ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Αλγόριθμοι Χρονοδρομολόγησης σε Συστήματα Μεγάλης Κλίμακας

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Βασίλειος Θ. Γλύκαντζης

Επιβλέπων : Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2013

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Αλγόριθμοι Χρονοδρομολόγησης σε Συστήματα Μεγάλης Κλίμακας

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Βασίλειος Θ. Γλύκαντζης

Επιβλέπων : Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 23η Ιουλίου 2013.

.....	.....	.....
Νεκτάριος Κοζύρης	Δημήτριος Σούντρης	Δημήτριος Φωτάκης
Καθηγητής Ε.Μ.Π	Επίκουρος Καθηγητής Ε.Μ.Π	Λέκτορας Ε.Μ.Π

Αθήνα, Ιούλιος 2013

.....

Βασίλειος Θ. Γλύκαντζης

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

Περίληψη

Τα υπολογιστικά συστήματα μεγάλης κλίμακας είναι πλέον ευρέως διαδεδομένα και κερδίζουν συνεχώς έδαφος στην επίλυση προβλημάτων στον χώρο της βιομηχανίας, της ακαδημαϊκής κοινότητας και της κυβερνητικής έρευνας. Όσο εξαπλώνεται όμως η χρήση τους, τόσο εντείνεται η ανάγκη για γρήγορη, αποτελεσματική χρήση και εξοικονόμηση στην κατανάλωση ενέργειας. Επιπλέον, η ταυτόχρονη εκτέλεση εργασιών στον ίδιο κόμβο εγείρει θέματα ανταγωνισμού για τους μοιραζόμενους πόρους, με αποτέλεσμα την επιδείνωση της επίδοσης. Οι μέθοδοι που έχουν προταθεί για την επίλυση αυτών των προβλημάτων αφορούν τόσο στο υλικό όσο και στο λογισμικό.

Η χρονοδρομολόγηση των εργασιών αποτελεί μία ακόμα ελκυστική λύση στην προσπάθεια επίλυσης του ανταγωνισμού των εργασιών για τους μοιραζόμενους πόρους και της εξοικονόμησης της ενέργειας του συστήματος, καθώς δεν απαιτεί επιπλέον υλικό και είναι σχετικά εύκολο να ενσωματωθεί σε ένα σύστημα.

Σκοπός της παρούσας διπλωματικής εργασίας είναι η μελέτη αλγορίθμων χρονοδρομολόγησης και προηγμένων αλγοριθμικών τεχνικών σε συστήματα μεγάλης κλίμακας, όπου οι χρήστες υποβάλλουν εργασίες προς εκτέλεση, και η ανάπτυξη μιας καινούριας πολιτικής χρονοδρομολόγησης, που θα αντιμετωπίσει το πρόβλημα του ανταγωνισμού του διαύλου μνήμης και θα οδηγήσει σε βελτιστοποίηση του συστήματος. Για την αξιολόγησή της, εισάγαμε την καινούρια μας πολιτική στον χρονοδρομολογητή Maui και πραγματοποιήσαμε μετρήσεις, χρησιμοποιώντας 3 διαφορετικά εργασιακά φορτία.

Λέξεις κλειδιά: Χρονοδρομολόγηση, συστήματα μεγάλης κλίμακας, HPC, ανταγωνισμός μνήμης, εύρος ζώνης μνήμης

Abstract

The large-scale computing systems are widely known and continuously gain appreciation in solving problems in the fields of industry, academia and governmental research. However, as their usage expands, the need for faster, more efficient use and reduction of energy consumption becomes a necessity. Moreover, the concurrent execution of tasks on the same node raises issues of competition for shared resources, resulting in deterioration of performance. The methods which have been proposed to solve these problems concern both hardware and software.

Job scheduling is an another attractive tool in solving the problem of competition for shared resources and reduction of system energy consumption, because it does not require extra hardware and is relatively easy to integrate into the system.

The purpose of this thesis is the study of scheduling algorithms and advanced algorithmic techniques in large scale system, where users submit jobs for execution, and the development of a new scheduling policy, which will address the problem of competition of the memory bus and will lead to the optimization of the system. For its evaluation, we inserted our new policy in the Maui scheduler and took measurements using 3 different workloads.

Keywords: Scheduling, large scale systems, HPC, memory contention, memory bandwidth

Ευχαριστίες

Δεν θα μπορούσα να παραλείψω να ευχαριστήσω τους ανθρώπους που με την προσφορά τους έκαναν εφικτή την πραγματοποίηση της παρούσας διπλωματικής εργασίας, συμβάλλοντας τόσο με την καθοδήγησή όσο και με την βοήθεια τους στην επίλυση των προβλημάτων που ανέκυψαν.

Θα ήθελα λοιπόν να ευχαριστήσω τον επιβλέποντα της διπλωματικής μου εργασίας, Καθηγητή ΕΜΠ κ. Νεκτάριο Κοζύρη για την δυνατότητα που μου προσέφερε να ασχοληθώ με αυτό το ενδιαφέρον και σύγχρονο θέμα. Επίσης, θα ήθελα να ευχαριστήσω τον υποψήφιο διδάκτορα Αλέξανδρο Χαριτάτο, που ήταν ο άνθρωπος που με καθοδήγησε καθ' όλη την διάρκεια διπλωματικής μου εργασίας. Για την υλοποίηση του cluster, αμέριστη ήταν η συμβολή του υποψηφίου διδάκτορα Ευάγγελου Αγγέλου. Τέλος, θα ήταν παράλειψη να μην αναφερθώ στον διδάκτορα Γεώργιο Γκούμα που με τις εύστοχες συμβουλές και υποδείξεις του συνέβαλε διαφωτιστικά στη συγγραφή και παρουσίαση της εν λόγω διπλωματικής εργασίας.

Πίνακας Περιεχομένων

Πίνακας Σχημάτων

Πίνακας Πινάκων

1 Εισαγωγή

Το εισαγωγικό αυτό κεφάλαιο πραγματεύεται το αντικείμενο της παρούσας διπλωματικής εργασίας και παρουσιάζει εν συντομία τα κεφάλαια που θα ακολουθήσουν.

1.1 Αντικείμενο της διπλωματικής

Ο όρος «σύστημα μεγάλης κλίμακας» χρησιμοποιείται για να περιγράψει υπολογιστικά συστήματα που αποτελούνται από μεγάλο πλήθος επεξεργαστών, σε πολλές περιπτώσεις δεκάδες ή εκατοντάδες χιλιάδες, αλλά και άλλων υπολογιστικών πόρων, όπως τεράστια συστήματα μνήμης και συστήματα εισόδου-εξόδου. Αν και η πρώτη τους εμφάνιση χρονολογείται πολύ πίσω, την δεκαετία του 1960, η άνθισή τους έλαβε χώρα κυρίως την τελευταία δεκαετία, λόγω της μείωσης του κατασκευαστικού κόστους, της εμφάνισης του διαδικτύου και της ανάπτυξης του ελευθέρου λογισμικού. Σήμερα, αποτελούν μια αρκετά μεγάλη αγορά, αφού τα ετήσια έσοδα από τις πωλήσεις συστημάτων μεγάλης κλίμακας αγγίζουν τα 10 δισεκατομμύρια $.

Η ραγδαία ανάπτυξη αυτών των συστημάτων οδήγησε στην ανάγκη βελτιστοποίησης της λειτουργίας τους, τόσο σε επίπεδο υλικού, όσο και σε επίπεδο λογισμικού. Πολλά προβλήματα απαιτούσαν άμεση λύση για να καταστίσουν αυτά τα συστήματα βιώσιμα. Μερικά από αυτά είναι η ψύξη των συστημάτων, οι διασυνδέσεις, οι υψηλές απαιτήσεις σε ενέργεια, η σωστή διαχείριση των πόρων από την πλευρά του λογισμικού. Η έρευνα σε όλους αυτούς τους τομείς είναι διαρκής και οδηγεί ακόμα και στην χρήση νέων τεχνολογιών, όπως θα περιγράψουμε αναλυτικά και σε επόμενα κεφάλαια.

Όπως προαναφέρθηκε, οι απαιτήσεις των συστημάτων σε ενέργεια είναι πολύ υψηλές, καθώς σε ορισμένες περιπτώσεις μόνο το κόστος λειτουργίας αυτών των συστημάτων μπορεί να ξεπεράσει τα 10 εκατομμύρια $ ετησίως. Για αυτόν τον λόγο είναι αναγκαία τόσο η προσπάθεια μείωσης της απαιτούμενης ενέργειας, όσο και η διαβεβαίωση ότι η ενέργεια χρησιμοποιείται βέλτιστα. Στην παρούσα εργασία κινούμαστε προς αυτήν την κατεύθυνση, μελετώντας την χρονοδρομολόγηση των συστημάτων μεγάλης κλίμακας.

Η χρονοδρομολόγηση στα συστήματα μεγάλης κλίμακας αφορά στην χρονοδρομολόγηση *batch* συστημάτων. Τα *batch* συστήματα είναι μια συλλογή υπολογιστικών πόρων (CPUs, μνήμη, δίσκος) στα οποία γίνεται υποβολή και εκτέλεση εργασιών. Η διαφορά τους όμως με τους συμβατικούς υπολογιστές είναι ότι αποτελούν συστήματα καταμερισμού χώρου. Οι συμβατικοί υπολογιστές θέλουμε να είναι αποκρίσιμοι στις εντολές των χρηστών, για αυτό και οι υπολογιστικοί πόροι δεν κατανέμονται μόνιμα σε μια διεργασία. Αντίθετα, ο χρόνος χωρίζεται σε διακριτά διαστήματα, τα οποία στην συνέχεια μοιράζονται σε ένα πλήθος εργασιών. Έτσι, δίνουν στον χρήστη την αίσθηση ότι πολλές εργασίες εκτελούνται ταυτόχρονα ακόμα και αν το σύστημα διαθέτει μόνο έναν επεξεργαστή. Αντίθετα, στα συστήματα καταμερισμού χώρου δεν μας ενδιαφέρει η αποκρισιμότητα του συστήματος, οπότε οι υπολογιστικοί πόροι παραχωρούνται στις εργασίες μέχρι αυτές να ολοκληρωθούν. Ουσιαστικά, η χρονοδρομολόγηση των *batch* συστημάτων μπορεί να παρομοιαστεί με μια δεξαμενή που περιέχει υπολογιστικούς πόρους. Όσο η δεξαμενή δεν είναι άδεια, οι πόροι παραχωρούνται σε εργασίες για την εκτέλεσή τους. Όταν μια εργασία ολοκληρωθεί

επιστρέφει τους πόρους και πάλι στην δεξαμενή. Αν όμως η δεξαμενή αδειάσει, τότε δεν είναι δυνατόν να παραχωρηθούν οι απαραίτητοι πόροι στις εργασίες που υποβάλλονται στο σύστημα, με αποτέλεσμα να αναβάλλεται η εκτέλεσή τους. Αυτές οι εργασίες είναι αναγκαίο να περιμένουν μέχρι κάποιες άλλες να ολοκληρωθούν, ελευθερώνοντας τους πόρους που είχαν δεσμεύσει.

Στην παρούσα εργασία, θα μελετήσουμε αρχικά 3 από τους πιο συνηθισμένους αλγορίθμους που χρησιμοποιούνται για την χρονοδρομολόγηση των συστημάτων μεγάλης κλίμακας. Για αυτόν τον σκοπό κατασκευάσαμε 3 εργασιακά φορτία. Θα τους συγκρίνουμε μεταξύ τους με κριτήρια την χρησιμοποίηση συστήματος, τον μέσο χρόνο εκτέλεσης, τον μέσο χρόνο αναμονής, τον μέσο χρόνο ολοκλήρωσης και τον συνολικό χρόνο ολοκλήρωσης των εργασιών, που μας δείχνει ουσιαστικά τον ρυθμό διεκπεραίωσης των εργασιών (throughput). Παράλληλα, θα εξετάσουμε και την χρησιμοποίηση κάποιων προηγμένων αλγοριθμικών τεχνικών, όπως οι αλγόριθμοι *advance reservation* και *backfill* (Κεφάλαιο 3).

Ένα από τα σύγχρονα θέματα μελέτης των υπολογιστικών συστημάτων είναι ο ανταγωνισμός εργασιών, που τρέχουν ταυτόχρονα σε ένα σύστημα, για τους μοιραζόμενους πόρους. Τέτοιοι πόροι μπορεί να είναι η κρυφή μνήμη, η κύρια μνήμη, το δίκτυο, τα συστήματα εισόδου και εξόδου.

Η κύρια μνήμη συγκεντρώνει το ενδιαφέρον της ερευνητικής κοινότητας λόγω του υψηλού χρόνου πρόσβασης που απαιτεί και του περιορισμένου εύρους ζώνης. Σύγχρονες επιστημονικές μελέτες δείχνουν ότι ο ανταγωνισμός των εργασιών για τον διάδρομο μνήμης μπορεί να επιδεινώσει σε μεγάλο βαθμό τις επιδόσεις τους [37, 38]. Στο 2^{ο} μέρος της παρούσας εργασίας κατασκευάζουμε μια νέα πολιτική χρονοδρομολόγησης με επίγνωση του ανταγωνισμού στο δίαυλο μνήμης (memory aware scheduling policy) και την εφαρμόζουμε ταυτόχρονα με τους αλγορίθμους που μελετήσαμε προηγουμένως, στα ίδια εργασιακά φορτία. Τέλος, μελετούμε τις μεταβολές στα κριτήρια που χρησιμοποιήσαμε και προηγουμένως, για να αποφανθούμε αν τελικά η *memory aware* πολιτική χρονοδρομολόγησης ανταποκρίνεται στις προσδοκίες μας.

Για την εφαρμογή των πειραμάτων χρησιμοποιήσαμε τον χρονοδρομολογητή Maui, που αποτελεί έναν ευρέως διαδεδομένο χρονοδρομολογητή στον χώρο των συστημάτων μεγάλης κλίμακας και υποστηρίζει πολλαπλές πολιτικές χρονοδρομολόγησης, απόδοση δυναμικών προτεραιοτήτων, κρατήσεις και δυνατότητες δίκαιης μοιρασιάς (fairshare).

1.2 Οργάνωση τόμου

Στο παρόν 1^{ο} κεφάλαιο, αναλύεται το αντικείμενο και ο σκοπός της διπλωματικής εργασίας και παρουσιάζονται εν συντομία τα περιεχόμενα κάθε κεφαλαίου.

Στο 2^{ο} κεφάλαιο γίνεται μια ιστορική αναδρομή των συστημάτων μεγάλης κλίμακας, με ιδιαίτερη έμφαση στις αιτίες που οδήγησαν στην ανάπτυξή τους την τελευταία δεκαετία. Αναλύονται επίσης, οι κατηγορίες των συστημάτων μεγάλης κλίμακας ανάλογα με τον τρόπο κατασκευής τους και τον τρόπο παροχής υπηρεσιών. Παρουσιάζονται οι εξελίξεις που έχουν γίνει, από την εποχή της εμφάνισής τους μέχρι σήμερα, στο υλικό και στο λογισμικό. Τέλος, γίνεται μια κατηγοριοποίηση των συστημάτων ανάλογα με τις

επιδόσεις τους και καταγράφονται οι κύριοι τομείς όπου χρησιμοποιούνται τα συστήματα μεγάλης κλίμακας σήμερα.

Το 3^{ο} κεφάλαιο επεξηγεί την έννοια «χρονοδρομολόγηση» στα συστήματα μεγάλης κλίμακας και ταξινομεί τα συστήματα σε 2 κατηγορίες, με ξεχωριστές ανάγκες χρονοδρομολόγησης. Επεξηγείται επίσης ο όρος «σύστημα διαχείρισης πόρων» και παρουσιάζονται τα συστατικά του στοιχεία, δηλαδή ο διαχειριστής πόρων και ο χρονοδρομολογητής. Εν συνεχεία, παρουσιάζονται οι κυριότεροι αλγόριθμοι χρονοδρομολόγησης που χρησιμοποιούνται στα συστήματα μεγάλης κλίμακας. Ιδιαίτερη αναφορά γίνεται σε προηγμένες αλγοριθμικές τεχνικές και σε πολιτικές, που έχουν ως σκοπό την βελτιστοποίηση των συνηθισμένων αλγορίθμων. Τέλος, αναφέρεται και αιτιολογείται η επιλογή του διαχειριστή πόρων και του χρονοδρομολογητή για την διεξαγωγή των πειραμάτων.

Το 4^{ο} κεφάλαιο είναι αφιερωμένο εξολοκλήρου στον χρονοδρομολογητή Maui. Περιγράφεται αναλυτικά η φιλοσοφία του χρονοδρομολογητή, καθώς και τα στοιχεία χρονοδρομολόγησης, ο κύκλος χρονοδρομολόγησης και η ροή των εργασιών. Ιδιαίτερη έμφαση δίνεται στην προτεραιότητα των εργασιών, την παραχώρηση των κόμβων στις εργασίες, τις προηγμένες αλγοριθμικές τεχνικές που χρησιμοποιεί και την ποιότητα των παρεχόμενων υπηρεσιών, καθώς αυτά είναι τα στοιχεία που θα μελετήσουμε και παρακάτω, κατά την διεξαγωγή των πειραμάτων. Επίσης, περιέχονται κεφάλαια που επεξηγούν πώς ο Maui συμπεριφέρεται σε θέματα δικαιοσύνης και παροχής στατιστικών στοιχείων.

Στο 5^{ο} κεφάλαιο πραγματοποιείται η 1^{η} σειρά πειραμάτων, που έχει ως στόχο την σύγκριση τριών ευρέως διαδεδομένων αλγορίθμων χρονοδρομολόγησης, δηλαδή των First Come First Served (FCFS), Largest Job First (LJF) και Smallest Job First (SJF), καθώς και της χρήσης των προηγμένων αλγοριθμικών τεχνικών *backfill* και *advance reservation*. Για αυτό το σκοπό, όπως αναφέραμε και προηγουμένως, κατασκευάσαμε 3 εργασιακά φορτία. Εν συνεχεία αποφαινόμαστε για τις συνθήκες επιλογής του κάθε αλγορίθμου.

Το 6^{ο} κεφάλαιο περιγράφει την κατασκευή μιας καινούριας *memory aware* πολιτικής, που θα βελτιστοποιήσει τους παραπάνω αλγορίθμους. Αρχικά, δικαιολογείται η ανάγκη για μια νέα πολιτική που θα έχει ως σκοπό την αντιμετώπιση του ανταγωνισμού για τον δίαυλο μνήμης. Στη συνέχεια, πραγματοποιούνται μετρήσεις που έχουν ως σκοπό την διερεύνηση τεχνικών επίλυσης του ανταγωνισμού για τον δίαυλο μνήμης. Ακολουθεί η περιγραφεί της νέας πολιτικής και η εφαρμογή των αλγορίθμων που μελετήσαμε προηγουμένως ταυτόχρονα με την *memory aware* πολιτική. Εν κατακλείδι, επισημαίνουμε τις συνθήκες υπό τις οποίες η *memory aware* πολιτική που κατασκευάσαμε παρουσιάζει θετικά αποτελέσματα και πραγματοποιούνται προτάσεις για περαιτέρω έρευνα πιθανών μελλοντικών επεκτάσεων της μελέτης που πραγματοποιήθηκε, η διερεύνηση των οποίων παρουσιάζει σημαντικό ενδιαφέρον.

Στο 7^{ο} κεφάλαιο παρουσιάζονται οι βιβλιογραφικές αναφορές που χρησιμοποιήθηκαν κατά την συγγραφή της παρούσας διπλωματικής.

Τέλος, ακολουθούν 3 παραρτήματα. Στο *Παράρτημα Ι* περιγράφεται η κατασκευή της συστοιχίας (cluster), όπου πραγματοποιήθηκε η ανάπτυξη του κώδικα. Στο *Παράρτημα ΙΙ* αναλύεται η διαδικασία εγκατάστασης του συστήματος διαχείρισης πόρων. Τέλος, το

Παράρτημα III περιγράφει αναλυτικά την διαδικασία τροποποίησης του κώδικα του χρονοδρομολογητή Maui, για την εισαγωγή της *memory aware* πολιτική.

2 *Συστήματα μεγάλης κλίμακας*

Στο κεφάλαιο αυτό πραγματοποιούμε μια σύντομη παρουσίαση των συστημάτων μεγάλης κλίμακας και εισάγουμε κάποιους όρους που θα μας φανούν χρήσιμοι στη συνέχεια.

2.1 *Ιστορική αναδρομή*

Αναφέροντας σε κάποιον τον όρο «υπερυπολογιστής» (supercomputer) κατευθείαν έρχεται στο μυαλό του η ιδέα ενός υπερβολικά περίπλοκου μηχανήματος, κόστους πολλών εκατομμυρίων, που λύνει προβλήματα που ενδιαφέρουν πολύ εξεζητημένες ομάδες ανθρώπων, όπως για παράδειγμα πυρηνικούς επιστήμονες ή επιστήμονες της NASA. Αυτή η αντίληψη δεν απείχε πολύ από την πραγματικότητα όταν οι υπερυπολογιστές εμφανίστηκαν για πρώτη φορά στο παγκόσμιο στερέωμα.

Ενώ ο όρος “Super Computing” είχε εμφανιστεί ήδη από το 1929 για να περιγράψει ένα μηχάνημα (tabulator) που είχε κατασκευάσει η IBM για το Columbia University, οι supercomputers εμφανίστηκαν για $1^{η}$ φορά την δεκαετία του 1960. Στα πρώτα τους βήματα χρησιμοποιούσαν λίγους επεξεργαστές. Όμως, από το 1990 και ύστερα άρχιζαν να εμφανίζονται μηχανήματα με χιλιάδες επεξεργαστές και προς το τέλος του $20^{ου}$ αιώνα supercomputers με δεκάδες χιλιάδες επεξεργαστές εμπορίου είχαν γίνει ο κανόνας.

Στο παρελθόν, την κατασκευή των supercomputers αναλάμβαναν ειδικές επιχειρήσεις που χρησιμοποιούσαν ειδικά κατασκευασμένα εξαρτήματα και αναλάμβαναν την συντήρηση, την αναβάθμιση, την υποστήριξη και την επεξήγηση της λειτουργίας στο προσωπικό, χρεώνοντας φυσικά μερικά εκατομμύρια. Το κόστος αυτό, που προερχόταν κυρίως από την κατασκευή ειδικών επεξεργαστών και συστημάτων μνήμης που εγγυόντουσαν υψηλή επίδοση, ήταν δυσβάσταχτο για τις περισσότερες επιχειρήσεις. Επιπρόσθετα, όσο η επίδοση αυτών των συστημάτων αυξανόταν, το κόστος κατασκευής μεγάλωνε εκθετικά.

Αντίθετα, οι εμπορικοί υπολογιστές βρίσκονταν σε ιδιαίτερη άνθιση. Όμως, οι επεξεργαστές που χρησιμοποιούσαν δεν παρείχαν υψηλή επίδοση καθώς ήταν σχεδιασμένοι για εφαρμογές γραφείου και δεν μπορούσαν να χρησιμοποιηθούν για την κατασκευή συστημάτων όπου η απόδοση ήταν το κυριότερο μέλημα. Αυτό άλλαξε όταν άρχισαν να περιλαμβάνουν μονάδες κινητής υποδιαστολής. Αυτό ήταν το κομβικό σημείο που οδήγησε στην σταθερή ανάπτυξη των εμπορικών επεξεργαστών και στην υιοθέτησή τους σε συστήματα μεγάλης κλίμακας (commodity clusters). [1,2]

Ένα άλλο στοιχείο που οδήγησε στην ανάπτυξη των commodity clusters ήταν το Internet. Πρώτον, πυροδότησε την ζήτηση για Web servers, για τα data center των οποίων χρησιμοποιήθηκαν χαμηλού κόστους commodity HPC clusters. Δεύτερον, οδήγησε σε παγκόσμια συνεργασία για την επίλυση προβλημάτων των πρώτων HPC (High Performance Computing). Δημιουργήθηκαν ολόκληρες κοινότητες με διαφορετικές προσεγγίσεις πάνω στα commodity-based HPC που οδήγησαν σε βαθμιαία ανάπτυξη των συστημάτων μεγάλης κλίμακας. [3]

Όλα τα παραπάνω σε συνδυασμό με την ανάπτυξη του Linux, που είναι ένα ανοιχτού λογισμικού λειτουργικό σύστημα που διατίθεται δωρεάν και επικράτησε στα συστήματα μεγάλης κλίμακας, ήταν οι κύριες αιτίες που έριξαν την τιμή των HPC συστημάτων από εκατομμύρια σε μερικές δεκάδες χιλιάδες και τα κατέστησαν πολύ πιο προσιτά σε πολύ μεγαλύτερο φάσμα του πληθυσμού.

2.2 Κατηγοριοποίηση

Υπάρχουν αρκετοί τρόποι με τους οποίους μπορούμε να δεσμεύσουμε υπολογιστικούς κύκλους για να επιλύσουμε διάφορα προβλήματα στα συστήματα μεγάλης κλίμακας. Παρακάτω παρουσιάζονται οι κυριότερες μορφές των συστημάτων μεγάλης κλίμακας [3]:

- **Εμπορικές Συστοιχίες (commodity HPC clusters):** Είναι κατασκευασμένοι από εξαρτήματα που είναι διαθέσιμα στο εμπόριο και διασυνδέσεις υψηλών ταχυτήτων. Όπως είπαμε και προηγουμένως αποτελεί την επικρατούσα μορφή μετά το 1990. Μπορεί να αποτελείται από εκατοντάδες μέχρι και δεκάδες χιλιάδες κόμβους που συνεργάζονται για την επίλυση ενός προβλήματος. Συνδυάζει υψηλή επίδοση, συμβατότητα (λόγω των εξαρτημάτων εμπορίου που χρησιμοποιεί) και χαμηλό κόστος.
- **Ειδικής κατασκευής υπερυπολογιστές (dedicated supercomputers):** Είναι η μορφή που είχε επικρατήσει κατά την εμφάνιση των συστημάτων μεγάλης κλίμακας. Δεν χρησιμοποιεί εξαρτήματα εμπορίου αλλά ειδικής κατασκευής επεξεργαστές και συστήματα μνήμης. Παρόλα αυτά, η κατασκευή τους δεν έχει εγκαταλειφθεί, καθώς ανάλογα με τις ανάγκες που πρέπει να καλυφθούν μπορεί να αποτελούν την καλύτερη επιλογή. Κυριότερο μειονέκτημά τους είναι το υψηλό κόστος κατασκευής.
- **HPC cloud computing:** Είναι μια νέα μέθοδος που βασίζεται στο Internet. Οι υπολογιστικοί πόροι που επιθυμούμε υπάρχουν σε ένα σύννεφο (cloud) και επιτρέπουν την απομακρυσμένη πρόσβαση στον χρήστη. Δηλαδή παρέχουν υπολογιστικούς πόρους στον χρήστη ως υπηρεσία. Ενώ το κόστος τους είναι σχετικά χαμηλό, η εισαγωγή επιπλέον στρωμάτων μεταξύ του χρήστη και του υλικού μειώνει την επίδοση. [10]
- **Grid (πλέγμα) computing:** Μοιάζει με το cloud computing αλλά απαιτεί μεγαλύτερο έλεγχο από τον τελικό χρήστη. Χρησιμοποιείται κυρίως για ακαδημαϊκές εργασίες όπου οι τοπικοί HPC clusters συνδέονται με μεγαλύτερους σε εθνικό ή και διεθνές επίπεδο. Οι υπολογιστικοί πόροι ενός grid μπορεί να είναι είτε στο εσωτερικό ενός οργανισμού είτε διασκορπισμένοι στην υφήλιο [4,10].

2.3 Δομή

Η δομή ενός συστήματος μεγάλης κλίμακας δεν είναι εύκολο να περιγραφεί, καθώς η φύση του εξαρτάται από τις εφαρμογές που καλείται να υλοποιήσει. Οι διαφορές μπορεί να είναι μικρές ή μεγάλες, υπάρχουν όμως κάποια κοινά χαρακτηριστικά που έχουν όλες οι συστοιχίες (clusters).

2.3.1 Hardware

Όπως αναφέραμε και προηγουμένως, η επικρατέστερη μορφή συστημάτων μεγάλης κλίμακας είναι οι commodity clusters που κατασκευάζονται από υλικά εμπορίου. Αυτό σημαίνει ότι τα διάφορα συστατικά του cluster, όπως οι επεξεργαστές, οι διασυνδέσεις, οι αποθηκευτικοί χώροι, προέρχονται από διαφορετικούς πωλητές και δεν υπάρχει μία εταιρεία που παίρνει την ευθύνη για όλο το cluster. Αυτό οδηγεί στην αποφυγή περιττών εξόδων, αφού οι τιμές των υλικών στην αγορά είναι αρκετά ανταγωνιστικές, και η ποικιλία τους ευνοεί την βελτιστοποίηση του συστήματος, ώστε να ικανοποιεί τις εκάστοτε ανάγκες. Από την άλλη βέβαια, απαιτείται ένας σημαντικός βαθμός εξειδίκευσης για την επιλογή του υλικού, την οποία δεν κατέχει ο μέσος χρήστης.

Ενώ κατά την εμφάνισή τους τα συστήματα μεγάλης κλίμακας διέθεταν μόνο μερικούς ειδικά κατασκευασμένους επεξεργαστές με υψηλές επιδόσεις, η πολιτική αυτή μεταβλήθηκε και σήμερα έχουμε καταλήξει σε μαζικά παράλληλα συστήματα, μερικά από τα οποία περιλαμβάνουν πάνω από 100000 επεξεργαστές. Οι περισσότεροι clusters έχουν έναν κεντρικό κόμβο που ονομάζεται head ή master node. Αυτός ο κόμβος αποτελεί την πύλη (gateway) που συνδέονται οι χρήστες. Επικοινωνεί μέσω ενός ή περισσότερων δικτύων (networks) με τους υπολογιστικούς κόμβους (worker ή computing node). Αυτά τα δίκτυα είναι ιδιωτικά (private) και είναι προσβάσιμα μόνο στο εσωτερικό του κόμβου.

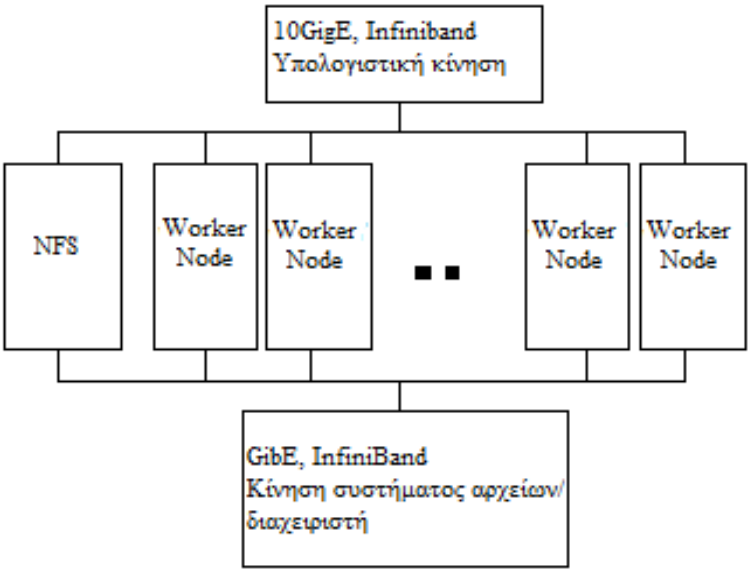
Η έρευνα πάνω στην βελτίωση τις υπολογιστικής δυναμικότητας των συστημάτων μεγάλης κλίμακας είναι διαρκής και πολύπλευρη. Καθώς η τιμή και η επίδοση των GPUs βελτιώνεται, αρχίζουν να παίζουν πρωταγωνιστικό ρόλο στα συστήματα μεγάλης κλίμακας [5]. Το 2012 ο Jaguar supercomputer μετατράπηκε σε Titan αντικαθιστώντας CPUs με GPUs και αποτελεί πλέον τον πιο γρήγορο supercomputer, ξεπερνώντας τα 17.5 Petaflops/s [6]. Οι πειραματισμοί που λαμβάνουν χώρα όμως δεν σταματούν στις GPUs. Έχουν κατασκευαστεί συστήματα ειδικού σκοπού που χρησιμοποιούν FPGA chips ή ειδικά VLSI chips παρέχοντας πολύ καλό συνδυασμό κόστους και επίδοσης. Μερικά παραδείγματα τέτοιων συστημάτων είναι το Belle [7], το Deep Blue [8] και το Hydra [9].

Σε ένα μαζικά παράλληλο σύστημα το δίκτυο επικοινωνίας των κόμβων αποτελεί πολύ σημαντικό παράγοντα για την επίδοση, καθώς είναι αναγκαία η συνεχής ανταλλαγή πληροφοριών. Η πιο συνηθισμένη περίπτωση είναι το Gigabit Ethernet (GigE), αν και το InfiniBand έχει αρχίσει και γίνεται αρκετά δημοφιλές. Για την τοπολογία των δικτύων μπορεί να χρησιμοποιηθούν είτε στατικά είτε δυναμικά δίκτυα διασυνδέσεων. Τα πιο δημοφιλή στατικά δίκτυα είναι ο δακτύλιος (ring), το πλέγμα (mesh), ο τόρος (torus), ο cube connected cycle (CCC) και η γενίκευση κ-δικός ν-κύβος. Τα δυναμικά δίκτυα που χρησιμοποιούνται είναι ο διάδρομος (bus), τα δίκτυα με σταυρωτούς διακόπτες (switches) και τα δίκτυα πολλών βαθμίδων (multistage networks) [10, 11].

Σε ένα cluster υπάρχουν 3 βασικές αιτίες κίνησης (traffic):

- Υπολογιστική κίνηση μεταξύ των κόμβων.
- Κίνηση στο σύστημα αρχείων. Συνήθως από NFS (Network File System) server, αν και χρησιμοποιείται πολλές φορές Direct Attach Storage καθώς έχει τα δικά του πλεονεκτήματα.
- Κίνηση που προκαλείται από τον διαχειριστή με σκοπό την παρακολούθηση των κόμβων και τον έλεγχο των εργασιών στο cluster.

Ανάλογα με την εφαρμογή που τρέχει, οι 2 πρώτες αιτίες είναι πιθανόν να φέρουν σημαντικές επιπτώσεις στην επίδοση. Για την αντιμετώπιση αυτού του προβλήματος συνήθως προστίθενται επιπλέον δίκτυα ώστε να βελτιώσουν τον ρυθμό διεκπεραίωσης των εργασιών (throughput). Οι πιο δημοφιλής επιλογές για αυτά τα δίκτυα είναι το 10GigE ή το InfiniBand. Φθηνές επιλογές υπάρχουν αλλά δεν επιφέρουν το ίδιο αποτέλεσμα στην επίδοση. Παρακάτω παρουσιάζεται σχηματικά η δομή ενός τυπικού cluster:



Σχήμα 2.1 Δομή μιας τυπικής συστοιχίας

Υπάρχουν πολλές παραλλαγές για τον σχεδιασμό μιας συστοιχίας. Για παράδειγμα ο master node μπορεί να αποτελείται από πολλούς διαφορετικούς κόμβους. Μπορεί να υπάρχουν πολλαπλοί κόμβοι αρχείων (file serving nodes), πολλαπλοί κόμβοι εισόδου του χρήστη (user login nodes) και ξεχωριστοί κόμβοι διαχειριστή (administration nodes).

Οι υπολογιστικοί κόμβοι τοποθετούνται σε ντουλάπες (equipment racks) και συνήθως απαιτούν ένα περιβάλλον όμοιο των κέντρων δεδομένων (data centers). Ο τύπος και ο αριθμός των επεξεργαστών, το μέγεθος της μνήμης, ακόμα και η παρουσία τοπικών συσκευών αποθήκευσης (local storage devices) καθορίζονται από τον χρήστη. Ένα παράδειγμα supercomputer παρουσιάζεται στην παρακάτω εικόνα:

Σχήμα 2.2 Ο supercomputer IBM Blue Gene/P στο Argonne National Laboratory

Ένα από τα πιο σημαντικά ζητήματα των συστημάτων μεγάλης κλίμακας είναι η διαχείριση της ενέργειας και των υψηλών θερμοκρασιών. Για παράδειγμα ο K computer της Fujitsu (1ος σε επίδοση τον 11/2011 και 3^{ος} τον 11/2012) απαιτεί πάνω από 12.5MW. Οπότε το κόστος για την λειτουργία και την ψύξη του συστήματος είναι αρκετά σημαντικό. Συγκεκριμένα, 12.5MW με τιμή 0.1$/kWh αντιστοιχεί σε 1250$ την ώρα ή περίπου 11 εκατομμύρια $ τον χρόνο.

Η σωστή διαχείριση των υψηλών θερμοκρασιών μπορεί να βελτιώσει σημαντικά την επίδοση των συστημάτων μεγάλης κλίμακας [13]. Οι τεχνολογίες που χρησιμοποιούνται ξεπερνούν τους συνηθισμένους τρόπους ψύξης των συμβατικών υπολογιστών. Η ενεργειακή επίδοση μετριέται σε FLOPS/Watt. Το *Green 500* είναι ένα site που περιέχει λίστες με τις αξιολογήσεις των συστημάτων μεγάλης κλίμακας πάνω στο θέμα της ενεργειακής επίδοσης [14].

Μερικές από τις πιο συχνές μεθόδους αντιμετώπισης των υψηλών θερμοκρασιών που χρησιμοποιούνται είναι η *υδρόψυξη* και ο συνδυασμός *υδρόψυξης* και *air conditioning*. Μια άλλη ενδιαφέρουσα μέθοδος είναι η χρησιμοποίηση επεξεργαστών με μικρή κατανάλωση ενέργειας (Blue Gene-IBM [15]). Τέλος, η IBM πρότεινε με τον supercomputer Aquasar (Πανεπιστήμιο ETH Ζυρίχη) μια καινοτόμα μέθοδο που ονομάζεται υδρόψυξη με ζεστό νερό (hot water cooling) και υποστηρίζει ότι καταναλώνει 40% λιγότερη ενέργεια [16,17]. Το νερό χρησιμοποιείται και για να θερμάνει κτίρια του πανεπιστημίου.

2.3.2 Software

Στα πρώτα βήματα των supercomputers, οι αρχιτεκτονικές εξελίσσονταν διαρκώς και το software προσπαθούσε να ακολουθήσει τις καινοτομίες τους, που άλλαζαν διαρκώς κατευθύνσεις. Αρχικά, όπως και στην περίπτωση του hardware, το software ήταν ειδικά κατασκευασμένο για κάθε supercomputer ώστε να ικανοποιεί τις ανάγκες του και να βελτιώνει την επίδοση του συστήματος. Αυτή η προσπάθεια όμως σύντομα οδήγησε σε αδιέξοδο καθώς σε πολλές περιπτώσεις το κόστος και η πολυπλοκότητα του software ήταν τόσο υψηλή που ανταγωνιζόταν αυτήν του hardware [18].

Την δεκαετία του 1980, το κόστος της ανάπτυξης software για τον supercomputer Cray ήταν ισάξιο του κόστους του hardware. Ήταν το πρώτο μεγάλο πλήγμα στην ανάπτυξη ειδικών λειτουργικών συστημάτων και οδήγησε σταδιακά στην υιοθέτηση ενός software γενικού σκοπού. Στα μέσα της ίδιας δεκαετίας εμφανίστηκε το πρώτο μεγάλο κύμα προς αυτήν την κατεύθυνση, καθώς το UNIX άρχισε να αντικαθιστά τα ειδικά λειτουργικά συστήματα [19].

Η επικράτηση του UNIX οφείλεται κυρίως στην διαδραστικότητα που προσέφερε στους χρήστες και στη χρησιμοποίηση της υψηλού επιπέδου γλώσσας προγραμματισμού C έναντι των γλωσσών assembly που χρησιμοποιούσαν παλαιότερα λειτουργικά συστήματα. Όμως, κάθε εταιρεία που υιοθετούσε το UNIX ως λειτουργικό σύστημα το τροποποιούσε σύμφωνα με τις ανάγκες της. Έτσι, λόγω των διαφορετικών αρχιτεκτονικών που χρησιμοποιούνταν δεν ήταν εύκολη η ανάπτυξη ενός καθολικού λειτουργικού συστήματος UNIX, που θα ικανοποιούσε όλα τα συστήματα μεγάλης κλίμακας [18].

Η σταθεροποίηση των λειτουργικών συστημάτων γενικού σκοπού απάλλαξε πολλούς ιδιοκτήτες supercomputer από το κόστος επαναδημιουργίας λειτουργικού συστήματος. Διαρκώς νέα χαρακτηριστικά εντάσσονταν στο UNIX, κάνοντας το πιο φιλικό και λειτουργικό. Αυτό είχε όμως ως συνέπεια την επιμήκυνση του κώδικά του, που όταν ξεπέρασε τις 500000 γραμμές δυσκόλευε πλέον την συντήρηση και την χρήση. Αυτό οδήγησε στην εμφάνιση των μικροπυρήνων (microkernels), που χρησιμοποιούσαν ένα πολύ μικρό σύνολο συναρτήσεων λειτουργικού συστήματος. Παραδείγματα συστημάτων που χρησιμοποιούσαν *microkernels* είναι το MACH στο Carnegie Mellon University και το Chorus στο INRIA [18]. Σταδιακά άρχιζαν να εμφανίζουν και άλλες παραλλαγές του UNIX, με πιο σημαντικές το AIX της IBM και το ανοιχτού λογισμικού λειτουργικό σύστημα Linux. Στον 21^{ο} αιώνα το Linux κατέχει πλέον την «μερίδα του λέοντος» στον χώρο των συστημάτων μεγάλης κλίμακας, κυρίως λόγω της ευκολίας της χρήσης του καθώς και επειδή δεν απαιτεί καταβολή τέλους για την αγορά της άδειας και δεν επιβαρύνεται με περιοριστικές συμφωνίες πνευματικής ιδιοκτησίας.

Η σύγχρονη τάση στα λειτουργικά συστήματα των συστημάτων μεγάλης κλίμακας είναι ο διαχωρισμός των λειτουργικών συστημάτων, καθώς οι σύγχρονοι supercomputers χρησιμοποιούν πολλά είδη διαφορετικών κόμβων: υπολογιστικούς κόμβους, κόμβους εισόδου/εξόδου (I/O). Έτσι, οι σύγχρονοι supercomputers χρησιμοποιούν «ελαφριούς» πυρήνες (kernels), όπως τον CNK ή τον CNL στους υπολογιστικούς κόμβους που έχουν απλό σχεδιασμό, για να επιτύχουν αποδοτική λειτουργία. Οι υπολογιστικοί κόμβοι δεν έχουν ούτε αρχεία εξόδου. Για την είσοδο/έξοδο δεδομένων υπεύθυνοι είναι οι κόμβοι εισόδου/εξόδου που συνήθως τρέχουν Linux, αφού είναι αναγκαίο να υποστηρίζουν ταυτόχρονη εκτέλεση πολλών ενεργειών (multitasking) [20, 21].

2.4 Επίδοση

Ανάλογα με τις επιδόσεις τα συστήματα μεγάλης κλίμακας διακρίνονται σε 2 κατηγορίες:

- **Συστήματα που βασίζονται στην δυναμικότητα (capability):** Πρόκειται για συστήματα που αφιερώνουν όλη την υπολογιστική τους δυναμικότητα στην επίλυση ενός μοναδικού προβλήματος μεγάλης διάστασης στο συντομότερο δυνατό χρόνο. Συχνά χρησιμοποιούνται για την επίλυση προβλημάτων που δεν μπορούν να λυθούν από συμβατικούς υπολογιστές, όπως για παράδειγμα η πρόγνωση και η μοντελοποίηση των καιρικών φαινομένων [23].
- **Συστήματα που βασίζονται στην χωρητικότητα (capacity):** Σε αντίθεση με τα capability HPC, τα capacity χρησιμοποιούν την υπολογιστική τους δυναμικότητα για να επιλύσουν έναν μεγάλο αριθμό μικρότερων προβλημάτων, όπως για παράδειγμα η πρόσβαση πολλών χρηστών σε βάσεις δεδομένων ή σε web site [23].

Η επίδοση των HPC συστημάτων μετριέται σε FLOPS (Floating Operations Per Second, πράξεις κινητής υποδιαστολής ανά δευτερόλεπτο). Από το 1993 οι γρηγορότεροι supercomputers ταξινομούνται στην λίστα του *TOP500* [5], σύμφωνα με τα αποτελέσματα του *Linpack benchmark* [24]. Βέβαια αυτή η λίστα δεν είναι απόλυτη, αφού το *Linpack benchmark* εκτελεί ουσιαστικά τον αλγόριθμο LU decomposition για έναν μεγάλο πίνακα προσφέροντας έτσι μια ένδειξη για την επίδοση σε πραγματικά προβλήματα. Σε άλλες εφαρμογές που μπορεί να απαιτούν γρηγορότερους υπολογισμούς ακεραίων, μεγαλύτερο memory bandwidth ή καλύτερη επίδοση του συστήματος εισόδου εξόδου, τα αποτελέσματα μπορεί να είναι αρκετά διαφορετικά.

2.5 Που χρησιμοποιούνται;

Η παγκόσμια αγορά των HPC συστημάτων αναπτύσσεται ραγδαία. Σύμφωνα με μια αναφορά που εξέδωσε το International Data Corporation (IDC) το 2009, τα έσοδα από τις πωλήσεις HPC συστημάτων για το 2008 υπολογίζονται σε 9.77 δισεκατομμύρια $ [22]. Η ίδια αναφορά προτείνει έναν διαχωρισμό των συστημάτων ανάλογα με το μέγεθός τους και υπολογίζει τα μερίδιά τους στην αγορά. Οι αντίστοιχοι πίνακες παρουσιάζονται παρακάτω:

Πίνακας 2.1 Διαχωρισμός HPC συστημάτων

Σύστημα	Περιγραφή
Supercomputers	Τιμή πώλησης πάνω από 500.000$, >= 512 κόμβοι
Divisional	Τιμή πώλησης από 250.000$ μέχρι 499.999, 128-512 κόμβοι
Departmental	Τιμή πώλησης από 100.000$ μέχρι 249.999, 16-128 κόμβοι
Workgroup	Τιμή πώλησης κάτω από 100.000$, 16 ή λιγότεροι κόμβοι

Πίνακας 2.2 Μερίδια αγοράς HPC συστημάτων

Σύστημα	Τμήμα Αγοράς (%)	Εκτιμώμενη Ποσότητα
Supercomputers	27	$\frac{\$9{,}77 \times 10^9 \times 0{,}27}{\$5 \times 10^5}$ = **5.279 Servers**
Divisional	14	$\frac{\$9{,}77 \times 10^9 \times 0{,}14}{\$3{,}75 \times 10^5}$ = **3.647 Servers**
Departmental	38	$\frac{\$9{,}77 \times 10^9 \times 0{,}38}{\$1{,}75 \times 10^5}$ = **21.215 Servers**
Workgroup	21	$\frac{\$9{,}77 \times 10^9 \times 0{,}21}{\$0{,}5 \times 10^5}$ = **41.034 Servers**

Τα HPC συστήματα χρησιμοποιούν κυρίως η βιομηχανία, οι κυβερνήσεις και οι ακαδημαϊκές κοινότητες. Παρακάτω παρουσιάζεται μία λίστα με τους σημαντικότερους τομείς όπου βρίσκουν εφαρμογή [3]:

- **Βιοεπιστήμες και ανθρώπινο γονιδίωμα:** Ανακάλυψη φαρμάκων, ανίχνευση/πρόληψη ασθενειών
- **Προγράμματα μηχανικών (CAE):** Σχεδιασμός και δοκιμή αυτοκίνησης, σχεδιασμός οδικών δικτύων, δομικός σχεδιασμός, μηχανολογικός σχεδιασμός
- **Χημική μηχανική:** Σχεδιασμός διεργασιών, μοριακός σχεδιασμός
- **Οικονομικά:** Ανάλυση ρίσκου, αυτόματες συνδιαλλαγές
- **Ηλεκτρονική:** Σχεδιασμός και επαλήθευση ηλεκτρονικών στοιχείων
- **Γεωεπιστήμες και γεωμηχανική:** Εξερεύνηση πετρελαίου και αερίου, μοντελοποήση δεξαμενών
- **Μηχανολογικός σχεδιασμός:** Δισδιάστατος και τρισδιάστατος σχεδιασμός και επαλήθευση, μοντελοποίηση
- **Άμυνα και ενέργεια:** Πυρηνική διαχείριση, βασική και εφαρμοσμένη έρευνα
- **Κυβερνητικά εργαστήρια:** Βασική και εφαρμοσμένη έρευνα
- **Πανεπιστήμια:** Βασική και εφαρμοσμένη έρευνα
- **Πρόγνωση καιρού:** Βραχυπρόθεσμη πρόγνωση και μοντελοποίηση κλίματος

3 Χρονοδρομολόγηση σε HPC Clusters

Τα τελευταία χρόνια παρατηρείται όλο και μεγαλύτερη στροφή στην κατασκευή συστημάτων μεγάλης κλίμακας, με αποτέλεσμα να αυξάνεται συνεχώς ο αριθμός των HPC clusters, προκειμένου να ικανοποιηθούν οι υπολογιστικές ανάγκες. Για αυτό είναι απαραίτητη η δημιουργία προηγμένων χρονοδρομολογητών (schedulers), που θα βοηθήσουν στην καλύτερη αξιοποίηση των πόρων (resource utilization) και ποιότητα των παρεχόμενων υπηρεσιών (Quality of Service). Στο κεφάλαιο αυτό θα κάνουμε μια σύντομη εισαγωγή στη χρονοδρομολόγηση (scheduling) εργασιών σε HPC clusters, θα ερμηνεύσουμε κάποιους χρήσιμους όρους για την κατανόηση των παρακάτω, θα αναφερθούμε σε τεχνικές χρονοδρομολόγησης που χρησιμοποιούνται ευρέως σήμερα και τις απαραίτητες προϋποθέσεις που πρέπει να εκπληρώνει ένας καλός χρονοδρομολογητής.

3.1 Κατηγοριοποίηση

Δύο είναι τα κύρια στοιχεία υλικού στα HPC clusters: οι κόμβοι και διασυνδέσεις δικτύου. Ανάλογα με το τι υλικά χρησιμοποιούνται, τα clusters χωρίζονται σε 2 κατηγορίες που διαφοροποιούνται ως προς τις ανάγκες χρονοδρομολόγησης [25].

Η 1η κατηγορία ονομάζεται **high-throughput computing clusters** και πρόκειται για clusters με υψηλή διεκπεραιωτική ικανότητα. Οι clusters αυτής της κατηγορίας συνδέουν ένα μεγάλο αριθμό κόμβων με πιο αργές διασυνδέσεις (low-end interconnections). Σε αυτά τα συστήματα ο κύριος στόχος είναι η αύξηση του ρυθμού διεκπεραίωσης εργασιών (throughput), περιορίζοντας την ανισοκατανομή του φορτίου στους υπολογιστικούς κόμβους. Η ισοκατανομή του φορτίου είναι ιδιαίτερα σημαντική εάν το cluster διαθέτει ετερογενείς κόμβους. Οι high-throughput computing clusters είναι κατάλληλοι για εκτέλεση παράλληλων εφαρμογών μικρής κλίμακας (loosely coupled) που δεν έχουν υψηλές απαιτήσεις επικοινωνίας μεταξύ των υπολογιστικών κόμβων.

Η 2η κατηγορία ονομάζεται **high-performance computing clusters** και δίνει μεγαλύτερη έμφαση στην επίδοση των συστημάτων και όχι τόσο στον ρυθμό διεκπεραίωσης εργασιών. Σε αντίθεση με τα high-throughput computing clusters, συνδέουν πιο ισχυρούς υπολογιστικούς κόμβους με ταχύτερες διασυνδέσεις (fast interconnections). Οι υψηλές διασυνδέσεις σχεδιάζονται κατάλληλα ώστε να έχουν μικρή λανθάνουσα περίοδο (latency) και μεγάλο εύρος ζώνης (bandwidth). Σε αυτή την κατηγορία εκτός από την ισοκατανομή φορτίου, σημαντικό ρόλο παίζει και η μείωση του κόστους επικοινωνίας (communication overhead), με τη σωστή τοποθέτηση των εφαρμογών σε διαθέσιμους υπολογιστικούς κόμβους. Οι high-performance computing clusters είναι ιδανικοί για εκτέλεση εφαρμογών μεγαλύτερης κλίμακας (tightly coupled) που έχουν υψηλές απαιτήσεις σε επικοινωνία και συγχρονισμό.

