gcc_s_2	69.5	83.3
502.gcc_r_3	68.1	87.8	603.bwaves_s_0	157.9	109.9
502.gcc_r_4	54.4	78.2	603.bwaves_s_1	153.7	108.2
503.bwaves_r_0	78.4	87.4	605.mcf_s	59.4	84.3
503.bwaves_r_1	76.7	88.0	607.cactuBSSN_s	112.2	90.4
503.bwaves_r_2	75.2	80.8	619.lbm_s	152.0	86.1
503.bwaves_r_3	78.8	87.8	620.omnetpp_s	60.3	83.6
505.mcf_r	63.7	87.7	621.wrf_s	97.9	97.5
507.cactuBSSN_r	72.4	92.7	623.xalancbmk_s	77.0	91.7
508.namd_r	105.0	109.0	625.x264_s_0	89.4	95.7
510.parest_r	60.9	82.3	625.x264_s_1	98.9	99.7
519.lbm_r	61.5	84.9	625.x264_s_2	94.8	94.9
520.omnetpp_r	58.8	82.0	628.pop2_s	75.3	83.4
521.wrf_r	82.0	93.9	631.deepsjeng_s	80.0	90.0
523.xalancbmk_r	77.7	92.3	638.imagick_s	97.7	98.5
525.x264_r_0	89.2	96.4	641.leela_s	93.8	96.0
525.x264_r_1	98.2	101.7	644.nab_s	97.3	98.6
525.x264$_r$_2	98.9	102.3	648.exchange2_s	101.3	103.0
526.blender_r	75.0	89.4	649.fotonik3d_s	156.6	93.9
527.cam4_r	77.0	87.9	654.roms_s	196.9	109.5
531.deepsjeng_r	84.6	91.7	657.xz_s_0	83.6	86.6
538.imagick_r	97.1	98.2	657.xz_s_1	88.6	82.1513
541.leela_r	93.1	101.4			
544.nab_r	101.3	98.6			
548.exchange2_r	103.6	100.0			
549.fotonik3d_r	60.7	76.0			
554.roms_r	65.1	83.7			
557.xz_r_0	56.4	80.4			
557.xz_r_1	57.1	75.0			
557.xz_r_2	60.1	78.2			

Πίνακας Α.7: Scenario 5: Αποδόσεις των benchmarks στις τοπολογίες remote και interleave σε σχέση με τη local

Performance %							
500s series				600s series			
Benchmark	**Topology 0**	**Topology 1**	**Topology 2**	**Benchmark**	**Topology 0**	**Topology 1**	**Topology 2**
500.perlbench_r_0	379.8	315.4	340.4	600.perlbench_s_0	379.8	311.7	343.1
500.perlbench_r_1	218.5	217.5	218.8	600.perlbench_s_1	263.4	254.9	262.8
500.perlbench_r_2	550.0	414.9	499.1	600.perlbench_s_2	652.4	483.4	570.9
502.gcc_r_0	724.9	519.6	661.0	602.gcc_s_0	1028.3	609.4	854.7
502.gcc_r_1	649.0	465.4	557.1	602.gcc_s_1	724.8	512.6	613.7
502.gcc_r_2	751.9	496.9	627.6	602.gcc_s_2	771.3	536.0	642.6
502.gcc_r_3	668.7	455.1	587.4	603.bwaves_s_0	206.0	325.2	226.3
502.gcc_r_4	912.1	496.6	713.7	603.bwaves_s_1	205.3	315.6	222.2
503.bwaves_r_0	307.0	240.7	268.4	605.mcf_s	948.8	563.5	800.3
503.bwaves_r_1	272.8	209.3	239.9	607.cactuBSSN_s	420.4	471.8	380.1
503.bwaves_r_2	276.0	207.5	223.0	619.lbm_s	239.9	364.7	206.5
503.bwaves_r_3	265.1	208.8	232.7	620.omnetpp_s	1173.0	707.3	980.5
505.mcf_r	740.7	471.5	649.4	621.wrf_s	1211.2	1186.0	1181.0
507.cactuBSSN_r	515.0	372.6	477.5	623.xalancbmk_s	522.6	402.3	479.4
508.namd_r	214.4	225.1	233.6	625.x264_s_0	234.1	209.2	224.0
510.parest_r	895.3	545.3	737.0	625.x264_s_1	216.9	214.6	216.2
519.lbm_r	789.6	485.3	670.7	625.x264_s_2	219.7	208.2	208.5
520.omnetpp_r	1211.8	712.8	993.6	628.pop2_s	378.2	284.8	315.4
521.wrf_r	242.0	198.4	227.4	631.deepsjeng_s	329.5	263.5	296.5
523.xalancbmk_r	533.1	414.4	492.0	638.imagick_s	180.5	176.3	177.8
525.x264_r_0	255.3	227.6	246.0	641.leela_s	244.1	229.1	234.3
525.x264_r_1	196.0	192.4	199.2	644.nab_s	203.0	197.4	200.1
525.x264$_r$_2	217.1	214.7	222.2	648.exchange2_s	170.2	172.5	175.3
526.blender_r	538.3	403.9	481.2	649.fotonik3d_s	238.8	374.1	224.1
527.cam4_r	423.7	326.2	372.3	654.roms_s	225.5	443.9	246.9
531.deepsjeng_r	308.6	261.1	282.9	657.xz_s_0	347.6	290.6	301.0
538.imagick_r	202.9	197.0	199.2	657.xz_s_1	493.0	437.0	405.0
541.leela_r	245.5	228.5	248.8				
544.nab_r	192.0	194.6	189.3				
548.exchange2_r	173.4	179.7	173.5				
549.fotonik3d_r	771.1	468.0	586.2				
554.roms_r	541.0	352.2	452.9				
557.xz_r_0	816.5	460.9	656.6				
557.xz_r_1	635.7	363.0	476.6				
557.xz_r_2	645.3	387.6	504.8				

Πίνακας Α.8: Scenario 5: Αποδόσεις των benchmarks στις τοπολογίες local, remote και interleave σε σχέση με τη baseline