3.2 Χρήσιμοι ορισμοί

Σε αυτό το υποκεφάλαιο παρατίθενται μερικοί χρήσιμοι ορισμοί που θα βοηθήσουν στην καλύτερη κατανόηση των παρακάτω [25]:

- **Daemon:** Στα multitasking λειτουργικά συστήματα υπολογιστών *daemon* είναι ένα πρόγραμμα που τρέχει ως μία διεργασία στο background, αντί να είναι υπό τον άμεσο έλεγχο ενός διαδραστικού χρήστη.
- **Χρόνος αναμονής (wait time):** Όταν μια εργασία υποβάλλεται στον resource manager, περιμένει ένα χρονικό διάστημα στην ουρά πριν χρονοδρομολογηθεί και εκτελεστεί. Αυτός ο χρόνος που σπαταλάει στην ουρά αναμονής ονομάζεται χρόνος αναμονής και εξαρτάται από αρκετούς παράγοντες, όπως η προτεραιότητα των εργασιών, το φορτίο του συστήματος και η διαθεσιμότητα των πόρων.
- **Χρόνος ολοκλήρωσης (turnaround time):** Είναι το χρονικό διάστημα μεταξύ της υποβολής μιας εργασίας και της ολοκλήρωσής της. Ο χρόνος ολοκλήρωσης εμπεριέχει τόσο τον χρόνο αναμονής όσο και τον χρόνο εκτέλεσης της εργασίας.
- **Χρησιμοποίηση πόρων (resource utilization):** Αντιπροσωπεύει την πραγματικά χρήσιμη δουλειά που έχει πραγματοποιηθεί κατά την διάρκεια ζωής μιας εργασίας.
- **Ρυθμός διεκπεραίωσης συστήματος (system throughput):** Ορίζεται ως ο αριθμός των εργασιών που ολοκληρώνονται σε μια μονάδα χρόνου.
- **Χρόνος απόκρισης (response time):** Αντιπροσωπεύει το πόσο γρήγορα ο χρήστης λαμβάνει απόκριση από το σύστημα, αφού η εργασία έχει υποβληθεί. Είναι μια πολύ σημαντική μετρική απόδοσης για τους χρήστες, οι οποίοι επιθυμούν να είναι όσο το δυνατόν μικρότερος. Αντίθετα, οι διαχειριστές του συστήματος ενδιαφέρονται περισσότερο για την συνολική χρησιμοποίηση των πόρων, καθώς θέλουν να μεγιστοποιήσουν την απόδοση του συστήματος, ιδιαίτερα στους high-performance computing clusters.

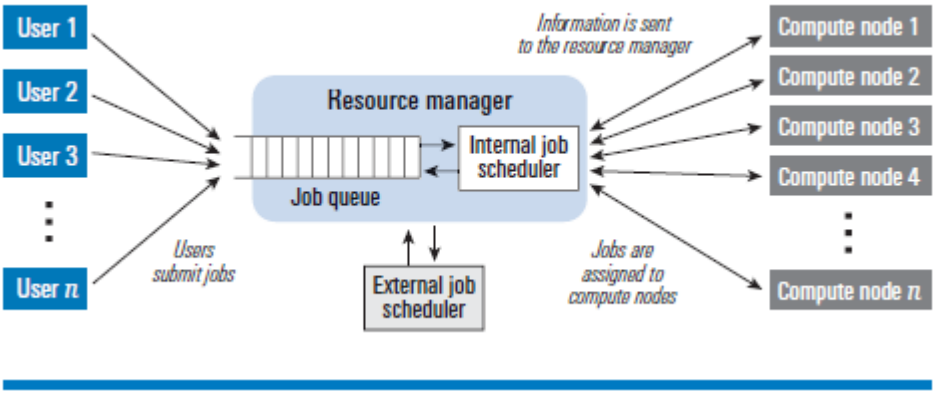
3.3 Σύστημα διαχείρισης πόρων

Το **σύστημα διαχείρισης πόρων (resource management system)** διαχειρίζεται τον εργασιακό φόρτο, εμποδίζοντας τις εργασίες (jobs) να ανταγωνίζονται μεταξύ τους για περιορισμένους υπολογιστικούς πόρους [25]. Αποτελείται από έναν **διαχειριστή πόρων (resource manager)** και έναν **χρονοδρομολογητή εργασιών (job scheduler)**. Σχεδόν όλοι οι resource managers διαθέτουν έναν ενσωματωμένο job scheduler, αλλά οι περισσότεροι διαχειριστές συστημάτων τον αντικαθιστούν συνήθως με έναν εξωτερικό για να βελτιώσουν την λειτουργικότητα του συστήματος. Σε κάθε περίπτωση, ο scheduler επικοινωνεί με τον resource manager για να πάρει πληροφορίες για τις ουρές (queues), τα φορτία των υπολογιστικών κόμβων και την διαθεσιμότητα των πόρων, ώστε να πάρει αποφάσεις για την χρονοδρομολόγηση εργασιών.

Συνήθως ο resource manager τρέχει αρκετούς daemon στον κύριο κόμβο και στους υπολογιστικούς κόμβους, συμπεριλαμβανομένου ενός scheduler daemon, που τρέχει συνήθως στον κύριο κόμβο. Ο resource manager δημιουργεί και ένα σύστημα ουράς (queuing system) ώστε να μπορούν οι χρήστες να υποβάλλουν εργασίες και ερωτήματα

για την κατάσταση των εργασιών. Επιπρόσθετα, ο resource manager διατηρεί μια λίστα από διαθέσιμους υπολογιστικούς πόρους και αναφέρει την κατάσταση των εργασιών που είχαν υποβληθεί προηγουμένως στον χρήστη. Βοηθάει στην οργάνωση των υποβληθέντων εργασιών ανάλογα με την προτεραιότητά τους, τους πόρους που ζητούν και την διαθεσιμότητα των πόρων.

Ο scheduler λαμβάνει περιοδικά στοιχεία από τον resource manager που αφορούν τις ουρές εργασιών και τους διαθέσιμους πόρους και φτιάχνει ένα πρόγραμμα που ορίζει την σειρά εκτέλεσης των εργασιών. Αυτό γίνεται πάντα σε συμφωνία με τις πολιτικές του διαχειριστή, που αφορούν το μέγεθος και το χρονοδιάγραμμα των πόρων για την εκτέλεση των εργασιών. Με βάση αυτές τις πληροφορίες, ο scheduler αποφασίζει ποια διεργασία θα εκτελεστεί, πότε και σε ποιον κόμβο. Ένα τυπικό resource management system παρουσιάζεται παρακάτω [25]:



Σχήμα 3.1 Ένα τυπικό resource management system

3.4 Χαρακτηριστικά ενός καλού χρονοδρομολογητή

Σε ένα τυπικό περιβάλλον παραγωγής υποβάλλονται στους clusters πολλά διαφορετικά είδη εργασιών. Αυτές οι διεργασίες μπορούν να χαρακτηριστούν από παράγοντες όπως ο αριθμός των επεξεργαστών που ζητούν (job size ή job width), η εκτίμηση του χρόνου εκτέλεσης, η προτεραιότητα, αν απαιτούν παράλληλη ή κατανεμημένη εκτέλεση, οι απαιτήσεις τους σε εισόδο/εξόδο [25].

Οι διαχειριστές καλούνται να δημιουργήσουν αρκετούς τύπους ουράς, που ο καθένας έχει διαφορετικό επίπεδο προτεραιότητας και ποιότητας παρεχόμενων υπηρεσιών (QoS). Για να δημιουργηθούν όμως «έξυπνοι» χρονοδρομολογητές χρειάζονται στοιχεία που αφορούν το μέγεθος των εργασιών, την προτεραιότητα και τον αναμενόμενο χρόνο εκτέλεσης (που παρέχονται από τον χρήστη), την άδεια πρόσβασης στους πόρους (όπως καθορίζεται από τον διαχειριστή του συστήματος) και την διαθεσιμότητα των πόρων (που λαμβάνεται από τον resource manager).

Στους high-performance computing clusters, η χρονοδρομολόγηση των παράλληλων εργασιών απαιτεί ιδιαίτερη προσοχή, καθώς οι παράλληλες εργασίες αποτελούνται από αρκετές δευτερεύουσες εργασίες (subtasks). Κάθε subtask ανατίθεται σε έναν μοναδικό υπολογιστικό κόμβο και οι κόμβοι επικοινωνούν συνεχώς μεταξύ τους κατά την

εκτέλεση της εργασίας. Ο τρόπος με τον οποίο τα subtasks υποβάλλονται στους κόμβους ονομάζεται χαρτογράφηση (mapping). Το mapping επηρεάζει σε μεγάλο βαθμό τον χρόνο εκτέλεσης και για αυτό ο χρονοδρομολογητής θα πρέπει να αναθέτει τα subtasks με προσοχή. Ο χρονοδρομολογητής θα πρέπει να εξασφαλίσει ότι οι κόμβοι που εκτελούν τα παράλληλα προγράμματα συνδέονται με γρήγορες διασυνδέσεις, ώστε να μειώνεται το κόστος επικοινωνίας. Για τις παράλληλες εργασίες, η αποδοτικότητα τους επηρεάζει επίσης και την χρησιμοποίηση των πόρων του συστήματος. Για να επιτύχουμε υψηλή χρησιμοποίηση πόρων είναι απαραίτητη τόσο η αποδοτικότητα των εργασιών, όσο και η προηγμένη χρονοδρομολόγηση.

Υπό συνθήκες υψηλού φορτίου είναι πολύ σημαντικό ο χρονοδρομολογητής να παρέχει μια δίκαιη μοιρασιά των πόρων του συστήματος στους χρήστες. Αυτό μπορεί να πραγματοποιηθεί χρησιμοποιώντας μια πολιτική δίκαιης μοιρασιάς (fair-share strategy), στην οποία ο χρονοδρομολογητής συλλέγει ιστορικά στοιχεία από εργασίες που πραγματοποιήθηκαν παλαιότερα και τα χρησιμοποιεί ώστε να προσαρμόσει δυναμικά την προτεραιότητα των εργασιών στην ουρά. Η ικανότητα του χρονοδρομολογητή να κάνει αλλαγές δυναμικά στην ουρά των εργασιών εξασφαλίζει την δίκαιη κατανομή των πόρων στους χρήστες.

Οι περισσότεροι χρονοδρομολογητές έχουν αρκετές παραμέτρους που μπορεί να προσαρμοστούν ώστε να ελέγχουν τις ουρές διεργασιών και τους αλγορίθμους χρονοδρομολόγησης, έτσι ώστε να παρέχουν διαφορετικούς χρόνους απόκρισης και ποσοστό χρησιμοποίησης. Συνήθως, υψηλή χρησιμοποίηση συστήματος μεταφράζεται σε μεγάλο μέσο χρόνο απόκρισης για τις εργασίες και όσο η χρησιμοποίηση συστήματος αυξάνεται τόσο ο μέσος χρόνος απόκρισης αυξάνεται πέρα από ένα συγκεκριμένο κατώφλι. Αυτό το κατώφλι εξαρτάται από τους αλγόριθμους επεξεργασίας των εργασιών και από το προφίλ των εργασιών. Στις περισσότερες περιπτώσεις, η βελτίωση της χρησιμοποίησης των πόρων έρχεται σε αντίθεση με την μείωση του χρόνου ολοκήρωσης. Η πρόκληση λοιπόν βρίσκεται στη ταυτόχρονη βελτίωση της χρησιμοποίησης των πόρων και διατήρηση αποδεκτών χρόνων απόκρισης των χρηστών. Ο παρακάτω πίνακας συνοψίζει τα επιθυμητά χαρακτηριστικά ενός καλού χρονοδρομολογητή [25]:

Πίνακας 3.1 Βασικά χαρακτηριστικά ενός καλού χρονοδρομολογητή

Χαρακτηριστικό	**Σχολιασμός**
Ευρύ πεδίο	Η φύση των εργασιών που υποβάλλονται σε ένα cluster μπορεί να ποικίλει, οπότε ο χρονοδρομολογητής θα πρέπει να υποστηρίζει διαφορετικά είδη εργασιών (πχ παράλληλα, κατανεμημένα, διαδραστικά ή μη) με την ίδια αποτελεσματικότητα.

Υποστήριξη αλγορίθμων	Ο χρονοδρομολογητής θα πρέπει να υποστηρίζει πολυάριθμους αλγορίθμους επεξεργασίας εργασιών (όπως FCFS, SJF, LJF, advance reservation, backfill). Επιπρόσθετα, ο χρονοδρομολογητής θα πρέπει να είναι σε θέση να εναλλάσσει αλγορίθμους και να εφαρμόζει διαφορετικούς αλγορίθμους σε διαφορετικές χρονικές στιγμές ή σε διαφορετικές ουρές ή και τα δύο.
Ευαισθησία στην αρχιτεκτονική των υπολογιστικών κόμβων και των διασυνδέσεων	Ο χρονοδρομολογητής πρέπει να ταιριάξει την κατάλληλη αρχιτεκτονική των υπολογιστικών κόμβων στο προφίλ των εργασιών, πχ χρησιμοποιώντας περισσότερους από έναν επεξεργαστές για να παρέχει βέλτιστη επίδοση για εφαρμογές που μπορούν να χρησιμοποιήσουν επιπλέον επεξεργαστές αποδοτικά.
Κλιμακωσιμότητα	Ο χρονοδρομολογητής θα πρέπει να είναι ικανός να κλιμακώνει σε χιλιάδες κόμβους και να επεξεργάζεται χιλιάδες εργασίες ταυτόχρονα.
Δυνατότητα για δίκαιη μοιρασιά (fair-share)	Ο χρονοδρομολογητής πρέπει να κατανέμει δίκαια τους πόρους σε συνθήκες υψηλού φορτίου και σε διαφορετικές χρονικές στιγμές.
Αποδοτικότητα	Το κόστος που σχετίζεται με την χρονοδρομολόγηση θα πρέπει να είναι αμελητέο και μέσα σε αποδεχτά πλαίσια. Προηγμένοι αλγόριθμοι χρονοδρομολόγησης μπορεί να χρειάζονται αρκετή ώρα για να τρέξουν. Για να είναι αποδοτικοί, οι αλγόριθμοι χρονοδρομολόγησης θα πρέπει να τρέχουν λιγότερο χρόνο από την αναμενόμενη εξοικονόμηση στο χρόνο εκτέλεσης εφαρμογών λόγω της βελτιωμένης χρονοδρομολόγησης.
Δυναμική ικανότητα προσθαφαίρεσης κόμβων	Ο χρονοδρομολογητής θα πρέπει να είναι ικανός να προσθέτει ή να αφαιρεί υπολογιστικούς πόρους δυναμικά (on the fly), υποθέτοντας ότι η εργασία μπορεί να προσαρμοστεί και να αξιοποιήσει τους επιπλέον πόρους.
Preemption	Επιτρέπει σε εργασίες μεγαλύτερης προτεραιότητας να αντικαταστήσουν εργασίες μικρότερης προτεραιότητας που βρίσκονται σε εκτέλεση. Μπορεί να συμβεί σε πολλά επίπεδα, πχ μπορεί μια εργασία να αναβληθεί ενώ τρέχει. Το checkpointing, δηλαδή η δυνατότητα να σταματήσει την εκτέλεση μιας εργασίας, να σώσει τα ενδιάμεσα αποτελέσματα και να την ξαναρχίσει αργότερα, μπορεί να εξασφαλίσει ότι τα αποτελέσματα δεν χάνονται.

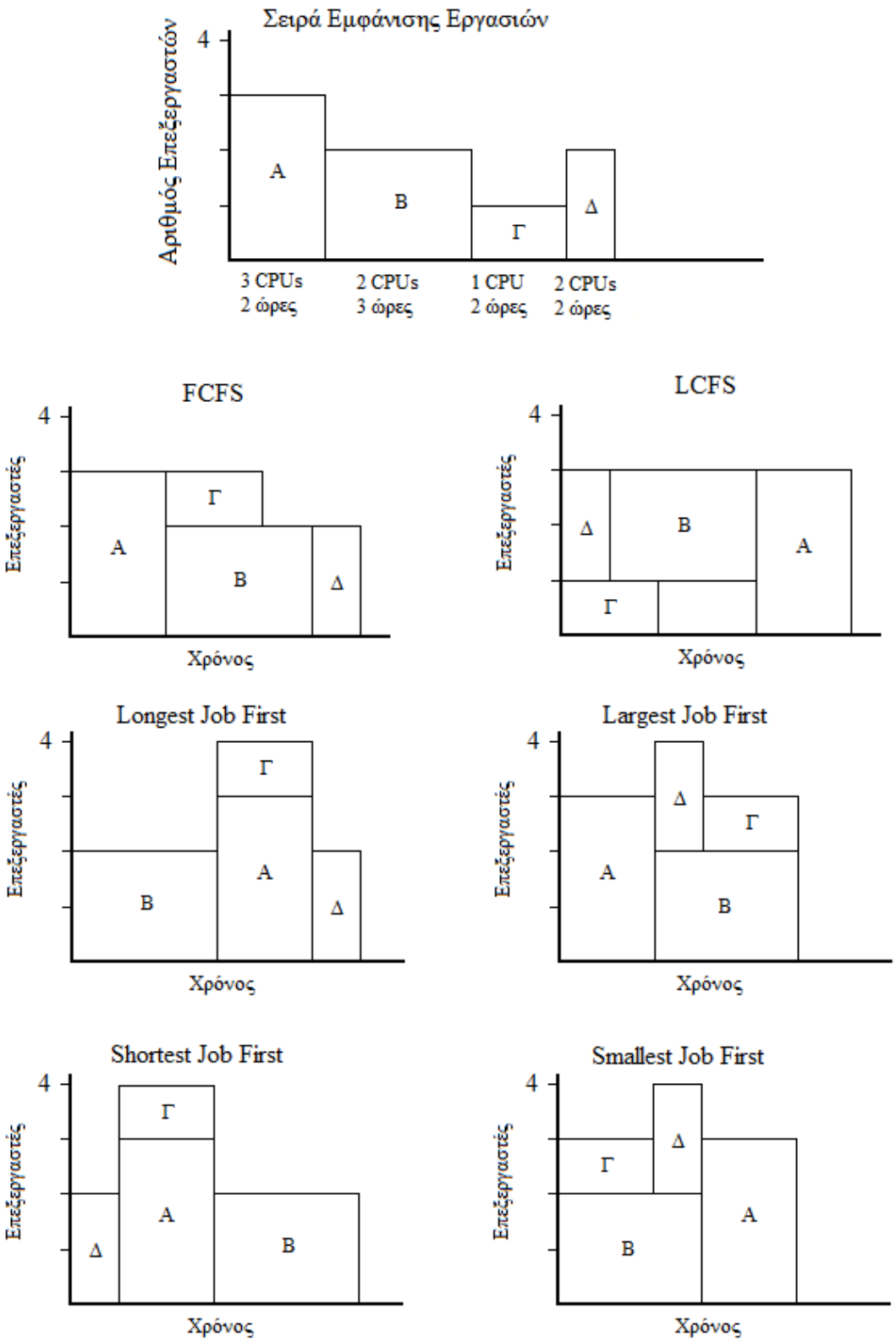
3.5 Αλγόριθμοι χρονοδρομολόγησης

Η κοινότητα παράλληλων και κατανεμημένων συστημάτων έχει αφιερώσει ουσιώδη ερευνητική προσπάθεια στην ανάπτυξη και στην κατανόηση αλγορίθμων χρονοδρομολόγησης εργασιών. Σήμερα, αρκετοί από αυτούς τους αλγορίθμους έχουν εφαρμοσθεί τόσο σε εμπορικούς χρονοδρομολογητές όσο και σε ανοιχτού κώδικα.

3.5.1 Βασικοί αλγόριθμοι χρονοδρομολόγησης

Οι αλγόριθμοι χρονοδρομολόγησης μπορούν να διακριθούν σε 2 κατηγορίες: τους αλγορίθμους καταμερισμού χρόνου (time-sharing) και τους αλγορίθμους καταμερισμού χώρου (space-sharing). Οι αλγόριθμοι καταμερισμού χρόνου χωρίζουν τον χρόνο σε ένα επεξεργαστή σε διακριτά χρονικά διαστήματα, που ονομάζονται slots. Τα slots στην συνέχεια ανατίθενται σε διαφορετικές εργασίες. Ως εκ τούτου, διαφορετικές εργασίες οποιαδήποτε στιγμή μπορούν να μοιράζονται τους ίδιους υπολογιστικούς πόρους. Αντίθετα, οι αλγόριθμοι καταμερισμού χώρου δίνουν τους ζητούμενους υπολογιστικούς πόρους σε μία εργασία μέχρι αυτή να ολοκληρωθεί.

Συνηθισμένοι αλγόριθμοι κατανομής χώρου είναι: first come - first served (FCFS), round robin στους κόμβους και όχι στον χρόνο (RR), shortest job first (SJF), longest job first (LJF). Όπως υποδηλώνει και το όνομα, ο FCFS εκτελεί τις εργασίες με την σειρά που υποβλήθηκαν στην ουρά. Είναι μια πολύ απλή στρατηγική ως προς την υλοποίηση και δουλεύει αρκετά καλά για χαμηλό φόρτο εργασιών. Ο RR αναθέτει τις εργασίες στους κόμβους όπως έρχονται, με κυκλικό τρόπο. Ο SJF ταξινομεί περιοδικά τις εισερχόμενες εργασίες και εκτελεί πρώτα τις πιο σύντομες εργασίες, βελτιώνοντας τον turnaround time τους. Ωστόσο, αυτή η στρατηγική προκαλεί καθυστερήσεις στις μεγαλύτερης διάρκειας εργασίες. Αντίθετα, ο LJF τείνει να μεγιστοποιήσει την χρησιμοποίηση του συστήματος με κόστος την αύξηση του turnaround time. Επειδή όμως οι προβλέψεις για τους χρόνους εκτέλεσης των εργασιών προέρχονται από εκτιμήσεις των χρηστών και συνήθως είναι υπερεκτιμημένες, οι αλγόριθμοι largest job first και smallest job first προτιμούνται από τους αλγορίθμους longest job first και shortest job first. Ο $1^{ος}$ αλγόριθμος ευνοεί τις εργασίες που απαιτούν μεγάλο αριθμό επεξεργαστών και ο $2^{ος}$ αυτές που απαιτούν μικρό αριθμό επεξεργαστών. Υπάρχουν και άλλοι αλγόριθμοι που χρησιμοποιούνται περιστασιακά για να ικανοποιήσουν τις πολιτικές του κάθε site. Στο [26] οι συγγραφείς ερευνούν τις διαφορές στην συμπεριφορά μεταξύ των shortest-remaining-processing-time (SRPT), FCFS, last come – first served (LCFS), SJF, foreground-background (FB), preemptive last come – first served (P-LCFS). Στο παρακάτω σχήμα παρουσιάζουμε ένα παράδειγμα εφαρμογής των σημαντικότερων αλγορίθμων στο ίδιο εργασιακό φορτίο, χωρίς την εφαρμογή προηγμένων αλγοριθμικών τεχνικών.



Σχήμα 3.2 Εφαρμογή των πιο συνηθισμένων αλγορίθμων χρονοδρομολόγησης

3.5.2 Προηγμένες αλγοριθμικές τεχνικές

Οι βασικοί αλγόριθμοι που περιγράφθηκαν παραπάνω μπορούν να βελτιωθούν όταν συνδυαστούν με προηγμένες αλγοριθμικές τεχνικές. Μερικές από αυτές είναι οι *advance reservation, backfill, preemption, checkpointing, node sets*.

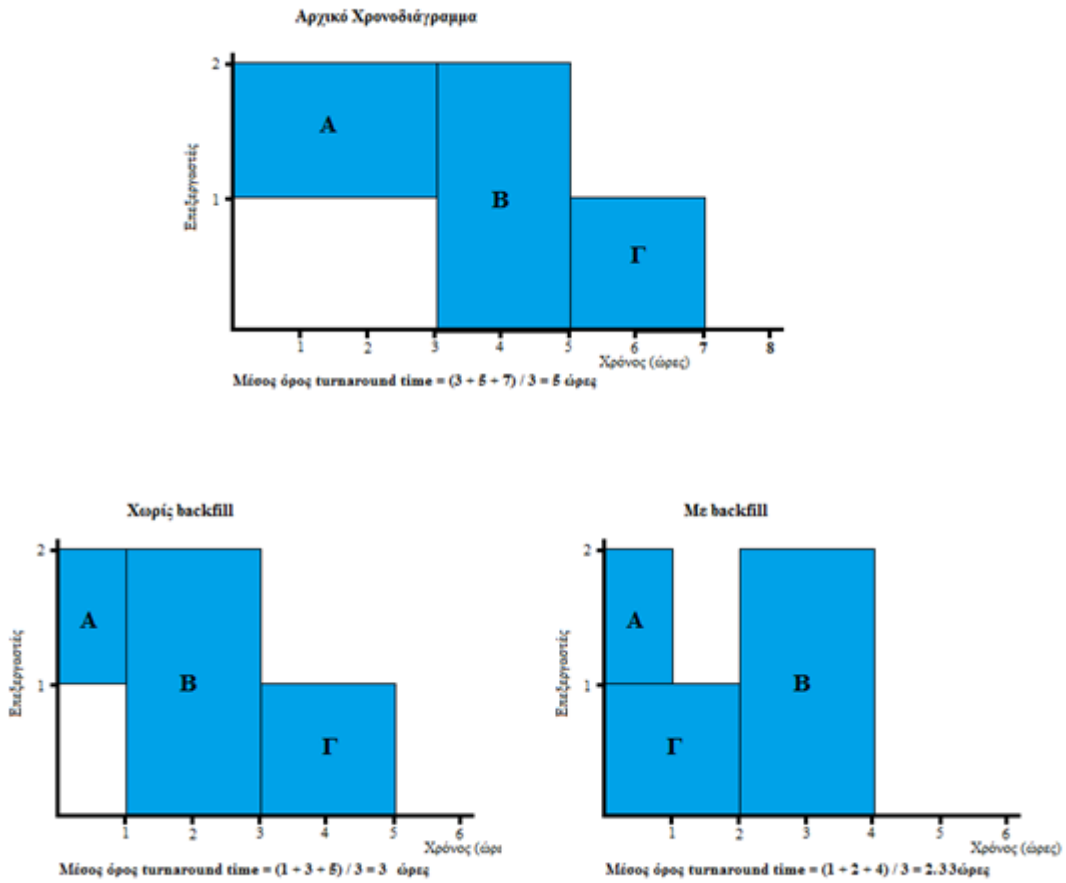
Ο αλγόριθμος *advance reservation* χρησιμοποιεί προβλέψεις των χρηστών για τους χρόνους εκτέλεσης των εργασιών ώστε να κρατήσει πόρους του συστήματος (όπως πχ CPUs ή μνήμη) για κάποιες εργασίες και να δημιουργήσει ένα χρονοδιάγραμμα. Είναι ευθύνη του χρονοδρομολογητή να εξασφαλίσει ότι αυτές οι κρατήσεις δεν θα παραβιαστούν.

Η τεχνική *backfill* επιτρέπει στον χρονοδρομολογητή να εκμεταλλευτεί σε μεγαλύτερο βαθμό τους διαθέσιμους πόρους του συστήματος εκτελώντας εργασίες εκτός σειράς. Δεδομένου ενός περιβάλλοντος όπου είναι σε εφαρμογή ο αλγόριθμος advance reservation και υπάρχουν λίστες με εργασίες χαμηλής και υψηλής προτεραιότητας, ο αλγόριθμος backfill προσπαθεί να «γεμίσει» τα κενά της χρονοδρομολόγησης με εργασίες χαμηλότερης προτεραιότητας, που έχουν μικρότερες απαιτήσεις σε πόρους συστήματος και χρόνο εκτέλεσης. Αυτό δεν μεταβάλλει την ακολουθία των εργασιών που είχαν ήδη προγραμματιστεί, αλλά βελτιώνει την χρησιμοποίηση του συστήματος, τρέχοντας εργασίες χαμηλής προτεραιότητας στο ενδιάμεσο της εκτέλεσης εργασιών υψηλότερης προτεραιότητας. Σε ένα τυπικό σύστημα η ενεργοποίηση του backfill μπορεί να αυξήσει την χρησιμοποίηση του συστήματος έως και 20% και τον turnaround time κατά ακόμα μεγαλύτερο ποσοστό. Λόγω του τρόπου λειτουργίας του, ο αλγόριθμος backfill ευνοεί κυρίως τις μικρότερες εργασίες, ενώ δεν επηρεάζει ιδιαίτερα τις μεγαλύτερες. Είναι αρκετά κοινό έως και το 90% των μικρότερων εργασιών να εκτελούνται νωρίτερα λόγω του αλγόριθμου backfill. Για την χρησιμοποίηση του αλγορίθμου, ο χρονοδρομολογητής απαιτεί μια πρόβλεψη από τον χρήστη για την διάρκεια των εργασιών, κατά την υποβολή τους.

Υπάρχουν όμως μειονεκτήματα στον αλγόριθμο backfill; Υπάρχουν, αλλά είναι αμελητέα μπροστά στα πλεονεκτήματα. Πρώτον, παραβιάζει την προτεραιότητα των εργασιών που έχει διαλέξει το site και μπορεί να απομακρύνει από την κατεύθυνση του φόρτου εργασίας που θέλει να επιβάλλει. Δεύτερον, μπορεί η ώρα έναρξης της εργασίας με την υψηλότερη προτεραιότητα να προστατεύεται από τον αλγόριθμο advance reservation, αλλά τι εμποδίζει την $3^{η}$ σε προτεραιότητα εργασία να ξεκινήσει νωρίτερα, καθυστερώντας την $2^{η}$ σε προτεραιότητα; Αυτό είναι ένα πρόβλημα που λύνεται σχετικά εύκολα, αυξάνοντας το βάθος του advance reservation, ώστε να γίνονται κρατήσεις για μεγαλύτερο αριθμό εργασιών.

Το $3^{ο}$ μειονέκτημα ονομάζεται *ψευδοκαθυστέρηση* (pseudo-delay) [28] και μπορεί να εξηγηθεί καλύτερα με ένα παράδειγμα. Έστω ένας κόμβος με 2 επεξεργαστές και 3 εργασίες Α, Β και Γ. Η Α είναι η εργασία με την υψηλότερη προτεραιότητα και απαιτεί 1 επεξεργαστή για 3 ώρες, η Β είναι η αμέσως επόμενη στην προτεραιότητα και απαιτεί 2 επεξεργαστές για 2 ώρες και η Γ είναι η εργασία με την χαμηλότερη προτεραιότητα και απαιτεί 1 επεξεργαστή για 2 ώρες. Οπότε ο χρονοδρομολογητής ξεκινάει την εργασία Α και κάνει μία κράτηση για την εργασία Β 3 ώρες μετά. Άρα η εργασία Γ μπορεί να ξεκινήσει νωρίτερα, λόγω του αλγορίθμου backfill. Όμως, η εργασία Α ολοκληρώνεται νωρίτερα, μετά από μία ώρα εκτέλεσης. Αυτό το φαινόμενο είναι πολύ συνηθισμένο, καθώς η χρήστες τείνουν να υπερβάλλουν στην εκτίμηση του χρόνου εκτέλεσης των εργασιών τους, για να αποφύγουν τον κίνδυνο του βίαιου τερματισμού

από τον χρονοδρομολογητή, χωρίς να έχει ολοκληρωθεί η εκτέλεσή τους. Οπότε η εργασία Β θα μπορούσε να ξεκινήσει άμεσα αλλά μπλοκάρετε από την εργασία Γ. Αυτό εννοούμε με τον όρο ψευδοκαθυστέρηση. Η εργασία Β θα ξεκινήσει τελικά στις 2 ώρες, δηλαδή 1 ώρα νωρίτερα από ότι είχε προγραμματιστεί. Έτσι έχουμε και καλύτερη χρησιμοποίηση του συστήματος και βελτίωση των χρόνων ολοκλήρωσης, όπως φαίνεται και στο παρακάτω σχήμα.



Σχήμα 3.3 Παράδειγμα ψευδοκαθυστέρησης λόγω backfill

Ο αλγόριθμος *preemption*, όπως είπαμε και προηγουμένως επιτρέπει σε εργασίες μεγαλύτερης προτεραιότητας να αντικαταστήσουν εργασίες μικρότερης, που βρίσκονται ήδη υπό εκτέλεση. Συνήθως πάει πακέτο με τον αλγόριθμο *checkpointing*, που επιτρέπει στον χρονοδρομολογητή να σταματήσει την εκτέλεση μιας εργασίας, να σώσει τα ενδιάμεσα αποτελέσματα και να την ξαναρχίσει αργότερα, εξασφαλίζοντας ότι τα αποτελέσματα δεν χάνονται.

Μια τελευταία τεχνική που χρησιμοποιείται ευρέως, ιδιαίτερα σε ετερογενείς clusters, είναι η τεχνική των node sets. Αυτή η τεχνική επιτρέπει στις εργασίες να ζητήσουν σύνολα ομοιόμορφων πόρων. Η ταχύτητα εκτέλεσης των περισσότερων παράλληλων εργασιών, που έχουν γραφεί σε δημοφιλής γλώσσες όπως το MPI, καθορίζεται από την ταχύτητα του πιο αργού κόμβου. Έτσι, η τεχνική των node sets βελτιώνει και την εκτέλεση των παράλληλων εργασιών, αφού επιτρέπει στον χρήστη να ζητήσει ομοιόμορφους πόρους υψηλού επιπέδου, και την χρησιμοποίηση του συστήματος. Για

παράδειγμα, έστω ότι ο χρονοδρομολογητής παρέχει σε μια εργασία 2 κόμβους, από τους οποίους ο ένας έχει επεξεργαστές με ταχύτητα 2 Ghz και ο άλλος επεξεργαστές με ταχύτητα 1 Ghz. Η ταχύτητα εκτέλεσης της εργασίας θα επηρεάζεται κυρίως από τους επεξεργαστές με την μικρότερη ταχύτητα, δηλαδή 1 Ghz. Άρα, οι επεξεργαστές των 2 Ghz θα είναι ουσιαστικά μια σπατάλη του συστήματος. Αν στο cluster υπάρχουν και άλλοι κόμβοι με ταχύτητες 1 ή 2 Ghz (ή κοντινές), τότε η τεχνική των node sets θα εξασφάλιζε και καλύτερη ταχύτητα εκτέλεσης, αφού θα μπορούσε να παραχωρήσει στην εργασία 2 κόμβους με ταχύτητα επεξεργαστών 2 Ghz, και καλύτερη χρησιμοποίηση των πόρων αφού δεν θα σπαταλούνταν άδικα πόροι του συστήματος. Σε συστήματα με αυξημένη ετερογένεια η τεχνική των node sets μπορεί να αυξήσει το throughput από 10% έως και 15% [29].

3.5.3 Πολιτικές χρονοδρομολόγησης

Μια άλλη σημαντική πτυχή της χρονοδρομολόγησης είναι οι πολιτικές [27]. Συνήθως κάθε site ακολουθεί τις δικές του πολιτικές. Για παράδειγμα στο [30], ερευνήθηκε η ιεραρχική χρονοδρομολόγηση, η οποία χρησιμοποιούσε χαρακτηρισμούς των εργασιών, ώστε να αναπτύξει ένα δέντρο ιεραρχίας χρονοδρομολόγησης, βασισμένο στο είδος των εργασιών. Στο [31], οι συγγραφείς πρότειναν μια τεχνική που ονομάζεται Flexible Co-Scheduling (FCS), όπου η εκτέλεση των εργασιών που απαιτούν υψηλή επικοινωνία μεταξύ των διεργασιών τους διακοπτόταν ενώ περίμεναν για συγχρονισμό των διεργασιών σε άλλες CPUs. Στο [32], ο συγγραφέας προτείνει μια προσαρμοστική πολιτική χρονοδρομολόγησης, όπου οι άεργοι επεξεργαστές σε ένα cluster ανακατανέμονται σε νέες εργασίες σε περιόδους υψηλού φορτίου και επιστρέφουν στο cluster όταν το φορτίο μειώνεται. Ένας αλγόριθμος που ονομάζεται Grid Backfilling προτάθηκε στο [33], ο οποίος χρησιμοποιεί εργασίες από πολλαπλά clusters, ώστε να κάνει backfill εργασίες σε γειτονικά clusters.

3.5.4 Queuing vs Planning

Η ανάπτυξη των παραπάνω αλγορίθμων οδήγησε σε μια μεγάλη αλλαγή στα συστήματα μεγάλης κλίμακας: περάσαμε από τα *queuing* συστήματα στα *planning*. Τα *queuing* συστήματα προσπαθούν να χρησιμοποιήσουν τους ελεύθερους πόρους για να ικανοποιήσουν τις υπάρχουσες αιτήσεις πόρων. Δεν υπάρχει μελλοντικός σχεδιασμός για όλες τις αιτήσεις που βρίσκονται σε αναμονή. Ως εκ τούτου, δεν υπάρχει εκτίμηση ώρας έναρξης για τις εργασίες που βρίσκονται σε αναμονή. Αντίθετα, τα *planning* συστήματα σχεδιάζουν και για το παρόν και για το μέλλον. Εκτιμήσεις για την ώρα έναρξης ανατίθενται σε όλες τις εργασίες και ένα πλήρες χρονοδιάγραμμα χρησιμοποίησης των πόρων υπολογίζεται και είναι διαθέσιμο στους χρήστες. Μια μεγαλύτερη ανάλυση των 2 συστημάτων παρουσιάζεται στο [34]. Ο παρακάτω πίνακας παραθέτει συνοπτικά τις διαφορές των 2 συστημάτων:

Πίνακας 3.2 Διαφορές Queuing και Planning συστημάτων

Ιδιότητα	Queuing	Planning
Χρονικός σχεδιασμός	Παρόν	Παρόν και Μέλλον
Υποβολή νέων αιτήσεων για πόρους συστήματος	Εισαγωγή στην ουρά	Επανασχεδιασμός
Εκτιμώμενος χρόνος έναρξης	Όχι	Σε όλες τις εργασίες
Εκτίμηση χρόνου εκτέλεσης	Όχι απαραίτητο	Υποχρεωτικό
Κρατήσεις	Όχι	Ναι
Backfill	Ναι	Ναι
Παραδείγματα	PBS, NQE/NQS, LL	CCS, Maui

3.6 Επιλογή εργαλείων για τα πειράματα

Παρακάτω γίνεται μια σύντομη περιγραφή των εργαλείων που χρησιμοποιήθηκαν για τα πειράματα. Όπως είπαμε και προηγουμένως, για την χρονοδρομολόγηση των συστημάτων μεγάλης κλίμακας απαραίτητα συστατικά στοιχεία είναι ένας διαχειριστής πόρων και ένας χρονοδρομολογητής.

3.6.1 Επιλογή διαχειριστή πόρων

Ο διαχειριστής πόρων παρέχει χαμηλού επιπέδου λειτουργικότητα για την εκκίνηση, την προσωρινή παύση, τη ματαίωση και την επίβλεψη των εργασιών. Χωρίς αυτές τις δυνατότητες ο χρονοδρομολογητής δεν μπορεί να ελέγξει τις εργασίες. Ως διαχειριστή πόρων επιλέξαμε τον TORQUE [35].

Ο TORQUE χρησιμοποιείται κυρίως για batch συστήματα. Τα batch συστήματα είναι μια συλλογή από επεξεργαστές και άλλους πόρους (δίκτυα, συστήματα αποθήκευσης) που λειτουργούν υπό το πρίσμα ότι το σύνολο είναι μεγαλύτερο από το άθροισμα των επιμέρους. Κάποια batch συστήματα αποτελούνται από μερικά μόνο μηχανήματα με έναν μόνο επεξεργαστή που διαχειρίζονται από τους ίδιους τους χρήστες. Άλλα πάλι αποτελούνται από χιλιάδες μηχανήματα που τρέχουν εργασίες χρηστών ενώ ταυτόχρονα

προσπαθούν να αποκτήσουν πρόσβαση σε συσκευές υλικού και αποθηκευτικά συστήματα. Τα batch συστήματα παρέχουν ένα μηχανισμό υποβολής, εκτέλεσης και παροχής πληροφοριών για εργασίες σε μοιραζόμενους πόρους. Η σωστή ρύθμιση των batch συστημάτων απαλλάσσει τους χρήστες από λεπτομέρειες που αφορούν την εκτέλεση και την διαχείριση εργασιών, επιτρέποντας υψηλότερη χρησιμοποίηση πόρων. Για παράδειγμα, αρκεί ο χρήστης να προσδιορίσει τους ελάχιστους πόρους που απαιτεί η εργασία του και δεν χρειάζεται να ξέρει τα ονόματα και τις ιδιότητες των κόμβων. Έτσι, τα batch συστήματα είναι ικανά να εκτελούν χιλιάδες εργασίες ταυτόχρονα.

Τα batch συστήματα αποτελούνται από 4 συστατικά στοιχεία που παρουσιάζονται στον παρακάτω πίνακα.

Πίνακας 3.3 Συστατικά στοιχεία των batch συστημάτων

Συστατικό στοιχείο	Περιγραφή
Κύριος κόμβος	Ένα batch σύστημα θα έχει έναν κύριο κόμβο όπου θα τρέχει ο *pbs_server*. Ανάλογα με τις ανάγκες των συστημάτων, ο κύριος κόμβος μπορεί να είναι αφοσιωμένος σε αυτήν την εργασία, ή μπορεί να εκπληρώνει και ρόλους των άλλων στοιχείων.
Κόμβοι υποβολής	Οι κόμβοι υποβολής παρέχουν ένα σημείο εισόδου στο σύστημα, ώστε οι χρήστες να διαχειρίζονται τις εργασίες τους. Σε αυτούς τους κόμβους οι χρήστες είναι ικανοί να υποβάλλουν και να ελέγχουν τις εργασίες τους. Επιπρόσθετα, κάποια sites έχουν 1 ή 2 κόμβους κρατημένους για διαδραστική χρήση, όπως η αντιμετώπιση προβλημάτων.
Υπολογιστικοί κόμβοι	Οι υπολογιστικοί κόμβοι είναι η κινητήριος δύναμη του συστήματος. Ο ρόλος τους είναι η εκτέλεση εργασιών. Σε κάθε υπολογιστικό κόμβο τρέχει το *pbs_mom*, που εκκινεί, τερματίζει και διαχειρίζεται τις υποβληθέντες εργασίες. Επικοινωνεί με τον pbs_server στον κύριο κόμβο. Ανάλογα με τις απαιτήσεις του συστήματος μπορεί να έχει και άλλους ρόλους.
Πόροι	Κάποια συστήματα οργανώνονται με σκοπό να διαχειρίζονται μια συλλογή από πόρους πέρα των υπολογιστικών κόμβων. Αυτοί οι πόροι μπορεί να είναι δίκτυα υψηλής ταχύτητας, αποθηκευτικοί χώροι, διαχειριστές αδειών κτλ. Η διαθεσιμότητα αυτών των πόρων είναι περιορισμένη και απαιτείται έξυπνη διαχείριση ώστε να υπάρχει δικαιοσύνη και αυξημένη χρησιμοποίηση.

Ο κύκλος ζωής μιας εργασίας στον Torque μπορεί να χωριστεί σε 4 φάσεις:

- **Δημιουργία:** Συνήθως η υποβολή των εργασιών γίνεται μέσω script, που καθορίζουν όλες τις παραμέτρους της εργασίας. Αυτοί οι παράμετροι μπορεί να καθορίζουν την διάρκεια της εργασίας, τους πόρους που απαιτούνται και τι πρέπει να εκτελεστεί.

- **Υποβολή:** Μια εργασία υποβάλλεται με την εντολή *qsub*. Μόλις υποβληθεί, οι πολιτικές που έχουν καθοριστεί από τους διαχειριστές και το τεχνικό προσωπικό καθορίζουν την προτεραιότητα της εργασίας και τον χρόνο έναρξης.
- **Εκτέλεση:** Οι εργασίες συνήθως καταναλώνουν το μεγαλύτερο διάστημα του κύκλου ζωής τους σε αυτό το στάδιο. Ενώ η εργασία εκτελείται, ο χρήστης μπορεί να ενημερωθεί για την κατάστασή της με την εντολή *qstat*.
- **Οριστικοποίηση:** Όταν μια εργασία ολοκληρώνεται τα αρχεία εξόδου (stdout) και σφάλματος (stderr) αντιγράφονται στον κατάλογο όπου έγινε η υποβολή των εργασιών.

Όπως αναλύσαμε και προηγουμένως, ένας TORQUE cluster αποτελείται από έναν κύριο κόμβο, που τρέχει το *pbs_server daemon,* και πολλούς υπολογιστικούς κόμβους, που τρέχουν τον *pbs_mom* daemon. Εντολές υποβολής και διαχείρισης εργασιών μπορούν να εγκατασταθούν σε οποιοδήποτε μηχάνημα, ακόμα και αν δεν τρέχει τον *pbs_server* ή τον *pbs_mom*.

Ο κύριος κόμβος τρέχει επίσης και έναν πρόγραμμα χρονοδρομολόγησης. Ο χρονοδρομολογητής αλληλεπιδρά με τον *pbs_server* για να πάρει αποφάσεις για την χρησιμοποίηση των πόρων και να κατανείμει υπολογιστικούς κόμβους στις εργασίες. Ένας απλός FIFO χρονοδρομολογητής, καθώς και κώδικας για την κατασκευή πιο σύνθετων χρονοδρομολογητών, παρέχεται στην διανομή του TORQUE. Οι περισσότεροι χρήστες του TORQUE όμως προτιμούν έναν πιο προηγμένο χρονοδρομολογητή, όπως τον Maui ή τον Moab.

Οι χρήστες υποβάλλουν εργασίες στον *pbs_server* χρησιμοποιώντας την εντολή *qsub*. Όταν ο *pbs_server* λαμβάνει μια καινούργια εργασία ενημερώνει τον χρονοδρομολογητή. Ο χρονοδρομολογητής βρίσκει κόμβους για την εργασία και στέλνει οδηγίες εκτέλεσης της εργασίας μαζί με μια λίστα από κόμβους στον *pbs_server*. Τότε ο *pbs_server* στέλνει την νέα εργασία στον πρώτο κόμβο της λίστας και δίνει εντολή για την εκτέλεση της εργασίας. Αυτός ο κόμβος ορίζεται υπεύθυνος για την εργασία και ονομάζεται ηγουμένη (Mother Superior). Οι υπόλοιποι κόμβοι που συμμετέχουν στην εκτέλεση της εργασίας ονομάζονται αδελφές μητέρες (sister moms).

3.6.2 Επιλογή χρονοδρομολογητή

Ως χρονοδρομολογητή διαλέξαμε τον Maui. Ο Maui είναι ένας προηγμένος χρονοδρομολογητής ανοιχτού κώδικα, σχεδιασμένος ώστε να βελτιστοποιεί την χρησιμοποίηση συστήματος σε ετερογενείς κόμβους με βάση διάφορες πολιτικές. Επικεντρώνεται κυρίως στην μείωση του χρόνου ολοκλήρωσης μεγάλων παράλληλων εργασιών, γεγονός που το καθιστά κατάλληλο για χρονοδρομολόγηση σε συστήματα μεγάλης κλίμακας. Τα χαρακτηριστικά του Maui που μας ενδιαφέρουν παρουσιάζονται αναλυτικότερα στο επόμενο κεφάλαιο.

4 Ο χρονοδρομολογητής Maui

Σε αυτό το κεφάλαιο παρουσιάζονται αναλυτικά κάποια βασικά χαρακτηριστικά του χρονοδρομολογητή Maui, δίνοντας ιδιαίτερη έμφαση στα χαρακτηριστικά που είναι απαραίτητα για την ολοκλήρωση του πειραματικού μέρους της εργασίας [28, 29, 36].

4.1 Φιλοσοφία

Στόχος του χρονοδρομολογητή από την ευρεία σκοπιά είναι να κάνει τους χρήστες, τους διαχειριστές και τους managers χαρούμενους. Οι χρήστες επιθυμούν να έχουν την ικανότητα να καθορίζουν τους πόρους που επιθυμούν, να έχουν χαμηλούς χρόνους ολοκλήρωσης στις εργασίες τους και να απολαμβάνουν αξιόπιστη κατανομή των πόρων. Οι managers επιθυμούν μεγιστοποίηση των κερδών της επένδυσης τους, που σημαίνει υψηλή χρησιμοποίηση συστήματος και την δυνατότητα να παρέχουν διαφορετική ποιότητα παροχής υπηρεσιών σε διαφορετικούς χρήστες και groups. Οι διαχειριστές επιθυμούν χαρούμενους χρήστες και managers. Θέλουν επίσης την δυνατότητα να γνωρίζουν τόσο τον φόρτο εργασίας όσο και τους διαθέσιμους πόρους. Αυτό περιλαμβάνει την παρούσα κατάσταση, προβλήματα, στατιστικά και πληροφορίες για το τι συμβαίνει στο παρασκήνιο. Χρειάζονται ένα εκτεταμένο σύνολο εργαλείων που θα τους επιτρέπει να εφαρμόζουν πολιτικές διαχείρισης και να συντονίζουν το σύστημα, ώστε να λαμβάνουν τα επιθυμητά χαρακτηριστικά.

Όπως είπαμε και προηγουμένως, τα batch συστήματα απλοποιούν την χρήση των κατανεμημένων πόρων ενός cluster, επιτρέποντας στους χρήστες να βλέπουν το cluster σαν ένα ενιαίο σύστημα, όσον αφορά την διαχείριση των εργασιών και το σύνολο των διαθέσιμων πόρων. Όμως, τα batch συστήματα θα πρέπει να κάνουν περισσότερα από το να παρέχουν μια καθολική εικόνα για το cluster. Όπως με πολλά μοιραζόμενα συστήματα, διάφορες περιπλοκές εμφανίζονται όταν προσπαθούμε να κατανέμουμε τους πόρους με έναν δίκαιο και αποτελεσματικό τρόπο. Αυτές οι περιπλοκές μπορεί να οδηγήσουν σε κακή επίδοση και σημαντικές ανισότητες στην χρήση. Σε ένα batch σύστημα ο χρονοδρομολογητής είναι υπεύθυνος για να αποφασίζει πότε, πού και πόσες εργασίες θα εκτελεστούν, ώστε να βελτιστοποιήσει την λειτουργία του cluster. Αυτές οι αποφάσεις αφορούν κυρίως 3 τομείς:

- **Έλεγχος της κυκλοφορίας:** Ο χρονοδρομολογητής είναι υπεύθυνος για να αποτρέπει τις εργασίες από το να παρεμβαίνουν μεταξύ τους. Εάν οι εργασίες επιτρέπεται να συναγωνίζονται για τους πόρους, θα προκαλέσουν στην γενική περίπτωση μείωση της επίδοσης του συστήματος, θα καθυστερήσουν την εκτέλεση εργασιών και πιθανότατα θα προκαλέσουν την αποτυχία εκτέλεσης ενός ή και περισσότερων εργασιών. Ο χρονοδρομολογητής είναι υπεύθυνος να εντοπίζει και να παρέχει αποκλειστική πρόσβαση των ζητούμενων πόρων σε μια εργασία, εμποδίζοντας την χρήση αυτών των πόρων από άλλες εργασίες.
- **Πολιτικές:** Οι clusters όπως και άλλα συστήματα μεγάλης κλίμακας δημιουργούνται συνήθως με συγκεκριμένους σκοπούς. Αυτοί οι σκοποί συνήθως καθορίζουν διάφορους κανόνες που αφορούν τον τρόπο λειτουργίας του συστήματος και ποιος ή τι επιτρέπεται να το χρησιμοποιεί. Για λόγους αποτελεσματικότητας, ο χρονοδρομολογητής θα πρέπει να παρέχει τους

μηχανισμούς ώστε το site να μπορεί να θέσει σε εφαρμογή τις πολιτικές που επιθυμεί μέσω της χρονοδρομολόγησης.

- **Βελτιστοποίηση:** Η υπολογιστική δύναμη ενός cluster είναι περιορισμένη και με την πάροδο του χρόνου είναι πολύ πιθανόν η ζήτηση να ξεπεράσει την παροχή. Έξυπνες αποφάσεις χρονοδρομολόγησης μπορούν να βελτιώσουν σημαντικά την αποδοτικότητα του συστήματος, με την αύξηση του αριθμού των εργασιών που εκτελούνται και μείωση του χρόνου εκτέλεσης. Παρά τους περιορισμούς του ελέγχου της κυκλοφορίας και των πολιτικών του site, είναι καθήκον του χρονοδρομολογητή να χρησιμοποιήσει οποιοδήποτε βαθμό ελευθερίας είναι διαθέσιμος ώστε να βελτιώσει την επίδοση τους συστήματος.

Το πόσο καλά συμπεριφέρεται ένας χρονοδρομολογητής μπορεί να καθοριστεί αν υπάρχουν εγκαταστημένοι μηχανισμοί επεξεργασίας στατιστικών. Ενώ οι στατιστικές είναι σημαντικές, η αξία τους είναι περιορισμένη αν δεν είναι διαθέσιμες βέλτιστες στατιστικές μετρήσεις για το τρέχον περιβάλλον, συμπεριλαμβανομένου του φόρτου εργασίας, των πόρων και των πολιτικών. Αν κάποιος μπορούσε να καθορίσει ότι ο τυπικός φόρτος εργασίας ενός site λαμβάνει ένα μέσο χρόνο αναμονής στην ουρά 3 ωρών σε ένα συγκεκριμένο σύστημα, αυτό θα ήταν μια καλή στατιστική. Όμως, αν κάποιος ήξερε ότι μέσω συντονισμού το σύστημα μπορούσε να πετύχει χρόνο αναμονής στην ουρά 1.2 ώρες με ασήμαντες αρνητικές επιπτώσεις, αυτό θα ήταν μια πολύτιμη γνώση.

Ο χρονοδρομολογητής Maui αναπτύχθηκε με εκτεταμένη δυνατότητα παροχής πληροφοριών σε χρήστες, διαχειριστές και managers. Στον πυρήνα του, είναι ένα εργαλείο σχεδιασμένο ώστε να διαχειρίζεται πραγματικά τους πόρους και να παρέχει χρήσιμες πληροφορίες για το τι γίνεται στην πραγματικότητα στο σύστημα. Δημιουργήθηκε για να ικανοποιήσει πραγματικές ανάγκες ενός διαχειριστή batch συστημάτων, καθώς προσπαθεί να ισορροπήσει τις ανάγκες των χρηστών, του προσωπικού και των managers.

4.2 Στοιχεία χρονοδρομολόγησης

Παρακάτω παρουσιάζονται κάποια βασικά στοιχεία χρονοδρομολόγησης που χειρίζεται ο Maui κατά την λειτουργία του.

4.2.1 Εργασίες

Οι πληροφορίες για τις εργασίες παρέχονται από τον διαχειριστή πόρων. Τα χαρακτηριστικά των εργασιών περιλαμβάνουν στοιχεία ιδιοκτησίας της εργασίας, της κατάστασης της εργασίας, του πλήθους και του τύπου των πόρων που απαιτούνται από την εργασία και μια εκτίμηση του χρόνου που απαιτούν οι πόροι. Μια εργασία αποτελείται από μία ή περισσότερες απαιτήσεις, η κάθε μία από τις οποίες απαιτεί έναν αριθμό πόρων συγκεκριμένου τύπου. Κάθε απαίτηση αποτελείται από ένα ή περισσότερα tasks, όπου ως task ορίζεται η μικρότερη δυνατή μονάδα ανεξάρτητων πόρων. Σε συμμετρικά πολυεπεξεργαστικά (SMP) περιβάλλοντα όμως, οι χρήστες ίσως επιθυμούν να συνδέσουν έναν ή περισσότερους επεξεργαστές με ένα συγκεκριμένο ποσό μνήμης και/ή άλλων πόρων.

4.2.2 Κόμβοι

Όσο αφορά τον Maui, οι κόμβοι είναι μια συλλογή από πόρους με ένα συγκεκριμένο σύνολο χαρακτηριστικών. Ένας κόμβος ορίζεται ως ένας ή περισσότεροι επεξεργαστές, μνήμη και πιθανότατα άλλοι υπολογιστικοί πόροι όπως τοπικοί δίσκοι, προσαρμογείς δικτύου, άδειες λογισμικού κτλ. Επιπρόσθετα, ο κόμβος μπορεί να περιγραφεί από ποικίλα άλλα χαρακτηριστικά όπως η αρχιτεκτονική του συστήματος και το λειτουργικό σύστημα. Οι κόμβοι ποικίλλουν στο μέγεθος από μονοεπεξεργαστικά συστήματα μέχρι μεγάλα (SMP) συστήματα, όπου κάθε κόμβος μπορεί να έχει εκατοντάδες επεξεργαστές και τεράστια ποσά μνήμης. Πληροφορίες για τους κόμβους παρέχονται στον χρονοδρομολογητή από τον διαχειριστή πόρων.

4.2.3 Advance Reservation

Ο όρος *advance reservation* υποδηλώνει ένα σύνολο συγκεκριμένων πόρων για συγκεκριμένη χρήση. Κάθε κράτηση (reservation) αποτελείται από μία λίστα πόρων, μια λίστα ελέγχου πρόσβασης και ένα χρονικό διάστημα για το οποίο θα εκτελείται. Η κράτηση εμποδίζει τους πόρους της λίστας από το να χρησιμοποιηθούν με τρόπο που δεν περιγράφεται από τη λίστα ελέγχου πρόσβασης κατά την διάρκεια του χρονικού διαστήματος που ορίζεται. Για παράδειγμα, μια κράτηση θα παρέχει 10 επεξεργαστές και 20 GB μνήμης σε 2 συγκεκριμένους χρήστες Bill και Alex από την Δευτέρα στις 3.00 μμ μέχρι την Τετάρτη στις 11.00 πμ. Ο Maui χρησιμοποιεί τον αλγόριθμο *advance reservation* για να κάνει *backfill* άλλες εργασίες, να εγγυηθεί διαθεσιμότητα πόρων για τις ενεργές εργασίες και να υποστηρίξει προθεσμίες. Ο Maui υποστηρίζει τόσο τακτικές κρατήσεις όσο και την δημιουργία δυναμικών κρατήσεων για ειδικές ανάγκες.

4.2.4 Πολιτικές

Οι πολιτικές καθορίζονται μέσω του αρχείου config του Maui και ελέγχουν το πώς και πότε μια εργασία ξεκινά. Οι πολιτικές περιλαμβάνουν την προτεραιότητα των εργασιών, πολιτικές δικαιοσύνης, δίκαιης μοιρασιάς και πολιτικές χρονοδρομολόγησης.

4.2.5 Πόροι

Οι εργασίες, οι κόμβοι και οι κρατήσεις όλοι σχετίζονται με την αφηρημένη έννοια των πόρων. Στον Maui οι πόροι είναι συνήθως:

- **Επεξεργαστές:** Προσδιορίζονται με μια απλή τιμή.
- **Μνήμη:** Πραγματική μνήμη ή RAM που προσδιορίζεται σε MB.
- **Swap:** Εικονική μνήμη που προσδιορίζεται σε MB.
- **Δίσκος:** Τοπικός δίσκος που προσδιορίζεται σε MB.

Εκτός από αυτά τα στοιχειώδη είδη πόρων υπάρχουν και 2 υψηλότερου επιπέδου έννοιες πόρων που χρησιμοποιούνται από τον Maui. Αυτές είναι τα tasks και το επεξεργαστικό ισοδύναμο (processor equivalent ή PE).

Το task είναι μια συλλογή στοιχειωδών πόρων που πρέπει να κατανεμηθούν μαζί σε ένα κόμβο. Για παράδειγμα, ένα task μπορεί να αποτελείται από ένα επεξεργαστή, 512 MB μνήμης και 2 GB τοπικού δίσκου. Μια κύρια πτυχή του task είναι ότι πρέπει να κατανεμηθεί ως μία ατομική μονάδα, χωρίς να εκτείνεται σε άλλους κόμβους. Δηλαδή ένα task που αποτελείται από 2 επεξεργαστές δεν μπορεί να ικανοποιηθεί από την

κατανομή 2 μονοεπεξεργαστικών κόμβων, ούτε μπορεί ένα task που χρειάζεται 1 επεξεργαστή και 1 GB μνήμης να ικανοποιηθεί κατανέμοντας έναν επεξεργαστή από έναν κόμβο και 1 GB μνήμης από ένα άλλον κόμβο.

Στον Maui, όταν μια εργασία ζητά πόρους, το κάνει με την μορφή των tasks, προσδιορίζοντας το task (task definition) και τον αριθμό τους (task count). Από προεπιλογή, ένα task αντιστοιχεί σε έναν επεξεργαστή μέσα σε μια εργασία και σε έναν πλήρη κόμβο μέσα σε μια κράτηση. Σε κάθε περίπτωση, αυτές οι προεπιλογές μπορούν να διαγραφούν με τον ορισμό ενός καινούριου task. Ανάλογα με τον ορισμό του task, είναι πιθανόν να υπάρξουν πολλαπλά task της ίδιας εργασίας στον ίδιο κόμβο. Για παράδειγμα, μια εργασία που ζητά 4 tasks, χρησιμοποιώντας τον προεπιλεγμένο ορισμό του 1 επεξεργαστή, μπορεί να ικανοποιηθεί από 2 κόμβους, που ο καθένας έχει 1 διπύρηνο επεξεργαστή.

Η έννοια του επεξεργαστικού ισοδύναμου (PE) προέκυψε από την ανάγκη να μεταφραστούν οι αιτήσεις πολλαπλών πόρων σε μία βαθμωτή τιμή. Δεν είναι ένας στοιχειώδης πόρος, αλλά μία μετρική πόρων. Είναι το μέτρο της πραγματικής επίπτωσης ενός συνόλου ζητούμενων πόρων από μια εργασία στους συνολικούς διαθέσιμους πόρους του συστήματος. Υπολογίζεται ως εξής:

$$PE = MAX\left(\begin{array}{c}\dfrac{Επεξεργαστές\ που\ ζητά\ η\ εργασία}{Συνολικός\ αριθμός\ επεξεργαστών},\\ \dfrac{Μνήμη\ που\ ζητά\ η\ εργασία}{Συνολική\ μνήμη},\\ \dfrac{Μέγεθος\ δίσκου\ που\ ζητά\ η\ εργασία}{Συνολικό\ μέγεθος\ δίσκου},\\ \dfrac{Swap\ που\ ζητά\ η\ εργασία}{Συνολικό\ swap}\end{array}\right) \times Συνολικός\ αριθμός\ επεξεργαστών$$

Για παράδειγμα, ας υποθέσουμε ότι έχουμε ένα cluster 25 κόμβων, όπου ο κάθε κόμβος έχει 4 επεξεργαστές και 4 GB μνήμης. Υποβάλλεται μια εργασία που απαιτεί 4 επεξεργαστές και 2 GB μνήμης. Το ισοδύναμο επεξεργαστή θα είναι:

$$PE = MAX\left(\frac{4}{4 \times 25}, \quad \frac{2}{4 \times 25}\right) \times 4 \times 25 = 4$$

Αυτό ο υπολογισμός δουλεύει τόσο σε ομογενή όσο και σε ετερογενή συστήματα.

4.2.6 Κλάση ή Ουρά

Η κλάση ή ουρά είναι ένα λογικό αντικείμενο που μπορεί να χρησιμοποιηθεί για να εφαρμόσει πολιτικές σε εργασίες. Στις περισσότερες περιπτώσεις, μια κλάση ορίζεται και διαμορφώνεται στον διαχειριστή πόρων και σχετίζεται με ένα ή περισσότερα χαρακτηριστικά ή περιορισμούς. Για παράδειγμα:

- **Προεπιλεγμένα χαρακτηριστικά εργασιών:** Μια ουρά μπορεί να σχετίζεται με μια προεπιλεγμένη διάρκεια, μέγεθος και απαίτηση πόρων.
- **Περιορισμός υπολογιστών:** Μια ουρά μπορεί να περιορίζει την εκτέλεση των εργασιών σε μια συγκεκριμένη ομάδα υπολογιστών.

- **Περιορισμός εργασιών:** Μια ουρά μπορεί να περιορίζει τα χαρακτηριστικά των εργασιών που υποβάλλονται, θέτοντας όρια όπως μέγιστος χρόνος εκτέλεσης, μέγιστος αριθμός επεξεργαστών κτλ.
- **Λίστα πρόσβασης:** Μια ουρά μπορεί να περιορίζει τον αριθμό των χρηστών που μπορούν να υποβάλλουν εργασίες, σύμφωνα με λίστες χρηστών, group κτλ.
- **Ειδική πρόσβαση:** Μια ουρά μπορεί να σχετίζεται με ειδικά προνόμια, όπως για παράδειγμα προσαρμοσμένη προτεραιότητα.

Όπως ειπώθηκε και προηγουμένως, οι περισσότεροι διαχειριστές πόρων παρέχουν την δυνατότητα πλήρης διαμόρφωσης των κλάσεων. Όταν χρειάζεται επιπλέον διαμόρφωση, μπορεί να χρησιμοποιηθεί η παράμετρος CLASSCFG του Maui. Ο Maui ανιχνεύει την χρησιμοποίηση των κλάσεων μέσω των εναρκτών κλάσεων (class initiators), περιορίζοντας έτσι τον αριθμό των εργασιών που χρησιμοποιούν μια συγκεκριμένη κλάση. Μπορεί να θεωρηθεί ως το εισιτήριο για να εκτελεστεί μια εργασία κάτω από μια κλάση. Χρησιμοποιώντας τις κλάσεις, ένα site μπορεί να εφαρμόσει διαφορετικές πολιτικές.

4.3 Κύκλος χρονοδρομολόγησης

Ο Maui έχει έναν αλγόριθμο χρονοδρομολόγησης 2 φάσεων. Κατά την διάρκεια της $1^{ης}$ φάσης, οι εργασίες υψηλότερης προτεραιότητας χρονοδρομολογούνται με την χρησιμοποίηση του αλγορίθμου *advance reservation*. Στην $2^{η}$ φάση, ο αλγόριθμος *backfill* χρησιμοποιείται για να δρομολογήσει εργασίες χαμηλότερης προτεραιότητας ανάμεσα στις εργασίες που έχουν ήδη χρονοδρομολογηθεί. Ο Maui χρησιμοποιεί την τεχνική της δίκαιης μοιρασιάς (fair-share) όταν παίρνει αποφάσεις χρονοδρομολόγησης που βασίζονται σε ιστορικά στοιχεία. Τα στάδια του κύκλου χρονοδρομολόγησης είναι τα παρακάτω:

- **Ανανέωση πληροφοριών κατάστασης:** Σε κάθε κύκλο ο χρονοδρομολογητής επικοινωνεί με τον διαχειριστή πόρων και του ζητάει τις τελευταίες διαθέσιμες πληροφορίες για τους υπολογιστικούς κόμβους, τον φόρτο εργασίας και την διαμόρφωση των πολιτικών.
- **Ανανέωση κρατήσεων**
- **Χρονοδρομολόγηση εργασιών για τις οποίες έχει γίνει κράτηση**
- **Χρονοδρομολόγηση εργασιών της υψηλότερης προτεραιότητας**
- **Backfill εργασιών**
- **Ανανέωση στατιστικών**
- **Εξυπηρέτηση αιτήσεων χρηστών:** Οι αιτήσεις των χρηστών μπορεί να αφορούν πληροφορίες κατάστασης, εντολές χειραγώγησης εργασιών ή πόρων.
- **Εκτέλεση του επόμενου κύκλου προγραμματισμού:** Όταν οι δραστηριότητες χρονοδρομολόγησης έχουν ολοκληρωθεί, ο Maui θα επεξεργαστεί τις αιτήσεις των χρηστών μέχρι ο διαχειριστής πόρων να λάβει ένα καινούργιο γεγονός ή να συμβεί ένα καινούριο εσωτερικό γεγονός. Τα γεγονότα που λαμβάνει ο διαχειριστής πόρων μπορεί να είναι η υποβολή μιας καινούριας εργασίας, η ολοκλήρωση μιας εργασίας, η προσθήκη πρόσθετων κόμβων ή αλλαγές στις πολιτικές του διαχειριστή πόρων. Εσωτερικά γεγονότα είναι οι αιτήσεις χρονοδρομολόγησης από διαχειριστές, η ενεργοποίηση ή απενεργοποίηση

κρατήσεων και η λήξη του RMPOLLINTERVAL χρονιστή, που δηλώνει την συχνότητα με την οποία θα πρέπει ο Maui να ανανεώνει τις πληροφορίες που λαμβάνει από τον διαχειριστή πόρων.

4.4 Ροή εργασιών

Παρακάτω παρουσιάζονται τα βήματα της ροής των εργασιών:

- **Καθορισμός των πραγματοποιήσιμων εργασιών:** Σε αυτό το στάδιο απορρίπτονται εργασίες που έχουν ματαιωθεί, που βρίσκονται σε άκυρες καταστάσεις (Completed, Not Queued, Defered κτλ) και δεν ικανοποιούν κάποιες προϋποθέσεις. Οι προϋποθέσεις μπορεί να αφορούν αρχεία εισόδου ή ολοκλήρωση κάποιων προκαταρκτικών σταδίων.
- **Καθορισμός της προτεραιότητας των εργασιών:** Μόλις δημιουργηθεί η λίστα των πραγματοποιήσιμων εργασιών, το επόμενο βήμα περιλαμβάνει τον καθορισμό της προτεραιότητας όλων των εργασιών της λίστας. Η προτεραιότητα κάθε εργασίας υπολογίζεται βασισμένη στα χαρακτηριστικά της εργασίας όπως τον χρήστη που την υπέβαλλε, το μέγεθος της εργασίας, την διάρκεια που βρίσκεται στην ουρά κτλ.
- **Εφαρμογή διαμορφωμένων στραγγαλιστικών πολιτικών:** Οι διαμορφωμένες πολιτικές εφαρμόζονται στην συνέχεια, περιορίζοντας τον αριθμό των εργασιών, κόμβων και επεξεργαστών που επιτρέπονται για κάθε αναγνωριστικό (credential), δηλαδή όνομα χρήστη, group, λογαριασμού, ουράς. Εργασίες που παραβιάζουν αυτές τις πολιτικές δεν τίθενται προς χρονοδρομολόγηση.
- **Καθορισμός διαθεσιμότητας πόρων:** Για κάθε εργασία, ο Maui προσπαθεί να εντοπίσει τους απαιτούμενους υπολογιστικούς πόρους. Για να γίνει το απαραίτητο ταίριασμα, ο κόμβος θα πρέπει να διαθέτει όλα τα απαιτούμενα χαρακτηριστικά που προσδιορίζονται από την εργασία και να κατέχει επαρκείς υπολογιστικούς πόρους για να ικανοποιεί τον περιορισμό TasksPerJob. Κανονικά, ο Maui καθορίζει ότι ένας κόμβος έχει επαρκείς πόρους εάν οι πόροι ούτε χρησιμοποιούνται ούτε είναι κατοχυρωμένοι σε μια άλλη εργασία χρησιμοποιώντας την εξίσωση:

$$Διαθέσιμοι = Διαμορφωμένοι - MAX(Κατοχυρωμένοι, Χρησιμοποιούμενοι)$$

Η παράμετρος RESOURCEAVAILABILITYPOLICY μπορεί να τροποποιηθεί για να προσαρμόσει αυτήν την συμπεριφορά.

- **Κατανομή πόρων στις εργασίες:** Εάν επαρκείς πόροι έχουν βρεθεί για μια εργασία, η πολιτική παραχώρησης πόρων εφαρμόζεται τότε για να επιλέξει το καλύτερο σύνολο πόρων. Αυτές οι πολιτικές κατανομής επιτρέπουν κριτήρια επιλογής όπως η ταχύτητα του κόμβου, ο τύπος κράτησης ή η παραχώρηση επιπλέον πόρων, να ληφθούν υπόψη κατά την απόφαση κατανομής για να βελτιώσουν την επίδοση της εργασίας ή/και να μεγιστοποιήσουν την ελευθερία του χρονοδρομολογητή σε μελλοντικές αποφάσεις χρονοδρομολόγησης.
- **Διαμοιρασμός των tasks εργασιών στους διατιθέμενους πόρους:** Μόλις επιλεχθούν οι πόροι, ο Maui αντιστοιχίζει τα tasks των εργασιών στους πόρους. Ο διαμοιρασμός των tasks βασίζεται γενικά σε απλούς αλγορίθμους κατανομής,

όπως ο round-robin ή max-blocking, αλλά μπορεί επίσης να ενσωματώσει μοτίβα από βιβλιοθήκες γλωσσών παράλληλου προγραμματισμού (MPI, PVM) για να μειώσει το κόστος διεργασιακής επικοινωνίας.

- **Εκτέλεση εργασιών:** Αφού έχουν επιλεχθεί οι πόροι και διαμοιραστεί τα tasks, ο χρονοδρομολογητής επικοινωνεί με τον διαχειριστή πόρων και τον ενημερώνει πού και πώς να εκτελέσει την εργασία. Ο διαχειριστής πόρων ξεκινά στη συνέχεια το εκτελέσιμο της εργασίας.

4.5 Προτεραιότητα εργασιών

Γενικά, η προτεραιότητα εργασιών είναι η διαδικασία καθορισμού των επιλογών που εκπληρώνουν καλύτερα τους γενικούς σκοπούς του site. Στην περίπτωση της χρονοδρομολόγησης, ένα site μπορεί να έχει πολλαπλούς και ανεξάρτητους στόχους, που μπορεί να περιλαμβάνουν μεγιστοποίηση της χρησιμοποίησης συστήματος, προτίμηση συγκεκριμένων χρηστών ή ότι μια εργασία δεν παραμένει στην ουρά για παραπάνω από ένα χρονικό διάστημα. Η προσέγγιση του Maui στην αντιπροσώπευση αυτού του πολύπλευρου συνόλου στόχων είναι η χρησιμοποίηση βαρών σε διάφορα αντικείμενα, ώστε κάθε εργασία να αποκτήσει μια συνολική τιμή ή προτεραιότητα. Με τον καθορισμό της προτεραιότητας των εργασιών, ο χρονοδρομολογητής μπορεί να πραγματοποιήσει τους στόχους του site, ξεκινώντας τις εργασίες με σειρά προτεραιότητας.

Ο μηχανισμός προτεραιότητας του Maui επιτρέπει τον καθορισμό βαρών τόσο στους βασικούς συντελεστές (components) αλλά και στους δευτερεύοντες (subcomponents), γεγονός που μας δίνει σημαντικό έλεγχο σε αυτήν την οπτική της χρονοδρομολόγησης. Για να ενεργοποιηθεί αυτός ο βαθμός ελέγχου, ο Maui χρησιμοποιεί μια απλή ιεραρχία προτεραιότητας-βάρους, όπου η συνεισφορά του κάθε subcomponent υπολογίζεται ως εξής:

<COMPONENT WEIGHT> × *<SUBCOMPONENT WEIGHT>* ×
× *<PRIORITY SUBCOMPONENT VALUE>*

Παρόλο που υπάρχουν πολλά components και subcomponents, ένα site χρειάζεται να επικεντρωθεί μόνο σε ένα μικρό υποσύνολο που ταιριάζει στις ανάγκες του. Αναμιγνύοντας και ταιριάζοντας τα βάρη προτεραιότητας, το site μπορεί να πετύχει την επιθυμητή συμπεριφορά. Οποιαδήποτε στιγμή, η εντολή *diagnose –p* μπορεί να χρησιμοποιηθεί για την ενημέρωση του χρήστη για τις επιπτώσεις των ρυθμίσεων προτεραιότητας και βαρών στις ανενεργές εργασίες. Επίσης, η εντολή *showgrid* βοηθάει τους διαχειριστές στην αξιολόγηση της αποτελεσματικότητας της προτεραιότητας με μετρήσεις χρησιμοποίησης του συστήματος, όπως το χρονικό διάστημα παραμονής στην ουρά και ο παράγοντας επέκτασης. Παρακάτω παρουσιάζεται ένας πίνακας που περιλαμβάνει τα διαθέσιμα components και subcomponents.

Πίνακας 4.1 Components-Subcomponents

Component	Subcomponent	Μετρική
CRED (αναγνωριστικά εργασίας)	USER	προτεραιότητα χρήστη (USERCFG)
	GROUP	προτεραιότητα group (GROUPCFG)
	ACCOUNT	προτεραιότητα λογαριασμού (ACCOUNTCFG)
	QOS	προτεραιότητα ποιότητας παρεχόμενων υπηρεσιών (QOSCFG)
	CLASS	προτεραιότητα κλάσης (CLASSCFG)
FS (δίκαιη μοιρασιά)	FSUSER	ιστορική χρήση χρήστη
	FSGROUP	ιστορική χρήση group
	FSACCOUNT	ιστορική χρήση λογαριασμού
	FSQOS	ιστορική χρήση ποιότητας παρεχόμενων υπηρεσιών
	FSCLASS	ιστορική χρήση κλάσης
RES (ζητούμενοι πόροι)	NODE	αριθμός ζητούμενων κόμβων
	PROC	αριθμός ζητούμενων επεξεργαστών
	MEM	συνολική ζήτηση πραγματικής μνήμης (MB)
	SWAP	συνολική ζήτηση εικονικής μνήμης (MB)
	DISK	συνολική ζήτηση τοπικού δίσκου (MB)
	PS	συνολική ζήτηση επεξεργαστών-δευτερολέπτων
	PE	συνολική ζήτηση επεξεργαστικού ισοδύναμου
	WALLTIME	συνολική ζήτηση χρόνου εκτέλεσης (δευτερόλεπτα)
SERV (ισχύον επίπεδο υπηρεσιών)	QUEUETIME	χρονικό διάστημα που η εργασία έχει παραμείνει στην ουρά (λεπτά)
	XFACTOR	ελάχιστος παράγοντας επέκτασης
	BYPASS	αριθμός φορών που η εργασία έχει προσπεραστεί από backfill
TARGET (στόχος επιπέδου υπηρεσιών)	TARGETQUEUETIME	χρονικό διάστημα μέχρι να επιτευχθεί ο χρονικός στόχος παραμονής στην ουρά (εκθετικό)
	TARGETXFACTOR	απόσταση από τον στόχο του παράγοντα επέκτασης (εκθετικό)

USAGE (πόροι που έχουν καταναλωθεί-μόνο ενεργές εργασίες)	CONSUMED	επεξεργαστές-δευτερόλεπτα που έχουν ανατεθεί μέχρι τώρα
	REMAINING	επεξεργαστές-δευτερόλεπτα που παραμένουν
	PERCENT	ποσοστό του ζητούμενου χρόνου εκτέλεσης που έχει καταναλωθεί

Η προτεραιότητα μιας εργασίας είναι το προσαρμοσμένο από το βάρος άθροισμα των ενεργοποιημένων subcomponents. Από προεπιλογή, η τιμή των βαρών όλων των components είναι 1 και όλων των subcomponents 0. Η μόνη εξαίρεση είναι το subcomponent QUEUETIME όπου έχει τιμή 1. Το αποτέλεσμα είναι η συνολική προτεραιότητα των εργασιών να ισούται με τον συνολικό χρόνο παραμονής τους στην ουρά, αναγκάζοντας τον Maui να ενεργεί σαν FIFO. Αφού υπολογιστεί η προτεραιότητα, περιορίζεται μεταξύ των τιμών 0 και MAX_PRIO_VAL, που έχει ορισθεί ως 1 δισεκατομμύριο. Σε καμία περίπτωση δεν πρέπει οι εργασίες να αποκτήσουν προτεραιότητα πάνω από MAX_PRIO_VAL μέσω των τιμών των subcomponents.

Χρησιμοποιώντας την εντολή *setspri*, οι διαχειριστές του site μπορούν να προσαρμόσουν την βασική υπολογισμένη προτεραιότητα εργασιών, είτε αναθέτοντας μια σχετική προσαρμογή προτεραιότητας, είτε μια απόλυτη. Η σχετική προσαρμογή προτεραιότητας θα προκαλέσει την αύξηση ή μείωση της βασικής προτεραιότητας κατά μία συγκεκριμένη τιμή. Η απόλυτη προσαρμογή, SPRIO, θα δώσει στην εργασία μια προτεραιότητα ίση με MAX_PRIO_VAL + SPRIO και θα εγγυηθεί ότι η εργασία θα έχει μεγαλύτερη προτεραιότητα από οποιαδήποτε άλλη εργασία, η προτεραιότητα της οποίας έχει προκύψει με την κλασική μέθοδο.

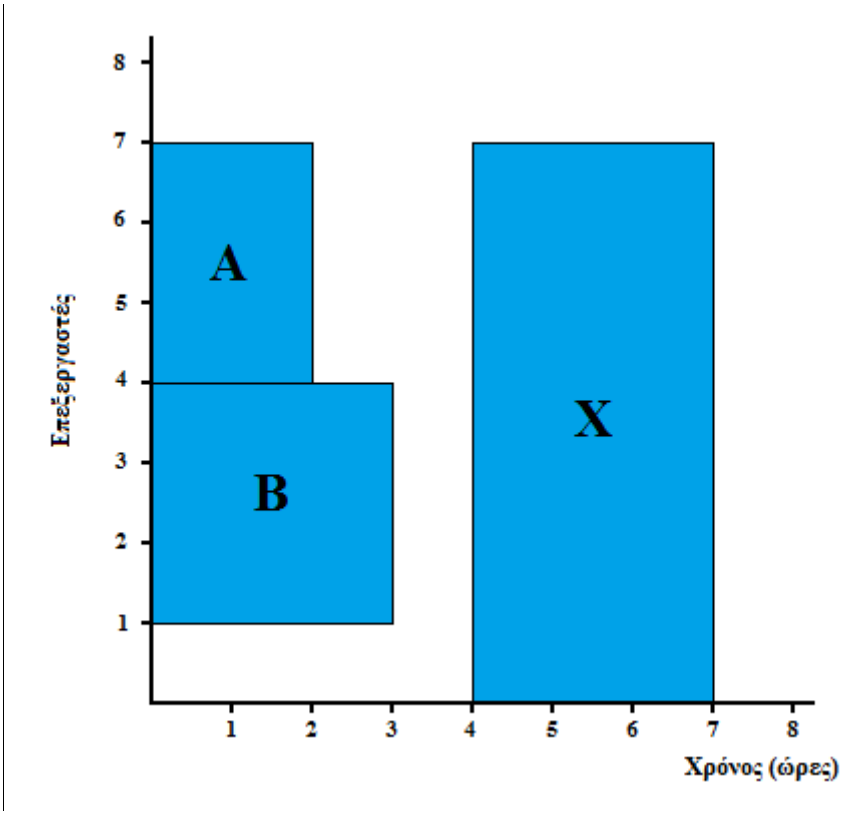
4.6 Κατανομή κόμβων

Ενώ η προτεραιότητα των εργασιών επιτρέπει σε ένα site να αποφασίσει ποιες εργασίες θα εκτελέσει, η κατανομή των κόμβων προσδιορίζει πώς οι διαθέσιμοι πόροι θα κατανεμηθούν σε κάθε εργασία. Ο αλγόριθμος καθορίζεται από την τιμή της παραμέτρου NODEALLOCATIONPOLICY.
Η κατανομή των κόμβων είναι ιδιαίτερα σημαντική στις εξής περιπτώσεις:

- **Ετερογενή συστήματα:** Εάν οι διαθέσιμοι υπολογιστικοί πόροι εμφανίζουν διαφορές και ένα υποσύνολο των υποβληθέντων εργασιών δεν μπορεί να εκτελεστεί σε όλους τους κόμβους, τότε οι αποφάσεις κατανομής μπορεί να επηρεάσουν σημαντικά την επίδοση του συστήματος. Για παράδειγμα, ένα σύστημα μπορεί να αποτελείται από 2 κόμβους, Α και Β, που έχουν ακριβώς τα ίδια χαρακτηριστικά, εκτός από την μνήμη RAM, αφού ο κόμβος Α έχει 256 MB και ο κόμβος Β 1 GB. Έστω 2 εργασίες που απαιτούν έναν επεξεργαστή, αλλά η $1^{η}$ χρειάζεται 256 MB ενώ η $2^{η}$ 1 GB. Η $1^{η}$ εργασία μπορεί να εκτελεστεί και στους 2 κόμβους, αλλά αν εκτελεστεί στον $2^{ο}$ κόμβο, τότε η $2^{η}$ εργασία θα μπλοκάρει. Ένας έξυπνος χρονοδρομολογητής θα παραχωρούσε τον $1^{ο}$ κόμβο στην $1^{η}$ εργασία, καθώς μπορεί να έχει λιγότερους πόρους αλλά και πάλι ικανοποιεί τις απαιτήσεις της, και τον $2^{ο}$ κόμβο στην $2^{η}$ εργασία. Με αυτόν

τον τρόπο και οι 2 εργασίες μπορούν να ξεκινήσουν άμεσα. Αυτή η προσέγγιση καλείται *bestfit* και είναι διαθέσιμη χρησιμοποιώντας την πολιτική MINRESOURCE.

- **Σύστημα μοιραζόμενων κόμβων:** Τα συστήματα μοιραζόμενων κόμβων έχουν συνήθως SMP κόμβους, αν και αυτό δεν είναι υποχρεωτικό. Σε κάθε περίπτωση όταν οι πόροι ενός κόμβου μοιράζονται σε tasks περισσότερων από μία εργασιών, εμφανίζονται θέματα ανταγωνισμού των πόρων (CPU, μνήμη, εύρος μνήμης) και κατακερματισμού. Ο ανταγωνισμός των πόρων μπορεί να οδηγήσει σε καθυστέρηση όλων των εργασιών που εμπλέκονται, γεγονός που μπορεί να έχει σημαντικές επιπτώσεις, εάν λαμβάνουν μέρος εργασίες με υψηλό βαθμό παραλληλισμού. Ο ενδοκομβικός κατακερματισμός από την άλλη είναι αρκετά συνηθισμένος σε μεγάλης κλίμακας SMP συστήματα (> 32 επεξεργαστές/κόμβο). Για παράδειγμα, έστω ένα σύστημα με 2 κόμβους Α και Β, που ο καθένας έχει 64 επεξεργαστές. Υποθέτοντας ότι έχουν φορτωθεί με αρκετές εργασίες, αυτήν την στιγμή είναι διαθέσιμοι 24 επεξεργαστές στον κόμβο Α και 12 στον κόμβο Β. Έστω και 2 εργασίες Χ και Υ, που απαιτούν 10 και 20 επεξεργαστές αντίστοιχα. Όπως και στην προηγούμενη περίπτωση, η εργασία Χ μπορεί να εκτελεστεί και στους 2 κόμβους, αλλά αν εκτελεστεί στον κόμβο Α, θα μπλοκάρει την εργασία Υ. Ένας αλγόριθμος για να αντιμετωπιστεί ο ενδοκομβικός κατακερματισμός είναι αρκετά απλός στην περίπτωση ενός πόρου, αλλά όταν η εργασία απαιτεί ένα συνδυασμό πόρων, το πρόβλημα γίνεται αρκετά πιο περίπλοκο. Αλγόριθμοι για να αντιμετωπίσουν αυτές τις περιπτώσεις περιλαμβάνονται στην βιβλιοθήκη επεκτάσεων G2.
- **Συστήματα που βασίζονται στις κρατήσεις (reservation based systems):** Τα συστήματα που βασίζονται στις κρατήσεις προσθέτουν και την μεταβλητή του χρόνου στις αποφάσεις κατανομής των κόμβων. Με τις κρατήσεις, οι πόροι των κόμβων πρέπει να αντιμετωπίζονται ως ένα σύστημα 2 μεταβλητών «κόμβος-χρόνος». Η κατανομή κόμβων στις εργασίες κατακερματίζει αυτόν τον χώρο και δυσκολεύει την χρονοδρομολόγηση στις εναπομείναντες θέσεις «κόμβων-χρόνου». Οι αποφάσεις κατανομής θα πρέπει να λαμβάνονται με γνώμονα την ελαχιστοποίηση του κατακερματισμού και την μεγιστοποίηση της ικανότητας του χρονοδρομολογητή να συνεχίζει να εκκινεί εργασίες στις διαθέσιμες θέσεις. Το παρακάτω σχήμα εξηγεί τις τυχόν ασάφειες. Στο σχήμα, οι εργασίες Α και Β βρίσκονται ήδη σε εκτέλεση. Επίσης, υπάρχει μια κράτηση Χ που είναι προγραμματισμένη να εκτελεστεί σε 4 ώρες. Υποθέτουμε ότι η εργασία Α έχει διάρκεια 2 ώρες ενώ η εργασία Β 3 ώρες. Υποβάλλονται δύο νέες εργασίες Γ και Δ, που απαιτούν 3 και 5 ώρες αντίστοιχα. Και οι 2 εργασίες θα μπορούσαν να εκτελεστούν στον κενό χώρο που βρίσκεται πάνω από την εργασία Α, όμως αν σε αυτό τον χώρο εκτελεστεί η εργασία Γ, τότε η εργασία Δ θα μπλοκάρει. Αν όμως εκτελεστεί σε αυτόν τον χώρο η εργασία Δ, τότε και η εργασία Γ μπορεί να εκτελεστεί άμεσα στον κενό χώρο κάτω από την εργασία Β. Αυτό το παράδειγμα παρουσιάζει την σημασία που έχουν οι κρατήσεις στην λήψη αποφάσεων για την κατανομή των κόμβων, τόσο κατά την έναρξη εργασιών, όσο και κατά την δημιουργία κρατήσεων. Οι επιπτώσεις αυξάνονται όσο μεγαλώνει ο αριθμός των κρατήσεων σε ένα σύστημα. Ο αλγόριθμος LASTAVAILABLE μπορεί να αντιμετωπίσει αυτό το πρόβλημα, καθώς εντοπίζει τους πόρους με το μικρότερο διάστημα μεταξύ του τέλους της εργασίας που προσπαθούμε να δρομολογήσουμε και της αρχής μιας μελλοντικής κράτησης.



Σχήμα 4.1 Χρήση της πολιτικής κατανομής κόμβων LASTAVAILABLE

- **Συστήματα με ανομοιόμορφο δίκτυο:** Σε συστήματα όπου οι συνδέσεις δικτύου δεν ομοιόμορφες μεταξύ των κόμβων, δηλαδή η καθυστέρηση και το εύρος ζώνης μεταξύ 2 οποιοδήποτε κόμβων διαφέρει αρκετά, τότε ο αλγόριθμος κατανομής κόμβων θα πρέπει να τοποθετήσει τα tasks μιας εργασίας όσο κοντά γίνεται μεταξύ τους, με στόχο την ελαχιστοποίηση των επιπτώσεων της διαφοράς του εύρους ζώνης και της καθυστέρησης.

Στον παρακάτω πίνακα παρουσιάζονται οι αλγόριθμοι που χρησιμοποιεί ο Maui για την κατανομή των κόμβων:

Πίνακας 4.2 Αλγόριθμοι κατανομής κόμβων

Αλγόριθμος	Περιγραφή
CPULOAD	Ο κόμβοι διαλέγονται με βάση ποιος έχει το μεγαλύτερο ποσό διαθέσιμης και αχρησιμοποίητης επεξεργαστικής δύναμης (CPU power). Είναι ένας καλός αλγόριθμος για συστήματα καταμερισμού χρόνου. Αυτός ο αλγόριθμος εφαρμόζεται σε εργασίες που ξεκινούν άμεσα. Για μελλοντικές κρατήσεις χρησιμοποιείται ο αλγόριθμος MINRESOURCE.
FIRSTAVAILABLE	Απλός first come - first served αλγόριθμος, όπου οι κόμβοι κατανέμονται με την σειρά που παρουσιάζονται από τον διαχειριστή πόρων. Είναι πολύ απλός και γρήγορος αλγόριθμος.

LASTAVAILABLE	Αλγόριθμος που διαλέγει τους πόρους έτσι ώστε να ελαχιστοποιήσει το ποσό του χρόνου μετά το τέλος της εργασίας και πριν την επόμενη κράτηση. Είναι ένας αλγόριθμος «καλύτερου ταιριάσματος στον χρόνο» που ελαχιστοποιεί τις επιπτώσεις του κατακερματισμού στο χώρο «κόμβοι-χρόνος». Είναι ιδιαίτερα χρήσιμο σε συστήματα με μεγάλο αριθμό κρατήσεων.
MACHINEPRIO	Αυτός ο αλγόριθμος επιτρέπει σε ένα site να προσδιορίσει την προτεραιότητα πολλών στατικών και δυναμικών πτυχών των υπολογιστικών κόμβων και να τους κατανείμει αναλόγως. Είναι ουσιαστικά μια πιο ευέλικτη μορφή του αλγόριθμου MINRESOURCE.
MINRESOURCE	Αυτός ο αλγόριθμος βάζει σε προτεραιότητα τους κόμβους με βάση τους πόρους τους. Επιλέγονται οι κόμβοι με τους λιγότερους πόρους που ικανοποιούν όμως τους περιορισμούς των εργασιών.
CONTIGUOUS	Αυτός ο αλγόριθμος θα κατανείμει τους κόμβους σε συνεχόμενα μπλοκ όπως απαιτείται από το Compaq RMS σύστημα.
MAXBALANCE	Αυτός ο αλγόριθμος θα προσπαθήσει να κατανείμει το πιο ισορροπημένο σύνολο κόμβων που είναι δυνατόν σε μια εργασία. Στις περισσότερες περιπτώσεις, αλλά όχι σε όλες, το κριτήριο της ισορροπίας είναι η ταχύτητα των κόμβων. Έτσι, εάν είναι δυνατόν, κόμβοι με όμοιες ταχύτητες θα κατανεμηθούν στην εργασία. Εάν δεν μπορούν να βρεθούν κόμβοι με όμοιες ταχύτητες, ο αλγόριθμος θα κατανείμει ένα υποσύνολο κόμβων με την μικρότερη δυνατή διαφορά ταχυτήτων.
FASTEST	Αυτός ο αλγόριθμος διαλέγει κόμβους με κριτήριο την ταχύτητα, βάζοντας σε προτεραιότητα τους ταχύτερους κόμβους. Οι κόμβοι θα επιλεγούν με κριτήριο την ταχύτητα κόμβου, αν αυτή προσδιορίζεται. Εάν όχι, οι κόμβοι θα επιλεγούν με κριτήριο την επεξεργαστική ταχύτητα. Αν τίποτα από τα 2 δεν προσδιορίζεται, οι κόμβοι θα επιλεγούν σε τυχαία σειρά.
LOCAL	Καλεί τον τοπικά κατασκευασμένο αλγόριθμο κατανομής κόμβων.

4.7 Δικαιοσύνη

Η ιδέα της δικαιοσύνης (fairness) ποικίλει από άνθρωπο σε άνθρωπο και από site σε site. Για κάποιους υπονοείται να δίνονται σε όλους τους χρήστες ισάξια πρόσβαση στους υπολογιστικούς πόρους. Όμως, πιο σύνθετες ιδέες περιλαμβάνουν ιστορικά στοιχεία χρήσης των πόρων, θέματα πολιτικής και σημασίας των εργασιών. Ενώ κανένας χρονοδρομολογητής δεν είναι σε θέση να χειριστεί όλους τους πιθανούς ορισμούς για το τι σημαίνει δικαιοσύνη, ο Maui παρέχει κάποια ευέλικτα εργαλεία που βοηθούν στους πιο συχνούς ορισμούς της διαχείρισης της δικαιοσύνης και των αναγκών. Ειδικότερα, η δικαιοσύνη κατά τον Maui, μπορεί να αντιμετωπιστεί ως έναν συνδυασμός των παρακάτω εννοιών:

Πίνακας 4.3 Fairshare facilities

Έννοια	Περιγραφή	Παράδειγμα
Στραγγαλιστικές πολιτικές (Throttling Policies)	Προσδιορίζει τα όρια των πόρων που μπορούν να χρησιμοποιηθούν κάθε στιγμή.	USERCFG[john] MAXJOB=3 GROUPCFG[DEFAULT] MAXPROC=64 GROUP[staff] MAXPROC=128 Επιτρέπει στον χρήστη john να εκτελέσει μέχρι 3 εργασίες ταυτόχρονα, στο group staff να χρησιμοποιήσει μέχρι 128 επεξεργαστές και στα υπόλοιπα groups μέχρι 64.
Προτεραιότητα εργασιών (Job Prioritization)	Προσδιορίζει τι είναι σημαντικό για τον χρονοδρομολογητή. Η χρησιμοποίηση παραγόντων Service μπορεί να επιτρέψει σε ένα site να ισορροπήσει τον χρόνο ολοκλήρωσης ή άλλες μετρικές απόδοσης.	SERVWEIGHT 1 QUEUETIMEWEIGHT 10 Προκαλεί αύξηση της προτεραιότητας μιας εργασίας κατά 10 για κάθε λεπτό παραμονής του στην ουρά.
Δίκαιη μοιρασιά (Fairshare)	Προσδιορίζει τους στόχους χρήσης για να περιορίσει την πρόσβαση στους πόρους ή να προσαρμόσει την προτεραιότητα βασιζόμενος σε ιστορικά στοιχεία χρήσης.	USERCFG[steve] FSTARGET=25.0+ FSWEIGHT 1 FSUSERWEIGHT 10 Ενεργοποιεί την δίκαιη μοιρασιά που βασίζεται στην προτεραιότητα και προσδιορίζει ένα στόχο δίκαιης μοιρασιάς για τον χρήστη steve, ευνοώντας τις εργασίες του στην προσπάθεια να καταλαμβάνουν τουλάχιστον το 25% των υπολογιστικών κύκλων.

Διαχείριση κατανομής (Allocation Management)	Προσδιορίζει μακροπρόθεσμα όρια βασιζόμενα στα αναγνωριστικά των εργασιών.	BANKTYPE QBANK BANKSERVER server.sys.net Ενεργοποιεί το QBANK σύστημα διαχείρισης κατά-νομής. Μέσα στον διαχειριστή πόρων μπορεί να διαμορφωθούν εκχωρήσεις πρότζεκτ ή λογαρια-σμών. Αυτές οι εκχωρήσεις, για παράδειγμα, μπορεί να επιτρέ-πουν ένα πρότζεκτ X να χρησιμοποιήσει 100,000 επε-ξεργαστές-ώρες κάθε τρίμηνο, μοιραζόμενη πρόσβαση κτλ.

4.8 Ποιότητα παρεχόμενων υπηρεσιών (QoS)

Το QoS επιτρέπει σε ένα site να παρέχει ειδική μεταχείριση σε ποικίλες κλάσεις εργασιών, χρηστών, groups κτλ. Κάθε αντικείμενο QoS μπορεί να θεωρηθεί ως ένα «δοχείο» ειδικών προνομίων που περιέχουν από εξαιρέσεις fairness, μέχρι ειδική προτεραιότητα εργασιών και προνομίων ειδικής πρόσβασης. Κάθε αντικείμενο QoS έχει επίσης εκτεταμένη λίστα χρηστών, groups και λογαριασμών που έχουν πρόσβαση σε αυτά τα προνόμια.

Τα sites μπορούν να διαμορφώσουν ποικίλα QoS, που το καθένα έχει το δικό του σετ προτεραιοτήτων, εξαιρέσεων από πολιτικές και ειδικές ρυθμίσεις πρόσβασης στους πόρους. Έπειτα, διαμορφώνουν τα δικαιώματα πρόσβασης χρηστών, groups, λογαριασμών και κλάσεων. Μια δεδομένη εργασία έχει ένα προεπιλεγμένο QoS, αλλά μπορεί να της επιτρέπεται η πρόσβαση και σε μια λίστα επιπλέον QoS. Όταν μια εργασία υποβάλλεται, ο υποβολέας μπορεί να απαιτήσει ένα συγκεκριμένο QoS ή να επιτρέψει την χρησιμοποίηση του προεπιλεγμένου QoS. Μόλις η εργασία υποβληθεί, ο χρήστης μπορεί να προσαρμόσει το QoS της εργασίας του οποιαδήποτε στιγμή με την εντολή *setqos*. Αυτή η εντολή θα επιτρέψει στον χρήστη να τροποποιήσει το QoS της εργασίας του, αλλάζοντας την σε κάποια άλλη, στην οποία έχει πρόσβαση. Οι διαχειριστές του Maui μπορούν να αλλάξουν το QoS οποιαδήποτε τιμή.

Στις εργασίες χορηγείται πρόσβαση σε προνόμια QoS τροποποιώντας τις ρυθμίσεις QDEF (προεπιλεγμένο QoS) και QLIST (λίστα επιτρεπόμενης πρόσβασης σε QoS) στο αρχείο *maui.cfg* μέσω της παραμέτρου QOSCFG. Γενικότερα, η διαχείριση των QoS γίνεται μέσω αυτής της παραμέτρου. Η εντολή *diagnose –Q* μπορεί να χρησιμοποιηθεί για την απόκτηση πληροφοριών σχετικά με την τρέχουσα διαμόρφωση.

Τα προνόμια που σχετίζονται με το QoS μπορούν να χωριστούν σε 3 κατηγορίες.

- **Απόδοση ειδικής προτεραιότητας (special prioritization)**
- **Πρόσβαση και περιορισμοί υπηρεσιών:** Οι υπηρεσίες ενεργοποιούνται ή απενεργοποιούνται ρυθμίζοντας την αντίστοιχη QoS σημαία.
- **Αψήφιση και εξαιρέσεις πολιτικών**

Παρακάτω παρουσιάζονται πίνακες που περιγράφουν τα στοιχεία των 2 πρώτων κατηγοριών:

Πίνακας 4.4 Χαρακτηριστικά Ειδικής Προτεραιότητας

Χαρακτηριστικό	Περιγραφή
FSTARGET	στόχος fairshare
PRIORITY	προτεραιότητα εργασιών που απαιτούν την συγκεκριμένη QoS
QTTARGET	στόχος QUEUETIME
QTWEIGHT	βάρος QUEUETIME
XFTARGET	στόχος XFACTOR
XFWEIGHT	βάρος XFACTOR

Πίνακας 4.5 Σημαίες πρόσβασης και περιορισμού υπηρεσιών

Σημαία	Περιγραφή
DEDICATED	Οι εργασίες δεν πρέπει να μοιράζονται υπολογιστικούς πόρους με άλλες εργασίες. Αυτές οι εργασίες θα εκτελούνται μόνο σε κόμβους που είναι ανενεργοί και δεν θα επιτρέπουν άλλες εργασίες να χρησιμοποιούν πόρους σε αυτούς τους κόμβους, ακόμα και αν είναι διαθέσιμοι.
NOBF	Οι εργασίες δεν μπορούν να γίνουν backfill.
NORESERVATION	Οι εργασίες δεν μπορούν να κάνουν κράτηση πόρων, ανεξάρτητα από την προτεραιότητα που έχουν.
PREEMPTEE	Η εργασία μπορεί να αναστείλει την εκτέλεσή της και να παραχωρήσει τους πόρους της σε εργασίες υψηλότερης προτεραιότητας.
PREEMPTOR	Η εργασία μπορεί να πάρει τους πόρους μιας PREEMPTEE εργασίας.
RESERVEALWAYS	Η εργασία θα πρέπει πάντα να κάνει κράτηση πόρων, ανεξαρτήτως προτεραιότητας.
RESTARTPREEMPT	Η εργασία μπορεί να πάρει τους πόρους μιας PREEMPTEE εργασίας, ξαναβάζοντάς την στην ουρά, αν αυτό της επιτρέψει να ξεκινήσει πιο γρήγορα.
USERESERVED[:<RESID>]	Η εργασία μπορεί να χρησιμοποιήσει πόρους μόνο σε κρατήσεις που έχει πρόσβαση. Εάν το RESID έχει προσδιοριστεί, η εργασία μπορεί να χρησιμοποιήσει τους πόρους της συγκεκριμένης κράτησης.