		Πειράματα					
Benchmark	**Baseline**	**1**	**2**	**3**	**4**	**5**	**6**
500.perlbench_r	249.8	379.7	456.2	345.1	409.6	436.5	344.9
508.namd_r	459.0	568.4	583.3	549.7	570.06	579.5	553.6
541.leela_r	694.1	901.7	963.6	853.1	937.6	956.6	851.7
544.nab_r	697.9	845.1	859.7	838.1	850.0	854.5	837.9
520.omnetpp_r	621.4	1089.7	1070.2	3248.8	1333.6	1307.1	2936.0
505.mcf_r	506.0	753.4	742.9	2123.8	909.9	891.9	1942.4
519.lbm_r	498.9	815.2	792.3	2458.0	1027.8	993.6	2223.6
549.fotonik3d_r	642.3	853.8	845.0	2906.8	1065.4	1052.6	2617.8

Πίνακας Α.9: Αποτελέσματα πρώτου σετ στατικών πειραμάτων resource manager (μη κανονικοποιημένα)

	Πειράματα					
Benchmark	**1**	**2**	**3**	**4**	**5**	**6**
500.perlbench_r	1.5	1.8	1.4	1.6	1.7	1.4
508.namd_r	1.2	1.3	1.2	1.2	1.3	1.2
541.leela_r	1.3	1.4	1.2	1.4	1.4	1.2
544.nab_r	1.2	1.2	1.2	1.2	1.2	1.2
520.omnetpp_r	1.8	1.7	5.2	2.1	2.1	4.7
505.mcf_r	1.5	1.5	4.2	1.8	1.8	3.8
519.lbm_r	1.6	1.6	4.9	2.1	2.0	4.5
549.fotonik3d_r	1.3	1.3	4.5	1.7	1.6	4.1
Overall Slowdown	1.4	1.5	2.4	1.6	1.6	2.3

Πίνακας Α.10: Αποτελέσματα πρώτου σετ στατικών πειραμάτων resource manager (κανονικοποιημένα ως προς baseline)

		Πειράματα					
Benchmark	**Baseline**	**1**	**2**	**3**	**4**	**5**	**6**
600.perlbench_s	250.1	3030.7	2885.4	3143.7	3561.0	2923.7	3113.9
641.leela_s	702.3	4724.9	4590.2	6200.5	5597.5	5174.5	6154.8
644.nab_s	791.5	3833.9	3770.7	4633.2	4067.2	3789.3	4570.8
648.exchange2_s	866.2	4875.5	4868.1	6522.3	5521.7	5523.0	6378.8
619.lbm_s	1391.0	3974.5	3891.2	18608.5	5248.6	5141.5	16525.2
649.fotonik3d_s	1022.9	3807.1	3729.2	15223.4	4838.8	4740.0	13566.1
654.roms_s	2483.8	5501.5	5430.4	22123.9	6956.5	6861.0	19649.1
603.bwaves_s	1408.2	5159.8	5093.1	14756.8	6111.4	6015.0	13475.8

Πίνακας Α.11: Αποτελέσματα δεύτερου σετ στατικών πειραμάτων resource manager (μη κανονικοποιημένα)

	Πειράματα					
Benchmark	**1**	**2**	**3**	**4**	**5**	**6**
600.perlbench_s	12.1	11.5	12.6	14.2	11.7	12.4
641.leela_s	6.7	6.5	8.8	8.0	7.4	8.8
644.nab_s	4.8	4.8	5.9	5.1	4.788	5.8
648.exchange2_s	5.6	5.6	7.5	6.4	6.4	7.4
619.lbm_s	2.9	2.8	13.4	3.8	3.7	11.9
649.fotonik3d_s	3.7	3.6	14.9	4.7	4.6	13.3
654.roms_s	2.2	2.2	8.9	2.8	2.8	7.9
603.bwaves_s	3.7	3.6	10.5	4.3	4.3	9.6
Overall Slowdown	4.6	4.5	9.9	5.5	5.2	9.3

Πίνακας Α.12: Αποτελέσματα δεύτερου σετ στατικών πειραμάτων resource manager (κανονικοποιημένα ως προς baseline)

	Πειράματα				
Benchmark	**1 (Best)**	**2 (Random)**	**3 (Random)**	**4 (Random)**	**5 (Worst)**
500.perlbench_r	401.951	355.317	365.701	348.767	346.846
508.namd_r	591.936	571.332	574.33	556.208	556.023
541.leela_r	959.371	968.344	906.064	922.424	889.128
544.nab_r	858.451	855.405	851.252	857.219	841.255
520.omnetpp_r	1102.099	1256.651	2068.693	1957.73	2418.422
505.mcf_r	744.94	893.675	1047.959	1459.95	1621.195
519.lbm_r	803.828	1062.14	1437.941	668.804	1199.076
549.fotonik3d_r	841.116	1063.593	796.679	2191.352	2305.784
600.perlbench_s	2913.221	2982.655	2994.77	2622.974	2544.024
641.leela_s	4707.604	4736.195	4731.09	4791.565	4805.243
644.nab_s	3824.005	3828.888	3808.196	3683.378	3771.418
648.exchange2_s	4876.157	4892.839	4947.423	5441.891	5477.17
619.lbm_s	3975.876	4100.609	4030.395	7034.35	3321.593
649.fotonik3d_s	3770.984	3712.533	3729.668	2835.442	5980.267
654.roms_s	5507.03	5530.273	5595.464	7825.303	7723.716
603.bwaves_s	5149.802	5169.905	5951.069	7404.667	6930.514

Πίνακας Α.13: Αποτελέσματα πρώτου σετ δυναμικών πειραμάτων resource manager (μη κανονικοποιημένα)

	Πειράματα				
Benchmark	**1 (Best)**	**2 (Random)**	**3 (Random)**	**4 (Random)**	**5 (Worst)**
500.perlbench_r	1	0.884	0.91	0.868	0.863
508.namd_r	1	0.965	0.97	0.94	0.939
541.leela_r	1	1.009	0.944	0.961	0.927
544.nab_r	1	0.996	0.992	0.999	0.98
520.omnetpp_r	1	1.14	1.877	1.776	2.194
505.mcf_r	1	1.12	1.407	1.96	2.1762
519.lbm_r	1	1.321	1.789	0.832	1.491
549.fotonik3d_r	1	1.265	0.947	2.605	2.741
600.perlbench_s	1	1.024	1.028	0.9	0.873
641.leela_s	1	1.006	1.005	1.018	1.021
644.nab_s	1	1.001	0.996	0.963	0.986
648.exchange2_s	1	1.003	1.015	1.116	1.123
619.lbm_s	1	1.031	1.014	1.769	0.835
649.fotonik3d_s	1	0.984	0.989	0.752	1.586
654.roms_s	1	1.004	1.016	1.421	1.403
603.bwaves_s	1	1.004	1.156	1.438	1.346
Overall Slowdown	1.0	1.047	1.099	1.188	1.25