Όσον αφορά την 3η κατηγορία, μπορούν να ανατεθούν νέα QoS σε εργασίες που ορίζουν νέα όρια και παρακάμπτουν τις ισχύουσες πολιτικές για χρήστες, groups και λογαριασμούς. Ειδικότερα, οι πολιτικές μπορούν να παρακαμφθούν ως εξής:

QOSCFG[staff] MAXJOB=48

Όπως είπαμε, εκτός από παράκαμψη πολιτικών, μέσω των QoS μπορεί να επιτραπεί στις εργασίες να αγνοήσουν πολιτικές, θέτοντας την QoS σημαία στο αντίστοιχο χαρακτηριστικό (πχ IGNJOBPERUSER, IGNPROCPERUSER κτλ).

4.9 Στατιστικά συστήματος και εργασιών

Ο Maui διατηρεί έναν μεγάλο αριθμό στατιστικών και παρέχει αρκετές εντολές που επιτρέπουν την εύκολη πρόσβαση σε αυτές τις πληροφορίες. Αυτές οι στατιστικές διακρίνονται σε 3 διαφορετικούς τύπους:

4.9.1 Στατιστικές πραγματικού χρόνου (Real Time Statistics)

Ο Maui παρέχει πληροφορίες για στατιστικές πραγματικού χρόνου που αφορούν τον τρόπο που το μηχάνημα τρέχει από την πλευρά της χρονοδρομολόγησης. Η εντολή *showstats* είναι στην πραγματικότητα μια σειρά από εντολές που παρέχουν λεπτομερείς πληροφορίες τόσο για την συνολική συμπεριφορά της χρονοδρομολόγησης, όσο και ειδικότερα για κάθε χρήστη, group, λογαριασμό και κόμβο. Οποιαδήποτε στιγμή, οι στατιστικές πραγματικού χρόνου μπορούν να μηδενιστούν με την εντολή *resetstats*. Εκτός από την εντολή *showstats*, μια άλλη πολύ χρήσιμη εντολή είναι η *showgrid*, που παρουσιάζει μια μεγάλη ποικιλία από μετρικές χρονοδρομολόγησης, όπως η αποτελεσματικότητα του αλγορίθμου backfill και ο μέσος χρόνος αναμονής στην ουρά.

4.9.2 Προφίλ στατιστικών ιστορικών χρήσης (Profiling Historical Usage)

Πληροφορίες ιστορικής χρήσης μπορούν να ληφθούν για ένα συγκεκριμένο χρονικό διάστημα, κλάσεις εργασιών και μέρους των πόρων με την εντολή *profiler*. Αυτή η εντολή χρησιμοποιεί στοιχεία που αποθηκεύονται στον φάκελο *stats* του Maui, στην μορφή ΌνομαΗμέρας_Μήνας_Ημέρα_Χρόνος (πχ Mon_Jul_16_2001). Όταν θέλουμε να φτιάξουμε των προφίλ των στατιστικών, τα αρχεία του φακέλου *stats* που καλύπτουν το χρονικό διάστημα που μας ενδιαφέρει πρέπει να συνενωθούν σε ένα αρχείο. Αυτό το αρχείο πρέπει να περαστεί σαν όρισμα στην εντολή *profiler* μαζί με έναν αριθμό σημαιών διαμόρφωσης που καθορίζουν τα δεδομένα που πρέπει να επεξεργαστούν και πως πρέπει να παρουσιαστούν. Η έξοδος της εντολής παρέχει εκτεταμένες και λεπτομερείς πληροφορίες για τις εργασίες που εκτελέστηκαν και το επίπεδο των υπηρεσιών χρονοδρομολόγησης που έλαβαν.

4.9.3 Στατιστικές δίκαιης μοιρασιάς (Fairshare Usage Statistics)

Ανεξάρτητα με το αν η fairshare είναι ενεργοποιημένη ή όχι, λεπτομερείς πληροφορίες δίκαιης μοιρασιάς για τα credentials των εργασιών διατηρούνται στον φάκελο *stats*. Τα στατιστικά fairshare βρίσκονται στην μορφή FS.<EPOCHTIME> και οι πληροφορίες μπορούν να παρουσιαστούν με την εντολή *diagnose –f*.

4.10 Αλγόριθμοι βελτιστοποίησης

Ο Maui διαθέτει αρκετούς αλγόριθμους βελτιστοποίησης της χρονοδρομολόγησης, όπως αυτούς που αναφέραμε και σε προηγούμενο κεφάλαιο. Μερικοί από αυτούς είναι ο αλγόριθμος *backfill*, *checkpoint*, *preemption*, *node sets*, *partitions*. Αναλυτική περιγραφή για την χρησιμοποίηση αυτών των αλγορίθμων με στόχο την βελτίωση της χρονοδρομολόγησης περιέχονται στο [29].

5 *Μελέτη αλγορίθμων χρονοδρομολόγησης*

Σε αυτό το κεφάλαιο γίνεται μελέτη των πιο συνηθισμένων αλγορίθμων χρονοδρομολόγησης. Συγκεκριμένα, μελετούνται οι αλγόριθμοι *first come-first served* (FCFS), *largest job first* (LJF) και *smallest job first* (SJF), καθώς και η τεχνική *backfill*. Για αυτό τον σκοπό κατασκευάστηκαν 3 εργασιακά φορτία, που διαφέρουν μεταξύ τους τόσο στον αριθμό των *memory intensive* εφαρμογών, όσο και στο μέγεθος και την σειρά εμφάνισης των εφαρμογών, σε μια προσπάθεια να καλυφθεί όσο το δυνατόν ευρύτερο φάσμα των πιθανών περιπτώσεων.

Η διεξαγωγή των μετρήσεων έγινε στην συστοιχία clones του Εργαστηρίου Υπολογιστικών Συστημάτων. Η χαρτογράφηση όμως των εφαρμογών, για να δούμε πως κατανέμονται στους υπολογιστικούς κόμβους ανάλογα με τον αλγόριθμο χρονοδρομολόγησης, έγινε σε ένα cluster από virtual machines στον termi2. Έχοντας διαθέσιμη την χαρτογράφηση των εφαρμογών για όλες τις περιπτώσεις που μας ενδιέφεραν, ήμασταν σε θέση να υποβάλουμε τις εφαρμογές στα clones με την κατάλληλη κατανομή, προσομοιώνοντας έτσι το αποτέλεσμα της χαρτογράφησης σε πραγματικό μηχάνημα. Η κατασκευή του cluster από virtual machines περιγράφεται στο *Παράρτημα Ι*, στο *Παράρτημα ΙΙ* περιγράφεται η εγκατάσταση του συστήματος διαχείρισης πόρων, ενώ στο *Παράρτημα ΙΙΙ* παρουσιάζεται αναλυτικά η διαδικασία ανάπτυξης του κώδικα.

5.1 *Υιοθέτηση Αλγορίθμων Χρονοδρομολόγησης*

Όπως αναφέραμε και προηγουμένως, η χρονοδρομολόγηση στον Maui βασίζεται στην απόδοση προτεραιοτήτων στις εργασίες. Παρακάτω περιγράφονται οι τροποποιήσεις που πρέπει να κάνουμε στο αρχείο maui.cfg για να υιοθετήσουμε τους επιθυμητούς αλγορίθμους χρονοδρομολόγησης.

FCFS

```
SERVWEIGHT          1
QUEUETIMEWEIGHT     1
```

LJF

```
RESWEIGHT           1
PROCWEIGHT          1
```

SJF

```
RESWEIGHT           1
PROCWEIGHT          -1
```

Επιπλέον, για να ενεργοποιήσουμε την προηγμένη αλγοριθμική τεχνική *backfill* πρέπει να προσθέσουμε:

BACKFILLPOLICY BESTFIT

Επίσης, χρησιμοποιήσαμε και την προηγμένη αλγοριθμική τεχνικά *advance reservation*, πραγματοποιώντας κράτηση μόνο για την εργασία υψηλότερης προτεραιότητας, ενώ ως πολιτική παραχώρησης κόμβων χρησιμοποιήσαμε την MINRESOURCE. Τα παραπάνω πραγματοποιήθηκαν με τις εξής προσθήκες στο maui.cfg:

RESERVATIONPOLICY CURRENTHIGHEST
NODEALLOCATIONPOLICY MINRESOURCE

5.2 CPU intensive εργασιακό φορτίο

Το 1ο εργασιακό φορτίο αποτελείται από 8 εφαρμογές, εκ των οποίων οι 7 είναι CPU intensive και η $8^{η}$ memory intensive, με εκτιμώμενο memory intensity 14. Ως memory intensity σε όλα τα σύνολα μετρήσεων θεωρούμε το ποσοστό του διαθέσιμου memory bandwidth που χρησιμοποιεί η εφαρμογή αν εκτελεστεί μόνο σε έναν κόμβο. Σίγουρα αυτή θεώρηση έχει και μειονεκτήματα, καθώς τα πραγματικά cluster μπορεί να είναι και ετερογενή, κάνοντας δύσκολη υπόθεση τον προσδιορισμό του memory intensity της εφαρμογής, καθώς αυτό θα εξαρτάται από τον κόμβο στον οποίο θα εκτελεστεί η εφαρμογή. Έτσι για την διευκόλυνση της μελέτης οι μετρήσεις έγιναν σε ομοιόμορφους κόμβους.

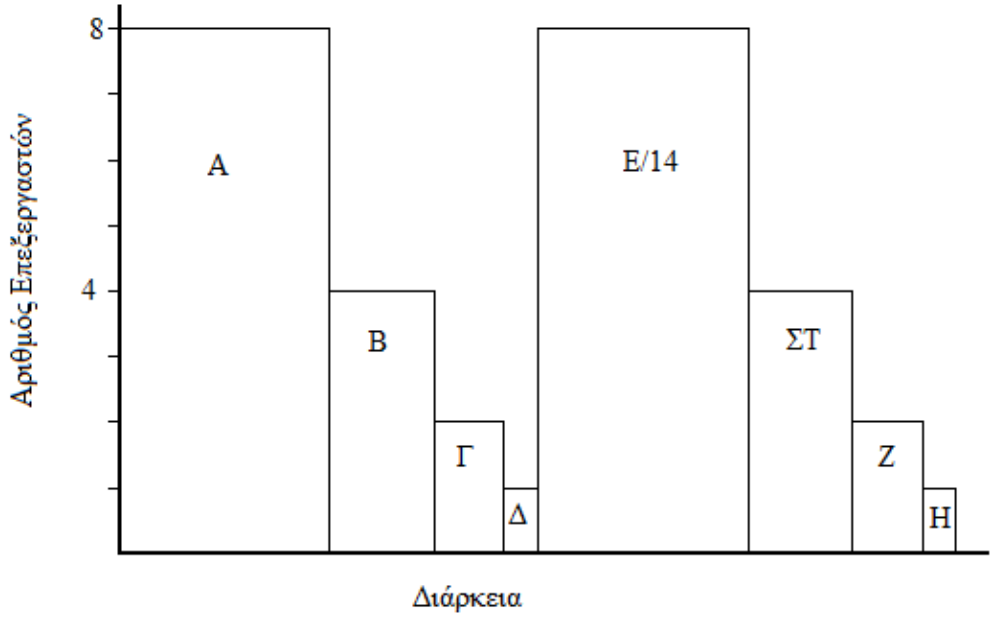
Είναι δύσκολο να οριστεί μια ανώτερη τιμή για το memory intensity. Όπως είπαμε, θεωρήσαμε το memory intensity ως ποσοστό, όπου το 0 αντιστοιχεί σε 0% του memory bandwidth και το 10 σε 100%. Οπότε, η memory intensive εφαρμογή που χρησιμοποιούμε σε αυτό το παράδειγμα απαιτεί το 140% τους διαθέσιμου memory bandwith του κόμβου.

Επιπρόσθετα, για την διεξαγωγή των μετρήσεων έγιναν οι παρακάτω παραδοχές προκειμένου να αυξήσουμε την αξιοπιστία των μετρήσεων:

- **Το χρονικό διάστημα μεταξύ του τερματισμού μιας εργασίας και της εκκίνησης της επόμενης είναι μηδενικό.** Όπως αναφέραμε και προηγουμένως, ο Maui κάνει χρονοδρομολόγηση των εφαρμογών είτε όταν πραγματοποιηθεί κάποιο καινούριο γεγονός (πχ υποβολή καινούριας εργασίας) είτε όταν λήξει ένα χρονικό διάστημα που ονομάζεται RMPOLLINTERVAL. Αυτό το χρονικό διάστημα είναι συνήθως μικρό (στο scirouter είναι 5s) και σε συνθήκες πραγματικής λειτουργίας, όπου η διάρκεια των εργασιών που υποβάλλονται στο cluster είναι της τάξης ωρών, είναι αμελητέο σε σχέση την διάρκεια των εργασιών. Όμως στην περίπτωσή μας, όπου η διάρκεια των εργασιών κυμαίνεται από 1-20s για διευκόλυνση της διεξαγωγής των μετρήσεων, αυτό το μικρό χρονικό διάστημα είναι συγκρίσιμο και σε πολλές περιπτώσεις αλλοιώνει το αποτέλεσμα των μετρήσεων, αφού προκαλεί καθυστερήσεις που μπορεί να αλλοιώσουν, για παράδειγμα, των αριθμό των εργασιών που τρέχουν μια δεδομένη στιγμή σε κάποιον κόμβο.
- **Η εκτίμηση του χρόνου εκτέλεσης των εργασιών είναι σχετικά ακριβής.** Αυτό συμβάλει στην αποδοτικότερη αξιοποίηση και μελέτη της τεχνικής *backfill*. Για παράδειγμα, αν στο συγκεκριμένο εργασιακό φορτίο, όπου οι

εφαρμογές μας διαρκούν από 1-20s, βάλουμε τον ίδιο εκτιμώμενο χρόνο εκτέλεσης σε όλες τις εργασίες, τότε είναι πιθανό να αποτραπούν οι μικρές εργασίες από το να γίνουν backfill, λόγω μελλοντικών κρατήσεων, μειώνοντας έτσι την αποδοτικότητα του συστήματος.

Παρακάτω παρουσιάζεται ένα σχήμα που περιγράφει την σειρά εμφάνισης (από τα αριστερά προς τα δεξιά) και το μέγεθος των εφαρμογών και ένας πίνακας που παρουσιάζει αναλυτικά στοιχεία των εφαρμογών, όπως το μέγεθος, ο αναμενόμενος χρόνος εκτέλεσης και το memory intensity. Για την απλοποίηση των υπολογισμών υποθέτουμε ότι όλες οι εργασίες υποβάλλονται την ίδια χρονική στιγμή και ότι η σειρά εμφάνισης αποτελεί ουσιαστικά την σειρά με την οποία εισάγονται σε μια FIFO ουρά προς εκτέλεση.



Σχήμα 5.1 Εργασίες του cpu intensive εργασιακού φορτίου

Πίνακας 5.1 Χαρακτηριστικά εργασιών cpu intensive εργασιακού φορτίου

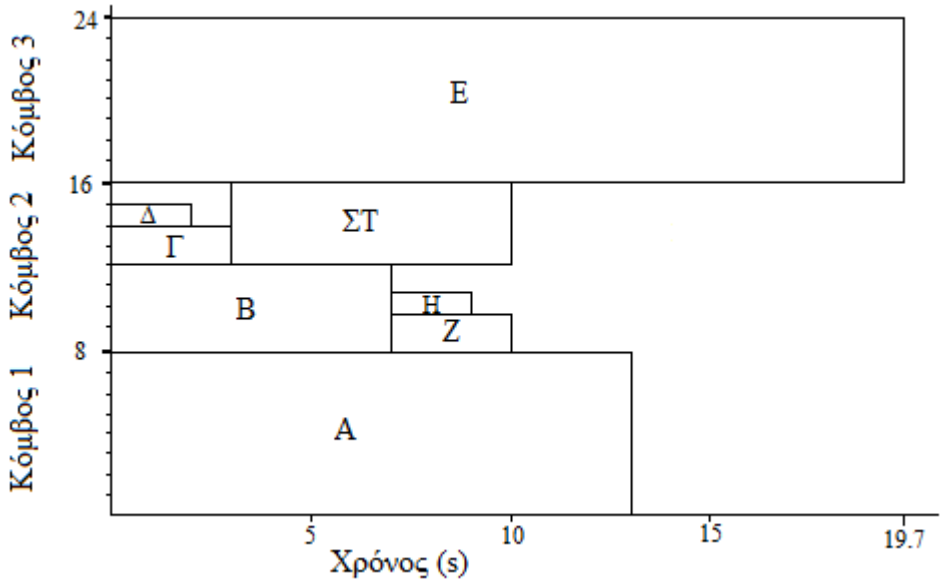
Εργασία	Αριθμός επεξεργαστών	Αναμενόμενος χρόνος εκτέλεσης (s)	Memory intensity
A	8	12.8	0
B	4	6.5	0
Γ	2	3.7	0
Δ	1	1.8	0
E	8	19.7	14
ΣΤ	4	6.5	0
Z	2	3.7	0
H	1	1.8	0

Στην συνέχεια παρουσιάζεται η εφαρμογή των αλγορίθμων στο συγκεκριμένο εργασιακό φορτίο.

5.2.1 FCFS

Παρακάτω παρουσιάζεται η χαρτογράφηση των εργασιών με την χρησιμοποίηση του αλγορίθμου FCFS καθώς και ένας πίνακας με μετρήσεις και χρήσιμες στατιστικές, τόσο στην περίπτωση χρησιμοποίησης του αλγορίθμου backfill όσο και χωρίς.

<u>Χωρίς backfill</u>

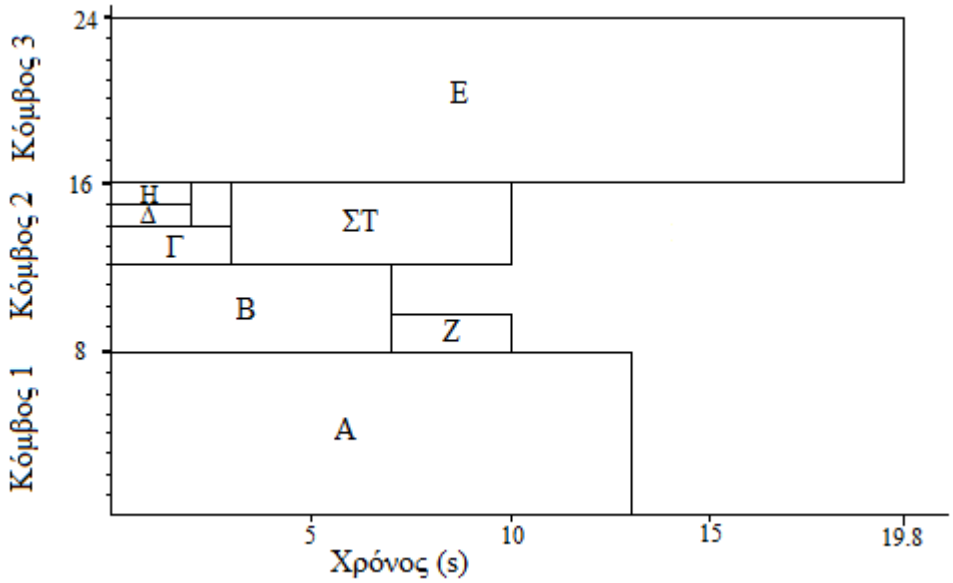


Σχήμα 5.2 Χαρτογράφηση εργασιών 1ου εργασιακού φορτίου-FCFS χωρίς backfill

Πίνακας 5.2 Αποτελέσματα 1ου εργασιακού φορτίου-FCFS χωρίς backfill

Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
A	12.88	0	12.88	**19.7**
B	7.09	0	7.09	
Γ	3.9	0	3.9	
Δ	1.86	0	1.86	
E	19.7	0	19.7	
ΣΤ	6.89	3.9	10.79	
Z	3.47	7.09	10.56	
H	1.83	7.09	8.92	
Μέσος Όρος	**7.20**	**2.26**	**9.46**	

Με backfill



Σχήμα 5.3 Χαρτογράφηση εργασιών 1ου εργασιακού φορτίου-FCFS με backfill

Πίνακας 5.3 Αποτελέσματα 1ου εργασιακού φορτίου-FCFS με backfill

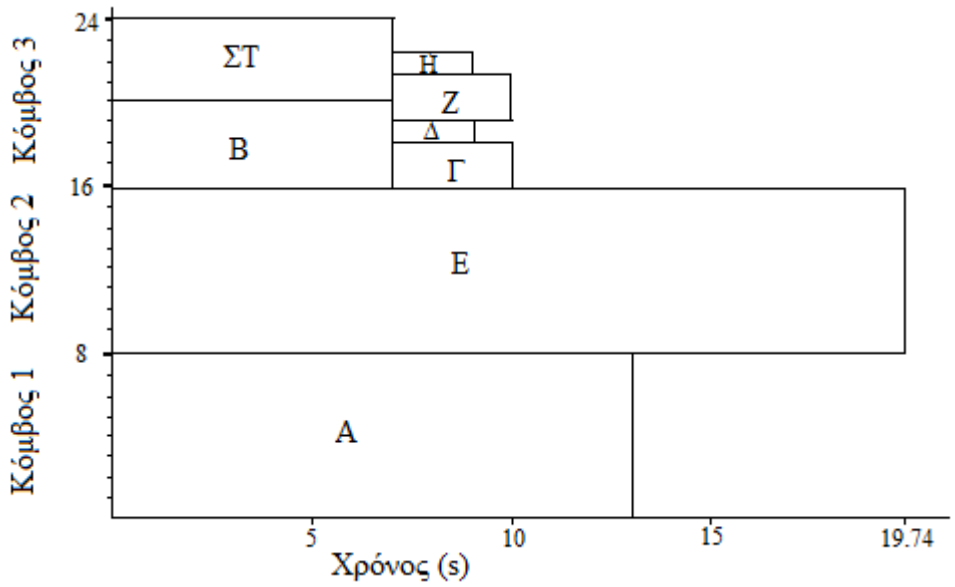
Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
A	12.93	0	12.93	**19.8**
B	6.93	0	6.93	
Γ	3.88	0	3.88	
Δ	1.86	0	1.86	
E	19.8	0	19.8	
ΣΤ	6.66	3.88	10.54	
Z	3.59	6.93	10.52	
H	1.85	0	1.85	
Μέσος Όρος	**7.19**	**1.35**	**8.54**	

Παρατηρήσεις

- Αρχικά παρατηρούμε ότι οι 2 περιπτώσεις διαφέρουν ελάχιστα μεταξύ τους. Αυτό συμβαίνει γιατί στο συγκεκριμένο παράδειγμα έχουμε αυξημένη χρησιμοποίηση συστήματος, καθώς όπως φαίνεται και από το σχήμα δεν υπάρχουν μεγάλα διαστήματα που οι επεξεργαστές των κόμβων μένουν ανενεργοί.
- Συγκεκριμένα μόνο η εργασία H γίνεται backfill. Παρόλα αυτά όμως βλέπουμε σημαντική βελτίωση τόσο στο μέσο χρόνο αναμονής που από 2.26 s γίνεται 1.35 s, δηλαδή βελτίωση 40.3%, αλλά και στο μέσο χρόνο ολοκλήρωσης που από 9.46 s γίνεται 8.54 s, δηλαδή βελτίωση 9.73%.

5.2.2 LJF

Παρακάτω παρουσιάζεται η χαρτογράφηση των εργασιών με την χρησιμοποίηση του αλγορίθμου LJF καθώς και ο αντίστοιχος πίνακας με τις μετρήσεις και χρήσιμες στατιστικές. Στην συγκεκριμένη περίπτωση η χρησιμοποίηση της τεχνικής backfill δεν επιφέρει κανένα απολύτως αποτέλεσμα, καθώς δεν υπάρχουν κενά διαστήματα όπου επεξεργαστές μένουν ανενεργοί, ώστε να καλυφθούν με εργασίες που γίνονται backfill.



Σχήμα 5.4 Χαρτογράφηση εργασιών 1ου εργασιακού φορτίου-LJF

Πίνακας 5.4 Αποτελέσματα 1ου εργασιακού φορτίου-LJF

Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
A	12.88	0	12.88	**19.74**
B	6.52	0	6.52	
Γ	3.91	6.52	10.43	
Δ	1.85	6.52	8.37	
E	19.74	0	19.74	
ΣΤ	6.91	0	6.91	
Z	3.93	6.91	10.84	
H	1.85	6.91	8.76	
Μέσος Όρος	**7.2**	**3.26**	**10.46**	

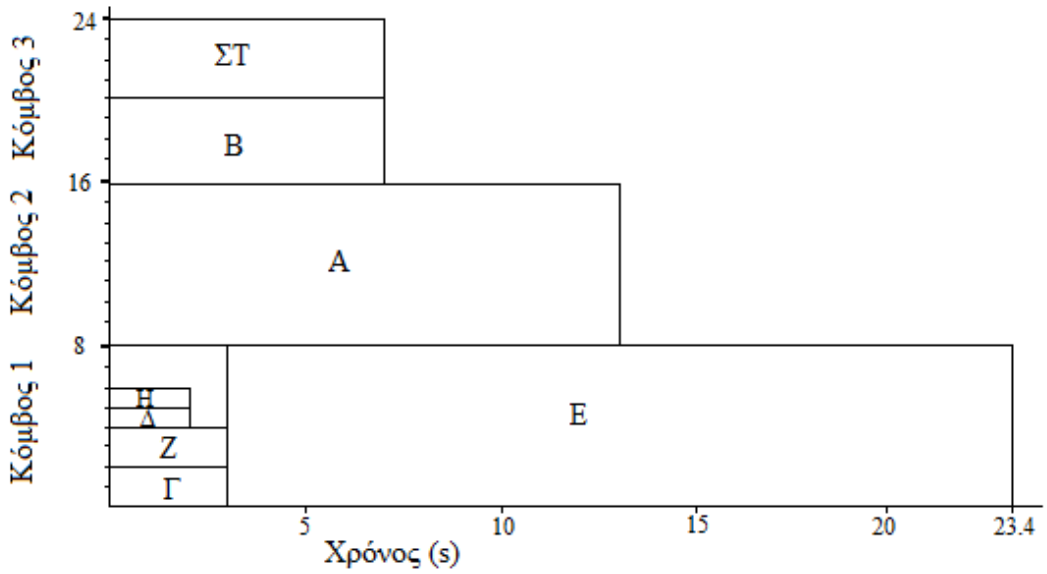
Παρατηρήσεις

- Παρατηρούμε ότι με την χρησιμοποίηση του αλγορίθμου LJF έχουμε άριστη χρησιμοποίηση συστήματος, καθώς δεν έχουμε καθόλου χρονικά διαστήματα όπου επεξεργαστές κόμβων μένουν ανενεργοί.
- Ο συνολικός χρόνος ολοκλήρωσης όλων των εργασιών παραμένει ουσιαστικά ο ίδιος, αφού επηρεάζεται μόνο από τον χρόνο εκτέλεσης της εργασίας Ε, που είναι η πιο χρονοβόρα.

- Ίδιος παραμένει και ο μέσος χρόνος εκτέλεσης, αφού δεν υπάρχει συναγωνισμός των εργασιών για κοινούς πόρους στο συγκεκριμένο εργασιακό φορτίο.
- Αντίθετα, ο μέσος χρόνος αναμονής και κατά συνέπεια ο μέσος χρόνος ολοκλήρωσης αυξάνονται σημαντικά, παρά την άριστη χρησιμοποίηση συστήματος, καθώς οι μικρές εργασίες μπλοκάρουν μέχρι να ολοκληρωθούν οι μεγαλύτερες.

5.2.3 SJF

Παρακάτω παρουσιάζεται η χαρτογράφηση των εργασιών με την χρησιμοποίηση του αλγορίθμου SJF καθώς και ο αντίστοιχος πίνακας με τις μετρήσεις και χρήσιμες στατιστικές. Στην περίπτωση του αλγορίθμου SJF δεν έχει ιδιαίτερο νόημα η εφαρμογή του αλγορίθμου backfill, καθώς οι μικρές εργασίες έτσι και αλλιώς εκτελούνται πριν τις μεγαλύτερες.



Σχήμα 5.5 Χαρτογράφηση εργασιών 1ου εργασιακού φορτίου-SJF

Πίνακας 5.5 Αποτελέσματα 1ου εργασιακού φορτίου-SJF

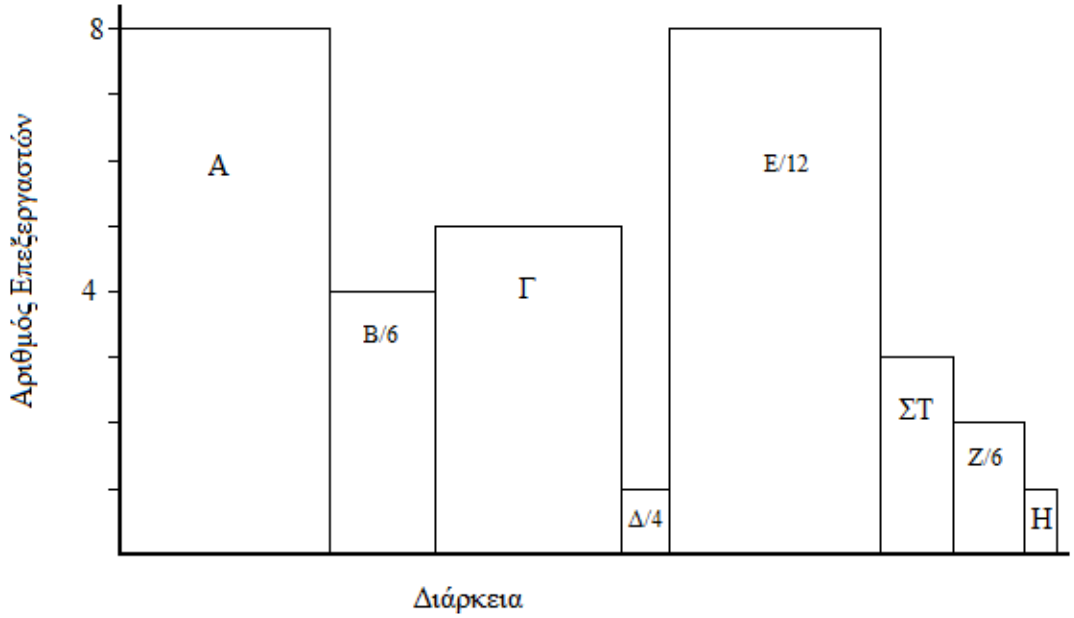
Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
A	12.88	0	12.88	**23.41**
B	6.42	0	6.42	
Γ	3.71	0	3.71	
Δ	1.87	0	1.87	
E	19.7	3.71	23.41	
ΣΤ	6.61	0	6.61	
Z	3.4	0	3.4	
H	1.86	0	1.86	
Μέσος Όρος	**7.06**	**0.46**	**7.52**	

Παρατηρήσεις

- Αρχικά παρατηρούμε ότι ο συνολικός χρόνος ολοκλήρωσης των εργασιών αυξάνεται, καθώς η εργασία Ε που έχει την μεγαλύτερη διάρκεια εκτέλεσης και στις προηγούμενες περιπτώσεις καθόριζε τον συνολικό χρόνο, μπλοκάρει πίσω από μικρότερες εργασίες. Έτσι, ο συνολικός χρόνος αυξάνεται κατά 15.7% σε σχέση με τους 2 προηγούμενους αλγορίθμους.
- Ο μέσος χρόνος εκτέλεσης παραμένει ο ίδιος για τον λόγο που αναφέραμε και προηγουμένως.
- Αντίθετα, ο μέσος χρόνος αναμονής βελτιώνεται σημαντικά, καθώς μόνο η εργασία Ε μπλοκάρει και για μικρό χρονικό διάστημα.
- Η βελτίωση του μέσου χρόνου αναμονής συμπαρασύρει και την βελτίωση του μέσου χρόνου ολοκλήρωσης των εργασιών.

5.3 Ισορροπημένο εργασιακό φορτίο

Το 2^{ο} εργασιακό φορτίο αποτελείται από 4 cpu intensive εφαρμογές και 4 memory intensive. Τροποποιήσαμε και τον αριθμό των επεξεργαστών που απαιτούν ορισμένες εργασίες για να μελετήσουμε την συμπεριφορά των αλγορίθμων σε περισσότερο ανομοιόμορφα σύνολα. Στο παρακάτω σχήμα παρουσιάζεται η σειρά εμφάνισης και το μέγεθος των εργασιών και στη συνέχεια ο πίνακας με τα χαρακτηριστικά τους. Η κάθετος δίπλα στο όνομα της εργασίας μας δείχνει πόσο memory intensive είναι η εφαρμογή.



Σχήμα 5.6 Εργασίες ισορροπημένου εργασιακού φορτίου

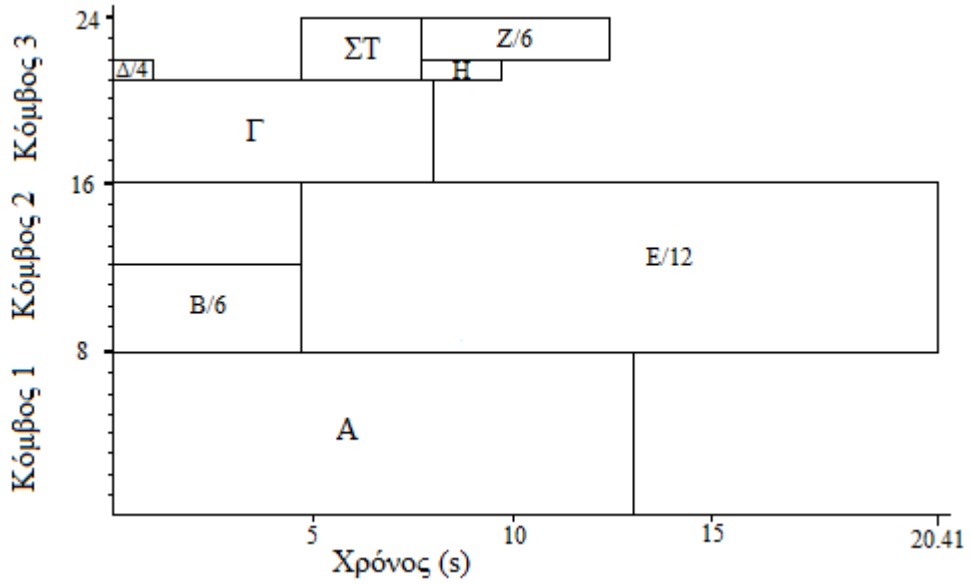
Πίνακας 5.6 Χαρακτηριστικά εργασιών ισορροπημένου εργασιακού φορτίου

Εργασία	Αριθμός επεξεργαστών	Αναμενόμενος χρόνος εκτέλεσης (s)	Memory intensity
Α	8	12.8	0
Β	4	4.7	6
Γ	5	8.5	0
Δ	1	1	4
Ε	8	15.5	12
ΣΤ	3	3.2	0
Ζ	2	4.7	6
Η	1	1.9	0

5.3.1 FCFS

Παρακάτω παρουσιάζεται η χαρτογράφηση των εργασιών με την χρησιμοποίηση του αλγορίθμου FCFS για το 2^{ο} εργασιακό φορτίο καθώς και ένας πίνακας με τις μετρήσεις και χρήσιμες στατιστικές, τόσο στην περίπτωση χρησιμοποίησης του αλγορίθμου backfill όσο και χωρίς.

Χωρίς backfill

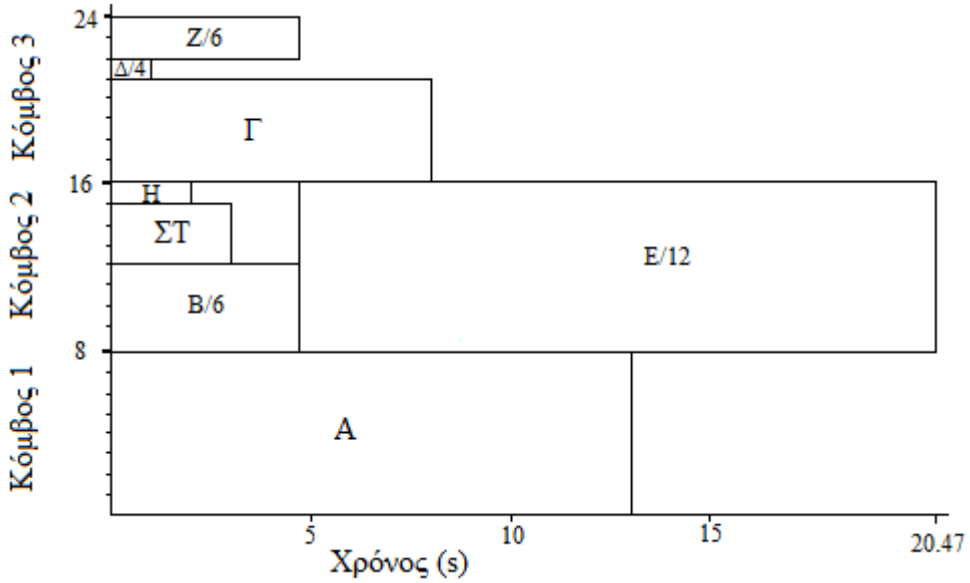


Σχήμα 5.7 Χαρτογράφηση εργασιών 2ου εργασιακού φορτίου-FCFS χωρίς backfill

Πίνακας 5.7 Αποτελέσματα 2ου εργασιακού φορτίου-FCFS χωρίς backfill

Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
Α	12.91	0	12.91	**20.41**
Β	4.69	0	4.69	
Γ	8.41	0	8.41	
Δ	0.98	0	0.98	
Ε	15.72	4.69	20.41	
ΣΤ	3.2	4.69	7.89	
Ζ	4.57	7.89	12.46	
Η	1.85	7.89	9.74	
Μέσος Όρος	**6.54**	**3.15**	**9.69**	

Με backfill



Σχήμα 5.8 Χαρτογράφηση εργασιών 2ου εργασιακού φορτίου-FCFS με backfill

Πίνακας 5.8 Αποτελέσματα 2ου εργασιακού φορτίου-FCFS με backfill

Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
Α	12.94	0	12.94	**20.47**
Β	4.62	0	4.62	
Γ	8.5	0	8.5	
Δ	1	0	1	
Ε	15.85	4.62	20.47	
ΣΤ	3.1	0	3.1	
Ζ	4.67	0	4.67	
Η	1.89	0	1.89	
Μέσος Όρος	**6.57**	**0.58**	**7.15**	

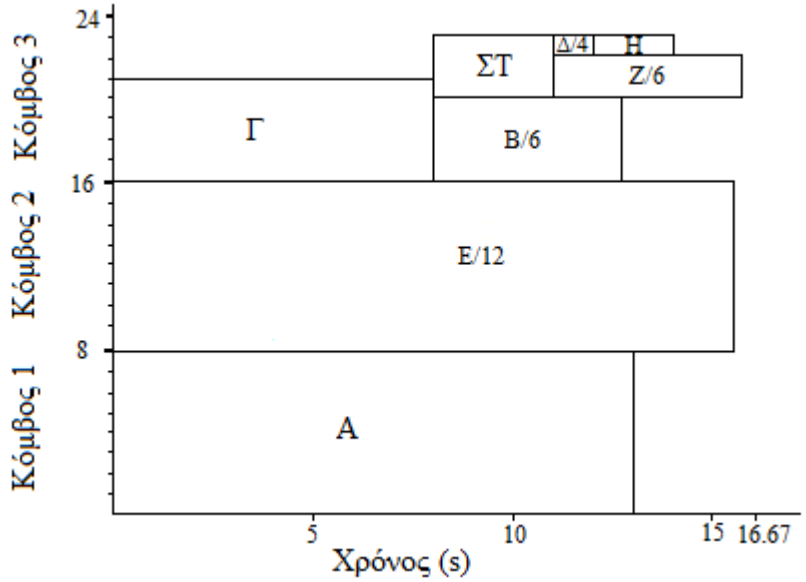
Παρατηρήσεις

- Αρχικά παρατηρούμε ότι ο συνολικός χρόνος ολοκλήρωσης των εργασιών είναι ίδιος και στις 2 περιπτώσεις, αφού καθορίζεται από το άθροισμα των χρόνων εκτέλεσης των εργασιών Β και Ε.
- Στην περίπτωση χρησιμοποίησης του backfill όμως, έχουμε πολύ καλύτερη χρησιμοποίηση των πόρων, αφού τα διαστήματα που μένουν ανενεργοί οι επεξεργαστές είναι πολύ λιγότερα.
- Ο μέσος χρόνος εκτέλεσης των εργασιών είναι ίδιος και στις 2 περιπτώσεις, αφού με τον τρόπο που κατανέμονται οι εργασίες στους κόμβους δεν έχουμε ανταγωνισμό για τον διάδρομο μνήμης. Το μέγιστο άθροισμα memory intensity των εργασιών σε έναν κόμβο οποιαδήποτε χρονική στιγμή δεν ξεπερνάει το 10, γεγονός που σημαίνει ότι το memory bandwidth του κόμβου αρκεί για να καλύψει τις απαιτήσεις των εργασιών.
- Η καλύτερη χρησιμοποίηση συστήματος οδηγεί σε τεράστια μείωση του μέσου χρόνου αναμονής αφού από 3.14 s γίνεται 0.58 s, δηλαδή μείωση 81.5%, και κατά συνέπεια μείωση του μέσου χρόνου ολοκλήρωσης εργασιών από 9.69 s σε 7.15 s, δηλαδή μείωση 26.3%.

5.3.2 LJF

Παρακάτω παρουσιάζεται η χαρτογράφηση των εργασιών με την χρησιμοποίηση του αλγορίθμου LJF για το 2^{ο} εργασιακό φορτίο, καθώς και ένας πίνακας με τις μετρήσεις και χρήσιμες στατιστικές, τόσο στην περίπτωση χρησιμοποίησης του αλγορίθμου backfill όσο και χωρίς, αφού σε αντίθεση με το 1^{ο} εργασιακό φορτίο δεν έχουμε μέγιστη χρησιμοποίηση συστήματος, με αποτέλεσμα η χρησιμοποίηση του αλγορίθμου backfill να έχει μεγάλη διαφορά.

Χωρίς backfill

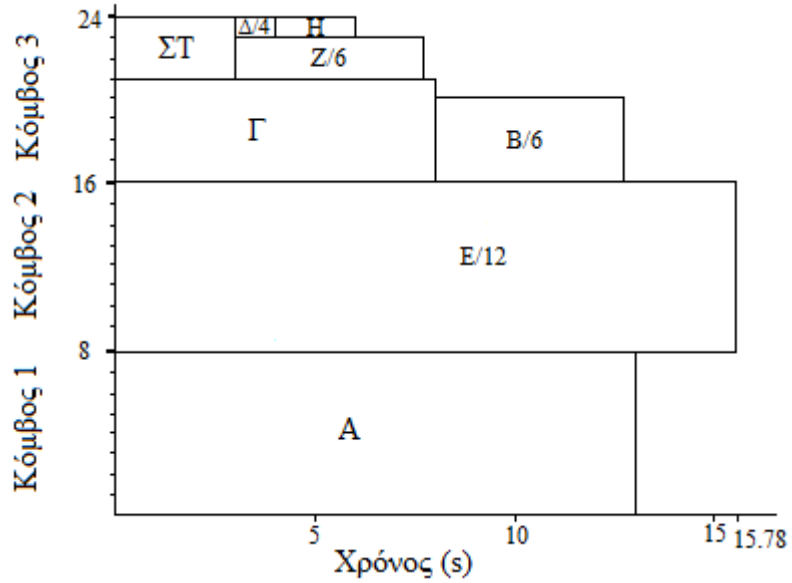


Σχήμα 5.9 Χαρτογράφηση εργασιών 2ου εργασιακού φορτίου-LJF χωρίς backfill

Πίνακας 5.9 Αποτελέσματα 2ου εργασιακού φορτίου-LJF χωρίς backfill

Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
Α	12.83	0	12.83	**16.67**
Β	5.12	8.53	13.65	
Γ	8.53	0	8.53	
Δ	1.28	11.65	12.93	
Ε	15.6	0	15.6	
ΣΤ	3.12	8.53	11.65	
Ζ	5.02	11.65	16.67	
Η	1.84	12.93	14.77	
Μέσος Όρος	**6.67**	**6.66**	**13.33**	

Με backfill



Σχήμα 5.10 Χαρτογράφηση εργασιών 2ου εργασιακού φορτίου-LJF με backfill

Πίνακας 5.10 Αποτελέσματα 2ου εργασιακού φορτίου-LJF με backfill

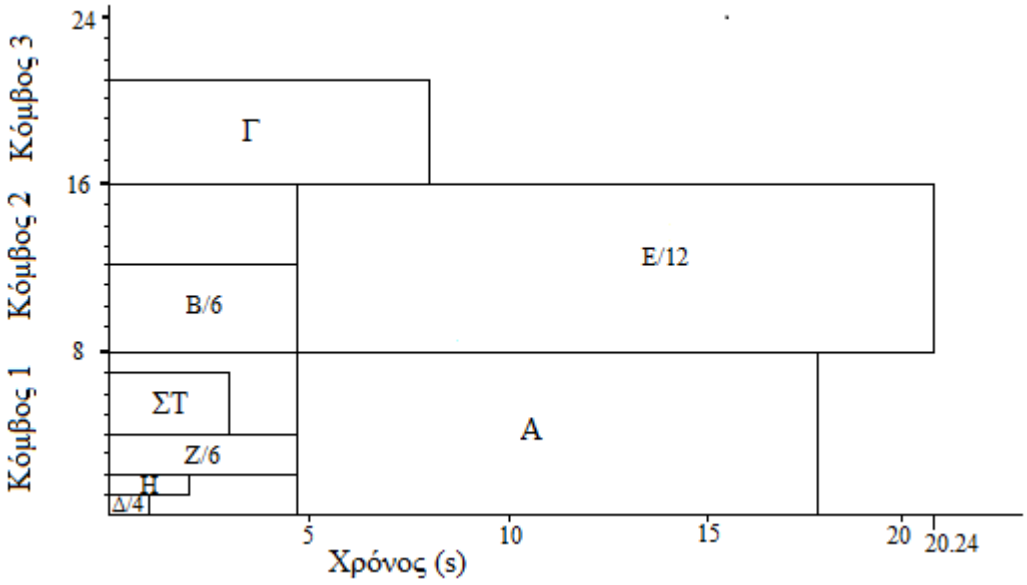
Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
Α	12.87	0	12.87	**15.78**
Β	4.69	8.5	13.19	
Γ	8.5	0	8.5	
Δ	0.97	3.22	4.19	
Ε	15.78	0	15.78	
ΣΤ	3.22	8.5	11.72	
Ζ	4.73	3.22	7.95	
Η	1.87	4.19	6.06	
Μέσος Όρος	**6.58**	**3.45**	**10.03**	

Παρατηρήσεις

- Παρατηρούμε ότι με την χρησιμοποίηση της τεχνικής backfill πετυχαίνουμε καλύτερα αποτελέσματα σχεδόν σε όλα τα μεγέθη στο συγκεκριμένο παράδειγμα.
- Ο συνολικός χρόνος ολοκλήρωσης μειώνεται καθώς οι μικρότερες εργασίες Δ, Z, H δεν μπλοκάρουν περιμένοντας να εκτελεστεί πρώτα η εργασία ΣΤ. Έτσι έχουμε μείωση περίπου 5% του συνολικού χρόνου εκτέλεσης, αφού από τα 16.67 s πηγαίνουμε στα 15.78 s.
- Για πρώτη φορά βλέπουμε μείωση στον χρόνο εκτέλεσης, αφού στην $1^{η}$ περίπτωση εκτελούνται στον κόμβο 3 για κάποιο χρονικό διάστημα ταυτόχρονα οι εργασίες Δ, Z, B που έχουν άθροισμα memory intensity 16, με αποτέλεσμα να ανταγωνίζονται έντονα για τον διάδρομο μνήμης και να αυξηθεί ο χρόνος εκτέλεσής τους. Σε αυτό το σημείο πρέπει να τονίσουμε ότι τεχνική backfill μπορεί να βοήθησε στην οριακή βελτίωση, αλλά θα μπορούσε σε κάποια άλλη περίπτωση να είχε τα αντίστροφα αποτελέσματα. Πιο συγκεκριμένα αν για παράδειγμα η εργασία B δεν ήταν memory intensive, αλλά ήταν η Γ, η τεχνική του backfill θα χειροτέρευε το μέσο χρόνο εκτέλεσης. Με λίγα λόγια η βελτίωση του μέσου χρόνου εκτέλεσης είναι ένα σχετικά τυχαίο γεγονός, που εξαρτάται από το εργασιακό φορτίο και δεν είναι αποτέλεσμα του αλγορίθμου backfill.
- Ο μέσος χρόνος αναμονής και ο μέσος χρόνος ολοκλήρωσης όμως, σχετίζονται σε μεγάλο βαθμό με τον αλγόριθμο backfill. Εδώ παρατηρούμε μείωση 48% του μέσου χρόνου αναμονής και 24.8% στον μέσο χρόνο ολοκλήρωσης.
- Σε σύγκριση με τον αλγόριθμο FCFS, ο αλγόριθμος LJF παρέχει καλύτερη χρησιμοποίηση συστήματος, αφού οι μεγάλες εργασίες προηγούνται και δεν καθυστερούνται από μικρότερες εργασίες, που μπλοκάρουν τους κόμβους. Έτσι, έχουμε βελτίωση στο συνολικό χρόνο ολοκλήρωσης, αφού οι χρονοβόρες και μεγάλες εργασίες ξεκινούν άμεσα, ενώ έχουμε επιδείνωση του μέσου χρόνου αναμονής και του μέσου χρόνου ολοκλήρωσης, καθώς οι λίγες μεγάλες εργασίες καταλαμβάνουν μεγάλα μέρη των κόμβων και οι πολλές μικρότερες εργασίες μπλοκάρουν και περιμένουν ολοκληρωθούν οι μεγαλύτερες.

5.3.3 SJF

Παρακάτω παρουσιάζεται η χαρτογράφηση των εργασιών με την χρησιμοποίηση του αλγορίθμου SJF για το $2^{ο}$ εργασιακό φορτίο καθώς και ένας πίνακας με τις μετρήσεις και χρήσιμες στατιστικές.



Σχήμα 5.11 Χαρτογράφηση εργασιών 2ου εργασιακού φορτίου-SJF

Πίνακας 5.11 Αποτελέσματα 2ου εργασιακού φορτίου-SJF

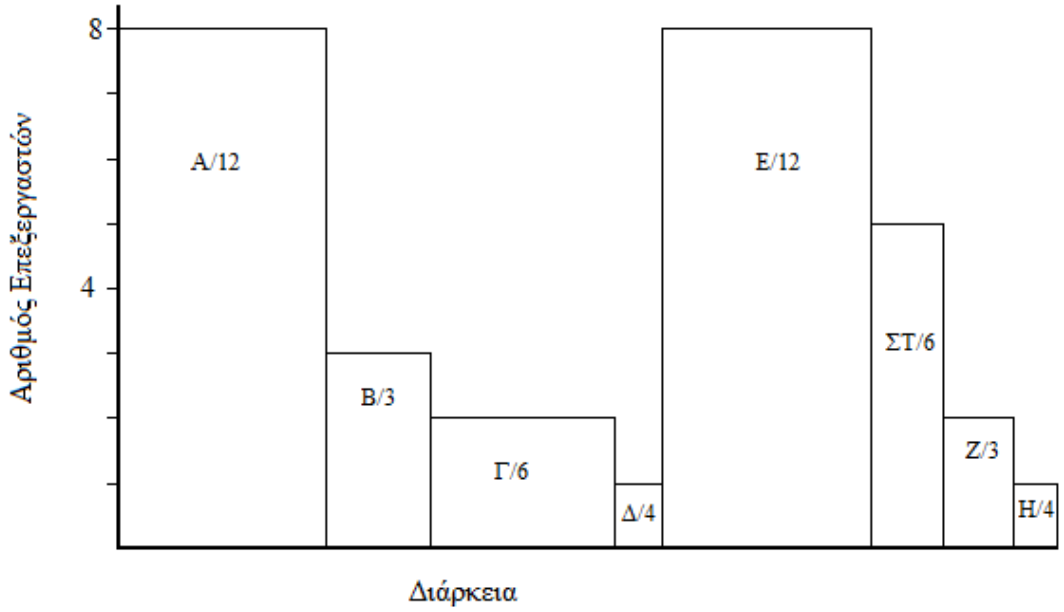
Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
A	12.9	4.67	17.57	**20.24**
B	4.7	0	4.7	
Γ	8.65	0	8.65	
Δ	1.02	0	1.02	
E	15.54	4.7	20.24	
ΣΤ	3.07	0	3.07	
Z	4.67	0	4.67	
H	1.85	0	1.85	
Μέσος Όρος	**6.55**	**1.17**	**7.72**	

Παρατηρήσεις

- Αρχικά παρατηρούμε ότι ο συνολικός χρόνος ολοκλήρωσης των εργασιών είναι παρόμοιος με του αλγορίθμου FCFS, καθώς η εργασία Ε μπλοκάρει πίσω από την εργασία Β.
- Ο μέσος χρόνος εκτέλεσης παραμένει σε φυσιολογικά επίπεδα, αφού δεν εκτελούνται ταυτόχρονα σε κάποιον κόμβο εργασίες, που ανταγωνίζονται έντονα για τον διάδρομο μνήμης.
- Τέλος, όπως και στο προηγούμενο σύνολο μετρήσεων, ο μέσος χρόνος αναμονής και ο μέσος χρόνος ολοκλήρωσης είναι μικρότεροι σε σύγκριση με τους αντίστοιχους του αλγορίθμου LJF, αλλά αυτήν την φορά μεγαλύτεροι από τους αντίστοιχους της FCFS, αφού εκτός από την εργασία Ε, καθυστερεί να ξεκινήσει και η εργασία Α.
- Ο συνολικός χρόνος εκτέλεσης είναι και πάλι μεγαλύτερος από τον αντίστοιχο της LJF, αλλά στα ίδια επίπεδα με της FCFS.