Πίνακας Α.14: Αποτελέσματα πρώτου σετ δυναμικών πειραμάτων resource manager (κανονικοποιημένα ως προς το πρώτο πείραμα)

	Πειράματα				
Benchmark	**1 (Best)**	**2 (Random)**	**3 (Random)**	**4 (Random)**	**5 (Worst)**
600.perlbench_s-ref-1-instance_0	3538.305	3595.663	3391.729	3517.946	3364.78
600.perlbench_s-ref-1-instance_1	3152.756	3595.663	3391.729	3517.946	3364.78
600.perlbench_s-ref-1-instance_2	3538.305	2826.886	3896.188	2785.919	2740.966
600.perlbench_s-ref-1-instance_3	3309.743	2963.48	3229.446	3517.946	2838.105
641.leela_s-ref-1-instance_3	5581.448	6481.112	5794.827	5242.242	5806.679
641.leela_s-ref-1-instance_2	5433.754	6163.841	5236.346	6896.638	5576.558
641.leela_s-ref-1-instance_1	5040.983	6238.389	6200.836	5242.242	5701.227
641.leela_s-ref-1-instance_0	6050.468	8077.162	5236.346	6180.144	6134.603
644.nab_s-ref-1-instance_2	4010.773	4333.805	4023.293	4082.519	4003.867
644.nab_s-ref-1-instance_3	4010.773	3829.475	4210.051	4082.519	4663.512
644.nab_s-ref-1-instance_0	4155.8	4333.805	4210.051	4302.917	4047.561
644.nab_s-ref-1-instance_1	4010.773	3992.343	4023.293	4082.519	4003.867
648.exchange2_s-ref-1-instance_0	5436.556	5875.354	5722.466	5900.767	6012.0
648.exchange2_s-ref-1-instance_1	5120.333	5890.328	5518.763	5252.106	5820.04
648.exchange2_s-ref-1-instance_2	5436.556	6366.27	5518.763	6104.092	5820.04
648.exchange2_s-ref-1-instance_3	5236.538	5371.186	5254.337	5573.232	6297.839
619.lbm_s-ref-1-instance_0	8034.974	4782.575	7476.835	4963.227	5822.892
619.lbm_s-ref-1-instance_1	4318.596	5913.55	3840.485	4542.42	5593.557
619.lbm_s-ref-1-instance_2	5036.351	8516.412	6365.461	7309.613	7600.004
619.lbm_s-ref-1-instance_3	4456.362	8009.465	4668.829	6457.295	9563.9
649.fotonik3d_s-ref-1-instance_1	4058.139	5397.94	6248.973	4256.337	5602.723
649.fotonik3d_s-ref-1-instance_0	3050.137	5933.56	3909.0	6275.835	7442.206
649.fotonik3d_s-ref-1-instance_3	3797.743	2735.218	3235.481	3734.327	8639.06
649.fotonik3d_s-ref-1-instance_2	4741.158	6571.337	4174.062	5669.359	5985.956
654.roms_s-ref-1-instance_2	6980.374	7735.149	8691.461	6747.913	5317.935
654.roms_s-ref-1-instance_1	6689.743	8633.004	7445.812	8049.25	9587.724
654.roms_s-ref-1-instance_0	9120.285	8824.483	8059.385	8316.948	10191.138
654.roms_s-ref-1-instance_3	6338.502	8734.726	8442.705	7409.731	10277.619
603.bwaves_s-ref-1-instance_3	6310.142	6005.093	7701.011	5701.067	8118.23
603.bwaves_s-ref-1-instance_2	6216.714	8098.87	4842.508	7089.189	5317.943
603.bwaves_s-ref-1-instance_1	5505.116	7115.386	5729.495	5943.276	7806.509
603.bwaves_s-ref-1-instance_0	8123.152	7766.36	7880.472	6938.162	7764.886

Πίνακας Α.15: Αποτελέσματα δεύτερου σετ δυναμικών πειραμάτων resource manager (μη κανονικοποιημένα)

	Πειράματα				
Benchmark	**1 (Best)**	**2 (Random)**	**3 (Random)**	**4 (Random)**	**5 (Worst)**
600.perlbench_s-ref-1-instance_0	1	1.016	0.959	0.994	0.951
600.perlbench_s-ref-1-instance_1	1	1.016	0.959	0.994	0.951
600.perlbench_s-ref-1-instance_2	1	0.799	1.101	0.787	0.775
600.perlbench_s-ref-1-instance_3	1	0.895	0.976	0.994	0.858
641.leela_s-ref-1-instance_3	1	1.161	1.038	0.939	1.04
641.leela_s-ref-1-instance_2	1	1.134	0.964	1.269	1.026
641.leela_s-ref-1-instance_1	1	1.238	1.23	0.939	1.131
641.leela_s-ref-1-instance_0	1	1.335	0.964	1.021	1.014
644.nab_s-ref-1-instance_2	1	1.081	1.003	1.018	0.998
644.nab_s-ref-1-instance_3	1	0.955	1.05	1.018	1.163
644.nab_s-ref-1-instance_0	1	1.081	1.05	1.035	0.974
644.nab_s-ref-1-instance_1	1	0.995	1.003	1.018	0.998
648.exchange2_s-ref-1-instance_0	1	1.081	1.053	1.085	1.106
648.exchange2_s-ref-1-instance_1	1	1.15	1.078	1.026	1.137
648.exchange2_s-ref-1-instance_2	1	1.171	1.078	1.123	1.137
648.exchange2_s-ref-1-instance_3	1	1.026	1.003	1.064	1.203
619.lbm_s-ref-1-instance_0	1	0.595	0.931	0.618	0.725
619.lbm_s-ref-1-instance_1	1	1.369	0.889	1.052	1.295
619.lbm_s-ref-1-instance_2	1	1.691	1.264	1.451	1.509
619.lbm_s-ref-1-instance_3	1	1.797	1.048	1.449	2.146
649.fotonik3d_s-ref-1-instance_1	1	1.33	1.54	1.049	1.381
649.fotonik3d_s-ref-1-instance_0	1	1.945	1.282	2.058	2.44
649.fotonik3d_s-ref-1-instance_3	1	0.72	0.852	0.983	2.275
649.fotonik3d_s-ref-1-instance_2	1	1.386	0.88	1.196	1.263
654.roms_s-ref-1-instance_2	1	1.108	1.245	0.967	0.762
654.roms_s-ref-1-instance_1	1	1.29	1.113	1.203	1.433
654.roms_s-ref-1-instance_0	1	0.968	0.884	0.912	1.117
654.roms_s-ref-1-instance_3	1	1.378	1.332	1.169	1.621
603.bwaves_s-ref-1-instance_3	1	0.952	1.22	0.903	1.287
603.bwaves_s-ref-1-instance_2	1	1.303	0.779	1.14	0.855
603.bwaves_s-ref-1-instance_1	1	1.293	1.041	1.08	1.418
603.bwaves_s-ref-1-instance_0	1	0.956	0.97	0.854	0.956
Overall Slowdown	1.0	1.13	1.045	1.053	1.163

Πίνακας Α.16: Αποτελέσματα δεύτερου σετ δυναμικών πειραμάτων resource manager (κανονικοποιημένα ως προς το πρώτο πείραμα)

	Πειράματα					
Benchmark	**1 (Best)**	**2 (Random)**	**3 (Random)**	**4 (Random)**	**5 (Random)**	**6 (Worst)**
600.perlbench_s-ref-1-instance_0	3318.643	3560.023	3728.544	3291.905	3302.75	3396.5
600.perlbench_s-ref-1-instance_1	3318.643	3007.933	2960.946	3526.576	3225.273	3152.408
600.perlbench_s-ref-1-instance_2	3318.643	4262.078	3002.074	2839.904	3959.28	3152.408
600.perlbench_s-ref-1-instance_3	3177.47	3661.151	2960.946	2936.086	3639.073	2847.561
641.leela_s-ref-1-instance_3	5059.329	5554.883	6902.81	5718.116	5643.195	5638.647
641.leela_s-ref-1-instance_2	4938.239	5168.407	5746.414	5202.083	5207.847	5638.647
641.leela_s-ref-1-instance_1	4841.708	5965.191	5232.366	6992.425	5963.979	5744.207
641.leela_s-ref-1-instance_0	5059.329	5400.081	6855.748	5274.996	4938.683	5638.647
644.nab_s-ref-1-instance_2	3863.402	4089.157	4660.373	3882.308	4325.995	4068.595
644.nab_s-ref-1-instance_3	3959.571	3951.389	3787.769	4217.811	4005.165	4068.595
644.nab_s-ref-1-instance_0	3891.77	4324.222	4161.683	3835.976	4005.165	4068.595
644.nab_s-ref-1-instance_1	3863.402	4089.157	3924.453	4417.244	4325.995	4068.595
648.exchange2_s-ref-1-instance_0	5040.879	5355.234	6170.144	5165.255	5114.602	5867.839
648.exchange2_s-ref-1-instance_1	4934.508	5807.994	5452.78	6285.298	5631.339	6137.967
648.exchange2_s-ref-1-instance_2	5040.879	5355.234	5452.78	5502.38	5266.812	6137.967
648.exchange2_s-ref-1-instance_3	4934.508	5657.97	5725.427	5618.074	5701.88	5867.839
619.lbm_s-ref-1-instance_0	4040.928	6050.804	6543.704	5649.288	4166.71	5001.743
619.lbm_s-ref-1-instance_1	4040.928	7365.391	7500.852	6889.903	4152.043	8497.384
619.lbm_s-ref-1-instance_2	4247.921	5111.883	7852.437	8311.093	3939.929	8198.467
619.lbm_s-ref-1-instance_3	4024.188	3995.152	4838.689	3558.868	6248.594	7646.483
649.fotonik3d_s-ref-1-instance_1	3847.832	4528.365	3191.838	6488.542	5481.127	6585.843
649.fotonik3d_s-ref-1-instance_0	3847.832	4435.019	6156.83	4513.263	3937.249	6403.828
649.fotonik3d_s-ref-1-instance_3	3527.082	6833.736	7038.46	5716.33	3843.723	5309.195
649.fotonik3d_s-ref-1-instance_2	4035.418	3420.679	2878.007	3875.807	4655.301	7756.112
654.roms_s-ref-1-instance_2	6279.666	7410.841	8081.315	10021.817	7101.303	9893.855
654.roms_s-ref-1-instance_1	5755.664	8099.052	8514.137	8475.503	7759.496	9358.355
654.roms_s-ref-1-instance_0	7014.963	7410.841	9138.033	7836.404	7728.779	9499.618
654.roms_s-ref-1-instance_3	6742.551	8178.555	8918.03	8133.035	7977.211	9092.802
603.bwaves_s-ref-1-instance_3	5781.604	5697.973	5490.664	6925.67	6788.217	7320.78
603.bwaves_s-ref-1-instance_2	5612.456	6957.038	6439.897	7309.787	6539.359	7709.019
603.bwaves_s-ref-1-instance_1	6423.241	6694.326	7048.746	8111.943	7007.769	7709.019
603.bwaves_s-ref-1-instance_0	6573.48	3963.409	7854.436	2759.092	6026.85	7320.78

Πίνακας Α.17: Αποτελέσματα τρίτου σετ δυναμικών πειραμάτων resource manager (μη κανονικοποιημένα)