5.4 *Memory intensive εργασιακό φορτίο*

Το 3ο εργασιακό φορτίο αποτελείται μόνο από memory intensive εφαρμογές. Μεταβάλαμε επίσης και την σειρά εμφάνισης των εργασιών σε σχέση με τον χρόνο εκτέλεσης που απαιτούν, για να ολοκληρώσουμε ένα σχετικά ευρύ φάσμα δυνατών περιπτώσεων. Στο παρακάτω σχήμα παρουσιάζεται η σειρά εμφάνισης και το μέγεθος των εργασιών και στην συνέχεια ο πίνακας με τα χαρακτηριστικά τους.



Σχήμα 5.12 Εργασίες memory intensive εργασιακού φορτίου

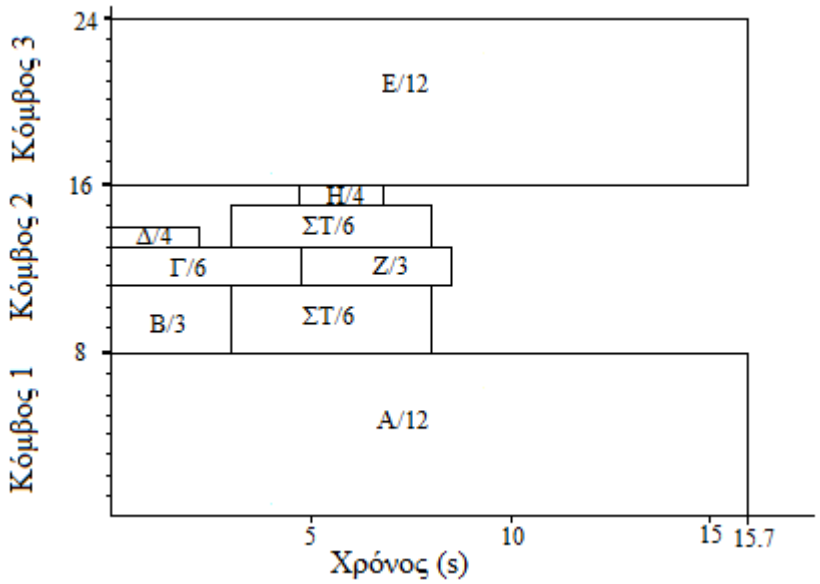
Πίνακας 5.12 Χαρακτηριστικά εργασιών memory intensive εργασιακού φορτίου

Εργασία	Αριθμός επεξεργαστών	Αναμενόμενος χρόνος εκτέλεσης (s)	Memory intensity
Α	8	15.5	0
Β	3	2.3	6
Γ	2	4.7	0
Δ	1	1	4
Ε	8	15.5	12
ΣΤ	5	4.7	0
Ζ	2	2.3	6
Η	1	1	0

5.4.1 FCFS

Παρακάτω παρουσιάζεται η χαρτογράφηση των εργασιών με την χρησιμοποίηση του αλγορίθμου FCFS για το 3^{ο} εργασιακό φορτίο, καθώς και ένας πίνακας με τις μετρήσεις και χρήσιμες στατιστικές, τόσο στην περίπτωση χρησιμοποίησης του αλγορίθμου backfill όσο και χωρίς.

Χωρίς backfill

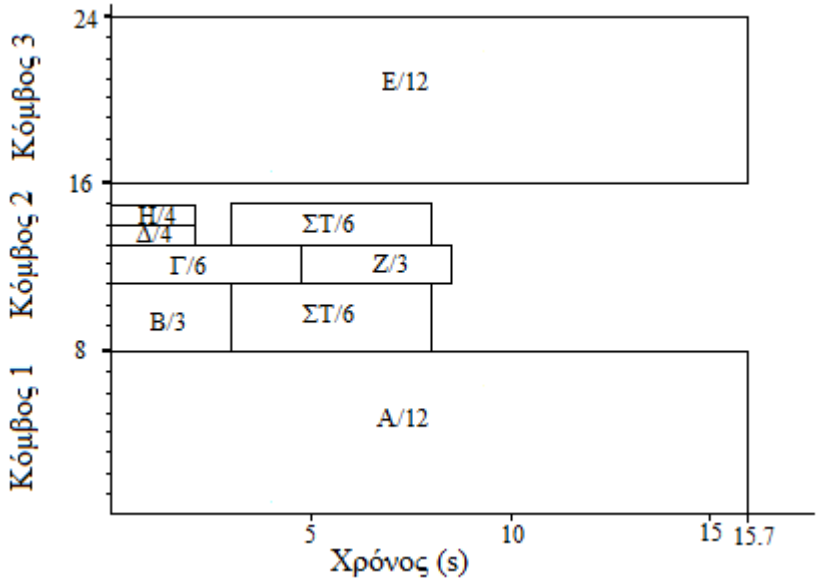


Σχήμα 5.13 Χαρτογράφηση εργασιών 3ου εργασιακού φορτίου-FCFS χωρίς backfill

Πίνακας 5.13 Αποτελέσματα 3ου εργασιακού φορτίου-FCFS χωρίς backfill

Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
A	15.7	0	15.7	**15.7**
B	2.89	0	2.89	
Γ	4.85	0	4.85	
Δ	2.2	0	2.2	
E	15.64	0	15.64	
ΣΤ	5.3	2.89	8.19	
Z	3.78	4.85	8.63	
H	2.1	4.85	6.95	
Μέσος Όρος	**6.56**	**1.57**	**8.13**	

Με backfill



Σχήμα 5.14 Χαρτογράφηση εργασιών 3ου εργασιακού φορτίου-FCFS με backfill

Πίνακας 5.14 Αποτελέσματα 3ου εργασιακού φορτίου-FCFS με backfill

Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
A	15.7	0	15.7	**15.7**
B	3.25	0	3.25	
Γ	5.02	0	5.02	
Δ	2.25	0	2.25	
E	15.64	0	15.64	
ΣΤ	4.65	3.25	7.9	
Z	2.3	5.02	7.32	
H	2.03	0	1.63	
Μέσος Όρος	**6.36**	**1.03**	**7.39**	

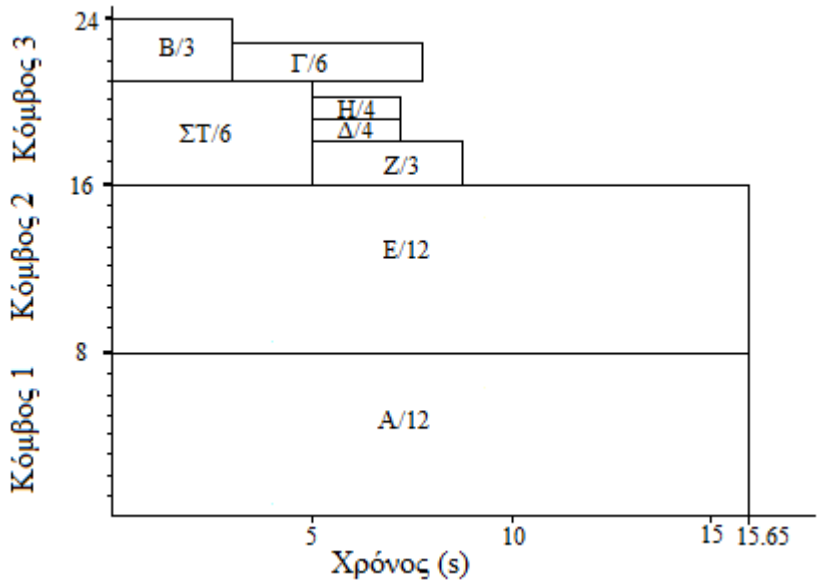
Παρατηρήσεις

- Αρχικά παρατηρούμε ότι ο συνολικός χρόνος εκτέλεσης είναι ο ίδιος, αφού εξαρτάται από τον χρόνο εκτέλεσης των εργασιών A, E.
- Ο μέσος χρόνος εκτέλεσης επηρεάζεται και στις 2 περιπτώσεις από την παράλληλη εκτέλεση memory intensive εργασιών στον ίδιο κόμβο, που ξεπερνούν το memory bandwidth του κόμβου.
- Ο μέσος χρόνος αναμονής είναι περίπου 35% μικρότερος όταν χρησιμοποιείται ο αλγόριθμος backfill, όπως άλλωστε είχαμε δει και στα προηγούμενα εργασιακά φορτία.
- Ο μέσος χρόνος ολοκλήρωσης των εργασιών μειώνεται με την χρησιμοποίηση του αλγορίθμου backfill, καθώς και ο μέσος χρόνος εκτέλεσης (τυχαίο γεγονός όπως είπαμε και προηγουμένως) και ο μέσος χρόνος αναμονής μειώνονται.

5.4.2 LJF

Παρακάτω παρουσιάζεται η χαρτογράφηση των εργασιών με την χρησιμοποίηση του αλγορίθμου LJF για το 3^{ο} εργασιακό φορτίο καθώς και ένας πίνακας με τις μετρήσεις και χρήσιμες στατιστικές. Σε αυτήν την περίπτωση ο αλγόριθμος backfill επιφέρει μικρές αλλαγές, αφού επηρεάζει τον χρόνο εκκίνησης μόνο μιας εργασίας.

<u>Χωρίς backfill</u>

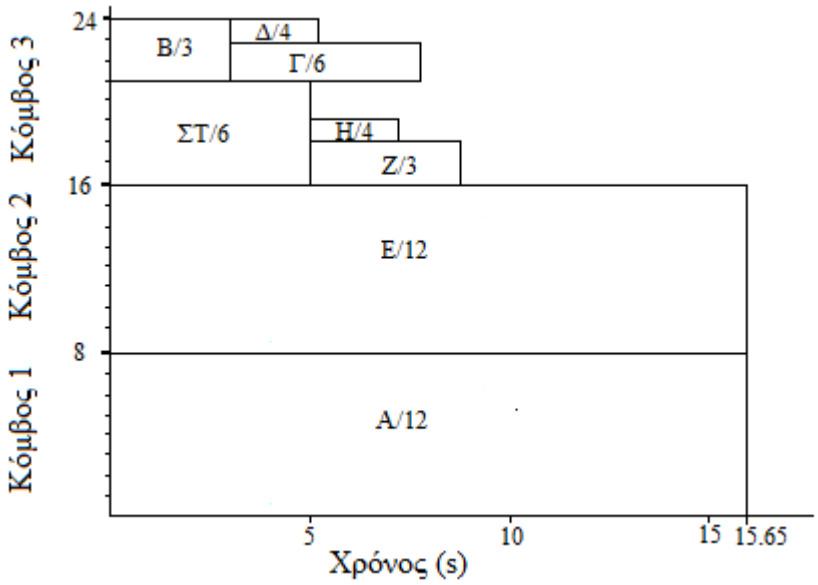


Σχήμα 5.15 Χαρτογράφηση εργασιών 3ου εργασιακού φορτίου-LJF χωρίς backfill

Πίνακας 5.15 Αποτελέσματα 3ου εργασιακού φορτίου-LJF χωρίς backfill

Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
Α	15.65	0	15.65	**15.65**
Β	2.32	0	2.32	
Γ	5.8	2.32	8.12	
Δ	3.6	4.65	8.25	
Ε	15.6	0	15.6	
ΣΤ	4.65	0	4.65	
Ζ	5.02	4.65	9.67	
Η	1.71	4.65	6.36	
Μέσος Όρος	**6.79**	**2.04**	**8.83**	

Με backfill



Σχήμα 5.16 Χαρτογράφηση εργασιών 3ου εργασιακού φορτίου-LJF με backfill

Πίνακας 5.16 Αποτελέσματα 3ου εργασιακού φορτίου-LJF με backfill

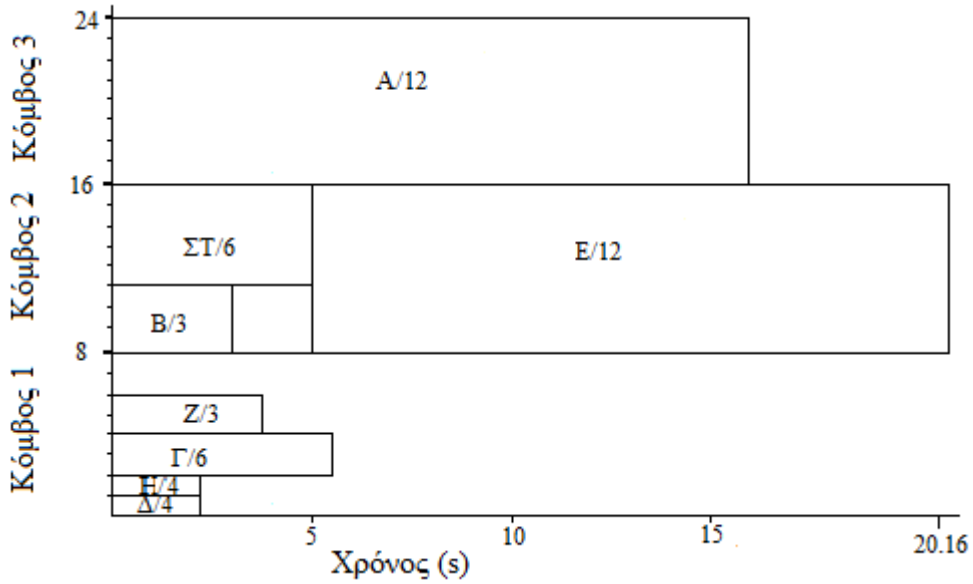
Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
Α	15.65	0	15.65	**15.65**
Β	2.32	0	2.32	
Γ	5.59	2.32	7.91	
Δ	3.34	2.32	5.66	
Ε	15.6	0	15.6	
ΣΤ	4.81	0	4.81	
Ζ	4.72	4.81	9.53	
Η	1.58	4.81	6.39	
Μέσος Όρος	**6.7**	**1.78**	**8.48**	

Παρατηρήσεις

- Ο συνολικός χρόνος ολοκλήρωσης των εργασιών είναι ο ίδιος και στις 2 περιπτώσεις και εξαρτάται από τους χρόνους εκτέλεσης των εργασιών Α, Ε.
- Ο μέσος χρόνος εκτέλεσης των εργασιών είναι περίπου ο ίδιος, αλλά λόγω του γεγονότος ότι η εργασία Δ γίνεται backfill, μειώνεται ο μέσος χρόνος αναμονής και ο μέσος χρόνος ολοκλήρωσης στην $2^{η}$ περίπτωση.
- Σε σχέση με τον αλγόριθμο FCFS, ο αλγόριθμος LJF παρουσιάζει και σε αυτήν την περίπτωση μεγαλύτερο μέσο χρόνο αναμονής και ολοκλήρωσης των εργασιών, ενώ ο συνολικός χρόνος είναι ο ίδιος.

5.4.3 SJF

Παρακάτω παρουσιάζεται η χαρτογράφηση των εργασιών με την χρησιμοποίηση του αλγορίθμου SJF για το 3ο εργασιακό φορτίο και ο πίνακας με τις μετρήσεις και στατιστικές.



Σχήμα 5.17 Χαρτογράφηση εργασιών 3ου εργασιακού φορτίου-SJF

Πίνακας 5.17 Αποτελέσματα 3ου εργασιακού φορτίου-SJF

Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
A	15.61	0	15.61	**20.16**
B	2.35	0	2.35	
Γ	5.39	0	5.39	
Δ	1.84	0	1.84	
E	15.54	4.62	20.16	
ΣΤ	4.62	0	4.62	
Z	2.9	0	2.9	
H	1.89	0	1.89	
Μέσος Όρος	**6.27**	**0.58**	**6.85**	

Παρατηρήσεις

- Ο συνολικός χρόνος εκτέλεσης των εργασιών είναι μεγαλύτερος περίπου κατά 20% από τους αντίστοιχους των 2 προηγούμενων αλγορίθμων.
- Ο μέσος χρόνος εκτέλεσης είναι περίπου ο ίδιος και στους 3 αλγορίθμους. Μικρή διαφορά παρουσιάζεται μόνο στον αλγόριθμο LJF.
- Ο μέσος χρόνος αναμονής και ολοκλήρωσης είναι μικρότερος και από τους 2 προηγούμενους αλγορίθμους, πράγμα που είναι λογικό λόγω της υψηλότερης προτεραιότητας των μικρότερων εργασιών.

5.5 Συμπεράσματα

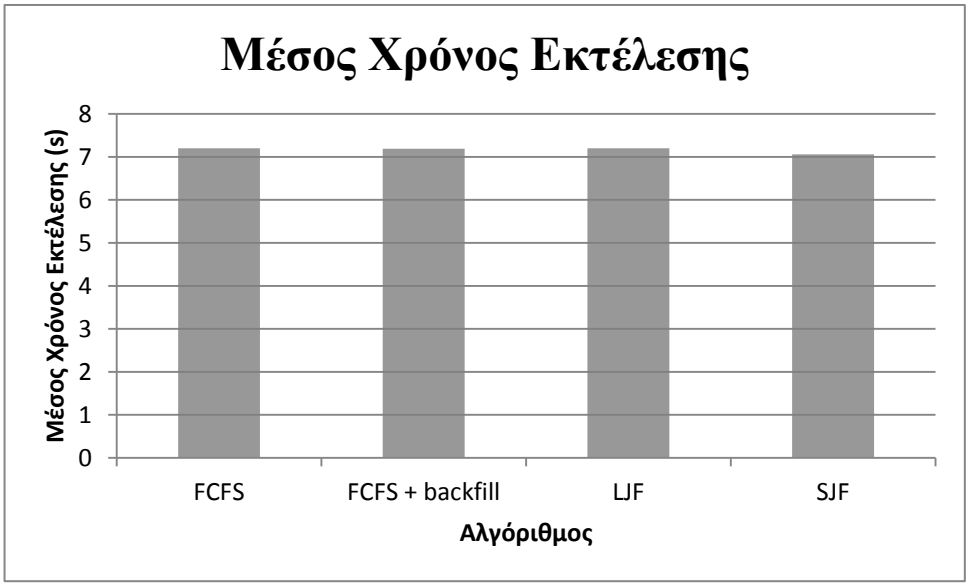
Στον παρακάτω πίνακα συνοψίζονται τα αποτελέσματα των 3 εργασιακών φορτίων:

Πίνακας 5.18 Σύνοψη αποτελεσμάτων

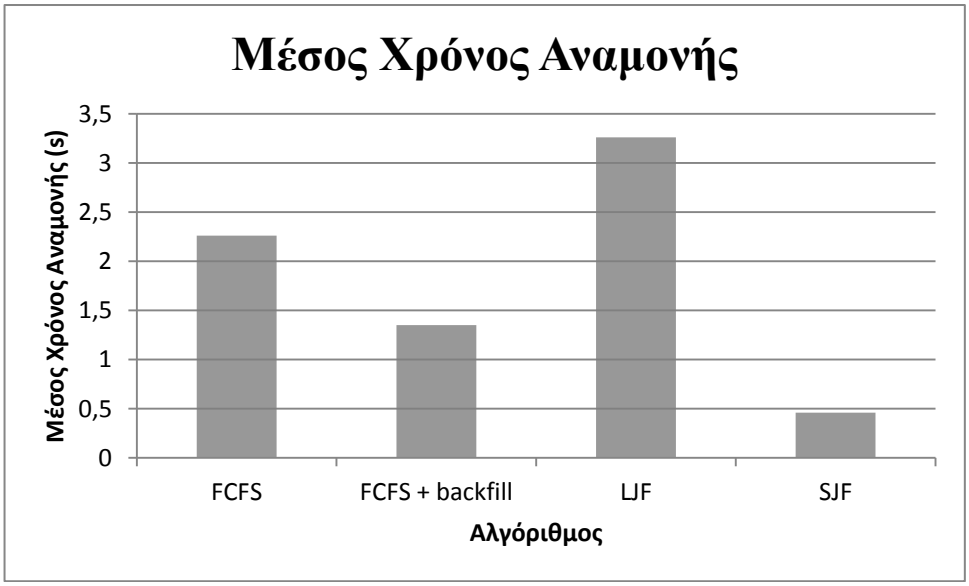
Εργασιακό Φορτίο	Αλγόριθμος	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
CPU intensive	**FCFS**	7.2	2.26	9.46	19.7
	FCFS + backfill	7.19	1.35	8.54	19.8
	LJF	7.2	3.26	10.46	19.74
	SJF	7.06	0.46	7.52	23.41
Balanced	**FCFS**	6.54	3.15	9.69	20.41
	FCFS + backfill	6.57	0.56	7.13	20.47
	LJF	6.67	6.66	13.33	16.67
	LJF + backfill	6.58	3.45	10.03	15.78
	SJF	6.55	1.17	7.72	20.24
Memory intensive	**FCFS**	6.56	1.57	8.13	15.7
	FCFS + backfill	6.36	1.03	7.39	15.7
	LJF	6.79	2.03	8.82	15.65
	LJF + backfill	6.7	1.78	8.48	15.65
	SJF	6.27	0.58	6.85	20.16

Για την καλύτερη εποπτεία των αποτελεσμάτων ακολουθούν γραφικές παραστάσεις των παραπάνω μετρικών, ταξινομημένες ανά εργασιακό φορτίο.

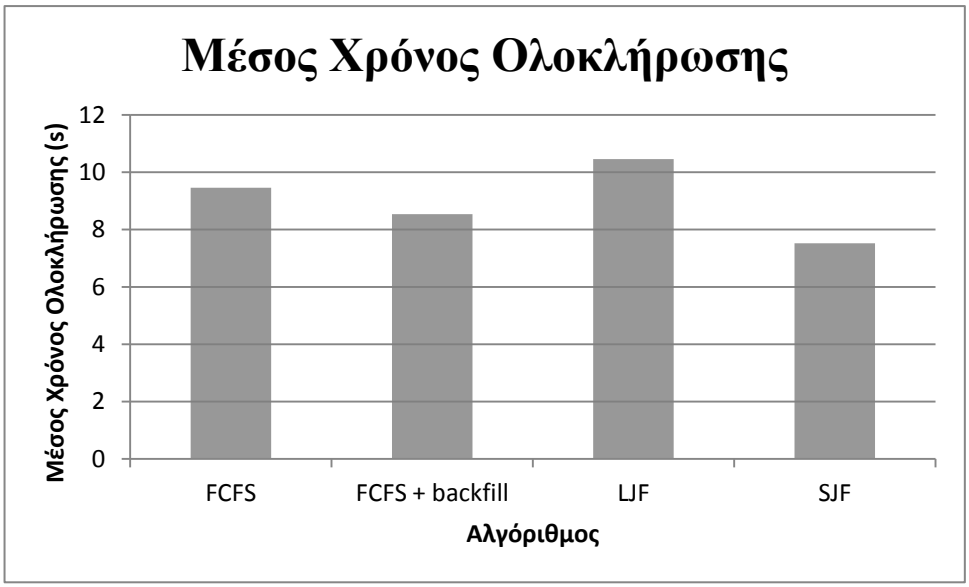
CPU intensive εργασιακό φορτίο



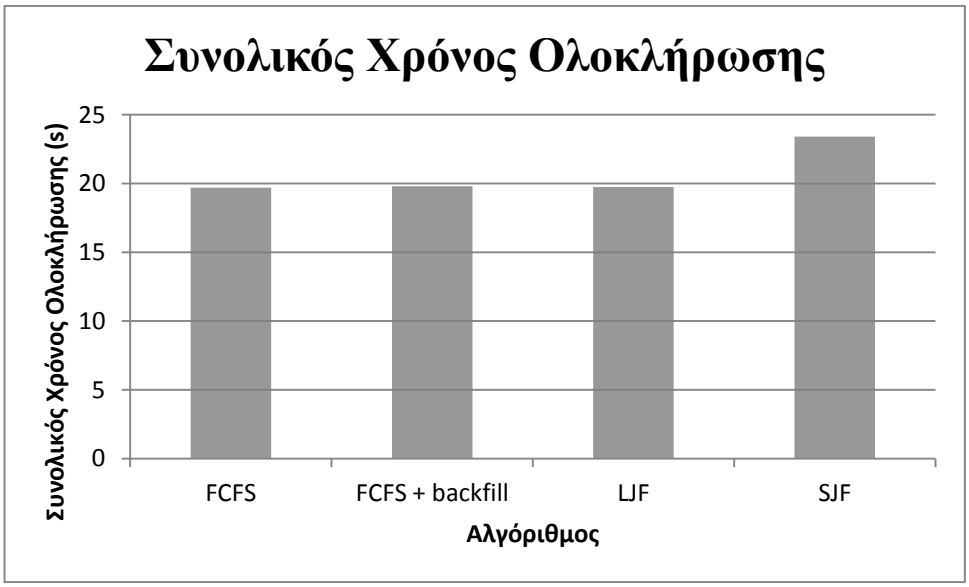
Σχήμα 5.18 CPU intensive workload-Μέσος Χρόνος Εκτέλεσης



Σχήμα 5.19 CPU intensive workload-Μέσος Χρόνος Αναμονής

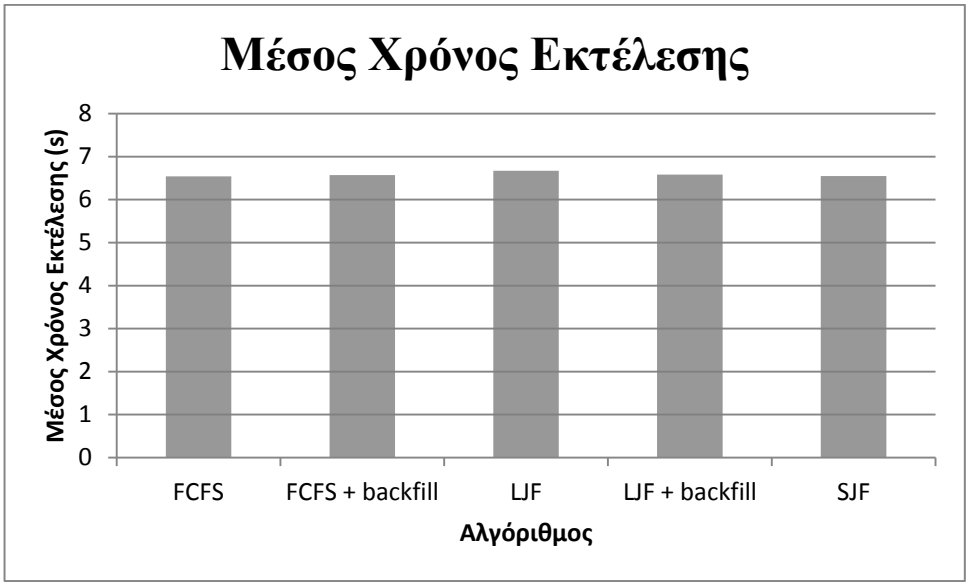


Σχήμα 5.20 CPU intensive workload-Μέσος Χρόνος Ολοκλήρωσης

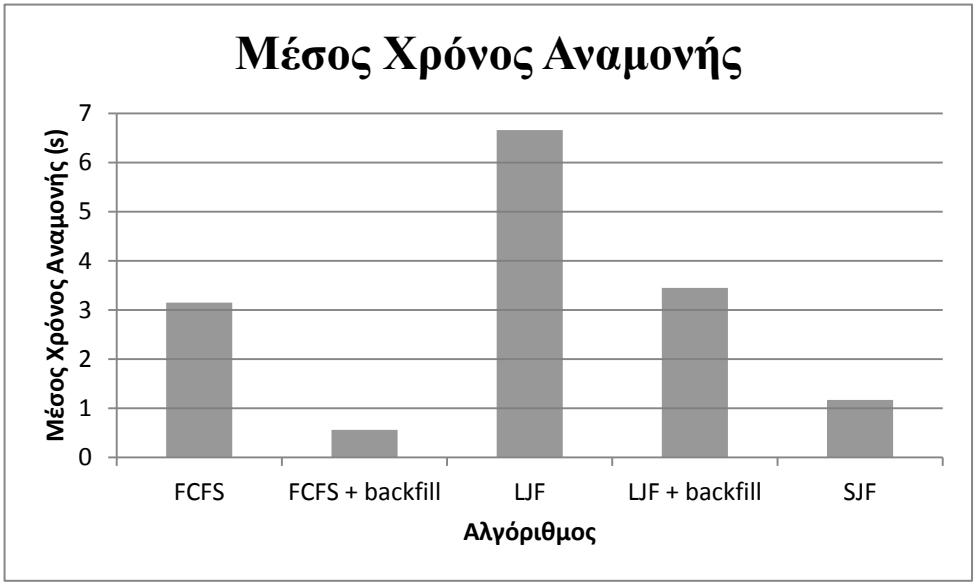


Σχήμα 5.21 CPU intensive workload-Συνολικός Χρόνος Ολοκλήρωσης

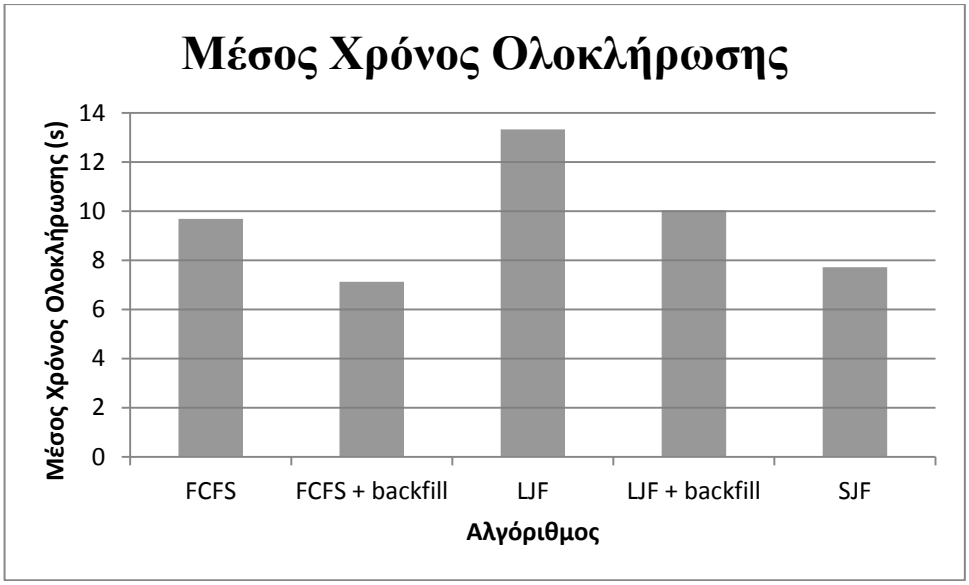
Ισορροπημένο εργασιακό φορτίο



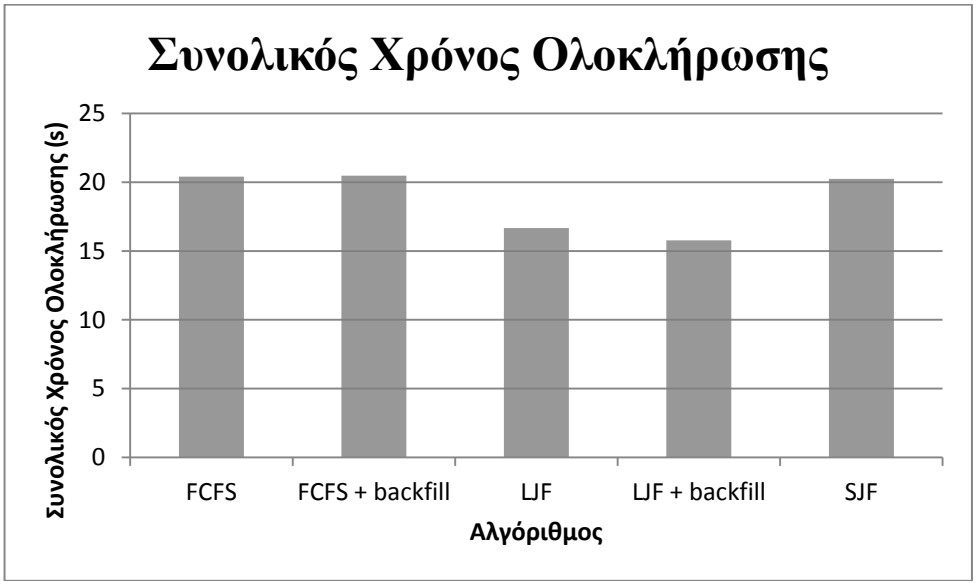
Σχήμα 5.22 Balanced workload-Μέσος Χρόνος Εκτέλεσης



Σχήμα 5.23 Balanced workload-Μέσος Χρόνος Αναμονής

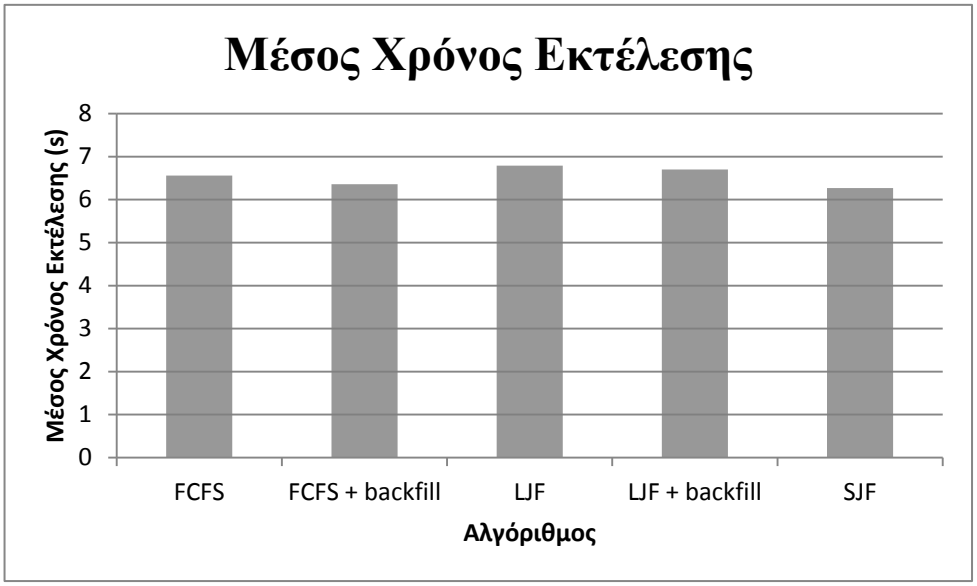


Σχήμα 5.24 Balanced workload-Μέσος Χρόνος Ολοκλήρωσης

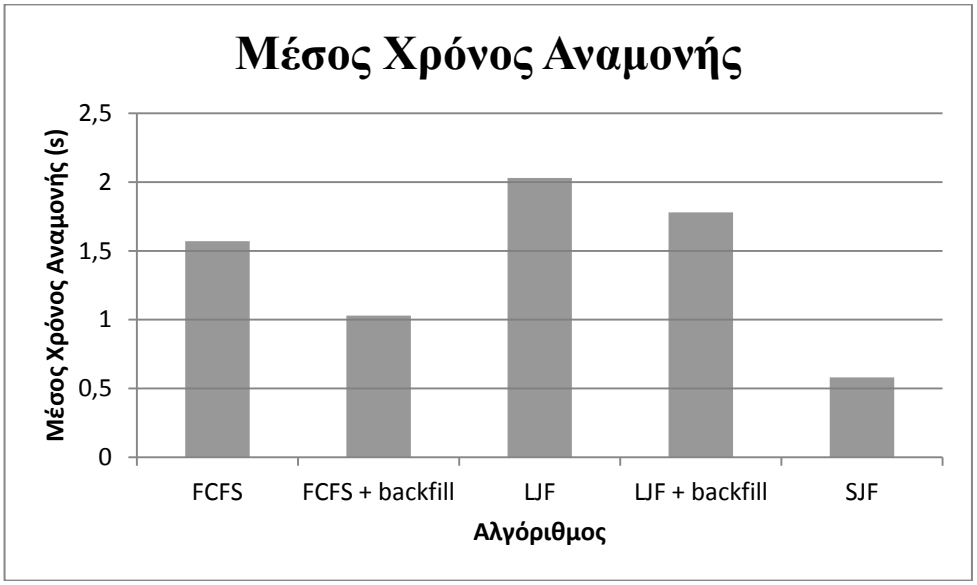


Σχήμα 5.25 Balanced workload-Συνολικός Χρόνος Ολοκλήρωσης

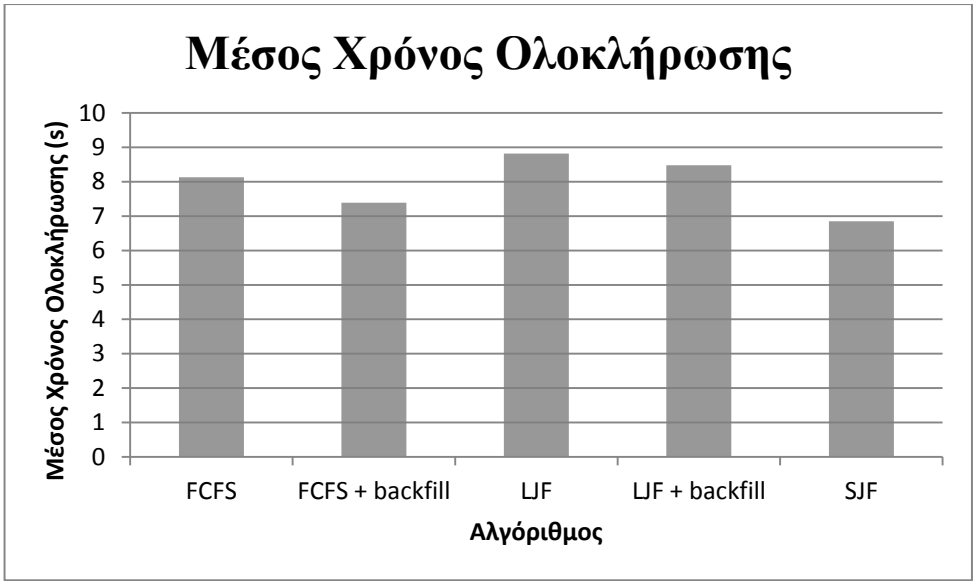
Memory intensive εργασιακό φορτίο



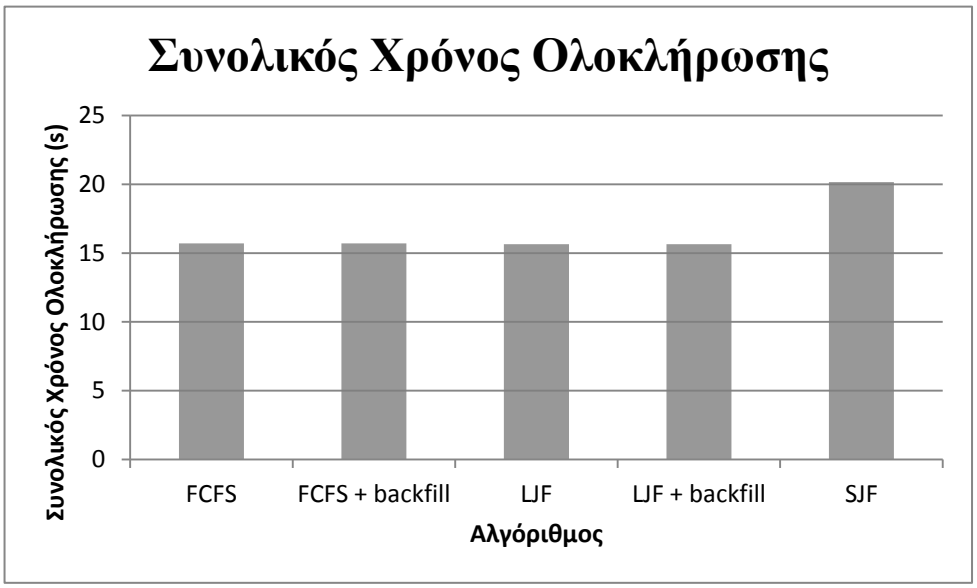
Σχήμα 5.26 Memory intensive workload-Μέσος Χρόνος Εκτέλεσης



Σχήμα 5.27 Memory intensive workload-Μέσος Χρόνος Αναμονής



Σχήμα 5.28 Memory intensive workload-Μέσος Χρόνος Ολοκλήρωσης



Σχήμα 5.29 Memory intensive workload-Συνολικός Χρόνος Ολοκλήρωσης

Αρχικά, συμπεραίνουμε ότι η χρήση του αλγορίθμου backfill, μπορεί να μην βοηθά στην μείωση του μέσου χρόνου εκτέλεσης των εργασιών, αλλά συμβάλει ουσιαστικά στην μείωση του μέσου χρόνου αναμονής και κατά συνέπεια του μέσου χρόνου ολοκλήρωσης των εργασιών.

Η μελέτη αυτήν δεν έγινε με σκοπό την επιλογή ενός αλγορίθμου που επικρατεί έναντι των υπολοίπων, καθώς και οι 3 είναι αλγόριθμοι που χρησιμοποιούνται σε πραγματικά cluster σήμερα. Ο καθένας έχει μειονεκτήματα και πλεονεκτήματα. Οι στόχοι και οι πολιτικές του site είναι αυτές που θα καθορίσουν την επιλογή του κατάλληλου αλγορίθμου χρονοδρομολόγησης.

Για παράδειγμα, παρατηρήσαμε ότι ο αλγόριθμος SJF έχει τις περισσότερες φορές καλύτερο μέσο χρόνο αναμονής και ολοκλήρωσης των εργασιών, ενώ υστερεί στον συνολικό χρόνο ολοκλήρωσης και χρησιμοποίηση συστήματος. Έτσι, ο αλγόριθμος SJF

είναι πιο πιθανό να επιλεγεί σε ένα site που έχει ως κύρια αξία την αποκρισιμότητα του συστήματος.

Αντίθετα, ο αλγόριθμος LJF παρουσιάζει καλύτερη χρησιμοποίηση συστήματος και έχει στις περισσότερες περιπτώσεις μικρότερο συνολικό χρόνο ολοκλήρωσης εργασιών, αλλά υστερεί σε μέσο χρόνο αναμονής και ολοκλήρωσης. Για αυτό, δεν θα ήταν η $1^{η}$ επιλογή σε ένα site που επιθυμεί αποκρισιμότητα, αλλά σε κάποιο που απαιτεί υψηλή χρησιμοποίηση συστήματος και ρυθμό διεκπεραίωσης εργασιών.

Ο αλγόριθμος FCFS αποτελεί μια μέση λύση ανάμεσα στους 2 προηγούμενους αλγορίθμους. Ένα επιπλέον πλεονέκτημα του αλγορίθμου FCFS, είναι ότι η εργασίες δεν κινδυνεύουν από «λιμοκτονία». Με τον όρο «λιμοκτονία» εννοούμε ότι μια εργασία έχει κάνει αίτηση για να αποκτήσει πρόσβαση σε κάποιους μοιραζόμενους πόρους, αλλά συνέχεια κάποια άλλη της «κλέβει» την θέση. Για παράδειγμα έστω ότι έχουμε έναν κόμβο με 8 επεξεργαστές και μια εργασία που απαιτεί για την εκτέλεσή της και τους 8. Αν έχουμε διαλέξει για αλγόριθμο χρονοδρομολόγησης τον SJF και υποβάλλονται διαρκώς εργασίες που απαιτούν λιγότερους από 8 επεξεργαστές, τότε η εργασία με τους 8 πυρήνες δεν θα εκτελεστεί ποτέ, αφού ο κόμβος θα είναι συνέχεια κατειλημμένος με μικρότερες εργασίες.

Για τον παραπάνω λόγο χρησιμοποιούνται συνήθως υβριδικοί αλγόριθμοι. Ο Maui για παράδειγμα μας δίνει την δυνατότητα να ορίσουμε έναν αλγόριθμο LJF που να λαμβάνει υπόψη του και τον χρόνο αναμονής στην ουρά των εργασιών. Αυτό γίνεται πιο εύκολα κατανοητό με ένα παράδειγμα.

Έστω ότι θέλουμε να φτιάξουμε μία LJF στον Maui, κάνοντας τις απαραίτητες τροποποιήσεις που αναφέραμε και προηγουμένως. Μια εργασία που απαιτεί 8 επεξεργαστές θα έχει προτεραιότητα:

PRIORITY = RESWEIGHT × PROCWEIGHT × NUMBER_OF_PROCS =
= 1 × 1 × 8 = 8

Έτσι μια εργασία που απαιτεί 1 πυρήνα δεν πρόκειται να εκτελεστεί αν συνέχεια γεμίζουν την ουρά μεγαλύτερες εργασίες. Έστω ότι τώρα θέλουμε να τροποποιήσουμε τον αλγόριθμο και να του προσθέσουμε επίσης τον παράγοντα αναμονής στην ουρά. Αυτό γίνεται προσθέτοντας στο maui.cfg:

SERVWEIGHT 1
QUEUETIME WEIGHT 1

Με αυτόν τον τρόπο μια εργασία που απαιτεί έναν επεξεργαστή και βρίσκεται στην ουρά για 8 λεπτά θα έχει προτεραιότητα:

PRIORITY = RESWEIGHT × PROCWEIGHT × NUMBER_OF_PROCS +
+ SERVWEIGHT × QUEUETIME × MINUTES_IN_QUEUE =
= 1 × 1 × 1 + 1 × 1 × 8 = 9

Δηλαδή θα ξεπεράσει σε προτεραιότητα μια εργασία 8 επεξεργαστών που μόλις υποβλήθηκε, αποφεύγοντας έτσι την λιμοκτονία.

6 *Μελέτη memory aware πολιτικής*

Σε αυτό το κεφάλαιο κατασκευάζουμε μια πολιτική που να λαμβάνει υπόψη της το memory intensity των εργασιών και να τις κατανέμει ανάλογα στους κόμβους.

6.1 Κίνητρο επιλογής memory aware πολιτικής

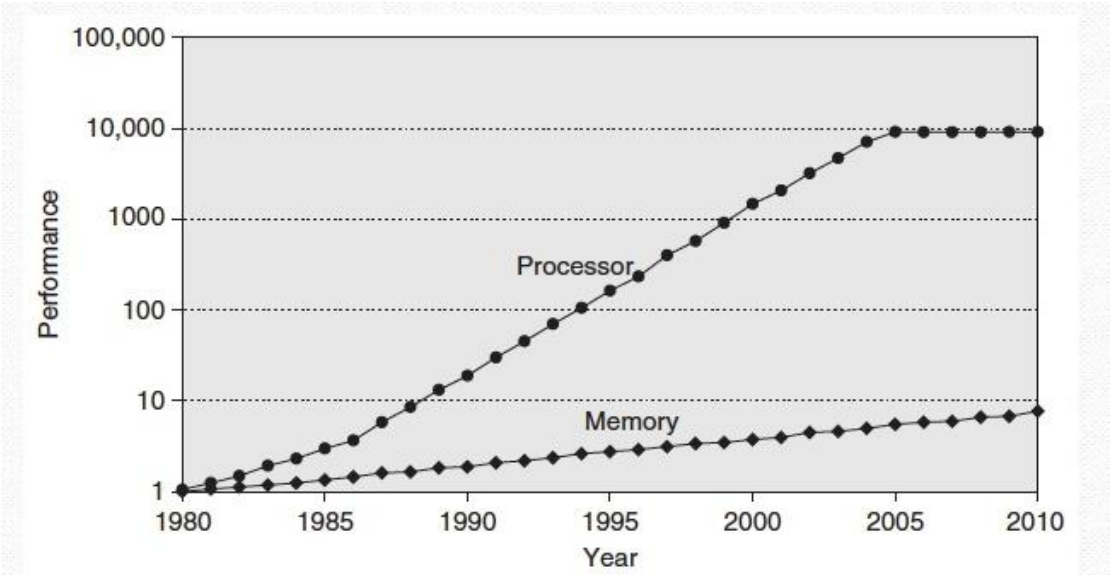
Στο προηγούμενο κεφάλαιο μελετήσαμε την συμπεριφορά τριών συχνά χρησιμοποιούμενων αλγορίθμων χρονοδρομολόγησης των συστημάτων μεγάλης κλίμακας. Σημειώσαμε τα πλεονεκτήματα και τα μειονεκτήματα αυτών των αλγορίθμων, εφαρμόζοντάς τους σε 3 διαφορετικά εργασιακά φορτία, που διέφεραν κυρίως στον αριθμό των *memory intensive* εργασιών. Τι μας οδήγησε όμως στην επιλογή των συγκεκριμένων εργασιακών φορτίων;

Στη σημερινή εποχή, τα πολυπύρηνα συστήματα έχουν κυριαρχήσει ακόμα και στους συμβατικούς υπολογιστές. Αυτό όμως εγείρει επιπλέον εμπόδια στην βέλτιστη λειτουργία των υπολογιστικών συστημάτων. Η ταυτόχρονη εκτέλεση εργασιών στο ίδιο υπολογιστικό σύστημα έχει σαν αποτέλεσμα τον ανταγωνισμό των εργασιών για τους μοιραζόμενους πόρους. Αυτός ο ανταγωνισμός έχει ως συνέπεια τόσο την επιδείνωση της επίδοσης των υπολογιστικών συστημάτων όσο και την αύξηση στην κατανάλωση ενέργειας, λόγω της διαιτησίας στην παραχώρηση των μοιραζόμενων πόρων.

Οι μοιραζόμενοι πόροι του συστήματος μπορεί να είναι η κρυφή μνήμη [39], η κύρια μνήμη [37, 38], το δίκτυο [38], ο δίσκος [40]. Εμείς στο πλαίσιο της παρούσας εργασίας αποφασίσαμε να ασχοληθούμε με το θέμα του ανταγωνισμού της κύριας μνήμης. Για αυτό τον λόγο διαλέξαμε τα εργασιακά φορτία με τέτοιο τρόπο ώστε να καλύπτουν ένα μεγάλο φάσμα περιπτώσεων ανταγωνισμού της κύριας μνήμης.

Ενώ μερικοί μοιραζόμενοι πόροι, όπως το δίκτυο και ο δίσκος είναι πολύ πιθανόν να μην χρησιμοποιούνται από μια εφαρμογή, η κύρια μνήμη είναι απαραίτητη για την εκτέλεση μιας εργασίας, αφού αποτελεί τον χώρο που θα φορτωθεί η εργασία για να εκτελεστεί. Συνειδητοποιούμε λοιπόν, ότι στον ανταγωνισμό για την κύρια μνήμη, συμμετέχουν όλες οι εργασίες, άλλες λιγότερο και άλλες περισσότερο.