	Πειράματα					
Benchmark	**1 (Best)**	**2 (Random)**	**3 (Random)**	**4 (Random)**	**5 (Random)**	**6 (Worst)**
600.perlbench_s-ref-1-instance_0	1	1.073	1.124	0.992	0.995	1.023
600.perlbench_s-ref-1-instance_1	1	0.906	0.892	1.063	0.972	0.95
600.perlbench_s-ref-1-instance_2	1	1.284	0.905	0.856	1.193	0.95
600.perlbench_s-ref-1-instance_3	1	1.152	0.892	0.924	1.145	0.896
641.leela_s-ref-1-instance_3	1	1.098	1.364	1.13	1.115	1.115
641.leela_s-ref-1-instance_2	1	1.047	1.164	1.053	1.055	1.115
641.leela_s-ref-1-instance_1	1	1.232	1.081	1.444	1.232	1.186
641.leela_s-ref-1-instance_0	1	1.067	1.355	1.043	0.976	1.115
644.nab_s-ref-1-instance_2	1	1.058	1.206	1.005	1.12	1.053
644.nab_s-ref-1-instance_3	1	0.998	0.957	1.065	1.012	1.053
644.nab_s-ref-1-instance_0	1	1.111	1.069	0.986	1.012	1.053
644.nab_s-ref-1-instance_1	1	1.058	1.016	1.143	1.12	1.053
648.exchange2_s-ref-1-instance_0	1	1.062	1.224	1.025	1.015	1.164
648.exchange2_s-ref-1-instance_1	1	1.177	1.105	1.274	1.141	1.244
648.exchange2_s-ref-1-instance_2	1	1.062	1.105	1.092	1.045	1.244
648.exchange2_s-ref-1-instance_3	1	1.147	1.16	1.139	1.156	1.164
619.lbm_s-ref-1-instance_0	1	1.497	1.619	1.398	1.031	1.238
619.lbm_s-ref-1-instance_1	1	1.823	1.856	1.705	1.027	2.103
619.lbm_s-ref-1-instance_2	1	1.203	1.849	1.957	0.927	1.93
619.lbm_s-ref-1-instance_3	1	0.993	1.202	0.884	1.553	1.9
649.fotonik3d_s-ref-1-instance_1	1	1.177	0.83	1.686	1.424	1.712
649.fotonik3d_s-ref-1-instance_0	1	1.153	1.6	1.173	1.023	1.664
649.fotonik3d_s-ref-1-instance_3	1	1.938	1.996	1.621	1.09	1.505
649.fotonik3d_s-ref-1-instance_2	1	0.848	0.713	0.96	1.154	1.922
654.roms_s-ref-1-instance_2	1	1.18	1.287	1.596	1.131	1.576
654.roms_s-ref-1-instance_1	1	1.407	1.479	1.473	1.348	1.626
654.roms_s-ref-1-instance_0	1	1.18	1.303	1.117	1.102	1.354
654.roms_s-ref-1-instance_3	1	1.213	1.323	1.206	1.183	1.349
603.bwaves_s-ref-1-instance_3	1	0.986	0.95	1.198	1.174	1.266
603.bwaves_s-ref-1-instance_2	1	1.24	1.147	1.302	1.165	1.374
603.bwaves_s-ref-1-instance_1	1	1.042	1.097	1.263	1.091	1.374
603.bwaves_s-ref-1-instance_0	1	0.603	1.195	0.42	0.917	1.266
Overall Slowdown	1.0	1.133	1.187	1.154	1.107	1.295

Πίνακας Α.18: Αποτελέσματα τρίτου σετ δυναμικών πειραμάτων resource manager (κανονικοποιημένα ως προς το πρώτο πείραμα)

	Πειράματα						
Benchmark	**1 (Best)**	**2 (Random)**	**3 (Random)**	**4 (Random)**	**5 (Worst)**	**6 (Best+)**	**7 (Worst+)**
600.perlbench_s-ref-1-instance_0	3281.152	3204.946	2894.734	3659.338	3129.815	3612.644	3150.184
600.perlbench_s-ref-1-instance_1	3240.819	3095.027	3602.871	2704.792	3129.815	3062.4	3150.184
600.perlbench_s-ref-1-instance_2	3848.616	3873.227	3112.4	5035.966	2991.997	4074.376	3150.184
600.perlbench_s-ref-1-instance_3	3505.458	2987.454	3602.871	3659.338	2991.997	3371.104	3150.184
641.leela_s-ref-1-instance_3	4944.472	5937.642	5686.389	5872.231	4883.606	4996.457	6473.072
641.leela_s-ref-1-instance_2	5302.133	6203.313	5794.862	7589.267	5695.577	5595.2	6473.072
641.leela_s-ref-1-instance_1	5048.864	5053.86	5686.389	5359.578	5795.762	4775.378	6504.261
641.leela_s-ref-1-instance_0	5105.063	5390.37	6457.509	6386.546	6016.422	5309.557	6473.072
644.nab_s-ref-1-instance_2	4075.058	4077.768	4910.055	5112.021	4073.378	4223.358	4162.146
644.nab_s-ref-1-instance_3	3897.225	3981.127	4115.105	4177.037	3769.736	3904.128	4026.462
644.nab_s-ref-1-instance_0	3897.225	3981.127	3974.124	4016.442	4073.378	3966.185	4162.146
644.nab_s-ref-1-instance_1	3956.163	3981.127	4115.105	4016.442	4073.378	3904.128	4162.146
648.exchange2_s-ref-1-instance_0	4968.843	5367.783	5785.469	6020.33	6114.758	5135.162	6261.245
648.exchange2_s-ref-1-instance_1	5087.313	5367.783	5339.245	5583.665	6114.758	4895.838	6261.245
648.exchange2_s-ref-1-instance_2	5087.313	6226.605	5897.952	6427.522	5858.992	5321.106	6261.245
648.exchange2_s-ref-1-instance_3	4968.843	5407.567	5785.469	5832.154	5200.854	5011.762	6261.245
619.lbm_s-ref-1-instance_0	3990.95	4354.142	7447.061	7324.172	8635.721	4280.519	7591.648
619.lbm_s-ref-1-instance_1	4306.082	5283.092	3759.754	3519.585	9495.189	3789.752	9262.036
619.lbm_s-ref-1-instance_2	3851.429	8135.994	7400.109	8088.397	5682.894	3429.681	7279.676
619.lbm_s-ref-1-instance_3	3798.615	6915.036	5034.182	6438.242	3304.321	4197.453	8268.165
649.fotonik3d_s-ref-1-instance_1	4080.672	3354.196	4763.5	6738.277	8950.003	3614.597	8644.164
649.fotonik3d_s-ref-1-instance_0	3818.644	4740.113	4469.064	5309.775	6686.456	4046.985	6386.399
649.fotonik3d_s-ref-1-instance_3	3626.526	2864.144	5383.273	4782.757	4917.59	3955.396	7064.441
649.fotonik3d_s-ref-1-instance_2	3626.526	7107.015	3791.448	6535.674	5615.449	4906.032	6698.63
654.roms_s-ref-1-instance_2	6229.204	8199.291	7952.038	8592.553	8890.355	5982.997	9415.573
654.roms_s-ref-1-instance_1	6179.603	7576.358	7040.315	7802.954	9845.368	6603.171	9777.353
654.roms_s-ref-1-instance_0	5842.472	8294.896	8308.596	7802.954	10002.661	6585.254	9604.906
654.roms_s-ref-1-instance_3	6697.077	7606.001	8262.523	8351.88	8537.465	5855.251	9916.937
603.bwaves_s-ref-1-instance_3	5945.328	6907.136	6899.534	7401.639	5591.942	6162.863	8170.172
603.bwaves_s-ref-1-instance_2	6089.302	7269.739	7607.919	7449.534	7578.451	5912.021	8170.172
603.bwaves_s-ref-1-instance_1	6175.665	6123.731	6246.862	6166.546	8368.751	5856.913	8560.013
603.bwaves_s-ref-1-instance_0	6310.178	6301.764	5966.765	7123.578	7867.491	5761.488	8170.172

Πίνακας Α.19: Αποτελέσματα τέταρτου σετ δυναμικών πειραμάτων resource manager (μη κανονικοποιημένα)

	Πειράματα						
Benchmark	**1 (Best)**	**2 (Random)**	**3 (Random)**	**4 (Random)**	**5 (Worst)**	**6 (Best+)**	**7 (Worst+)**
600.perlbench_s-ref-1-instance_0	1	0.977	0.882	1.115	0.954	1.101	0.96
600.perlbench_s-ref-1-instance_1	1	0.955	1.112	0.835	0.954	0.945	0.96
600.perlbench_s-ref-1-instance_2	1	1.006	0.809	1.309	0.777	1.059	0.96
600.perlbench_s-ref-1-instance_3	1	0.852	1.112	1.115	0.777	0.962	0.96
641.leela_s-ref-1-instance_3	1	1.201	1.15	1.188	0.988	1.011	1.309
641.leela_s-ref-1-instance_2	1	1.17	1.093	1.431	1.074	1.055	1.309
641.leela_s-ref-1-instance_1	1	1.001	1.15	1.062	1.148	0.946	1.288
641.leela_s-ref-1-instance_0	1	1.056	1.265	1.251	1.179	1.04	1.309
644.nab_s-ref-1-instance_2	1	1.001	1.205	1.254	1.0	1.036	1.021
644.nab_s-ref-1-instance_3	1	1.022	1.056	1.072	0.967	1.002	1.033
644.nab_s-ref-1-instance_0	1	1.022	1.02	1.031	1.0	1.018	1.021
644.nab_s-ref-1-instance_1	1	1.022	1.056	1.031	1.0	1.002	1.021
648.exchange2_s-ref-1-instance_0	1	1.08	1.164	1.212	1.231	1.033	1.26
648.exchange2_s-ref-1-instance_1	1	1.08	1.05	1.098	1.231	0.962	1.26
648.exchange2_s-ref-1-instance_2	1	1.224	1.159	1.263	1.152	1.046	1.26
648.exchange2_s-ref-1-instance_3	1	1.088	1.164	1.174	1.047	1.009	1.26
619.lbm_s-ref-1-instance_0	1	1.091	1.866	1.835	2.164	1.073	1.902
619.lbm_s-ref-1-instance_1	1	1.227	0.873	0.817	2.205	0.88	2.151
619.lbm_s-ref-1-instance_2	1	2.112	1.921	2.1	1.476	0.89	1.89
619.lbm_s-ref-1-instance_3	1	1.82	1.325	1.695	0.87	1.105	2.177
649.fotonik3d_s-ref-1-instance_1	1	0.822	1.167	1.651	2.193	0.886	2.118
649.fotonik3d_s-ref-1-instance_0	1	1.241	1.17	1.39	1.751	1.06	1.672
649.fotonik3d_s-ref-1-instance_3	1	0.79	1.484	1.319	1.356	1.091	1.948
649.fotonik3d_s-ref-1-instance_2	1	1.96	1.045	1.802	1.548	1.353	1.847
654.roms_s-ref-1-instance_2	1	1.316	1.277	1.379	1.427	0.96	1.512
654.roms_s-ref-1-instance_1	1	1.226	1.139	1.263	1.593	1.069	1.582
654.roms_s-ref-1-instance_0	1	1.42	1.422	1.263	1.712	1.127	1.644
654.roms_s-ref-1-instance_3	1	1.136	1.234	1.247	1.275	0.874	1.481
603.bwaves_s-ref-1-instance_3	1	1.162	1.16	1.245	0.941	1.037	1.374
603.bwaves_s-ref-1-instance_2	1	1.194	1.249	1.223	1.245	0.971	1.374
603.bwaves_s-ref-1-instance_1	1	0.992	1.012	0.999	1.355	0.948	1.386
603.bwaves_s-ref-1-instance_0	1	0.999	0.946	1.129	1.247	0.913	1.374
Overall Slowdown	1.0	1.135	1.159	1.248	1.226	1.011	1.381

Πίνακας Α.20: Αποτελέσματα τέταρτου σετ δυναμικών πειραμάτων resource manager (κανονικοποιημένα ως προς το πρώτο πείραμα)