Η κύρια μνήμη συγκέντρωνε πάντα το ενδιαφέρον της ερευνητικής κοινότητας τόσο λόγω του μεγάλου χρόνου πρόσβασης όσο και του περιορισμένου εύρους ζώνης που διαθέτει. Ειδικά σε αντίθεση με την ταχύτητα αύξηση των επιδόσεων των CPUs, η συμπεριφορά της κύριας μνήμης αποτελούσε πάντα ένα «αγκάθι» στον τομέα των υπολογιστικών συστημάτων. Ο όρος «Memory Wall» [41] χρησιμοποιείται για να περιγράψει την αυξανόμενη διαφορά στην ταχύτητα της CPU και της κύριας μνήμης. Ένας σημαντικός λόγος για αυτήν την ανισότητα είναι το περιορισμένο εύρος ζώνης στην επικοινωνία μεταξύ στοιχείων που ανήκουν σε διαφορετικά τσιπ, ή στην συγκεκριμένη περίπτωση το περιορισμένο *memory bandwidth*. Από το 1986 έως το 2000, η ταχύτητα των CPUs βελτιωνόταν κατά 55% σε ετήσια βάση, ενώ η ταχύτητα της μνήμης μόνο κατά 10%. Με τα εξής δεδομένα, ήταν σίγουρο ότι ο λανθάνων χρόνος της μνήμης θα γινόταν ένα σημαντικό εμπόδιο στην επίδοση των υπολογιστικών συστημάτων. Στο παρακάτω σχήμα παρουσιάζεται η μεταβολή στις επιδόσεις των CPUs και την κύριας μνήμης με την πάροδο του χρόνου.



Σχήμα 6.1 Memory Wall

Όλα τα παραπάνω έχουν οδηγήσει την ερευνητική κοινότητα των υπολογιστικών συστημάτων προς την εύρεση λύσεων στο θέμα του ανταγωνισμού. Οι λύσεις που έχουν προταθεί αφορούν τόσο στο υλικό όσο και στο λογισμικό. Για παράδειγμα, η ανακάλυψη των *double date rate* μνημών, που επιτρέπουν την μεταφορά δεδομένων τόσο στην αρνητική όσο και στην θετική ακμή του ρολογιού, έχουν ουσιαστικά διπλασιάσει το *memory bandwidth.* Επίσης, η αύξηση των καναλιών μνήμης αποτελεί μια ακόμα κίνηση προς την ίδια κατεύθυνση.

Η χρονοδρομολόγηση των εργασιών αποτελεί μία ακόμα ελκυστική λύση στην προσπάθεια επίλυσης του ανταγωνισμού για τον διάδρομο μνήμης, καθώς δεν απαιτεί επιπλέον υλικό και είναι σχετικά εύκολο να ενσωματωθεί σε ένα σύστημα. Στο [37] οι συγγραφείς μελετούν τους τρόπους που οι εργασίες επηρεάζουν η μία την άλλη όταν ανταγωνίζονται για τους μοιραζόμενους πόρους. Η έρευνά τους δεν περιορίζεται μόνο στις κρυφές μνήμες, αλλά καλύπτει και τους ελεγκτές μνήμης, τον διάδρομο μνήμης και το prefetching hardware. Στο τέλος συμπεραίνουν, ότι το μεγαλύτερο αντίκτυπο ενός *contention aware scheduler* δεν είναι τόσο στην βελτίωση της επίδοσης του συνολικού εργασιακού φορτίου, αλλά βρίσκεται κυρίως στην βελτίωση των παρεχόμενων υπηρεσιών σε συγκεκριμένες εφαρμογές και στην βελτιστοποίηση της κατανάλωσης ενέργειας.

Στο [38] οι συγγραφείς, παρουσιάζουν πειραματικά αποτελέσματα που επισημαίνουν την επιδείνωση του χρόνου εκτέλεσης των εργασιών, λόγω του ανταγωνισμού των μοιραζόμενων πόρων και συγκεκριμένα του διαδρόμου μνήμης και του δικτύου. Στην συνέχεια προτείνουν μια αλγοριθμική τεχνική που να αποτρέπει τον κορεσμό αλλά και την ανεπαρκή χρήση του διαδρόμου μνήμης και του δικτύου, με σκοπό την αύξηση του ρυθμού διεκπεραίωσης των εργασιών. Για αυτόν τον σκοπό, χωρίζουν τον χρόνο σε κβάντα, στην λογική των συστημάτων καταμερισμού χρόνου, και υπολογίζουν τις απαιτήσεις σε *memory* και *network bandwidth* από τα προηγούμενα χρονικά κβάντα. Στην συνέχεια προχωρούν σε χρονοδρομολόγηση των εργασιών ανάλογα με τις απαιτήσεις τους. Τα αποτελέσματα έδειξαν μέχρι και 48% βελτίωση στον ρυθμό διεκπεραίωσης των εργασιών.

Στην ίδια λογική, προσπαθούμε να κατασκευάσουμε μια νέα *memory aware* πολιτική, που να προσπαθεί να αντιμετωπίσει το πρόβλημα του ανταγωνισμού του διαδρόμου μνήμης, και να την προσαρμόσουμε στους αλγόριθμους χρονοδρομολόγησης που μελετήθηκαν παραπάνω. Η πολιτική που προτείνουμε ενσωματώνεται σε έναν πραγματικό χρονοδρομολογητή, τον Maui. Σκοπός μας είναι μελετήσουμε όχι μόνο την συμπεριφορά του ρυθμού διεκπεραίωσης, αλλά και άλλων μετρικών, όπως ο μέσος χρόνος εκτέλεσης, ο μέσος χρόνος αναμονής και ο μέσος χρόνος ολοκλήρωσης των εργασιών. Επιπλέον, προσπαθούμε να αναδείξουμε και τυχόν μειονεκτήματα μιας τέτοιας προσέγγισης και κάποιους τρόπους αντιμετώπισής τους.

6.2 Ανάλυση συμπεριφοράς memory intensive εργασιών

Σε αυτό το υποκεφάλαιο ελέγχουμε την συμπεριφορά των memory intensive εργασιών ως προς τον χρόνο εκτέλεσής τους για να δούμε κατά πόσο αξίζει να δημιουργήσουμε μια νέα πολιτική χρονοδρομολόγησης που να λαμβάνει υπόψη της τις memory intensive εργασίες.

6.2.1 Ταυτόχρονη εκτέλεση memory intensive εργασιών στον ίδιο κόμβο

Σε πρώτη φάση ελέγξαμε αν η κατανομή memory intensive εφαρμογών στον ίδιο κόμβο επηρεάζει τον χρόνο εκτέλεσης τους. Για τις παρακάτω μετρήσεις χρησιμοποιήσαμε τον αλγόριθμο LU decomposition. Οι μετρήσεις έγιναν επίσης στην ουρά clones.

Αρχικά πήραμε τον σειριακό αλγόριθμο και τον τρέξαμε σε έναν κόμβο με 8 επεξεργαστές. Διαλέξαμε για είσοδο έναν πίνακα double με μέγεθος 1024 , ώστε να έχουμε συνολικό μέγεθος δεδομένων 1024×1024×8 = 8 MB που είναι παραπάνω από τα 4MB cache που διέθετε ο κόμβος που κάναμε τις μετρήσεις. Έτσι, η εργασία θα αναγκαζόταν να χρησιμοποιήσει τον διάδρομο μνήμης. Στην συνέχεια αρχίσαμε να προσθέτουμε και άλλες εργασίες στον κόμβο, που εκτελούσαν τον αλγόριθμο LU decomposition σειριακά, για να δούμε σε τι βαθμό επηρεάζεται ο χρόνος εκτέλεσης όταν εκτελούνται πολλές εργασίες ταυτόχρονα. Τα αποτελέσματα παρουσιάζονται στον παρακάτω πίνακα, όπου το πλήθος δείχνει τον αριθμό των εργασιών που εκτελούνται ταυτόχρονα στον κόμβο και η μεταβολή την αύξηση στον μέσο χρόνο εκτέλεσης των εργασιών σε σχέση με τον χρόνο εκτέλεσης όταν μόνο μια εργασία εκτελείται στον κόμβο. Για καλύτερη εποπτεία παρουσιάζουμε και μία γραφική παράσταση.

Πίνακας 6.1 Ταυτόχρονη εκτέλεση σειριακών εργασιών στον ίδιο κόμβο

Πλήθος	Μέσος Χρόνος Εκτέλεσης (s)	Μεταβολή (%)
1	0.84	0
2	0.86	2.4
3	1.13	34.5
4	1.27	51.2
5	1.66	97.6
6	1.9	126.2
7	2.38	183.3
8	4	376.2

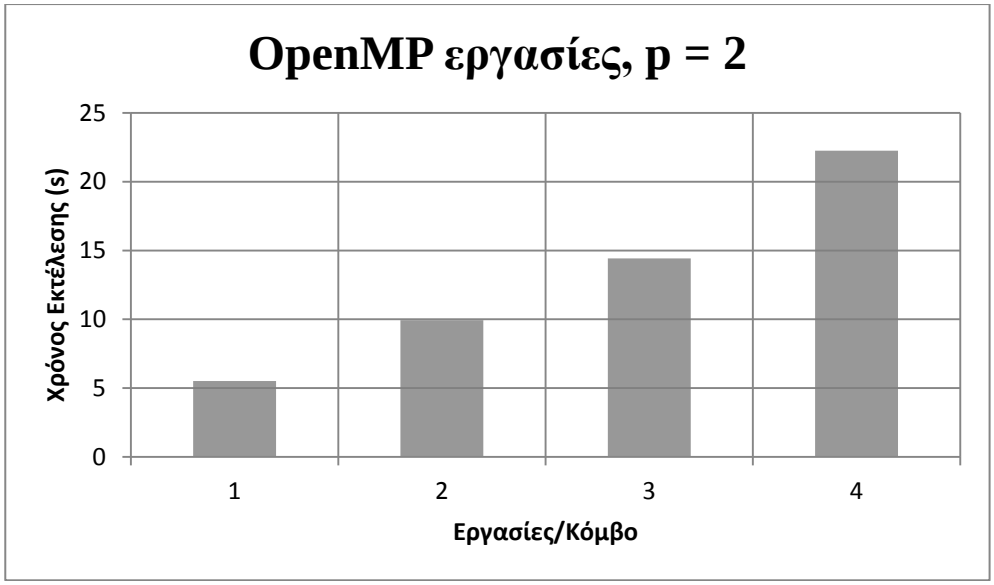
Σχήμα 6.2 Ταυτόχρονη εκτέλεσης σειριακών εργασιών στον ίδιο κόμβο

Βλέπουμε ότι όταν τρέξουμε μέχρι και 2 εργασίες στον ίδιο κόμβο τα αποτελέσματα είναι πανομοιότυπα, γεγονός που σημαίνει ότι η ταυτόχρονη εκτέλεση τους δεν απαιτεί μεγαλύτερο memory bandwidth από αυτό του κόμβου. Όμως, από την στιγμή που εκτελούμε και $3^{η}$ εργασία ταυτόχρονα, ο χρόνος εκτέλεσης αρχίζει να αυξάνεται σημαντικά, γεγονός που σημαίνει ότι έχουμε ξεπεράσει το memory bandwidth του κόμβου.

Στην συνέχεια κάνουμε παρόμοια πειράματα χρησιμοποιώντας πάλι τον αλγόριθμο LU decomposition, αλλά αυτή την φορά εκτελώντας τον παράλληλα τόσο με το μοντέλο της μοιραζόμενης μνήμης όσο και με το μοντέλο της ανταλλαγής μηνυμάτων. Στο $2^{ο}$ πείραμα εκτελέσαμε τον αλγόριθμο με 2 threads και μέγεθος πίνακα 2048, χρησιμοποιώντας το πρότυπο OpenMP. Τα αποτελέσματα παρουσιάζονται παρακάτω:

Πίνακας 6.2 Ταυτόχρονη εκτέλεση OpenMp εργασιών στον ίδιο κόμβο

Πλήθος	**Μέσος Χρόνος Εκτέλεσης (s)**	**Μεταβολή (%)**
1	5.51	0
2	9.93	80.2
3	14.43	161.9
4	22.25	303.8



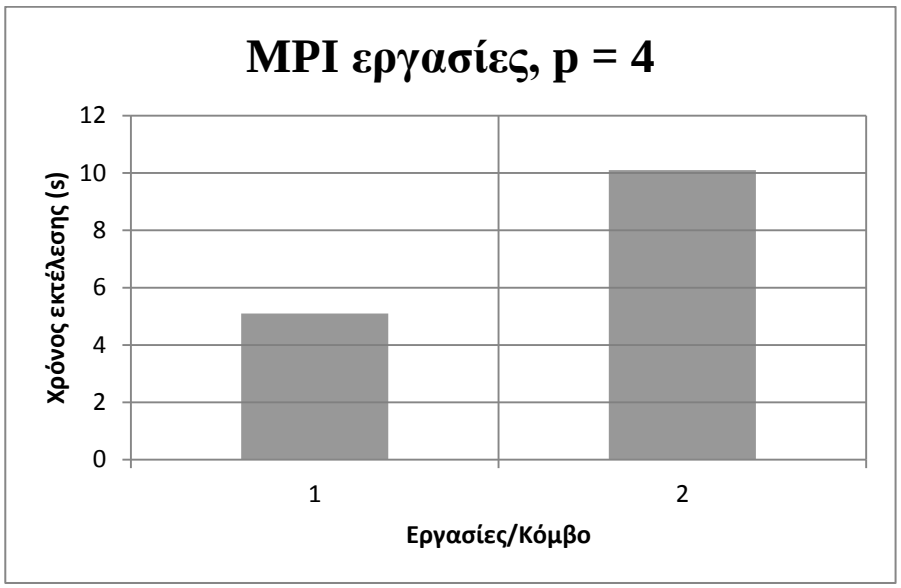
Σχήμα 6.3 Ταυτόχρονη εκτέλεση OpenMp εργασιών στον ίδιο κόμβο

Και σε αυτήν την περίπτωση παρατηρούμε σημαντική μεταβολή στον χρόνο εκτέλεσης όταν αυξάνουμε τον αριθμό των εργασιών που εκτελούνται ταυτόχρονα στον ίδιο κόμβο.

Τέλος, εκτελέσαμε τον αλγόριθμο και με το πρωτόκολλο ανταλλαγής μηνυμάτων. Έτσι χρησιμοποιήσαμε το MPI για να εκτελέσουμε τον αλγόριθμο με μέγεθος πίνακα 2048 και 4 επεξεργαστές. Τα αποτελέσματα φαίνονται παρακάτω:

Πίνακας 6.3 Ταυτόχρονη εκτέλεση MPI εργασιών στον ίδιο κόμβο

Πλήθος	Μέσος Χρόνος Εκτέλεσης (s)	Μεταβολή (%)
1	5,1	0
2	10,1	98



Σχήμα 6.4 Ταυτόχρονη εκτέλεση MPI εργασιών στον ίδιο κόμβο

Και εδώ παρατηρούμε ότι η ταυτόχρονη εκτέλεση MPI memory intensive εργασιών στον ίδιο κόμβο οδηγεί σε «εκτόξευση» του χρόνου εκτέλεσης. Από τα παραπάνω συμπεραίνουμε, ότι αν το σύστημά μας διαθέτει περισσότερους από 1 κόμβους, είναι αποδοτικότερο από την σκοπιά του χρόνου εκτέλεσης να μην τοποθετήσουμε τις εργασίες σε έναν κόμβο, όπως θα υπέβαλλε η αρχή της βέλτιστης αξιοποίησης συστήματος, αλλά να τις διασκορπίσουμε στους κόμβους, έτσι ώστε σε κανέναν κόμβο οι εργασίες που εκτελούνται να μην ξεπερνούν το memory bandwidth του κόμβου. Ειδικότερα αν είχαμε ένα σύστημα 4 κόμβων με παρόμοια χαρακτηριστικά με τον κόμβο στον οποίο έγιναν οι μετρήσεις θα είχαμε:

- Στην $1^{η}$ περίπτωση, θα μπορούσαμε να τρέχουμε μέχρι 2 εργασίες στον ίδιο κόμβο. Έτσι, ο συνολικός χρόνος εκτέλεσης θα ήταν περίπου 0.86s, δηλαδή κατά 78.5% μικρότερος από την περίπτωση που διαλέγαμε να εκτελέσουμε όλες τις εργασίες στον ίδιο κόμβο.
- Στην $2^{η}$ περίπτωση, θα μπορούσαμε να εκτελέσουμε μία εργασία σε κάθε κόμβο με αποτέλεσμα μείωση του συνολικού χρόνου εκτέλεσης κατά 75.2%.
- Στην $3^{η}$ περίπτωση, αν εκτελούσαμε τις εργασίες ξεχωριστά σε 2 κόμβους θα είχαμε μείωση στον συνολικό χρόνο εκτέλεσης κατά 49.5%.

Αξιοσημείωτο είναι ότι στην $2^{η}$ και $3^{η}$ περίπτωση, όπου οι εφαρμογές είναι αρκετά πιο απαιτητικές από ότι στην $1^{η}$, αν διαλέγαμε να τρέξουμε τις εφαρμογές σειριακά την μία μετά την άλλη αντί για ταυτόχρονα, θα είχαμε περίπου το ίδιο αποτέλεσμα στον συνολικό χρόνο ολοκλήρωσης.

6.2.2 Παραχώρηση CPU από διαφορετικούς κόμβους σε MPI εργασίες

Για τις σειριακές εργασίες και τις εργασίες OpenMP δεν μπορούμε να κάνουμε κάτι για να βελτιώσουμε τον χρόνο εκτέλεσης, εκτός από το να φροντίζουμε να μην εκτελούνται σε κόμβους όπου το συνολικό memory bandwidth των εργασιών ξεπερνάει αυτό του κόμβου. Όμως, για τις εργασίες που χρησιμοποιούν το πρωτόκολλο ανταλλαγής μηνυμάτων μπορούμε να πάρουμε επιπλέον μέτρα. Πιο συγκεκριμένα, αν γνωρίζουμε ότι μια εφαρμογή είναι τόσο memory intensive που το memory bandwidth του κόμβου δεν την ικανοποιεί, θα μπορούσαμε να την εκτελέσουμε σε περισσότερους από ένα κόμβους. Έτσι, σε αυτήν την σειρά πειραμάτων ελέγχουμε αν η κατανομή επεξεργαστών από διαφορετικούς κόμβους συμβάλει στην μείωση του χρόνου εκτέλεσης εργασιών, που χρησιμοποιούν το πρωτόκολλο ανταλλαγής μηνυμάτων.

Για αυτό το σύνολο πειραμάτων χρησιμοποιήθηκε και πάλι ο αλγόριθμος LU decomposition. Παρακάτω ακολουθούν πίνακες και διαγράμματα που παρουσιάζουν τα αποτελέσματα των πειραμάτων που διεξήχθησαν. Ο όρος μεταβολή αναφέρεται στην μεταβολή του συνολικού χρόνου, αφού αυτός είναι που μας ενδιαφέρει.

Πίνακας 6.4 Μέγεθος Πίνακα 6144, 8 επεξεργαστές

Επεξεργαστές/Κόμβο	Χρόνος υπολογισμού (s)	Χρόνος επικοινωνίας (s)	Συνολικός χρόνος (s)	Μεταβολή (%)
8	158,4	1,8	160,2	0
4	147,4	2,6	150	-6,37
3	113,3	3,4	116,7	-27,15
2	99,86	3,69	103,55	-35,36

Πίνακας 6.5 Μέγεθος Πίνακα 6144, 4 επεξεργαστές

Επεξεργαστές/Κόμβο	Χρόνος υπολογισμού (s)	Χρόνος επικοινωνίας (s)	Συνολικός χρόνος (s)	Μεταβολή (%)
4	211,1	1	212,1	0
2	152,45	2,76	155,21	-26,82

Πίνακας 6.6 Μέγεθος Πίνακα 4096, 8 επεξεργαστές

Επεξεργαστές/Κόμβο	Χρόνος υπολογισμού (s)	Χρόνος επικοινωνίας (s)	Συνολικός χρόνος (s)	Μεταβολή (%)
8	44,63	0,84	45,47	0
4	39,95	1,8	41,75	-8,18
3	30,47	2,19	32,66	-28,17
2	25,12	2,57	27,69	-39,1

Πίνακας 6.7 Μέγεθος Πίνακα 4096, 4 επεξεργαστές

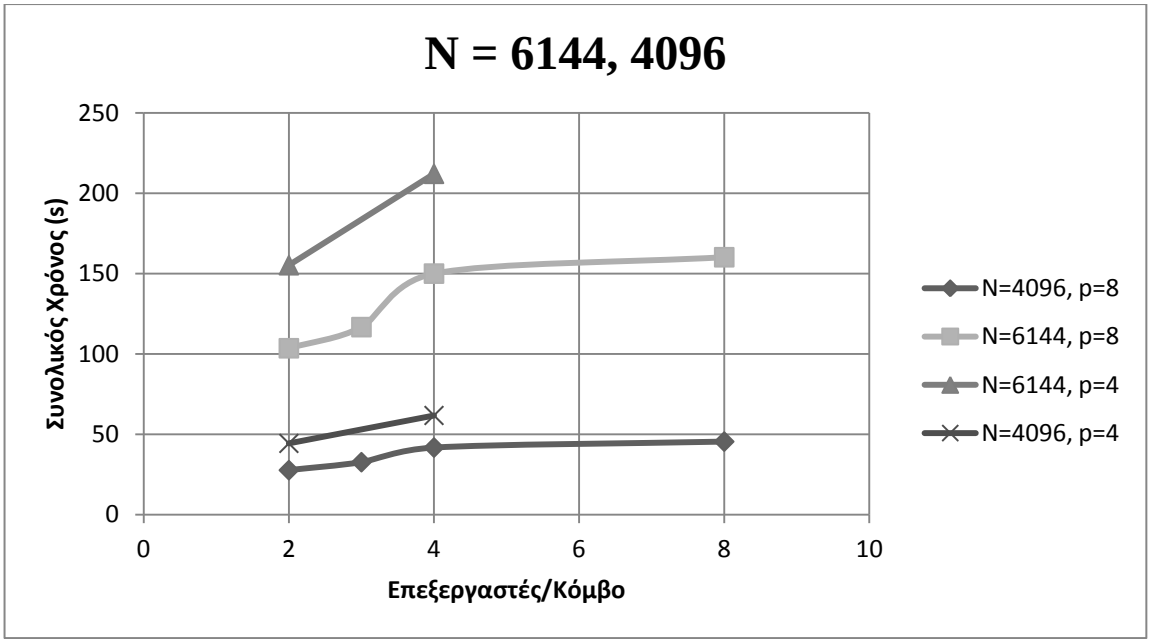
Επεξεργαστές/Κόμβο	Χρόνος υπολογισμού (s)	Χρόνος επικοινωνίας (s)	Συνολικός χρόνος (s)	Μεταβολή (%)
4	61,28	0,46	61,74	0
2	42,93	1,43	44,36	-28,15

Πίνακας 6.8 Μέγεθος Πίνακα 2048, 8 επεξεργαστές

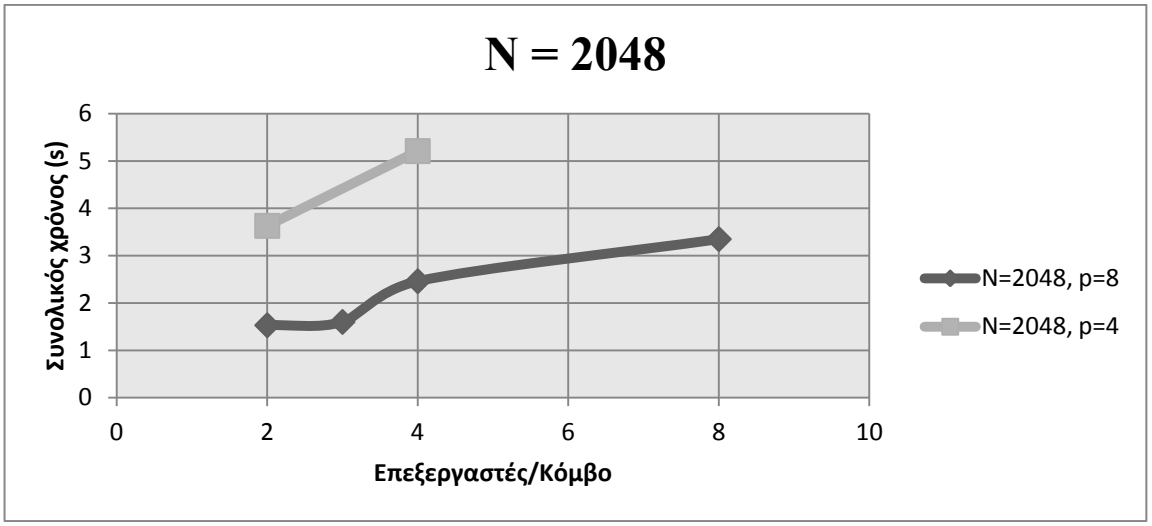
Επεξεργαστές/Κόμβο	Χρόνος υπολογισμού (s)	Χρόνος επικοινωνίας (s)	Συνολικός χρόνος (s)	Μεταβολή (%)
8	3,15	0,2	3,35	0
4	1,87	0,59	2,46	-26,57
3	0,8	0,8	1,6	-52,24
2	0,73	0,8	1,53	-54,33

Πίνακας 6.9 Μέγεθος Πίνακα 2048, 4 επεξεργαστές

Επεξεργαστές/Κόμβο	Χρόνος υπολογισμού (s)	Χρόνος επικοινωνίας (s)	Συνολικός χρόνος (s)	Μεταβολή (%)
4	5,1	0,11	5,21	0
2	3,13	0,5	3,63	-30,33



Σχήμα 6.5 Συνολικός χρόνος LU Decomposition N = 6144, 4096



Σχήμα 6.6 Συνολικός χρόνος LU Decomposition N = 2048

Από τα παραπάνω πειράματα βλέπουμε ότι ενώ ο χρόνος επικοινωνίας αυξάνεται (σε κάποιες περιπτώσεις έως και διπλασιάζεται), η τιμή του όμως είναι σχετικά αμελητέα σε σχέση με την τιμή του χρόνου υπολογισμού. Αντίθετα, ο χρόνος υπολογισμού μειώνεται σημαντικά, πετυχαίνοντας σε ορισμένες περιπτώσεις παραπάνω από 50% μείωση. Ο χρόνος υπολογισμού είναι λογικό να μειώνεται, αφού όταν παραχωρούμε σε μια παράλληλη εργασία επεξεργαστές από διαφορετικούς κόμβους, μειώνεται ο ανταγωνισμός για τον διάδρομο μνήμης στο εσωτερικό του κάθε κόμβου. Σε αυτό το σημείο να σημειώσουμε ότι οι πίνακες που χρησιμοποιήθηκαν ήταν αρκετά μεγάλοι, έτσι ώστε να μην χωράνε στις caches σε οποιαδήποτε από τις παραπάνω περιπτώσεις. Ενδεικτικά, για μέγεθος πίνακα 2048 έχουμε συνολικό μέγεθος δεδομένων 2KB × 2KB × 8 = 32 MB. Οι κόμβοι που έγιναν οι μετρήσεις είχαν 2 caches μεγέθους 4 MB. Όμως, ακόμα και στην περίπτωση που χρησιμοποιούνται 2 επεξεργαστές/κόμβο, ο χρονοδρομολογητής τις τοποθετεί σε μια cache, άρα έχουμε σύνολο 16 MB. Αυτό δικαιολογείται και από τις μετρήσεις, αφού αν καθώς γινόταν η παραχώρηση

επεξεργαστών στην εργασία, το συνολικό άθροισμα της χωρητικότητας των caches ήταν τουλάχιστον 32 MB, θα παρατηρούσαμε μεγάλη αλλαγή στον χρόνο υπολογισμού, καθώς πηγαίναμε από τους 3 στους 2 επεξεργαστές/κόμβο. Τέλος, καθώς ο συνολικός χρόνος εξαρτάται κυρίως από τον χρόνο υπολογισμού, έχουμε σημαντική μείωση που σε κάποιες περιπτώσεις ξεπερνά το 50%. Άρα, συμπεραίνουμε ότι αξίζει να παραχωρούμε σε memory intensive εργασίες που χρησιμοποιούν το πρωτόκολλο ανταλλαγής μηνυμάτων επεξεργαστές από διαφορετικούς κόμβους.

6.3 Περιγραφή memory aware πολιτικής

Σε αυτό το υποκεφάλαιο περιγράφουμε την memory aware πολιτική που κατασκευάσαμε. Η πολιτική κατασκευάστηκε με βάση τις παρατηρήσεις που κάναμε προηγουμένως για τις memory intensive εργασίες. Αναλυτικά η κατασκευή της πολιτικής περιγράφεται στο *Παράρτημα ΙΙΙ*.

Για την εφαρμογή της πολιτικής χρησιμοποιήσαμε την δομή QoS που προσφέρει ο Maui. Έτσι αρχικά προσθέσαμε ένα καινούργιο πεδίο στην δομή QoS, που το ονομάσαμε MEMINTENSITY. Αυτό το πεδίο δείχνει πόσο memory intensive είναι μία εφαρμογή. Όπως είπαμε και προηγουμένως, το MEMINTENSITY παίρνει μόνο θετικές τιμές, αλλά δεν έχει μέγιστη τιμή. Ουσιαστικά μας δείχνει το ποσοστό του memory bandwidth που απαιτεί η εργασία, με το 10 να συμβολίζει το 100%. Η default τιμή αυτής της παραμέτρου θα είναι 0, έτσι ώστε αν δεν δηλωθεί να έχουμε την συνηθισμένη συμπεριφορά του Maui. Η πολιτική που κατασκευάσαμε βασίζεται στο ότι σε ένα κόμβο, οποιαδήποτε στιγμή, δεν μπορούν να εκτελούνται ταυτόχρονα εργασίες που το άθροισμα των MEMINTENSITY τιμών τους ξεπερνάει το 10, ή με άλλα λόγια απαιτούν συνολικά μεγαλύτερο memory bandwidth από αυτό που προσφέρει το σύστημα. Η εισαγωγή αυτής της παραμέτρου ικανοποιεί την $1^{η}$ παρατήρηση που κάναμε στο προηγούμενο υποκεφάλαιο.

Στην συνέχεια προσθέσαμε μια καινούργια σημαία στην δομή QoS με το όνομα MEMINTENSIVE. Αυτή η σημαία, όταν είναι ενεργοποιημένη για μια εργασία, ελέγχει το MEMINTENSITY και αν είναι μεγαλύτερο από 10, προσθέτει κόμβους στην κράτηση της εργασίας και ισομοιράζει την κατανομή των επεξεργαστών μέχρι το MEMINTENSITY να μην ξεπερνάει το 10. Αν το MEMINTENSITY είναι εξαρχής μικρότερο από 10, προφανώς καμία ενέργεια δεν λαμβάνει χώρα. Σε αυτό το σημείο πρέπει να τονίσουμε ότι η παράμετρος MEMINTENSITY και η σημαία MEMINTENSIVE είναι ανεξάρτητες. Με λίγα λόγια, μια εργασία μπορεί να έχει τιμή στο πεδίο MEMINTENSITY χωρίς να έχει δηλωθεί MEMINTENSIVE και το ανάποδο. Με αυτή την σημαία ικανοποιούμε την $2^{η}$ παρατήρηση που κάναμε για τις memory intensive εργασίες.

Στο επόμενο κεφάλαιο εφαρμόζουμε την πολιτική που μόλις περιγράψαμε στα εργασιακά φορτία του προηγούμενου κεφαλαίου, για να δούμε πώς αυτή η πολιτική επηρεάζει τους 3 αλγορίθμους που μελετήσαμε. Πιο συγκεκριμένα, δίνουμε τιμή στο πεδίο MEMINTENSITY όλων των εργασιών και ενεργοποιούμε την σημαία MEMINTENSIVE για όλες τις εργασίες που έχουν MEMINTENSITY μεγαλύτερο από 10. Αυτό γίνεται τροποποιώντας το maui.cfg ως εξής:

QOSCFG[memint12] QFLAG=MEMINTENSIVE MEMINTENSITY=12
QOSCFG[mem5] MEMINTENSITY=5

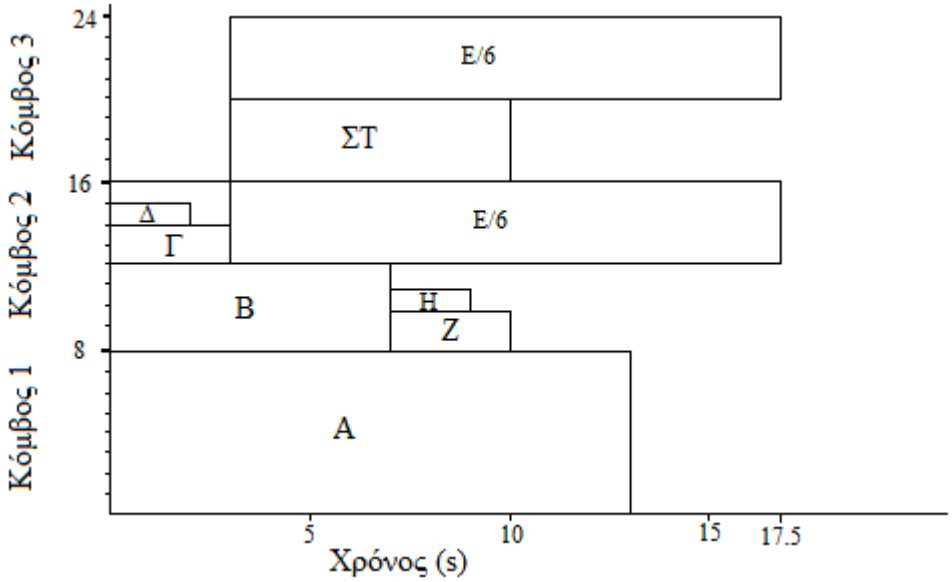
Με την 1η γραμμή δηλώνουμε ότι οι εργασίες με το συγκεκριμένο QoS έχουν MEMINTENSITY ίσο με 12 και ότι θέλουμε να γίνει προσπάθεια να τους παραχωρηθούν παραπάνω από ένας κόμβοι (2 στην συγκεκριμένη περίπτωση), αφού η σημαία MEMINTENSIVE είναι ενεργοποιημένη. Με την 2η γραμμή απλά δηλώνουμε ότι οι εργασίες με το συγκεκριμένο QoS έχουν MEMINTENSITY ίσο με 5.

6.4 CPU intensive εργασιακό φορτίο

Παρακάτω παρουσιάζονται σε σχήματα οι χαρτογραφήσεις των εργασιών του 1ου εργασιακού φορτίου που χρησιμοποιήσαμε και στο προηγούμενο κεφάλαιο για τους 3 αλγορίθμους χρονοδρομολόγησης, καθώς και πίνακες με χρήσιμες μετρικές.

6.4.1 memFCFS

<u>Χωρίς backfill</u>

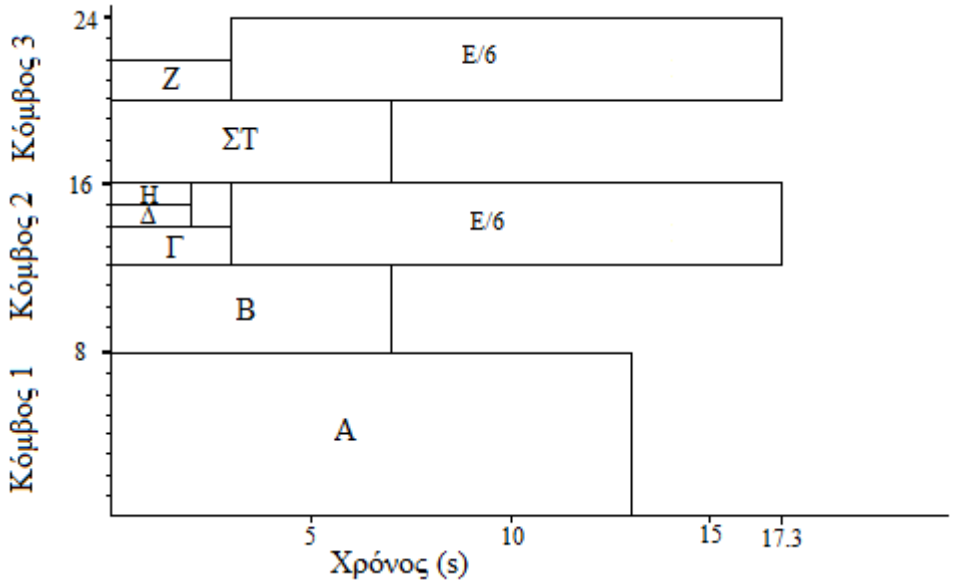


Σχήμα 6.7 Memory aware FCFS χωρίς backfill-cpu intensive workload

Πίνακας 6.10 Memory aware FCFS χωρίς backfill-cpu intensive workload

Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
Α	12.82	0	12.82	**17.45**
Β	6.5	0	6.5	
Γ	3.81	0	3.81	
Δ	1.85	0	1.85	
Ε	13.64	3.81	17.45	
ΣΤ	6.61	3.81	10.42	
Ζ	3.65	6.5	10.15	
Η	1.85	6.5	8.35	
Μέσος Όρος	**6.34**	**2.58**	**8.92**	

Με backfill



Σχήμα 6.8 Memory aware FCFS με backfill-cpu intensive workload

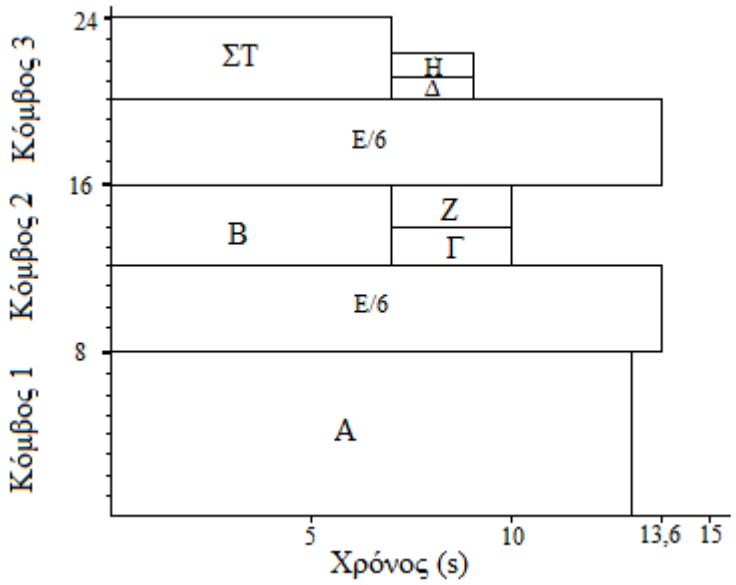
Πίνακας 6.11 Memory aware FCFS με backfill-cpu intensive workload

Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
Α	12.88	0	12.88	**17.33**
Β	6.55	0	6.55	
Γ	3.89	0	3.89	
Δ	1.85	0	1.85	
Ε	13.44	3.89	17.33	
ΣΤ	6.71	0	6.71	
Ζ	3.85	0	3.85	
Η	1.86	0	1.86	
Μέσος Όρος	**6.39**	**0.49**	**6.87**	

Παρατηρήσεις

- Για μία ακόμα φορά βλέπουμε ότι η τεχνική backfill μειώνει τον μέσο χρόνο αναμονής και κατά συνέπεια και τον μέσο χρόνο ολοκλήρωσης.
- Σε σχέση με την απλή FCFS+backfill, η memory aware FCFS+backfill μειώνει τον μέσο χρόνο εκτέλεσης κατά 11.3%. Αυτό γίνεται επειδή μειώνεται ο χρόνος εκτέλεσης της εργασίας Ε από 19.8 s σε 13.44 s, δηλαδή 32.1% μείωση.
- Έχουμε επίσης βελτίωση στον μέσο χρόνο αναμονής, αφού η διάσπαση της εργασίας Ε προκαλεί την αναγκαστική καθυστέρησή της και έτσι δίνεται η δυνατότητα στις μικρότερες εργασίες Ζ και ΣΤ να εκτελεστούν πρώτες.
- Λόγω των παραπάνω βελτιώνεται και ο μέσος χρόνος ολοκλήρωσης, αφού έχουμε μείωση κατά 19.6%.
- Τέλος, μειώνεται και ο συνολικός χρόνος ολοκλήρωσης των εργασιών κατά 12.5%, αφού ο χρόνος εκτέλεσης της εργασίας Ε που τον καθόριζε και προηγουμένως μειώνεται σημαντικά, παρόλο που η εργασία Ε καθυστερεί να ξεκινήσει.

6.4.2 memLJF



Σχήμα 6.9 Memory aware LJF-cpu intensive workload

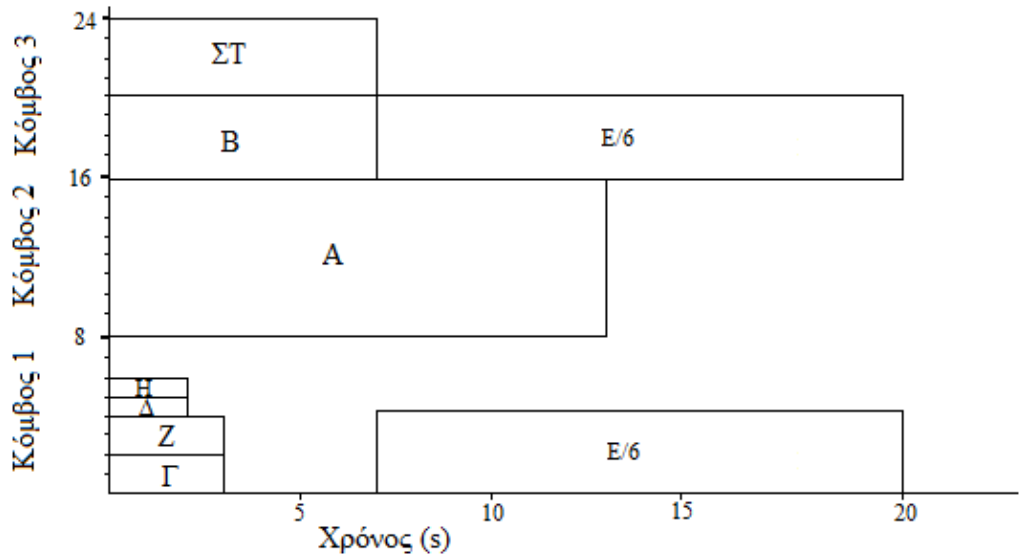
Πίνακας 6.12 Memory aware LJF-cpu intensive workload

Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
Α	12.82	0	12.82	**13.6**
Β	6.57	0	6.57	
Γ	3.71	6.57	10.28	
Δ	1.87	6.51	8.44	
Ε	13.6	0	13.6	
ΣΤ	6.51	0	6.51	
Ζ	3.75	6.57	10.26	
Η	1.85	6.51	8.36	
Μέσος Όρος	**6.34**	**3.27**	**9.61**	

Παρατηρήσεις

- Και σε αυτήν την περίπτωση βλέπουμε ότι ο μέσος χρόνος εκτέλεσης μειώνεται αρκετά, κατά 11,7% για τον ίδιο λόγο με προηγουμένως.
- Ο μέσος χρόνος αναμονής παραμένει σχεδόν σταθερός (αύξηση 0.3%).
- Κατά συνέπεια ο μέσος χρόνος ολοκλήρωσης μειώνεται, συγκεκριμένα κατά 7.9%
- Τέλος, όπως και στον αλγόριθμο FCFS, και εδώ βελτιώνεται σημαντικά ο συνολικός χρόνος ολοκλήρωσης των εργασιών (κατά 31.1%), αφού μειώνεται ο χρόνος εκτέλεσης της εργασίας Ε που τον καθορίζει.

6.4.3 memSJF



Σχήμα 6.10 Memory aware SJF-cpu intensive workload

Πίνακας 6.13 Memory aware SJF-cpu intensive workload

Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
Α	12.75	0	12.75	**19.95**
Β	6.63	0	6.63	
Γ	3.63	0	3.63	
Δ	1.86	0	1.86	
Ε	13.32	6.63	19.95	
ΣΤ	6.51	0	6.51	
Ζ	3.56	0	3.56	
Η	1.85	0	1.85	
Μέσος Όρος	**6.26**	**0.83**	**7.09**	

Παρατηρήσεις

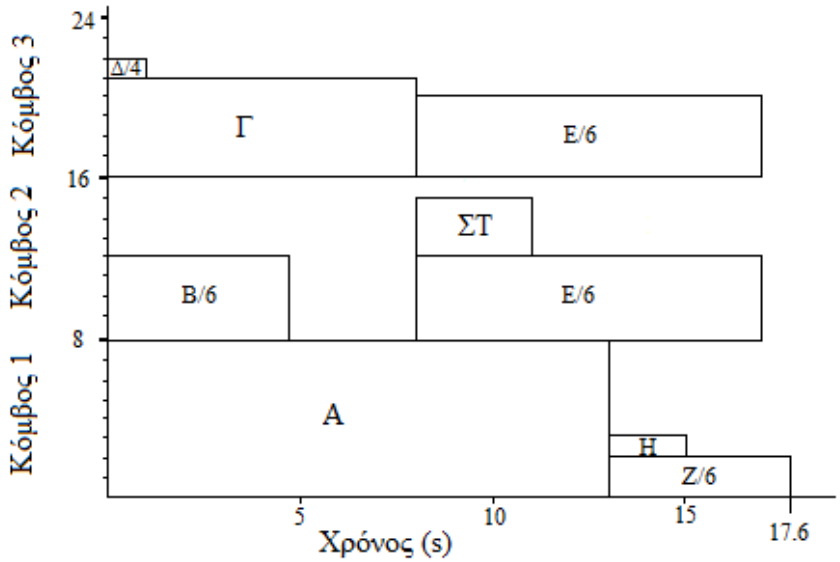
- Ο μέσος χρόνος εκτέλεσης μειώνεται κατά 11.3% για τους ίδιους λόγους με τις προηγούμενες περιπτώσεις.
- Ο μέσος χρόνος αναμονής διπλασιάζεται, αλλά και πάλι παραμένει μικρός σε απόλυτη τιμή. Αυτό συμβαίνει γιατί η εργασία Ε χρειάζεται πλέον 2 κόμβους για να εκτελεστεί και πρέπει να περιμένει περισσότερο για να γίνουν διαθέσιμοι.
- Επειδή η μείωση του μέσου χρόνου εκτέλεσης είναι μεγαλύτερη κατά απόλυτη τιμή από την αύξηση στον μέσο χρόνο αναμονής, μειώνεται οριακά και ο μέσος χρόνος ολοκλήρωσης, κατά 5.7%.
- Τέλος, και εδώ έχουμε σημαντική μείωση στον συνολικό χρόνο ολοκλήρωσης των εργασιών κατά 14,8%.

6.5 *Ισορροπημένο εργασιακό φορτίο*

Παρακάτω παρουσιάζονται σε σχήματα οι χαρτογραφήσεις των εργασιών του 2ου εργασιακού φορτίου που χρησιμοποιήσαμε και στο προηγούμενο κεφάλαιο, για τους 3 αλγορίθμους χρονοδρομολόγησης, καθώς και πίνακες με χρήσιμες μετρικές.

6.5.1 *memFCFS*

Χωρίς backfill

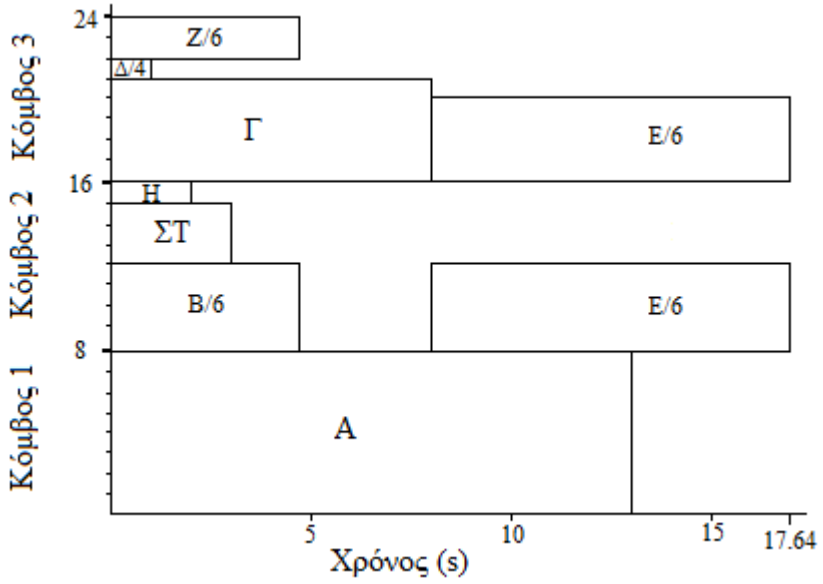


Σχήμα 6.11 Memory aware FCFS χωρίς backfill-balanced workload

Πίνακας 6.14 Memory aware FCFS χωρίς backfill-balanced workload

Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
Α	12.89	0	12.89	**17.59**
Β	4.45	0	4.45	
Γ	8.63	0	8.63	
Δ	1.02	0	1.02	
Ε	8.95	8.63	17.58	
ΣΤ	3.33	8.63	11.96	
Ζ	4.7	12.89	17.59	
Η	1.85	12.89	14.74	
Μέσος Όρος	**5.73**	**5.38**	**11.11**	

Με backfill



Σχήμα 6.12 Memory aware FCFS με backfill-balanced workload

Πίνακας 6.15 Memory aware FCFS με backfill-balanced workload

Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
A	12.97	0	12.97	**17.64**
B	4.55	0	4.55	
Γ	8.59	0	8.59	
Δ	1.03	0	1.03	
E	9.05	8.59	17.64	
ΣΤ	3.23	0	3.23	
Z	4.6	0	4.6	
H	1.86	0	1.86	
Μέσος Όρος	**5.74**	**1.07**	**6.81**	

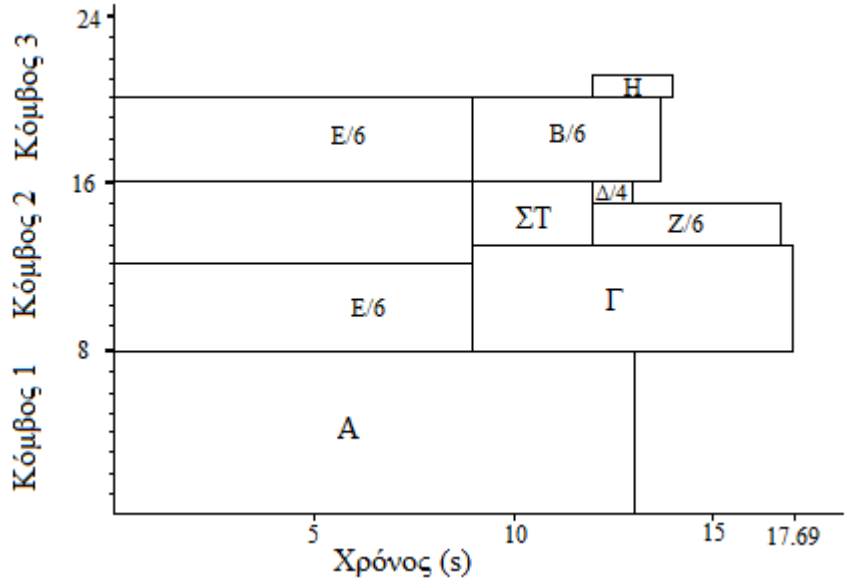
Παρατηρήσεις

- Σε αυτό το παράδειγμα η τεχνική backfill έχει τεράστιο αντίκτυπο, καθώς μειώνει τον μέσο χρόνο αναμονής κατά 80% και τον μέσο χρόνο ολοκλήρωσης κατά 38.7%.
- Η εισαγωγή της memory aware πολιτική στην FCFS+backfill μειώνει τον μέσο χρόνο εκτέλεσης των εργασιών κατά 12.6%.
- Αντίθετα, ο μέσος χρόνος αναμονής αυξάνεται κατά 91.1%. Η αύξηση όμως είναι μικρή κατά απόλυτη τιμή, μόλις 0.51 s, σε σύγκριση με την μείωση του μέσου χρόνου εκτέλεσης που είναι 0.83 s.
- Έτσι, μειώνεται και ο μέσος χρόνος ολοκλήρωσης των εργασιών, οριακά βέβαια κατά 0.32 s ή 4.5%. Ενδεικτικά, αν συγκρίνουμε τους αλγορίθμους χωρίς την εφαρμογή της τεχνικής backfill, παρατηρούμε ότι ο μέσος χρόνος ολοκλήρωσης αυξάνεται κατά 14.7%.