	Πειράματα			
Benchmark	**1 (Best)**	**2 (Best+)**	**3 (Worst)**	**4 (Worst+)**
520.omnetpp_r-ref-1-instance_3	7555.414	7061.483	9079.079	10326.669
520.omnetpp_r-ref-1-instance_0	7880.866	8029.206	8681.083	10279.158
520.omnetpp_r-ref-1-instance_1	9409.804	8034.544	9184.796	10232.613
520.omnetpp_r-ref-1-instance_2	9494.12	8553.089	10023.527	11650.634
505.mcf_r-ref-1-instance_0	7488.435	7498.857	7062.96	9379.346
505.mcf_r-ref-1-instance_1	8607.966	7347.96	6334.152	9219.564
505.mcf_r-ref-1-instance_2	9189.841	8156.726	7689.583	11099.476
505.mcf_r-ref-1-instance_3	7216.343	6640.525	6334.152	8375.441
519.lbm_r-ref-1-instance_2	9135.859	7626.163	8623.852	11136.825
519.lbm_r-ref-1-instance_3	7278.844	6658.355	7124.932	8637.157
519.lbm_r-ref-1-instance_0	7305.423	7529.621	7305.167	9487.774
519.lbm_r-ref-1-instance_1	8643.872	7370.945	7305.167	9556.97
549.fotonik3d_r-ref-1-instance_3	7323.336	6768.327	7462.136	11445.9
549.fotonik3d_r-ref-1-instance_2	9254.7	8267.128	9131.13	10570.014
549.fotonik3d_r-ref-1-instance_1	9078.141	7769.089	9878.663	10020.767
549.fotonik3d_r-ref-1-instance_0	7371.4	7624.17	8384.55	10022.564
619.lbm_s-ref-1-instance_0	6219.616	5093.664	8364.765	8435.139
619.lbm_s-ref-1-instance_1	6219.616	4960.282	7160.002	5720.232
619.lbm_s-ref-1-instance_2	7923.265	6475.206	8847.961	9157.994
619.lbm_s-ref-1-instance_3	4864.79	4844.877	4403.673	7943.005
649.fotonik3d_s-ref-1-instance_1	5302.85	4450.723	7307.919	7222.465
649.fotonik3d_s-ref-1-instance_0	3351.918	4450.723	6911.603	7222.465
649.fotonik3d_s-ref-1-instance_3	4361.751	3740.016	5499.689	5261.967
649.fotonik3d_s-ref-1-instance_2	2998.936	4076.988	7555.244	7403.179
654.roms_s-ref-1-instance_2	8700.521	7924.772	9890.008	10679.18
654.roms_s-ref-1-instance_3	6454.997	6370.967	9572.243	10225.151
654.roms_s-ref-1-instance_0	6454.997	7293.442	6781.351	9544.476
654.roms_s-ref-1-instance_1	8161.973	7055.34	9890.008	9544.476
603.bwaves_s-ref-1-instance_0	6660.9	6157.089	6847.474	6424.283
603.bwaves_s-ref-1-instance_3	6546.07	5141.498	8274.453	8917.138
603.bwaves_s-ref-1-instance_2	6925.908	6883.56	6289.633	8177.849
603.bwaves_s-ref-1-instance_1	7063.124	6883.56	6430.958	7914.44

Πίνακας Α.21: Αποτελέσματα πέμπτου σετ δυναμικών πειραμάτων resource manager (μη κανονικοποιημένα)

	Πειράματα			
Benchmark	**1 (Best)**	**2 (Best+)**	**3 (Worst)**	**4 (Worst+)**
520.omnetpp_r-ref-1-instance_3	1	0.935	1.202	1.367
520.omnetpp_r-ref-1-instance_0	1	1.019	1.102	1.304
520.omnetpp_r-ref-1-instance_1	1	0.854	0.976	1.087
520.omnetpp_r-ref-1-instance_2	1	0.901	1.056	1.227
505.mcf_r-ref-1-instance_0	1	1.001	0.943	1.253
505.mcf_r-ref-1-instance_1	1	0.854	0.736	1.071
505.mcf_r-ref-1-instance_2	1	0.888	0.837	1.208
505.mcf_r-ref-1-instance_3	1	0.92	0.736	1.161
519.lbm_r-ref-1-instance_2	1	0.835	0.944	1.219
519.lbm_r-ref-1-instance_3	1	0.915	0.979	1.187
519.lbm_r-ref-1-instance_0	1	1.031	1.0	1.299
519.lbm_r-ref-1-instance_1	1	0.853	1.0	1.106
549.fotonik3d_r-ref-1-instance_3	1	0.924	1.019	1.563
549.fotonik3d_r-ref-1-instance_2	1	0.893	0.987	1.142
549.fotonik3d_r-ref-1-instance_1	1	0.856	1.088	1.104
549.fotonik3d_r-ref-1-instance_0	1	1.034	1.137	1.36
619.lbm_s-ref-1-instance_0	1	0.819	1.345	1.356
619.lbm_s-ref-1-instance_1	1	0.798	1.151	0.92
619.lbm_s-ref-1-instance_2	1	0.817	1.117	1.156
619.lbm_s-ref-1-instance_3	1	0.996	0.905	1.633
649.fotonik3d_s-ref-1-instance_1	1	0.839	1.378	1.362
649.fotonik3d_s-ref-1-instance_0	1	0.839	2.062	1.362
649.fotonik3d_s-ref-1-instance_3	1	0.857	1.261	1.206
649.fotonik3d_s-ref-1-instance_2	1	1.359	2.519	2.469
654.roms_s-ref-1-instance_2	1	0.911	1.137	1.227
654.roms_s-ref-1-instance_3	1	0.987	1.483	1.584
654.roms_s-ref-1-instance_0	1	1.13	1.051	1.479
654.roms_s-ref-1-instance_1	1	0.864	1.137	1.479
603.bwaves_s-ref-1-instance_0	1	0.924	1.028	0.964
603.bwaves_s-ref-1-instance_3	1	0.785	1.264	1.362
603.bwaves_s-ref-1-instance_2	1	0.994	0.908	1.181
603.bwaves_s-ref-1-instance_1	1	0.994	0.91	1.121
Overall Slowdown	1.0	0.92	1.099	1.275

Πίνακας Α.22: Αποτελέσματα πέμπτου σετ δυναμικών πειραμάτων resource manager (κανονικοποιημένα ως προς το πρώτο πείραμα)