- Τέλος, όπως είδαμε και σε περιπτώσεις του προηγούμενου εργασιακού φορτίου, ο συνολικός χρόνος ολοκλήρωσης των εργασιών μειώνεται κατά 13.8%, παρόλο που η εργασία Ε που καθυστερεί περισσότερο να ξεκινήσει.

6.5.2 memLJF

<u>Χωρίς backfill</u>

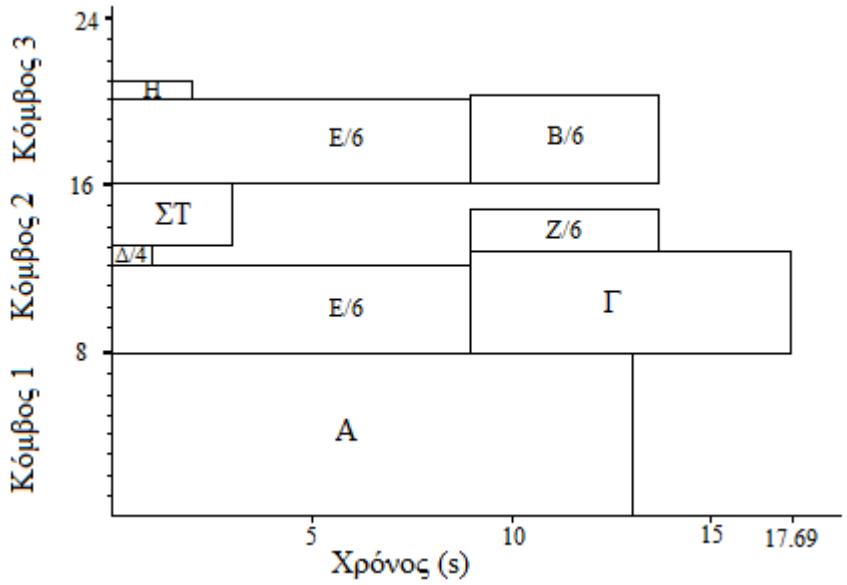


Σχήμα 6.13 Memory aware LJF χωρίς backfill-balanced workload

Πίνακας 6.16 Memory aware LJF χωρίς backfill-balanced workload

Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
Α	12.87	0	12.87	**17.69**
Β	4.8	9.15	13.95	
Γ	8.54	9.15	17.69	
Δ	1.04	12.87	13.91	
Ε	9.15	0	9.15	
ΣΤ	3.22	9.15	12.37	
Ζ	4.67	12.87	17.54	
Η	1.86	12.87	14.73	
Μέσος Όρος	**5.77**	**8.26**	**14.03**	

Με backfill



Σχήμα 6.14 Memory aware LJF με backfill-balanced workload

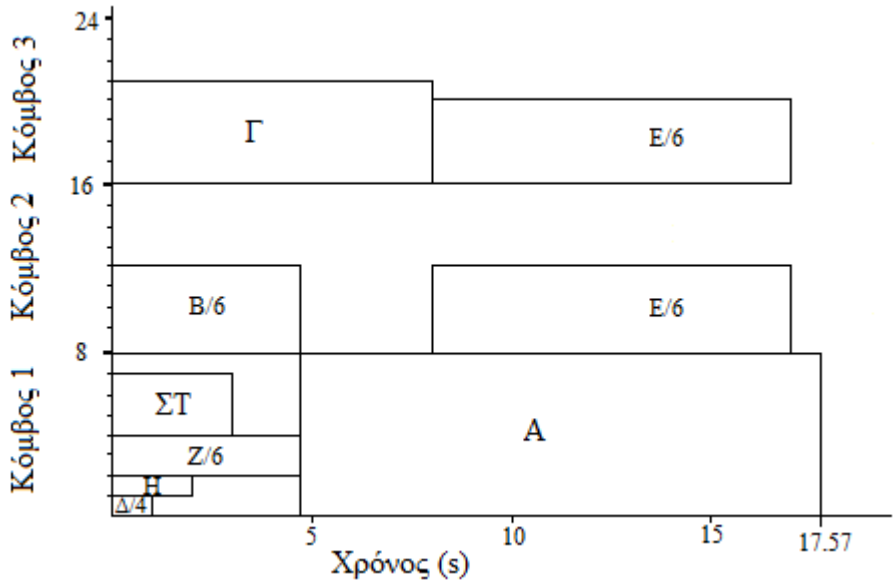
Πίνακας 6.17 Memory aware LJF με backfill-balanced workload

Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
A	12.92	0	12.92	**17.69**
B	4.65	9.04	13.69	
Γ	8.48	9.04	17.52	
Δ	0.97	0	0.97	
E	9.04	0	9.04	
ΣΤ	3.32	0	3.32	
Z	4.77	9.04	17.69	
H	1.85	0	1.85	
Μέσος Όρος	**5.75**	**3.39**	**9.14**	

Παρατηρήσεις

- Η τεχνική backfill μειώνει, για ακόμα μία φορά, σημαντικά τον χρόνο αναμονής (59%) και τον μέσο χρόνο ολοκλήρωσης (34.8%).
- Ο μέσος χρόνος εκτέλεσης μειώνεται, όπως ήταν λογικό, λόγω της χρησιμοποίησης της memory aware πολιτικής, κατά 12.6%.
- Ο μέσος χρόνος αναμονής όμως, όπως είδαμε και σε άλλα παραδείγματα αυξάνεται οριακά, κατά 1.7%.
- Ο μέσος χρόνος ολοκλήρωσης βέβαια μειώνεται κατά 8.9%, καθώς η μείωση στον μέσο χρόνο εκτέλεσης έχει μεγαλύτερο αντίκτυπο. Και εδώ αν δεν χρησιμοποιηθεί backfill, ο μέσος χρόνος ολοκλήρωσης αυξάνεται κατά 5.3%.
- Για πρώτη φορά όμως βλέπουμε αύξηση στον συνολικό χρόνο ολοκλήρωσης των εργασιών κατά 12.1%. Αυτό γίνεται επειδή η memory aware πολιτική μπλοκάρει την εκκίνηση της απαιτητικής σε αριθμό επεξεργαστών εργασίας Γ, αφού πλέον καταλαμβάνει 2 κόμβους για την εκτέλεσή της.

6.5.3 memSJF



Σχήμα 6.15 Memory aware SJF-balanced workload

Πίνακας 6.18 Memory aware SJF-balanced workload

Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
Α	12.8	4.77	17.57	**17.57**
Β	4.75	0	4.75	
Γ	8.45	0	8.45	
Δ	1	0	1	
Ε	8.87	8.45	17.32	
ΣΤ	3.27	0	3.27	
Ζ	4.77	0	4.77	
Η	1.88	0	1.88	
Μέσος Όρος	**5.72**	**1.65**	**7.37**	

Παρατηρήσεις

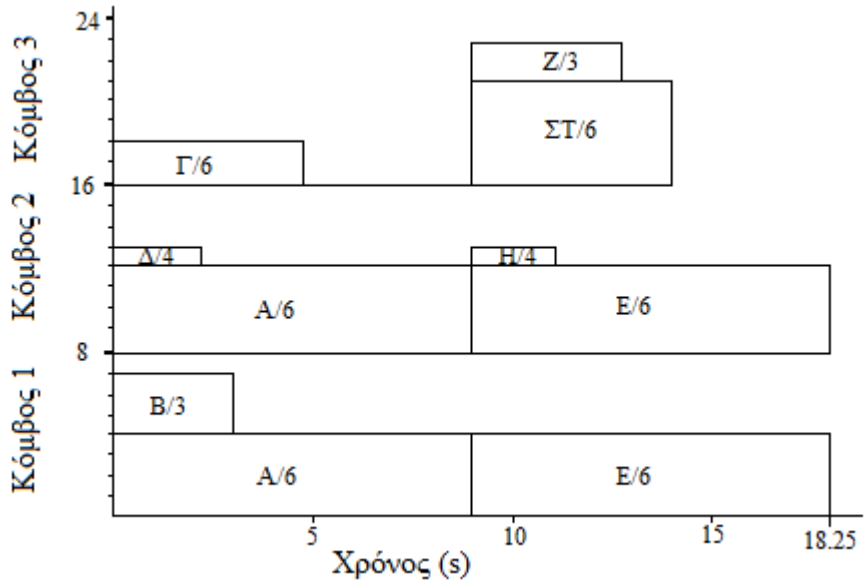
- Η εφαρμογή της πολιτικής μας προφανώς και προκαλεί μείωση του μέσου χρόνου εκτέλεσης, κατά 12.7%.
- Ο μέσος χρόνος αναμονής αυξάνεται κατά 41%, έχοντας βέβαια σχετικά μικρή απόλυτη τιμή.
- Κατά συνέπεια, ο μέσος χρόνος ολοκλήρωσης των εργασιών μειώνεται, κατά 4.5%.
- Τέλος, και ο συνολικός χρόνος εκτέλεσης των εργασιών μειώνεται κατά 13.2%, παρόλο που η εργασία Ε που τον καθορίζει καθυστερεί να ξεκινήσει. Αυτό συμβαίνει λόγω της σημαντικής μείωσης στον χρόνο εκτέλεσης της εργασίας Ε, που είναι 43%.

6.6 *Memory intensive εργασιακό φορτίο*

Παρακάτω παρουσιάζονται σε σχήματα οι χαρτογραφήσεις των εργασιών του 3ου εργασιακού φορτίου που χρησιμοποιήσαμε και στο προηγούμενο κεφάλαιο για τους 3 αλγορίθμους χρονοδρομολόγησης, καθώς και πίνακες με χρήσιμες μετρικές.

6.6.1 memFCFS

Χωρίς backfill

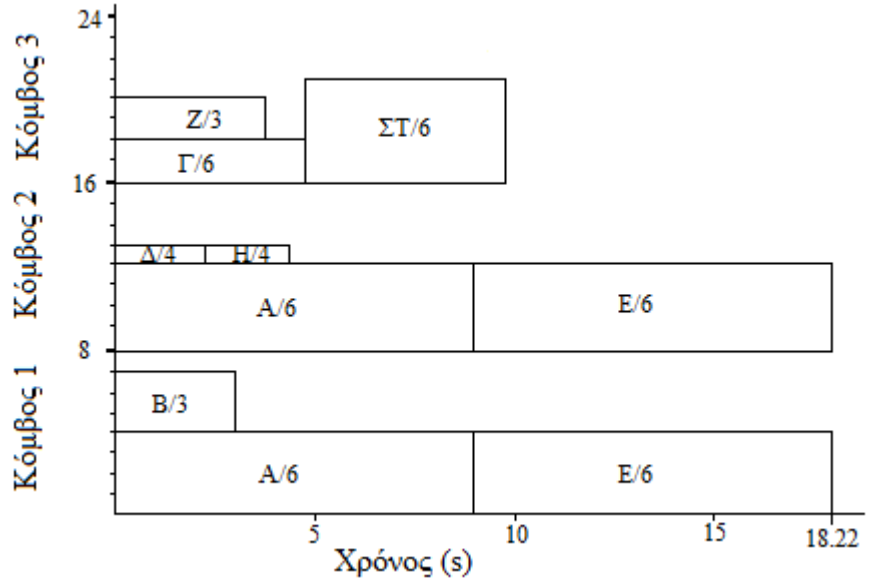


Σχήμα 6.16 Memory aware FCFS χωρίς backfill-memory intensive workload

Πίνακας 6.19 Memory aware FCFS χωρίς backfill-memory intensive workload

Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
Α	9.2	0	9.2	**18.25**
Β	2.3	0	2.3	
Γ	4.72	0	4.72	
Δ	1.01	0	1.01	
Ε	9.05	9.2	18.25	
ΣΤ	4.65	9.2	13.85	
Z	2.2	9.2	11.4	
Η	0.97	9.2	10.17	
Μέσος Όρος	**4.26**	**4.6**	**8.86**	

Με backfill



Σχήμα 6.17 Memory aware FCFS με backfill-memory intensive workload

Πίνακας 6.20 Memory aware FCFS με backfill-memory intensive workload

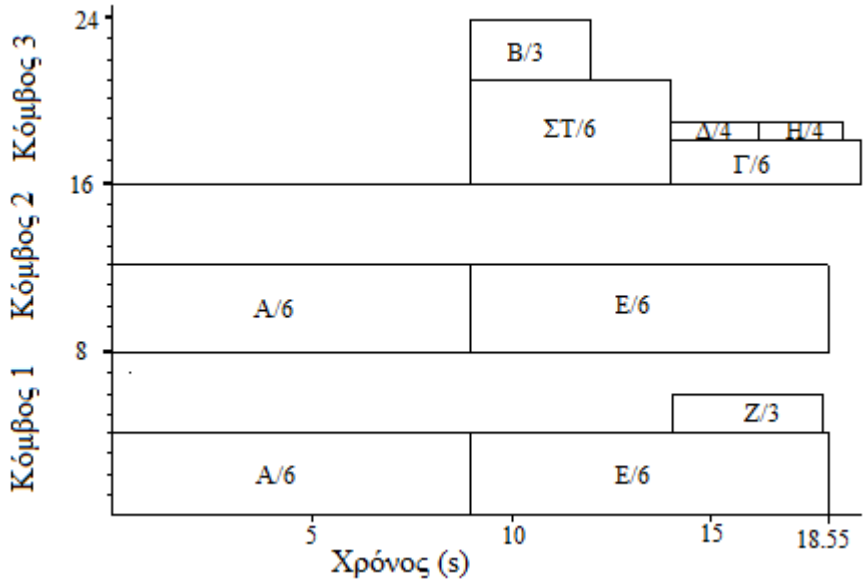
Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
A	9.08	0	9.08	**18.22**
B	2.3	0	2.3	
Γ	4.78	0	4.78	
Δ	1.01	0	1.01	
E	9.14	9.08	18.22	
ΣΤ	4.74	4.78	9.52	
Z	2.3	0	2.3	
H	1.03	1.01	2.04	
Μέσος Όρος	**4.3**	**1.86**	**6.16**	

Παρατηρήσεις

- Η τεχνική backfill κατά τα γνωστά προκαλεί μείωση του μέσου χρόνου αναμονής (60%) και του μέσου χρόνου ολοκλήρωσης των εργασιών (30.5%).
- Η memory aware πολιτική μας επιφέρει εντυπωσιακή βελτίωση στον χρόνο εκτέλεσης των εργασιών, καθώς μειώνεται κατά 32.4%.
- Από την άλλη πλευρά, αυξάνει ο μέσος χρόνο αναμονής των εργασιών κατά 80.6%. Η ίδια μετρική χωρίς τον αλγόριθμο backfill αυξάνεται κατά 193%.
- Παρόλο αυτά, ο μέσος χρόνος ολοκλήρωσης συνεχίζει να μειώνεται (16.6%), αν χρησιμοποιηθεί η τεχνική backfill. Αν δεν χρησιμοποιηθεί, ο μέσος χρόνος ολοκλήρωσης αυξάνεται κατά 9%.
- Τέλος, παρατηρείται σημαντική επιδείνωση του συνολικού χρόνου ολοκλήρωσης των εργασιών, που οφείλεται στην αύξηση του χρόνου αναμονής της εργασίας E.

6.6.2 memLJF

<u>Χωρίς backfill</u>

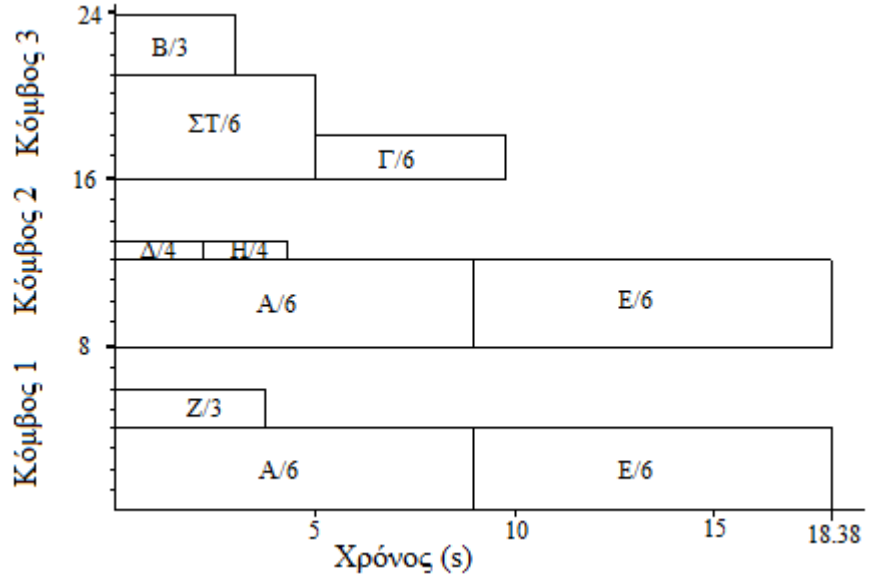


Σχήμα 6.18 Memory aware LJF χωρίς backfill-memory intensive workload

Πίνακας 6.21 Memory aware LJF χωρίς backfill-memory intensive workload

Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
Α	9.14	0	9.14	**18.55**
Β	2.35	9.14	11.49	
Γ	4.78	13.77	18.55	
Δ	1	13.77	14.77	
Ε	9.28	9.14	18.42	
ΣΤ	4.63	9.14	13.77	
Ζ	2.18	13.77	15.95	
Η	1.02	14.77	15.79	
Μέσος Όρος	**4.3**	**10.44**	**14.74**	

Με backfill



Σχήμα 6.19 Memory aware LJF με backfill-memory intensive workload

Πίνακας 6.22 Memory aware LJF με backfill-memory intensive workload

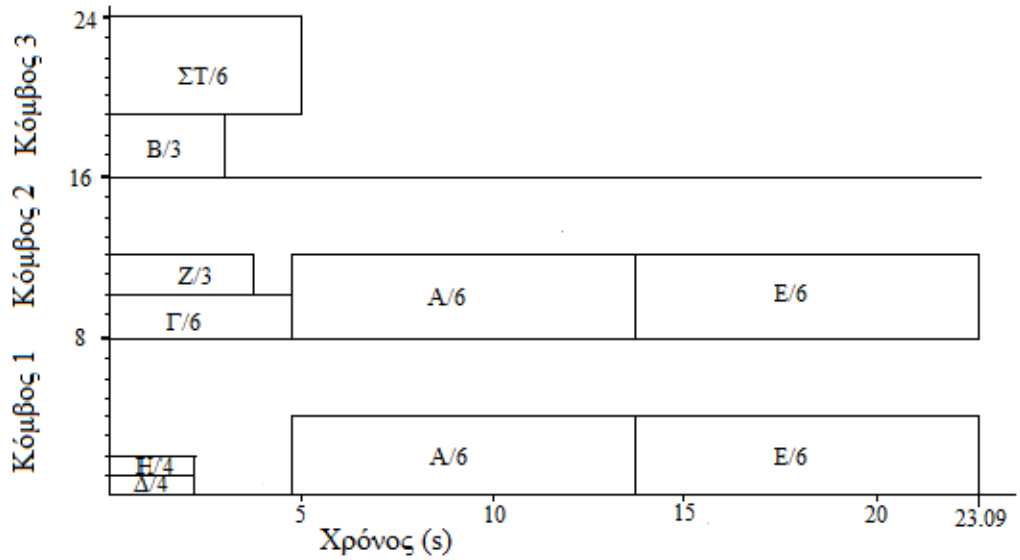
Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
A	9.2	0	9.2	**18.38**
B	2.3	0	2.3	
Γ	4.65	4.75	9.4	
Δ	1.02	0	1.02	
E	9.18	9.2	18.38	
ΣΤ	4.75	0	4.75	
Z	2.24	0	2.24	
H	1	1.02	2.02	
Μέσος Όρος	**4.29**	**1.87**	**6.16**	

Παρατηρήσεις

- Όπως είδαμε και στο προηγούμενο εργασιακό φορτίο, η εφαρμογή της τεχνικής backfill έχει τεράστια πλεονεκτήματα όταν χρησιμοποιείται η memory aware πολιτική. Αυτό συμβαίνει επειδή ο συνδυασμός LJF και memory aware πολιτικής προκαλεί μεγάλα διαστήματα ανενεργίας των κόμβων, που μπορεί να εκμεταλλευτεί ο αλγόριθμος backfill για να βελτιώσει την επίδοση. Εδώ συγκεκριμένα ο μέσος χρόνος αναμονής μειώνεται κατά 82% και ο μέσος χρόνος ολοκλήρωσης κατά 58.2%.
- Ο μέσος χρόνος εκτέλεσης, όπως και στον αλγόριθμο FCFS, βελτιώνεται κατά 36% με την εφαρμογή της memory aware πολιτικής.
- Ο μέσος χρόνος αναμονής αυξάνεται σε μικρό βαθμό όταν χρησιμοποιείται η τεχνική backfill (5.1%). Σε αντίθετη περίπτωση έχουμε τραγικής αύξηση 414%.
- Ο μέσος χρόνος ολοκλήρωσης των εργασιών βελτιώνεται σημαντικά, αφού παρατηρείται μείωση κατά 27.4%. Χωρίς την χρησιμοποίηση του αλγορίθμου backfill έχουμε αύξηση 67.1%.

- Παρόλο την ενθαρρυντική συμπεριφορά της πολιτικής στις παραπάνω μετρικές, ο συνολικός χρόνος ολοκλήρωσης των εργασιών αυξάνεται σε σημαντικό βαθμό (17.4%), λόγω της καθυστέρησης στην εκκίνηση της εργασίας Ε.

6.6.3 memSJF



Σχήμα 6.20 Memory aware SJF-memory intensive workload

Πίνακας 6.23 Memory aware SJF-memory intensive workload

Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
Α	9.18	4.68	13.86	**23.09**
Β	2.33	0	2.33	
Γ	4.68	0	4.68	
Δ	1.02	0	1.02	
Ε	9.23	13.86	23.09	
ΣΤ	4.7	0	4.7	
Ζ	2.28	0	2.28	
Η	1	0	1	
Μέσος Όρος	**4.3**	**2.32**	**6.62**	

Παρατηρήσεις

- Η χρησιμοποίηση της memory aware πολιτικής μειώνει και εδώ σημαντικά τον μέσο χρόνο εκτέλεσης (31.4%).
- Η επίπτωση στον μέσο χρόνο αναμονής όμως είναι τραγική, αφού έχουμε αύξηση 300%, λόγω της πολύ κακής χρησιμοποίησης συστήματος.
- Παρόλα αυτά όμως, η μείωση στο μέσο χρόνο εκτέλεσης υπερισχύει, με αποτέλεσμα να έχουμε οριακή μείωση στο μέσο χρόνο ολοκλήρωσης εργασιών κατά 3.4%.
- Ο συνολικός χρόνος εκτέλεσης των εργασιών αυξάνεται και σε αυτή τη σειρά μετρήσεων (14.5%) για τον λόγο που προαναφέρθηκε και στις προηγούμενες περιπτώσεις.

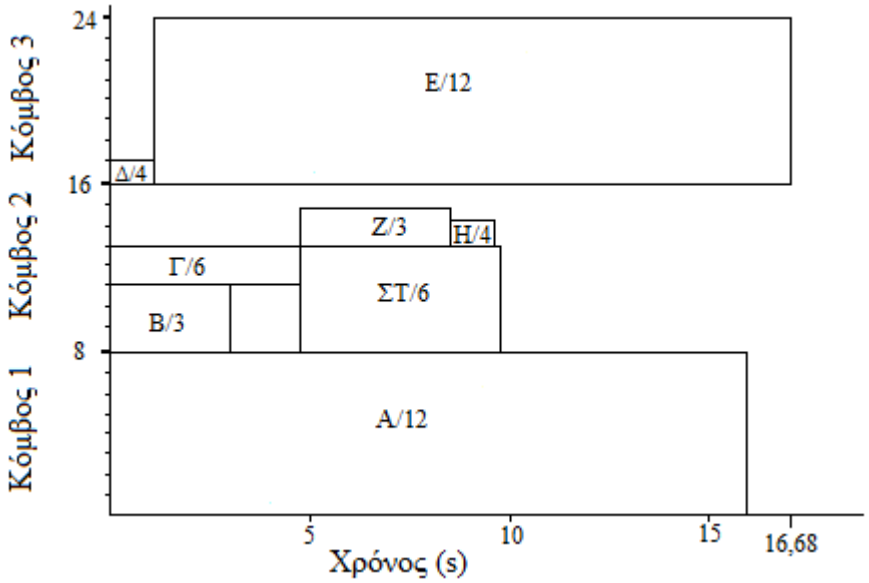
6.7 Memory intensive εργασιακό φορτίο χωρίς τη σημαία MEMINTENSIVE

Η εφαρμογής της memory aware πολιτικής, όπως είδαμε στο προηγούμενο υποκεφάλαιο, σε σύνολα που περιέχουν μεγάλο βαθμό memory intensive εργασιών βελτιώνει τον μέσο χρόνο εκτέλεσης και τον μέσο χρόνο ολοκλήρωσης, με σημαντική όμως επίπτωση στον μέσο χρόνο αναμονής και κυρίως στον συνολικό χρόνο ολοκλήρωσης. Για αυτό τον λόγο αποφασίσαμε να κάνουμε άλλη μια σειρά μετρήσεων στο memory intensive εργασιακό φορτίο, όπου θα αφήσουμε ενεργή την πολιτική ότι το άθροισμα των πεδίων MEMINTENSITY των εργασιών που εκτελούνται ταυτόχρονα σε έναν κόμβο να μην ξεπερνάει το 10 και θα απενεργοποιήσουμε την σημαία MEMINTENSIVE στις εργασίες που έχουν MEMINTENSITY μεγαλύτερο από 10.

Στην συνέχεια παρουσιάζονται οι χαρτογραφήσεις των εργασιών του 3ου εργασιακού φορτίου και οι πίνακες στατιστικών, χωρίς την εφαρμογή της σημαίας MEMINTENSIVE.

6.7.1 mem2FCFS

<u>Χωρίς backfill</u>

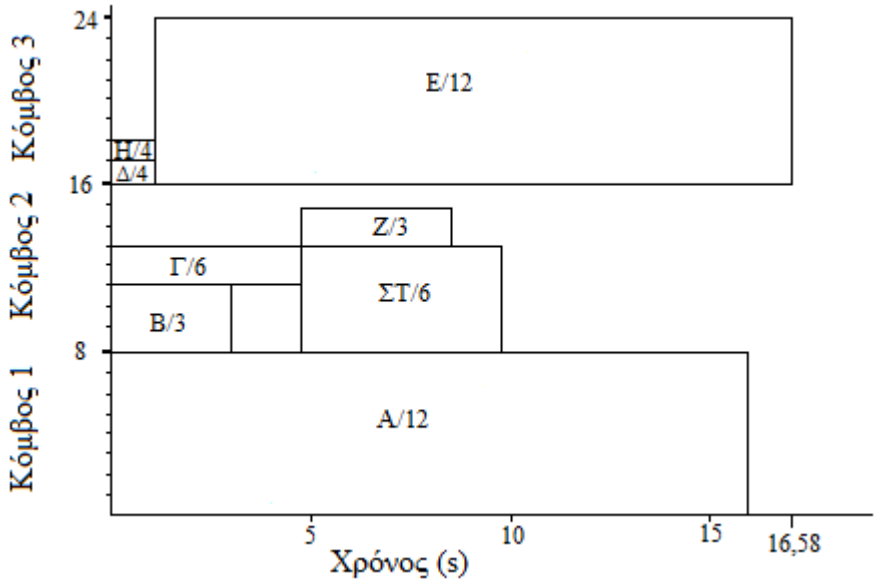


Σχήμα 6.21 mem2FCFS χωρίς backfill και σημαία MEMINTENSIVE

Πίνακας 6.24 mem2FCFS χωρίς backfill και σημαία MEMINTENSIVE

Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
Α	15.6	0	15.6	**16.68**
Β	2.34	0	2.34	
Γ	4.72	0	4.72	
Δ	1.03	0	1.03	
Ε	15.65	1.03	16.68	
ΣΤ	4.61	4.72	9.33	
Ζ	2.18	4.72	6.9	
Η	0.95	9.33	10.28	
Μέσος Όρος	**5.89**	**2.17**	**8.06**	

<u>Με backfill</u>



Σχήμα 6.22 mem2FCFS με backfill και χωρίς σημαία MEMINTENSIVE

Πίνακας 6.25 mem2FCFS με backfill και σημαία MEMINTENSIVE

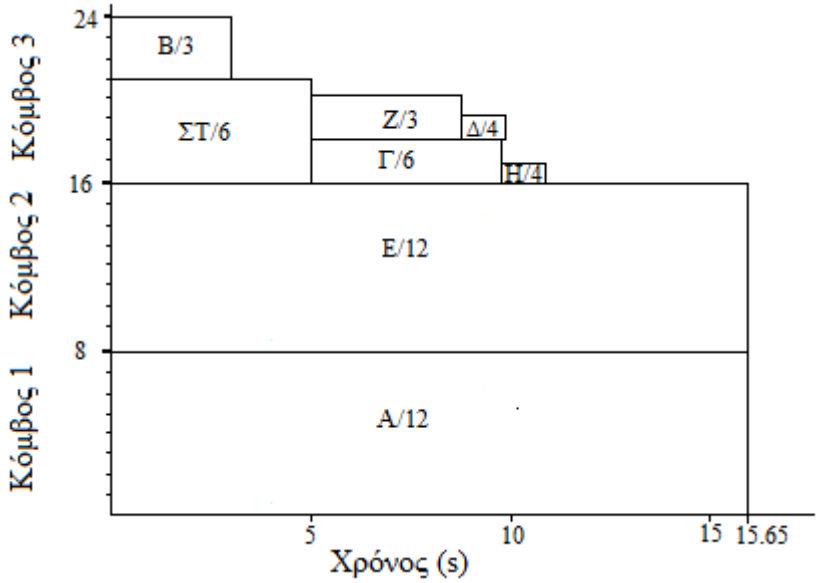
Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
Α	15.64	0	15.64	**16.58**
Β	2.34	0	2.34	
Γ	4.72	0	4.72	
Δ	1.03	0	1.03	
Ε	15.55	1.03	16.58	
ΣΤ	4.61	4.72	9.33	
Ζ	2.18	4.72	6.9	
Η	0.98	0	0.98	
Μέσος Όρος	**5.88**	**1.31**	**7.19**	

Παρατηρήσεις

- Ο μέσος χρόνος εκτέλεσης βελτιώνεται και σε αυτήν την περίπτωση, σε μικρότερο βαθμό βέβαια από ότι προηγουμένως (7.5% αντί για 32.4%)
- Ο μέσος χρόνος αναμονής αυξάνεται μόνο κατά 27.2% σε σύγκριση με το 80.6%, αν δεν χρησιμοποιηθεί η σημαία MEMINTENSIVE.
- Ο μέσος χρόνος ολοκλήρωσης εργασιών βελτιώνεται οριακά κατά 2.7%.
- Τέλος, ο συνολικός χρόνος ολοκλήρωσης αυξάνεται κατά 5.6%, που είναι σχετικά αποδεκτό σε σχέση με το 16.1% που είχαμε προηγουμένως.

6.7.2 mem2LJF

Χωρίς backfill

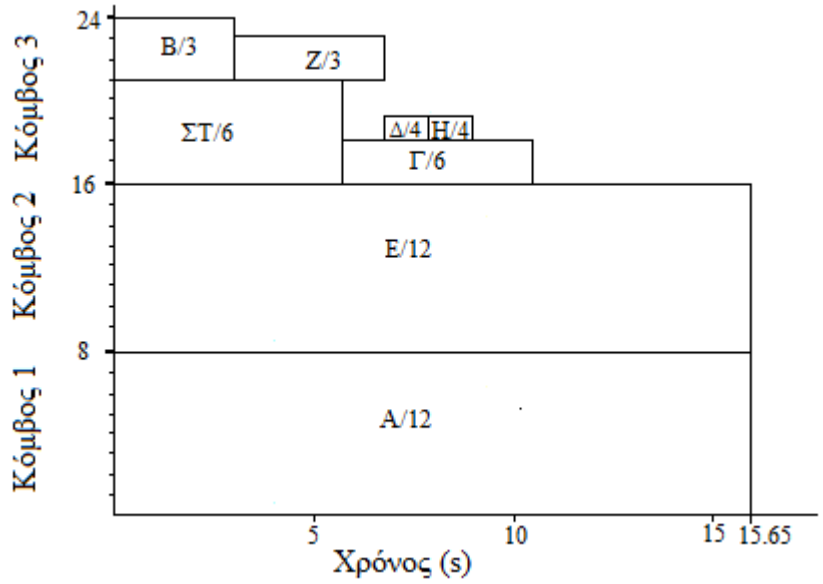


Σχήμα 6.23 mem2LJF χωρίς backfill και σημαία MEMINTENSIVE

Πίνακας 6.26 mem2LJF χωρίς backfill και σημαία MEMINTENSIVE

Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
A	15.65	0	15.65	**15.65**
B	2.31	0	2.31	
Γ	4.65	4.65	9.3	
Δ	1.03	6.88	7.91	
E	15.6	0	15.6	
ΣΤ	4.65	0	4.65	
Z	2.23	4.65	6.88	
H	1	9.3	10.3	
Μέσος Όρος	**5.89**	**3.19**	**9.08**	

Με backfill



Σχήμα 6.24 mem2LJF με backfill και χωρίς σημαία MEMINTENSIVE

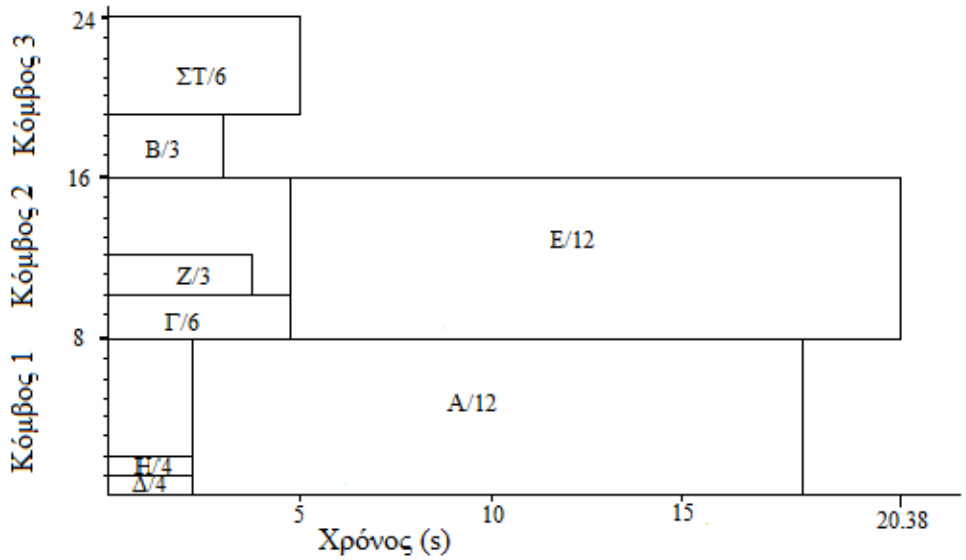
Πίνακας 6.27 mem2LJF με backfill και χωρίς σημαία MEMINTENSIVE

Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
A	15.61	0	15.61	**15.65**
B	2.31	0	2.31	
Γ	4.65	4.65	9.3	
Δ	1.03	4.54	5.57	
E	15.65	0	15.65	
ΣΤ	4.65	0	4.65	
Z	2.23	2.31	4.54	
H	1	5.57	4.57	
Μέσος Όρος	**5.89**	**2.13**	**8.02**	

Παρατηρήσεις

- Και εδώ έχουμε μικρότερη βελτίωση του μέσου χρόνου εκτέλεσης αφού μειώνεται μόνο κατά 12.1%, όταν χρησιμοποιείται η τεχνική backfill.
- Ο μέσος χρόνος αναμονής εμφανίζει αύξηση 19.7%.
- Η επίπτωση στον μέσο χρόνο ολοκλήρωσης είναι εμφανής, καθώς μειώνεται μόνο κατά 5.4% αντί για 27.4%.
- Τέλος, είναι φανερή η βελτίωση στον συνολικό χρόνο ολοκλήρωσης, καθώς παραμένει σταθερός. Αποφεύγουμε δηλαδή την 17.4% αύξηση που είχαμε προηγουμένως.

6.7.3 mem2SJF



Σχήμα 6.25 mem2SJF χωρίς σημαία MEMINTENSIVE

Πίνακας 6.28 mem2SJF χωρίς σημαία MEMINTENSIVE

Εργασία	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
Α	15.63	1.02	16.65	**20.38**
Β	2.33	0	2.33	
Γ	4.68	0	4.68	
Δ	1.02	0	1.02	
Ε	15.7	4.68	20.38	
ΣΤ	4.7	0	4.7	
Ζ	2.28	0	2.28	
Η	1	0	1	
Μέσος Όρος	**5.92**	**0.71**	**6.63**	

Παρατηρήσεις

- Για μία ακόμη φορά παρατηρούμε μικρότερη βελτίωση στον μέσο χρόνο εκτέλεσης κατά 5.6%.
- Η βελτίωση στον μέσο χρόνο αναμονής είναι προφανής, αφού από 300% πηγαίνει στο μόλις 22.4%.
- Η βελτίωση στον μέσο χρόνο ολοκλήρωσης κυμαίνεται στα ίδια επίπεδα με προηγουμένως.
- Και σε αυτήν την περίπτωση έχουμε πολύ καλύτερη συμπεριφορά στον συνολικό χρόνο ολοκλήρωσης των εργασιών, αφού από 14.5% πηγαίνουμε σε μόλις 1.1% αύξηση.

6.8 Συμπεράσματα

Παρακάτω παρουσιάζονται 2 πίνακες για τη σύνοψη των αποτελεσμάτων. Ο 1ος πίνακας παρουσιάζει τα αποτελέσματα των μετρήσεων με την εφαρμογή της memory aware πολιτικής και ο 2ος την επί τις εκατό μεταβολή από τα αποτελέσματα χωρίς την χρήση της memory aware πολιτικής (5ο κεφάλαιο):

Πίνακας 6.29 Σύνοψη memory aware πολιτικής

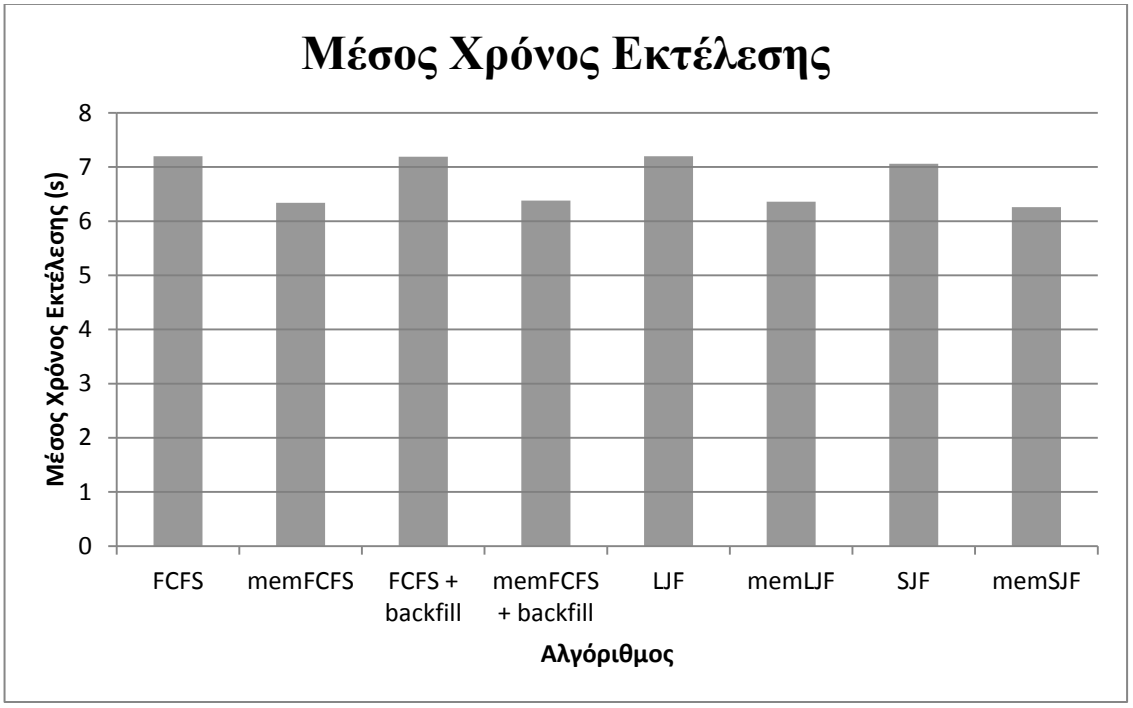
Εργασιακό Φορτίο	Αλγόριθμος	Μέσος Χρόνος Εκτέλεσης (s)	Μέσος Χρόνος Αναμονής (s)	Μέσος Χρόνος Ολοκλήρωσης (s)	Συνολικός Χρόνος (s)
CPU intensive	**memFCFS**	6.34	2.58	8.92	17.45
	memFCFS + backfill	6.38	0.49	6.87	17.33
	memLJF	6.34	3.27	9.61	13.6
	memSJF	6.26	0.83	7.09	19.95
Balanced	**memFCFS**	5.73	5.38	11.11	17.59
	memFCFS + backfill	5.74	1.07	6.81	17.64
	memLJF	5.77	8.26	14.03	17.69
	memLJF + backfill	5.75	3.39	9.14	17.69
	memSJF	5.72	1.65	7.37	17.57
Memory intensive	**memFCFS**	4.26	4.6	8.86	18.25
	memFCFS + backfill	4.3	1.86	6.16	18.22
	memLJF	4.3	10.44	14.74	18.55
	memLJF + backfill	4.29	1.87	6.16	18.38
	memSJF	4.3	2.32	6.62	23.09
Memory intensive 2	**mem2FCFS**	5.89	2.17	8.06	16.68
	mem2FCFS + backfill	5.88	1.31	7.19	16.58
	mem2LJF	5.89	3.19	9.08	15.65
	mem2LJF + backfill	5.89	2.13	8.02	15.65
	mem2SJF	5.92	0.71	6.63	20.38

Πίνακας 6.30 Σύγκριση αποτελεσμάτων

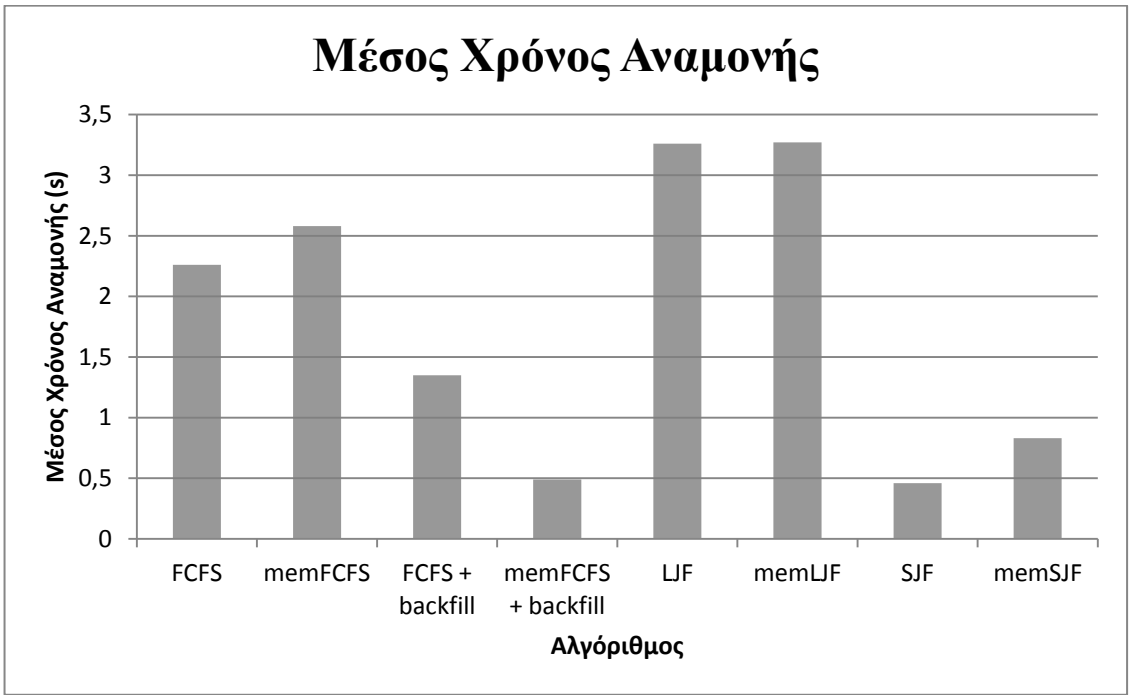
Εργασιακό Φορτίο	Αλγόριθμος	Μέσος Χρόνος Εκτέλεσης (%)	Μέσος Χρόνος Αναμονής (%)	Μέσος Χρόνος Ολοκλήρωσης (%)	Συνολικός Χρόνος (%)
CPU intensive	**memFCFS**	-11,9	14,2	-5,7	-11,4
	memFCFS + backfill	-11,3	-63,7	-19,6	-12,5
	memLJF	-11,7	0,3	-7,9	-31,1
	memSJF	-11,3	80,4	-5,7	-14,8
Balanced	**memFCFS**	-12,4	70,8	14,7	-13,8
	memFCFS + backfill	-12,6	91,1	-4,5	-13,8
	memLJF	-13,5	24	5,3	6,1
	memLJF + backfill	-12,6	-1,7	-8,9	12,1
	memSJF	-12,7	41	-4,5	-13,2
Memory intensive	**memFCFS**	-35,1	193	9	16,2
	memFCFS + backfill	-32,4	80,6	-16,6	16,1
	memLJF	-36,7	414,3	67,1	18,5
	memLJF + backfill	-36	5,1	-27,4	17,4
	memSJF	-31,4	300	-3,4	14,5
Memory intensive 2	**mem2FCFS**	-10,2	38,2	-0,9	6,2
	mem2FCFS + backfill	-7,5	27,2	-2,7	5,6
	mem2LJF	-13,3	57,1	2,9	0
	mem2LJF + backfill	-12,1	19.7	-5.4	0
	mem2SJF	-5,6	22,4	-3,2	1,1

Για καλύτερη εποπτεία των αποτελεσμάτων τα παρουσιάζουμε και σε γραφικές παραστάσεις:

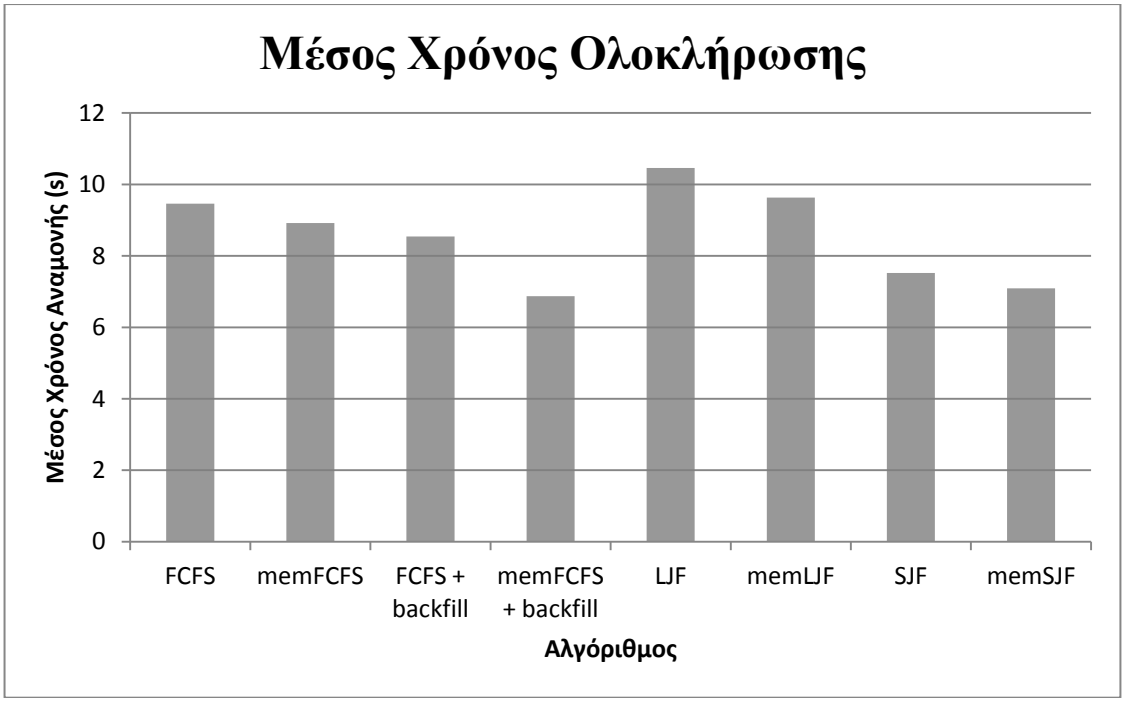
CPU intensive εργασιακό φορτίο



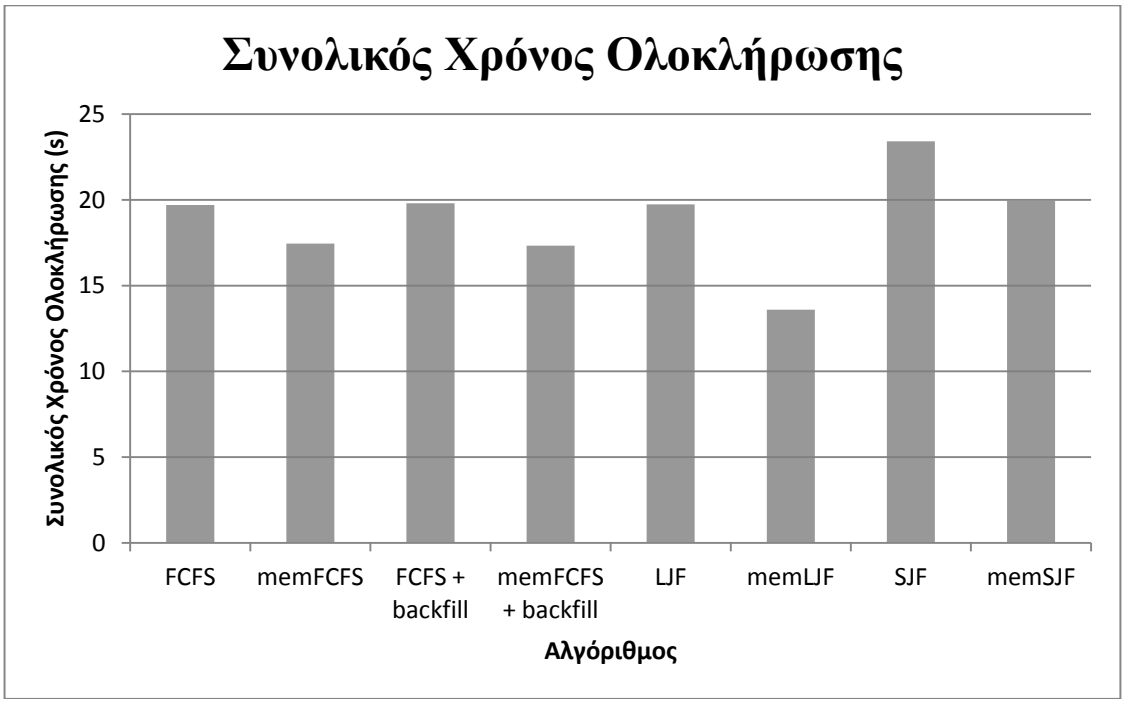
Σχήμα 6.26 CPU intensive workload-Μέσος Χρόνος Εκτέλεσης