Παράρτημα Β

Κώδικας του Resource Manager

Παρακάτω παρατίθεται ο κώδικας της συνάρτησης *apply_model* που υλοποιεί το μοντέλο του Resource Manager που κατασκευάσαμε στα πλαίσια της παρούσας διπλωματικής εργασίας γραμμένο στη γλώσσα προγραμματισμού Python:

```
def apply_model(params):

  open_files = params["open_files"]
  BW_file = params["BW_file"]
  bench_log_file = params["bench_log_file"]

  global PIDs
  global process_info, log_file, change_pages
  overall_mem_pages = [0] * 4

  lines = BW_file.readlines()

  if lines:
    last_iteration = lines[-1:][0]
    if not last_iteration.startswith(';'):
      tokens = last_iteration.split(';')
      new_tokens = []
      for token in tokens:
        new_tokens.append(token.strip())

      try:
        BW_node_0 = float(new_tokens[9])
        BW_node_1 = float(new_tokens[17])
        BW_node_2 = float(new_tokens[25])
        BW_node_3 = float(new_tokens[33])
        Total_BW = float(new_tokens[36])
      except IndexError:
        bench_log_file.write("\nBW Stats index error")

  # Loop over running PIDs to collect updated perf counters
  for pid in PIDs.keys():
    lines = open_files[pid].readlines()
    # Fetch last update to perf output file
    last_iteration = lines[-4:]

    try:
      perf_counters = []
      for line in last_iteration:
        tokens = line.split()
        perf_counters.append(int(tokens[1]))

      cycles = perf_counters[0]
      instructions = perf_counters[1]

      if (cycles != 0 and instructions != 0):
        process_info[pid]["CPI"] = float(cycles / instructions)
```

```
47
48      if perf_interval:
49        process_info[pid]["BW"] = (perf_counters[2] + perf_counters[3]) / perf_interval
50
51    except ValueError:
52      log_file.write("\nError in fetching performance counters of process with pid " + str(pid))
53
54    # Fetch memory allocation of process in numa system
55    numa_file_path = "/proc/" + str(pid) + "/numa_maps"
56    try:
57      numa_file = open(numa_file_path, "r")
58      lines = numa_file.readlines()
59      numa_file.close()
60    except (OSError,IOError):
61      log_file.write("\nCould not open file " + numa_file_path)
62
63    node = [0] * 4
64    for line in lines:
65      tokens = line.split()
66      for token in tokens:
67        if token.startswith("N0"):
68          node[0] = node[0] + int(token[3:])
69          overall_mem_pages[0] = overall_mem_pages[0] + int(token[3:])
70        elif token.startswith("N1"):
71          node[1] = node[1] + int(token[3:])
72          overall_mem_pages[1] = overall_mem_pages[1] + int(token[3:])
73        elif token.startswith("N2"):
74          node[2] = node[2] + int(token[3:])
75          overall_mem_pages[2] = overall_mem_pages[2] + int(token[3:])
76        elif token.startswith("N3"):
77          node[3] = node[3] + int(token[3:])
78          overall_mem_pages[3] = overall_mem_pages[3] + int(token[3:])
79
80    process_info[pid]["mem_pages"] = node
81    node_sum = node[0] + node[1] + node[2] + node[3]
82
83    # Fetch CPU affinity
84    try:
85      res = subprocess.check_output(['taskset', '-p', str(pid)])
86      res = res.decode('ASCII')
87      tokens = res.split()
88      cpu_affinity = tokens[5]
89      process_info[pid]["cpu_affinity"] = cpu_affinity
90    except subprocess.CalledProcessError:
91      log_file.write("\nCould not fetch CPU affinity for process with pid " + str(pid))
92
93    max_memory_node = node.index(max(node))
94
95    # Default CPU affinity per node
96    node_cpu = ["ff000000ff", "ff000000ff00", "ff000000ff0000", "ff000000ff000000"]
97
98    if (PIDs[pid]["priority"] == "high"):
99      # If process is of high priority gather all memory pages to one node and migrate CPU to
100     # corresponding node in order to run locally
101     for i in range(0,3):
102       if (i == max_memory_node):
103         continue
104       try:
105         res = subprocess.check_output(['migratepages', str(pid), str(i), str(max_memory_node)
    ])
106       except subprocess.CalledProcessError:
107         log_file.write("\nError in migrating pages of process with pid " + str(pid))
108
```

```
    if (cpu_affinity != node_cpu[max_memory_node]):
      try:
        res = subprocess.check_output(['taskset', '-p', node_cpu[max_memory_node], str(pid)])
      except subprocess.CalledProcessError:
        log_file.write("\nError in changing CPU affinity of process with pid " + str(pid))

# If user wants page migration
if (change_pages["flag"]):
  BWs_loc = []
  BWs_rem = []
  for pid in PIDs.keys():
    # High priority benchmarks are not affected by page migration
    if (PIDs[pid]["priority"] == "low"):
      if (process_info[pid]["cpu_affinity"] == node_cpu[change_pages["dst"]]):
        BWs_loc.append((pid, process_info[pid]["BW"]))
      else:
        BWs_rem.append((pid, process_info[pid]["BW"]))

  # Sort local (to dst node) benchmarks in descending BW order
  BWs_loc.sort(key=lambda tuple: tuple[1], reverse=True)
  # Sort remote (to dst node) benchmarks in ascending BW order
  BWs_rem.sort(key=lambda tuple: tuple[1])

  total_pages = 0
  # Start migrating pages of benchmarks running on dst node cpu
  # in order to increase the proportion of their local memory
  for item in BWs_loc:
    try:
      pages = process_info[item[0]]["mem_pages"][change_pages["src"]]
      total_pages = total_pages + pages
      res = subprocess.check_output(['migratepages', str(item[0]),
            str(change_pages["src"]), str(change_pages["dst"])])

      if (total_pages >= change_pages["num"]):
        change_pages["flag"] = False
        break
    except subprocess.CalledProcessError:
      log_file.write("\nError in migrating pages of process with pid " + str(item[0]))

  # If total_pages migrated are less than those specified in the input command
  # continue with benchmarks running on remote nodes (to dst node)
  if change_pages["flag"]:
    for item in BWs_rem:
      try:
        pages = process_info[item[0]]["mem_pages"][change_pages["src"]]
        total_pages = total_pages + pages
        res = subprocess.check_output(['migratepages', str(item[0]),
              str(change_pages["src"]), str(change_pages["dst"])])

        if (total_pages >= change_pages["num"]):
          change_pages["flag"] = False
          break
      except subprocess.CalledProcessError:
        log_file.write("\nError in migrating pages of process with pid " + str(item[0]))

  # If total_pages migrated are less than those specified in the input command
  if (total_pages < change_pages["num"]):
    bench_log_file.write("\n\tCan not migrate any more pages")
    change_pages["flag"] = False
```