Σχήμα 6.27 CPU intensive workload-Μέσος Χρόνος Αναμονής

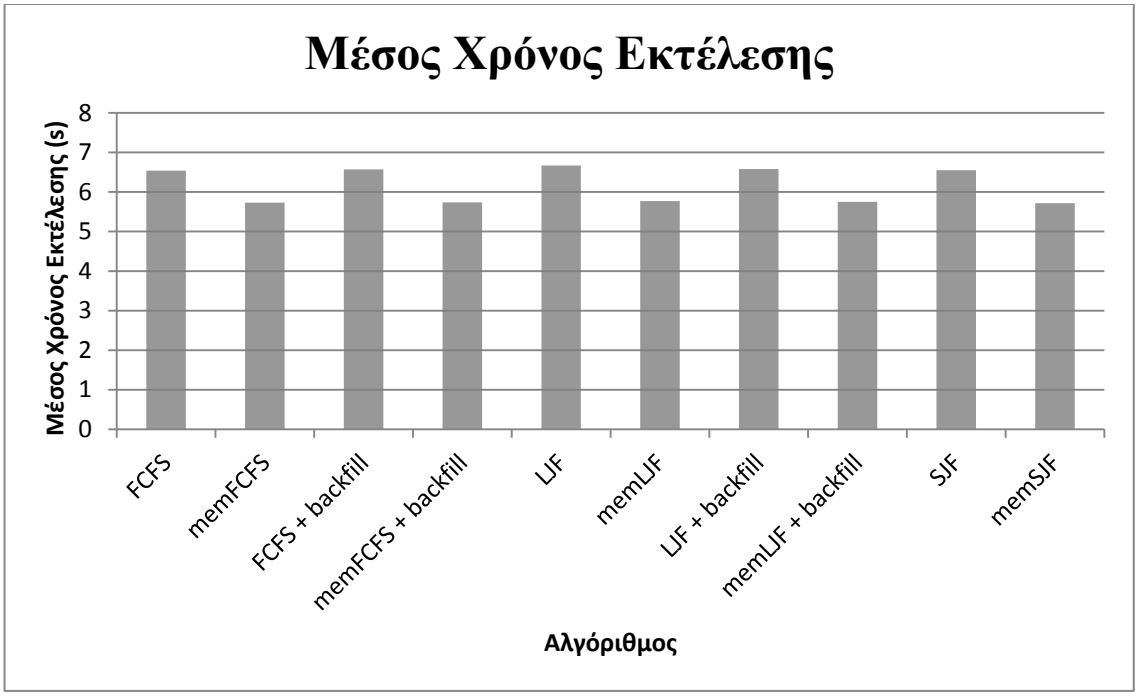


Σχήμα 6.28 CPU intensive workload-Μέσος Χρόνος Ολοκλήρωσης

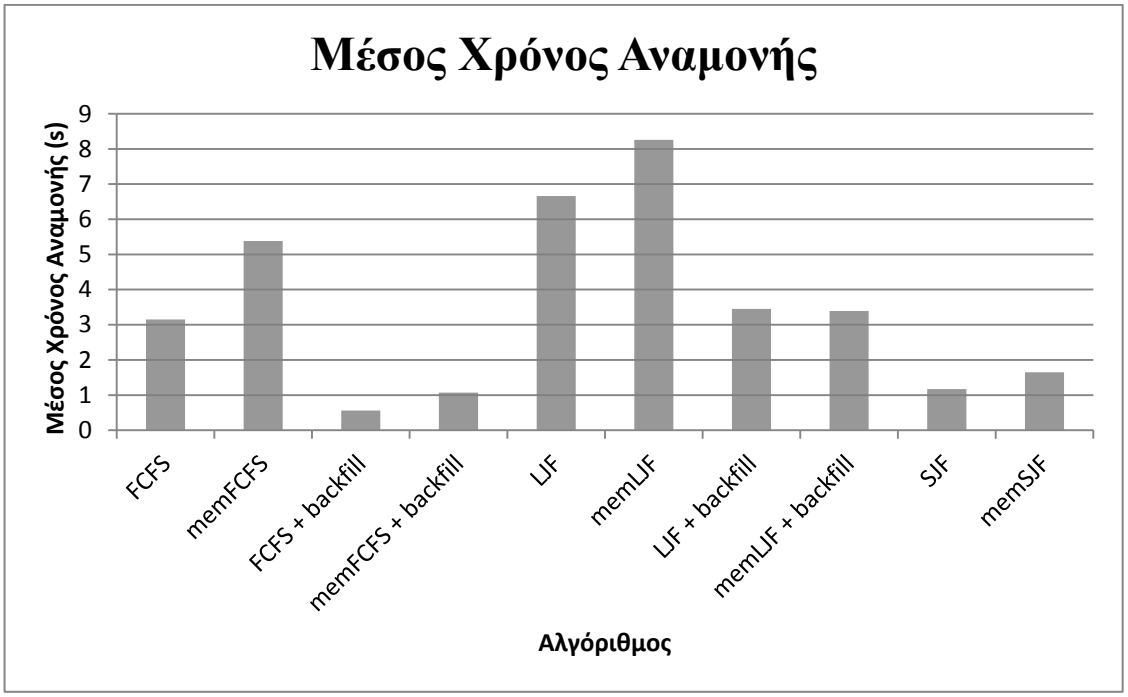


Σχήμα 6.29 CPU intensive workload-Συνολικός Χρόνος Ολοκλήρωσης

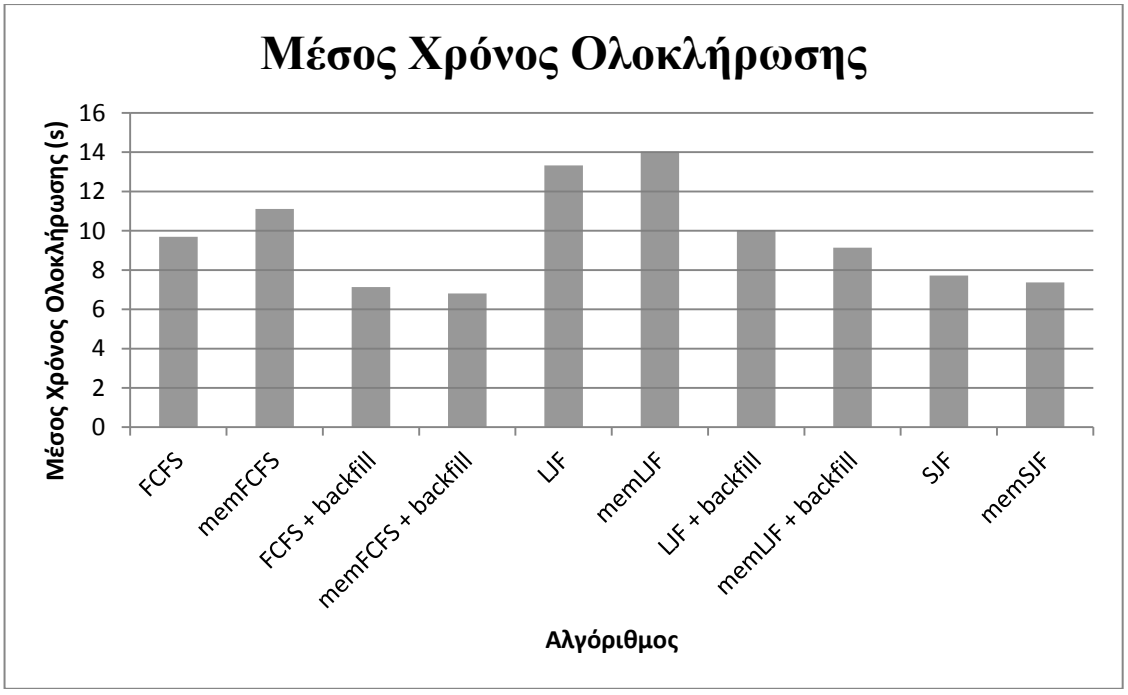
Ισορροπημένο εργασιακό φορτίο



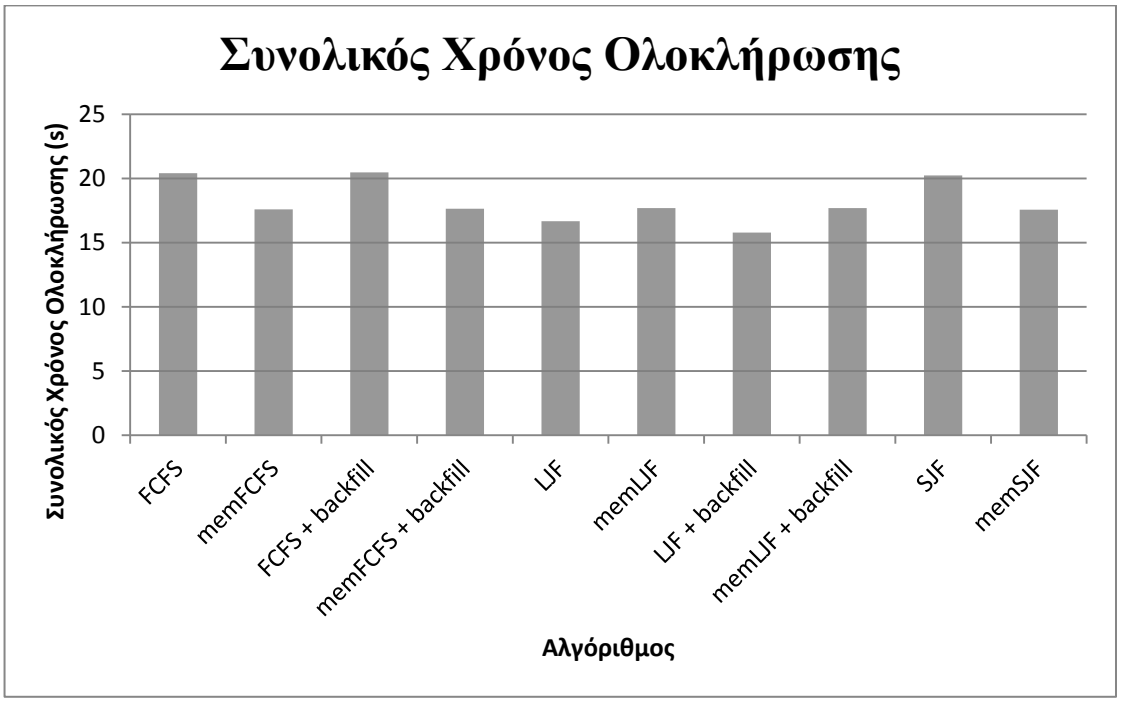
Σχήμα 6.30 Balanced workload-Μέσος Χρόνος Εκτέλεσης



Σχήμα 6.31 Balanced workload-Μέσος Χρόνος Αναμονής

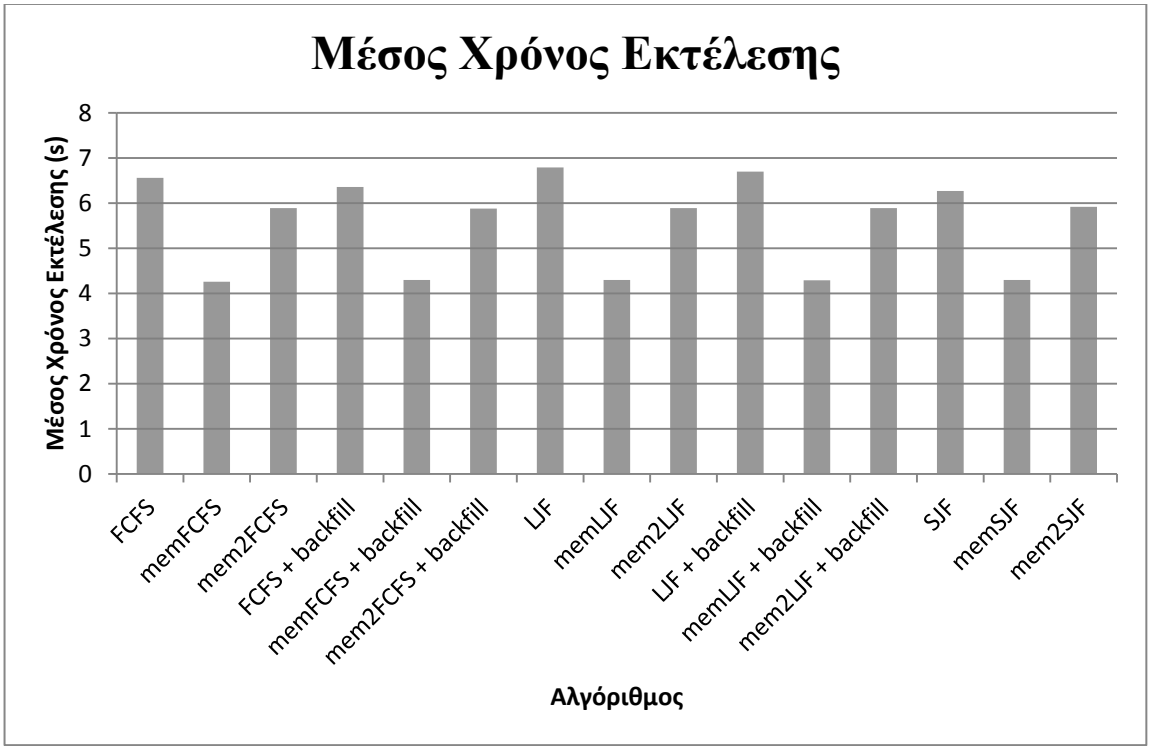


Σχήμα 6.32 Balanced workload-Μέσος Χρόνος Ολοκλήρωσης



Σχήμα 6.33 Balanced workload-Συνολικός Χρόνος Ολοκλήρωσης

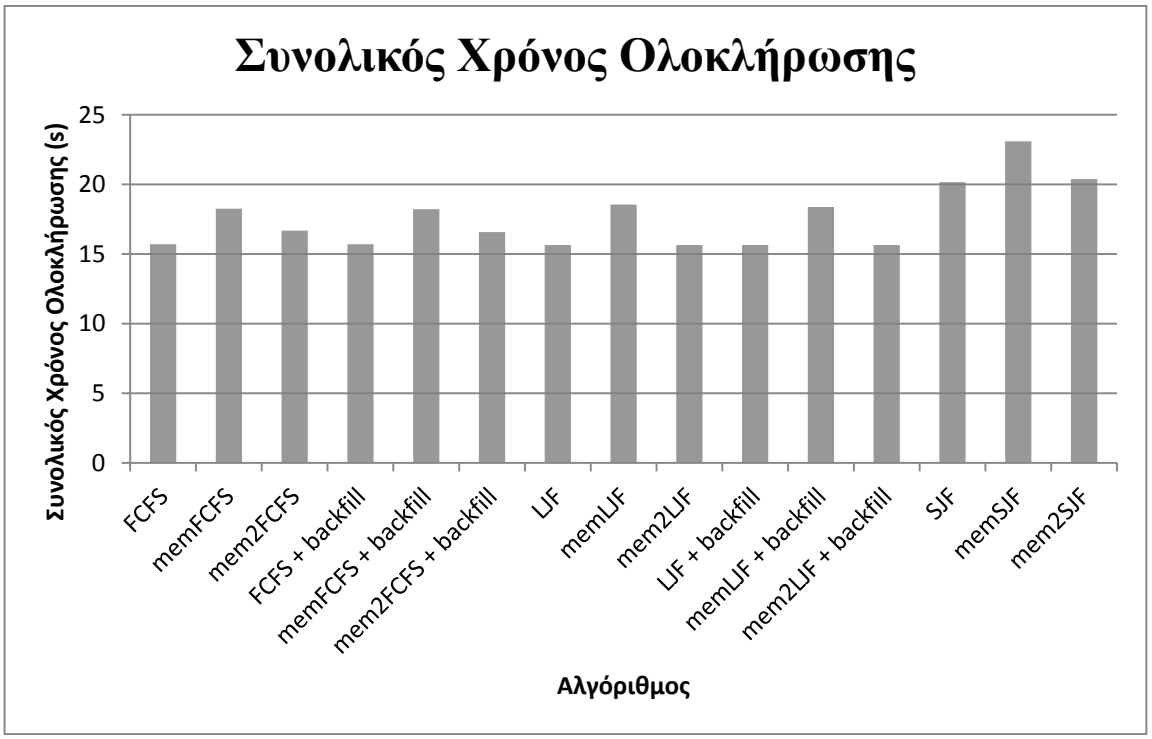
Memory intensive εργασιακό φορτίο



Σχήμα 6.34 Memory intensive workload-Μέσος Χρόνος Εκτέλεσης

Σχήμα 6.35 Memory intensive workload-Μέσος Χρόνος Αναμονής

Σχήμα 6.36 Memory intensive workload-Μέσος Χρόνος Ολοκλήρωσης



Σχήμα 6.37 Memory intensive workload-Συνολικός Χρόνος Ολοκλήρωσης

Το 1^{ο} πράγμα που πρέπει να σημειώσουμε είναι ότι η εφαρμογή μιας memory aware πολιτικής βελτιώνει τον μέσο χρόνο εκτέλεσης των εργασιών. Αυτό είναι λογικό γιατί κατανέμουμε τις εργασίες στους υπολογιστικού κόμβους με τέτοιον τρόπο ώστε να αποφύγουμε τον ανταγωνισμό για τον διάδρομο μνήμης. Έτσι, μπορούμε να παρατηρήσουμε βελτίωση στον μέσο χρόνος εκτέλεσης πάνω από 10 % σε σύνολα που δεν περιέχουν πολλές memory intensive εργασίες και έως και 30-35% σε σύνολα που έχουν μεγάλο ποσοστό τέτοιων εργασιών.

Μια 2η σημαντική παρατήρηση είναι η σημαντική συνεισφορά της τεχνικής *backfill*. Αυτό το είχαμε επισημάνει και στα συμπεράσματα του κεφαλαίου 5, αλλά με την εφαρμογή της memory aware πολιτικής η τεχνική αυτή αποκτάει έναν πολύ πιο σημαντικό ρόλο. Η memory aware πολιτική, διασκορπίζει τις επεξεργαστικές μονάδες που απαιτούν memory intensive εργασίες σε περισσότερους από έναν κόμβους, με αποτέλεσμα να δημιουργούνται μεγάλα διαστήματα όπου επεξεργαστικές μονάδες κόμβων μένουν ανενεργές, καθώς μικρότερες εργασίες μπλοκάρουν πίσω από μεγαλύτερες. Η τεχνική *backfill* μπορεί να γεμίσει αυτά τα κενά, διορθώνοντας σε μεγάλο βαθμό αυτό το πρόβλημα που δημιουργεί η memory aware πολιτική. Όπως, παρατηρήσαμε σε πολλά παραδείγματα έχουμε αύξηση του μέσου και του συνολικού χρόνου ολοκλήρωσης των εργασιών, αν χρησιμοποιηθεί η memory aware πολιτική χωρίς την ταυτόχρονη εφαρμογή της τεχνικής *backfill*.

Ο μέσος χρόνος αναμονής αυξάνεται σημαντικά σχεδόν σε όλα τα πειράματα που πραγματοποιήσαμε. Όμως, ο μέσος χρόνος ολοκλήρωσης μειώνεται σε όλες τις περιπτώσεις που εξετάσαμε, δεδομένου ότι χρησιμοποιείται η τεχνική backfill. Αυτό συμβαίνει γιατί ο μέσος χρόνος αναμονής ήταν σημαντικά μικρότερος κατά απόλυτη τιμή από τον μέσο χρόνο εκτέλεσης, με αποτέλεσμα η βελτίωση στο χρόνο εκτέλεσης να υπερκαλύπτει την επιδείνωση του χρόνου αναμονής.

Η μετρική που ίσως παρουσιάζει το μεγαλύτερο ενδιαφέρον είναι ο συνολικός χρόνος ολοκλήρωσης των εργασιών. Παρατηρούμε ότι όταν το ποσοστό των memory intensive εργασιών είναι μικρό, έχουμε σημαντική μείωση που ξεπερνά το 10%. Το ίδιο συμβαίνει και στην περίπτωση που οι memory και cpu intensive εργασίες είναι μοιρασμένες, με μοναδική εξαίρεση τον αλγόριθμο LJF. Αυτό συμβαίνει επειδή ο διασκορπισμός των επεξεργαστικών μονάδων που κατανέμονται στις memory intensive εργασιών αυξάνουν το άθροισμα του MEMINTESITY αρκετών κόμβων, με αποτέλεσμα να καθυστερεί η εκκίνηση πολλών εργασιών με μέτριο MEMINTESITY και της δημιουργίας μεγάλων διαστημάτων ανενεργίας επεξεργαστών. Αυτές οι εργασίες εκτελούνται τελικά μετά την ολοκλήρωση των memory intensive εργασιών, με αποτέλεσμα να επιμηκύνουν την συνολική διάρκεια εκτέλεσης του συνόλου.

Εκεί που τα αποτελέσματα στον συνολικό χρόνο ολοκλήρωσης είναι τραγικά είναι στο 3ο εργασιακό φορτίο, όπου έχουμε μεγάλο ποσοστό memory intensive εργασιών. Η αύξηση σε ορισμένες περιπτώσεις αγγίζει το 20%. Αυτό είναι λογικό γιατί έχουμε σε μεγαλύτερο βαθμό το φαινόμενο που περιγράψαμε για τον αλγόριθμο LJF του 2ου εργασιακού φορτίου. Δηλαδή, εργασίες με ενδιάμεσο MEMINTESITY δεν μπορούν να εκτελεστούν σε κόμβους που διαθέτουν τους απαιτούμενους επεξεργαστές, καθώς ξεπερνούν το επιτρεπτό άθροισμα MEMINTESITY.

Για να απαλύνουμε αυτή τη σημαντική επίπτωση, πραγματοποιήσαμε μια ακόμα σειρά μετρήσεων στο 3ο εργασιακό φορτίο, όπου εμφανίστηκε το πρόβλημα, διατηρώντας τον κανόνα ότι το επιτρεπτό άθροισμα MEMINTESITY σε έναν κόμβο δεν πρέπει να ξεπερνά το 10, αλλά καταργώντας τον διασκορπισμό των επεξεργαστικών μονάδων των memory intensive εργασιών σε περισσότερους από έναν κόμβους. Σε αυτό το σημείο να επισημάνουμε ότι μια εργασία με MEMINTESITY 12 δεν θα δρομολογηθεί, λόγω του τρόπου με τον οποίο κατασκευάστηκε ο αλγόριθμος. Ο τρόπος κατασκευής του αλγορίθμου απαιτεί όταν χρησιμοποιείται το πεδίο MEMINTESITY των εργασιών, χωρίς να χρησιμοποιείται η σημαία MEMINTESIVE, όλες οι εργασίες που έχουν MEMINTESITY περισσότερο από 10, να υποβάλλονται με MEMINTESITY 10, για να

μπορέσουν να δρομολογηθούν. Τα αποτελέσματα που παρατηρήσαμε με αυτήν την μετατροπή είναι ο περιορισμός της μείωσης στον μέσο χρόνο εκτέλεσης και ολοκλήρωσης εργασιών, αλλά σημαντική βελτίωση του μέσου χρόνου αναμονής και του συνολικού χρόνου ολοκλήρωσης.

Επίσης, πρέπει να επισημάνουμε ότι η εφαρμογή της *memory aware* πολιτικής θα έχει και ως συνέπεια την μείωση της ενέργειας που καταναλώνει το σύστημα. Αυτό συμβαίνει επειδή γλιτώνουμε σε μεγάλο βαθμό το κόστος διαιτησίας του διαδρόμου μνήμης αλλά και επειδή η σημαντική μείωση στον χρόνο εκτέλεσης των εργασιών θα επιτρέψει στο σύστημα να μεταβεί γρηγορότερα σε κατάσταση χαμηλής κατανάλωσης ενέργειας.

Συνοψίζοντας, η εφαρμογή της *memory aware* πολιτικής που κατασκευάσαμε ενδείκνυται σε περιπτώσεις που το σύνολο εργασιών έχουν χαμηλό ή μέσο ποσοστό memory intensive εργασιών ή ισοδύναμα όταν έχουμε ικανοποιητικό πλήθος υπολογιστικών κόμβων σε σχέση με τον αριθμό των memory intensive εργασιών, αφού βελτιώνει τόσο την αποκρισιμότητα του συστήματος (που εκφράζεται μέσω του μέσου χρόνου ολοκλήρωσης) αλλά και το συνολικό χρόνο ολοκλήρωσης και το ρυθμό διεκπεραίωσης των εργασιών. Σε περιπτώσεις όπου έχουμε μεγάλο ποσοστό memory intensive εργασιών, η χρησιμοποίηση της memory aware πολιτικής εξαρτάται από τους στόχους και τις αξίες του site. Αν το site αξιολογεί ως σημαντικότερα την αποκρισιμότητα του συστήματος και την σημαντική μείωση στην κατανάλωση ενέργειας, τότε η επιλογή αυτής της πολιτικής μπορεί να επιφέρει το επιθυμητό αποτέλεσμα. Σε περίπτωση όμως που δίνει μεγαλύτερη βαρύτητα στον ρυθμό διεκπεραίωσης των εργασιών, η χρησιμοποίηση της *memory aware* πολιτικής δεν ενδείκνυται. Βέβαια, σε αυτήν την περίπτωση μπορεί να γίνει ο συμβιβασμός που περιγράψαμε προηγουμένως με την κατάργηση της MEMINTENSIVE σημαίας. Με αυτό τον τρόπο το site θα συμβιβαστεί με μικρή βελτίωση στον μέσο χρόνο ολοκλήρωσης, με αντάλλαγμα οριακή επιδείνωση του ρυθμού διεκπεραίωσης εργασιών.

6.9 Μελλοντικές επεκτάσεις

Υπάρχουν πολλές επεκτάσεις της εφαρμογής που θα είχαν ερευνητικό και πρακτικό ενδιαφέρον. Κάποιες από αυτές αναφέρονται στη συνέχεια.

6.9.1 Βελτιστοποίηση κώδικα (Optimization)

Ο κώδικας που κατασκευάσαμε εξυπηρετεί τις ανάγκες της έρευνας των επιδόσεων της *memory aware* πολιτικής και θα απαιτούσε κάποιες βελτιστοποιήσεις για να είμαστε σε θέση να τον εφαρμόσουμε σε μόνιμη βάση σε ένα πραγματικό σύστημα μεγάλης κλίμακας. Μερικά από τα μειονεκτήματα του κώδικα, που μπορούν να γίνουν αντικείμενο περαιτέρω μελέτης, παρουσιάζονται παρακάτω.

Ένα μειονέκτημα του κώδικα είναι ότι όταν χρησιμοποιείται η σημαία MEMINTENSIVE, πραγματοποιείται μια ισοκατανομή των επεξεργαστών που παραχωρούνται στην MPI εργασία στους υπολογιστικούς κόμβους. Για παράδειγμα, αν μια εργασία έχει MEMINTENSITY ίση με 12 και απαιτεί 4 επεξεργαστές, ο αλγόριθμος που κατασκευάσαμε θα προσπαθήσει να βρει 2 κόμβους που να έχουν από 3 ελεύθερους επεξεργαστές και άθροισμα MEMINTENSITY κάτω από 4 (αφού η

καινούρια MEMINTENSITY της εργασίας σε κάθε κόμβο θα είναι ίση με $\frac{12}{2} = 6$). Όμως, είναι πολύ πιθανόν να μην υπάρχουν 2 τέτοιοι κόμβοι την δεδομένη χρονική στιγμή στο σύστημα. Σε αυτή την περίπτωση, η εργασία θα αναγκαστεί να περιμένει. Υπάρχει όμως η περίπτωση να υπάρχει ένας κόμβος με 4 ελεύθερους επεξεργαστές και ένας με 2, που ικανοποιούν παράλληλα και τον περιορισμό του μέγιστου αθροίσματος MEMINTENSITY. Σε αυτήν την περίπτωση, η εργασία θα έπρεπε να ξεκινήσει με αυτήν κατανομή επεξεργαστών. Επιπρόσθετα, το άθροισμα του MEMINTENSITY στον κόμβο με τους 4 ελεύθερους επεξεργαστές θα πρέπει να αυξηθεί κατά 8 και στον κόμβο με τους 2 κατά 4. Όμως, σε αυτή την περίπτωση αυξάνεται αρκετά η πολυπλοκότητα του χρονοδρομολογητή και όπως είπαμε και προηγουμένως, τα μειονεκτήματα από την αύξηση της πολυπλοκότητας του χρονοδρομολογητή δεν θα πρέπει να υπερβαίνουν τα οφέλη της νέας πολιτικής. Μια μελέτη πάνω σε αυτό το αντικείμενο θα ήταν αρκετά ενδιαφέρουσα.

Επιπρόσθετα, ο κώδικας, αλλά και γενικότερα η πολιτική που κατασκευάσαμε, κάνει την παραδοχή ότι το σύστημά μας είναι ομογενές. Αυτό γίνεται εύκολα αντιληπτό, αφού ορίσαμε ως *memory intensity* το ποσοστό του *memory bandwidth* που θα απαιτούσε η εργασία αν έτρεχε μόνη της στον κόμβο. Σε ένα σύστημα μεγάλης κλίμακας όμως, είναι πιθανόν να υπάρχουν κόμβοι με διαφορετικά στοιχεία (πχ διαφορετικό μέγεθος cache, διαφορετικό memory bandwidth), με αποτέλεσμα να εξαρτάται η τιμή του *memory intensity* από τον κόμβο που θα εκτελεστεί. Μια πρόχειρη λύση σε αυτό το ζήτημα ίσως να είναι η χρησιμοποίηση της προηγμένης αλγοριθμικής τεχνική *node sets*. Με αυτήν την τεχνική φτιάχνουμε σύνολα κόμβων, στα οποία μπορούν να τρέχουν ορισμένες εργασίες. Έτσι, μπορούμε να κατασκευάσουμε σύνολα που αποτελούνται από ομοιόμορφους κόμβους ως προς το *memory intensity*, δηλαδή κόμβους όπου οι εργασίες θα έχουν το ίδιο *memory intensity* αν εκτελεστούν σε αυτούς. Με αυτό τον τρόπο μπορούμε ίσως να αντιμετωπίσουμε αποτελεσματικά το πρόβλημα των ετερογενών κόμβων.

Τέλος, τα πειράματα που προηγήθηκαν έγιναν σε εργασίες που απαιτούσαν σχετικά μικρό αριθμό επεξεργαστών και κόμβων. Έτσι, ο κύριος παράγοντας που επηρέαζε τον χρόνο εκτέλεσης των εργασιών ήταν ο χρόνος υπολογισμών, τον οποίο καταφέραμε να μειώσουμε με την εφαρμογή της συγκεκριμένης πολιτικής. Σε περιπτώσεις όμως που οι εργασίες απαιτούν μεγάλο αριθμό κόμβων και επεξεργαστών, ο χρόνος επικοινωνίας μπορεί να είναι συγκρίσιμος με τον χρόνο υπολογισμού. Σε μια τέτοια περίπτωση, η εφαρμογή της παραπάνω πολιτικής μπορεί να επιδεινώσει τα αποτελέσματα, αφού η παραχώρηση στην εργασία επεξεργαστών από περισσότερους κόμβους θα αυξήσει σημαντικά τον χρόνο επικοινωνίας. Θα ήταν πολύ ενδιαφέρον να προσδιοριστεί σε μια μελλοντική μελέτη ένα άνω όριο, πάνω από το οποίο να σταματάμε να εφαρμόζουμε την memory aware πολιτική, λόγω της επιδείνωσης του χρόνου επικοινωνίας.

6.9.2 Μελέτη ανταγωνισμού σε άλλους τομείς

Η κύρια μνήμη δεν είναι ο μόνος μοιραζόμενος πόρος για τον οποίο υπάρχει περίπτωση να ανταγωνίζονται οι εργασίες, κατά την ταυτόχρονη εκτέλεσή τους στον ίδιο κόμβο. Ανάλογες έρευνες έχουν πραγματοποιηθεί και για τον ανταγωνισμό των caches [39], τον ανταγωνισμό για το δίκτυο [38] αλλά και για τον ανταγωνισμό για τον δίσκο [40]. Ενδιαφέρουσα θα ήταν η κατασκευή ενός χρονοδρομολογητή που αντιμετωπίζει όλες τις παραπάνω περιπτώσεις και η αξιολόγησή του τόσο ως προς τις επιδόσεις του, την

κατανάλωση ενέργειας και γενικότερα των πλεονεκτημάτων του σε σχέση με την πολυπλοκότητά του.

6.9.3 Επέκταση σε συστήματα καταμερισμού χρόνου

Η μελέτη που πραγματοποιήσαμε αφορά στα συστήματα καταμερισμού χώρου. Όμως, το θέμα ανταγωνισμού των εργασιών δεν εμφανίζεται μόνο στα batch συστήματα. Σήμερα, ακόμα και οι συμβατικοί υπολογιστές είναι κατά κύριο λόγο πολυπύρηνα μηχανήματα, οπότε η περίπτωση ανταγωνισμού για τους κοινούς πόρους είναι συνηθισμένη. Ένας έξυπνος χρονοδρομολογητής θα μπορούσε να αποτρέπει *memory intensive* εργασίες από το να εκτελούνται ταυτόχρονα, αποφεύγοντας έτσι το κόστος διαιτησίας του διαδρόμου μνήμης.

7 Βιβλιογραφία

[1] Computer Science and Technology Board, National Research Council, Academy Industry Program and National Academy of Sciences, Supercomputers: directions in technology and applications, 1990

[2] Mark D. Hill , Norman P. Jouppi and Gurindar S. Sohi, *Readings in computer architecture*, 1999

[3] Douglas Eadline, *High Performance Computing for Dummies*

[4] Radu Prodan and Thomas Fahringer, *Grid computing: experiment management, tool integration and scientific workflows, 2007*

[5] Rainer Keller, David Kramer and Jan-Philipp Weiss, *Facing the Multicore-Challenge: Aspects of New Paradigms and Technologies in Parallel Computing*, 2010

[6] www.top500.org

[7] Condon, J.H. and K.Thompson, "Belle Chess Hardware", In *Advances in Computer Chess 3*(ed.M.R.B.Clarke), Pergamon Press, 1982

[8] Feng-Hsiung Hsu, *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion*, Princeton University Press, 2002

[9] Chrilly Donninger and Ulf Lorenz, *The Chess Monster Hydra*

[10] Ian Foster, Yong Zhao, Ioan Raicu and Shiyong Lu, *Cloud Computing and Grid Computing 360-Degree Compared*

[11] William James Dally and Brian Patrick Towles, *Principles and Practices of Interconnection Networks (The Morgan Kaufmann Series in Computer Architecture and Design*)

[12] Jose Duato, Sudhakar Yalamanchili and Lionel M. Ni, *Interconnection Networks: An Engineering Approach*

[13] Alexander A. Balandin, *Better Computing Through CPU Cooling*, http://spectrum.ieee.org/semiconductors/materials/better-computing-through-cpu-cooling/0, September 2009

[14] www.green500.org

[15] Timothy Prickett Morgan, *IBM uncloaks 20 petaflops BlueGene/Q super*, http://www.theregister.co.uk/2010/11/22/ibm_blue_gene_q_super/, November 2010

[16] http://www.hpcwire.com/hpcwire/2010-07-02/ibm_hot_water-cooled_supercomputer_goes_live_at_eth_zurich.html, July 2010

[17] Timothy Prickett Morgan, *IBM 'Blue Waters' super node washes ashore in August*, http://www.theregister.co.uk/2011/07/15/power_775_super_pricing/, July 2011

[18] David Padua, *Encyclopedia of Parallel Computing*, 2011

[19] Donald MacKenzie, *Knowing machines: essays on technical change*, 1998

[20] Marco Danelutto, Marco Vanneschi and Domenico Laforenza, *Euro-Par 2004 Parallel Processing: 10th International Euro-Par Conference* 2004

[21] Sadaf R. Alam et al, *An Evaluation of the Oak Ridge National Laboratory Cray XT3*, International Journal of High Performance Computing Applications, February 2008, vol. 22 no. 1 52-80

[22] G. Shanier, *Hpc market and interconnects report*, May 2009.

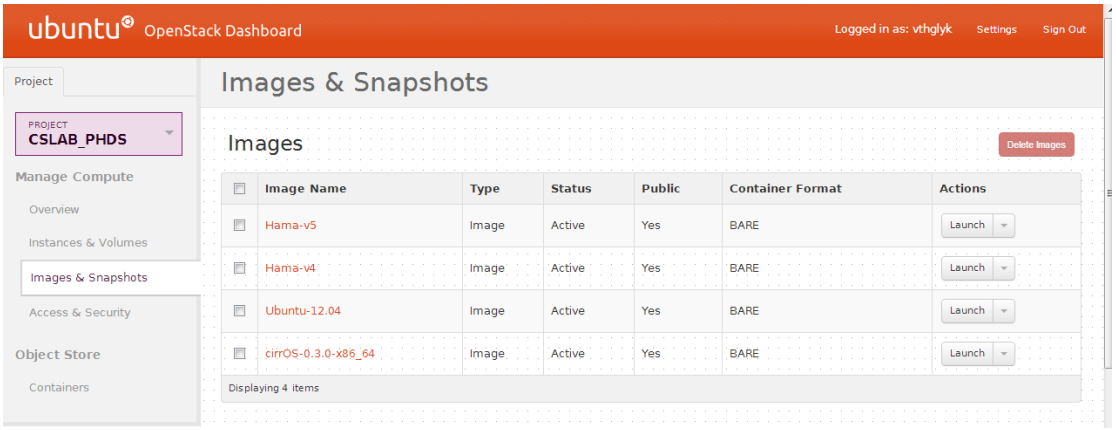
[23] Committee on the Potential Impact of High-End Computing on Illustrative Fields of Science and Engineering and National Research Council, *The Potential*

Impact of High-End Capability Computing on Four Illustrative Fields of Science and Engineering, October 2008
[24] Jack J. Dongarra, Piotr Luszczek and Antoine Petitet, *The LINPACK Benchmark: past, present and future*, July 2002
[25] S. Iqbal, R. Gupta, and Y. Fang, *Planning considerations for job scheduling in hpc clusters*, Dell Power Solutions, Tech. Rep., Feb 2005.
[26] N. Bansal and M. Harchol-Balter, *Analysis of srpt scheduling: Investigating unfairness*, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, Tech. Rep., 1999.
[27] W. B. Hurst, S. Ramaswamy, R. B. Lenin and D. Hoffman, *Modeling and Simulation of HPC Systems Through Job Scheduling*
[28] David Jackson, Quinn Snell, and Mark Clement, *Core Algorithms of the Maui Scheduler*
[29] Maui Administrator's Guide
[30] T. Thanalapati and S. Dandamudi, *An efficient adaptive scheduling scheme for distributed memory multicomputers*, IEEE Transactions on Parallel and Distributed Systems, vol. 12, no. 7, pp. 758–768, July 2001.
[31] E. Frachtenberg, D. G. Feitelson, F. Petrini, and J. Fernandez, *Adaptive parallel job scheduling with flexible co-scheduling*, IEEE transactions on Parallel and Distributed Systems, Tech. Rep. Vol. 16, No. 11, Nov 2005.
[32] J. Abawajy, *An efficient adaptive scheduling policy for highperformance computing*, ScienceDirect, Future Generation Computer Systems, www.elsevier.com/locate/fgcs, Tech. Rep. Vol. 25, p. 364-370, Apr 2006.
[33] F. Guim, I. Rodero, and J. Corbalan, *A multi-site scheduling architecture with data mining prediction techniques*,.
[34] Matthias Hovestadt, Odej Kao, Axel Keller and Achim Streit,*Scheduling in HPC Resource Management Systems: Queuing vs. Planning*
[35] Torque administration guide
[36] David B Jackson, *New Issues and New Capabilities in HPC Scheduling with the Maui Scheduler*
[37] Sergey Blagodurov, Sergey Zhuravlev and Alexandra Fedorove, *Contention-Aware Scheduling*
[38] Evangelos Koukis and Nectarios Koziris, *Memory and Network Bandwidth Aware Scheduling of Multiprogrammed Workloads on Clusters of SMPs*
[39] *Sergey Blagodurov Sergey Zhuravlev Serge Lansiquot Alexandra Fedorova, Addressing Shared Resource Contention in Multicore Processors Via Scheduling*
[40] Emilia Rosti, Giuseppe Serazzi, Evgenia Smirni and Mark S. Squillante, *The impact of I/O on program behavior and parallel scheduling*
[41] Wm. A. Wulf and Sally A. McKee, *Hitting the memory wall: implications of the obvious*, SIGARCH Comput. Archit. News 23, 1 (March 1995), 20-24

Παράρτημα Ι

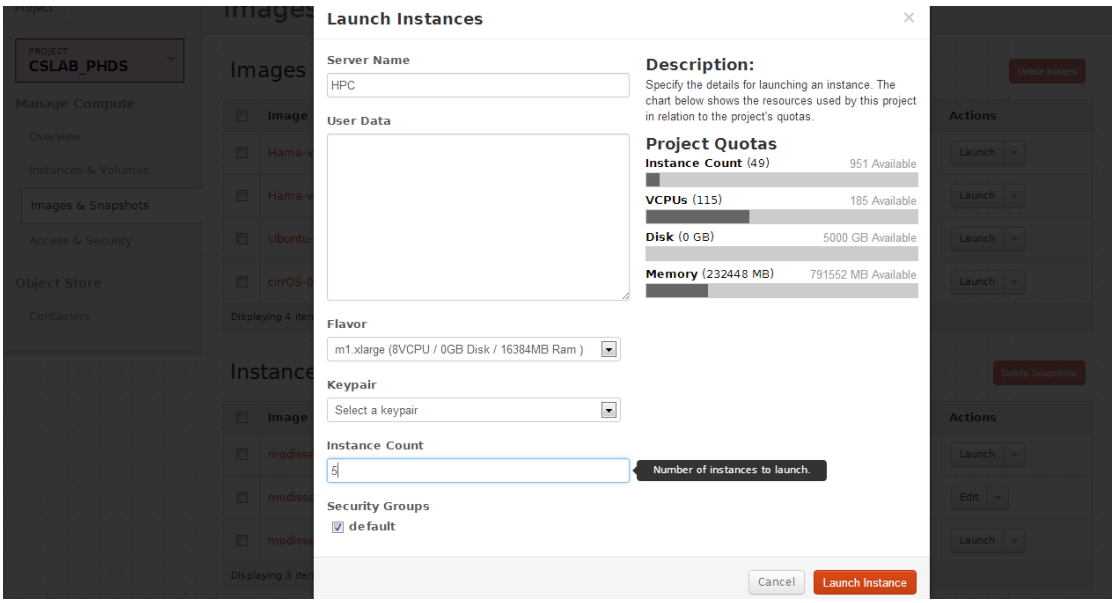
Σε αυτό το παράρτημα περιγράφουμε την διαδικασία κατασκευής του cluster στο οποίο έγινε η ανάπτυξη και η εισαγωγή της νέας μας πολιτικής στον Maui.

Το 1^{ο} βήμα της διαδικασίας είναι η κατασκευή των virtual machines (VMs). Για αυτό το λόγο πηγαίνουμε στο site http://termi2.cslab.ece.ntua.gr/. Για να ξεκινήσουμε VMs πηγαίνουμε στο *Images & Snapshots*, επιλέγουμε το *image* που χρειαζόμαστε (στην περίπτωσή μας το *Hana-v5*) και πατάμε *Launch*.



Σχήμα 7.0.1 Επιλογή virtual machines

Συμπληρώνουμε τα απαραίτητα στοιχεία (στην περίπτωσή μας επιλέξαμε 5 VMs, με 8 CPUs το καθένα, με το όνομα HPC) και πατάμε *Launch Instance*. Πληροφορίες για τα VMs μπορούμε να πάρουμε από την καρτέλα *Images & Snapshots*.



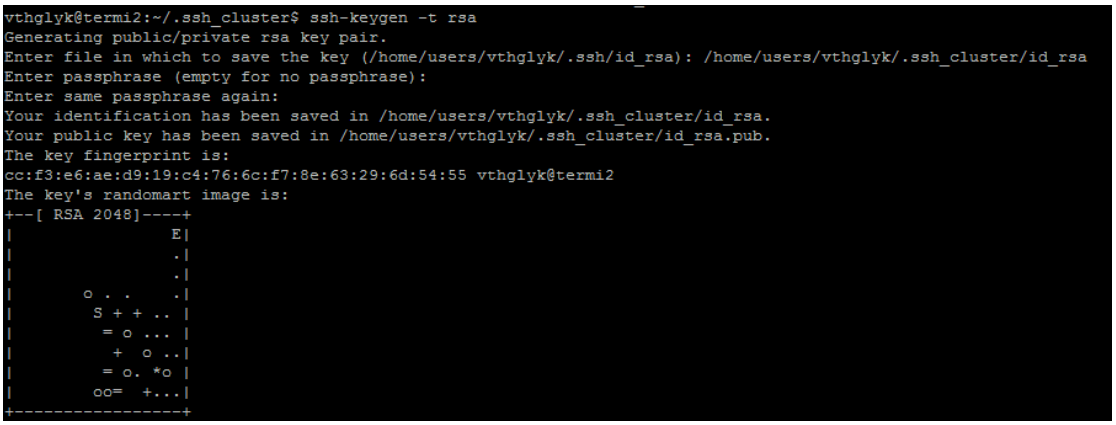
Σχήμα 7.0.2 Συμπλήρωση απαραίτητων στοιχείων

Για να αποκτήσουμε πρόσβαση στα VM μας χρειαζόμαστε πρόσβαση στον termi2. Για αυτό κάνουμε *ssh vthglyk@termi2.cslab.ece.ntua.gr*. Το επόμενο βήμα είναι η κατασκευή ενός κλειδιού rsa, για να εξασφαλιστεί η ασφαλής πρόσβαση στα VMs και να διευκολυνθεί η μεταξύ τους επικοινωνία. Έτσι, κατασκευάζουμε έναν καινούργιο

φάκελο *.ssh_cluster* και μέσα σε αυτόν εκτελούμε την εντολή για την δημιουργία ενός καινούργιου ssh key pair.

```
>> vthglyk@termi2:~$ mkdir .ssh_cluster
>> vthglyk@termi2:~$ cd .ssh_cluster/
>> vthglyk@termi2:~/.ssh_cluster$ ssh-keygen -t rsa
```

Συμπληρώνουμε την τοποθεσία που θέλουμε να αποθηκευτεί το κλειδί και δίνουμε ένα passphrase. Με αυτό το passphrase στην συνέχεια θα μπορούμε να αποκτάμε πρόσβαση στον κόμβο. Η διαδικασία φαίνεται στην παρακάτω εικόνα:

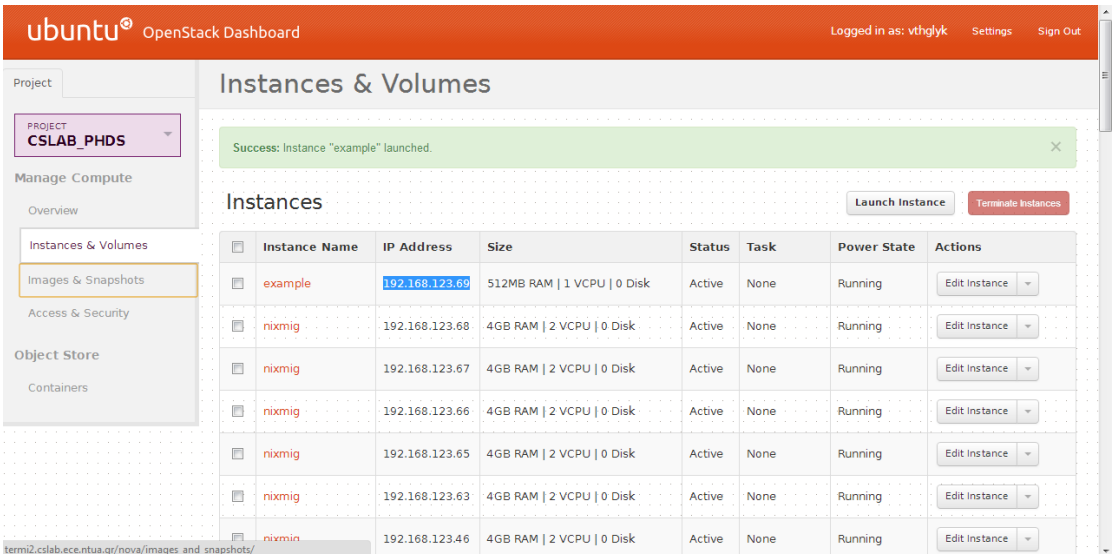


Σχήμα 7.0.3 Δημιουργία rsa κλειδιού

Στη συνέχεια αποθηκεύουμε το περιεχόμενο του *id_rsa.pub* σε ένα καινούργιο αρχείο *authorized_keys*, που είναι το αρχείο που περιέχει όλα τα κλειδιά με τα οποία μπορούν να αποκτήσουν πρόσβαση στο σύστημα χρήστες από απόσταση.

```
>> vthglyk@termi2:~/.ssh_cluster$ cat id_rsa.pub \
> >> authorized_keys
```

Συνεχίζουμε με την αντιγραφή του φακέλου *.ssh_cluster* στον master node. Χρησιμοποιούμε στην εντολή scp και ως προορισμό την διεύθυνση ip του master node, την οποία μπορούμε να δούμε από το site του termi2, στην καρτέλα *Instances & Volumes.*



Σχήμα 7.0.4 Instances & Volumes

Στην ερώτηση αν θέλουμε να συνεχίσουμε, επειδή η αυθεντικότητα του προορισμού δεν μπορεί να διαπιστωθεί, πληκτρολογούμε *yes*. Επιπρόσθετα, πληκτρολογούμε τον απαραίτητο κωδικό του *root*.

```
vthglyk@termi2:~$ scp -r .ssh_cluster root@192.168.123.69:~/
The authenticity of host '192.168.123.69 (192.168.123.69)' can't be established.
ECDSA key fingerprint is b9:94:b8:93:9b:17:a3:6b:48:10:c5:7c:b4:26:e3:ec.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.123.69' (ECDSA) to the list of known hosts.
root@192.168.123.69's password:
authorized_keys
id_rsa
id_rsa.pub
vthglyk@termi2:~$
```

Σχήμα 7.0.5 Αντιγραφή του φακέλου .ssh στον master

Κάνουμε *ssh* στον master node, διαγράφουμε τον *.ssh* φάκελο που προϋπάρχει αλλά δεν περιέχει ακόμα τίποτα, μετονομάζουμε τον φάκελο *.ssh_cluster* σε *.ssh* και προσθέτουμε τα κατάλληλα permissions στον φάκελο και το αρχείο *authorized_keys*.

```
>> vthglyk@termi2:~$ ssh  root@192.168.123.69
>> root@hpc:~# rm -r .ssh
>> root@hpc:~# mv .ssh_cluster/ .ssh
>> root@hpc:~# chmod 700 .ssh/
>> root@hpc:~# chmod 600 .ssh/authorized_keys
```

Στην συνέχεια θα πάμε να τροποποιήσουμε το αρχείο */etc/hosts* και θα προσθέσουμε τα ονόματα και τις διευθύνσεις των υπολοίπων κόμβων, για να αποκτάμε πρόσβαση στον κόμβο μόνο με το όνομά του και όχι την διεύθυνση *ip* του, και το */etc/hostname* για να ξεχωρίζουμε σε ποιον κόμβο είμαστε. Κάνουμε reboot για να εφαρμοστεί η αλλαγή στο όνομα.

```
>> root@hpc:~# vi /etc/hosts
```

```
127.0.0.1 localhost
192.168.123.69 master
192.168.123.41 node1
192.168.123.42 node2
192.168.123.43 node3
192.168.123.46 node4

# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts

~
~
~
~
~
~
~
~
"/etc/hosts" 15L, 328C                                        1,1           All
```

Σχήμα 7.0.6 Τροποποίηση του αρχείου /etc/hosts

```
>> root@hpc:~# vi /etc/hostname
```

Σχήμα 7.0.7 Τροποποίηση του αρχείου /etc/hostname

```
>> root@hpc:~# reboot
```

Για την πιο εύκολη επικοινωνία μεταξύ των κόμβων αφαιρέσαμε το passphrase από το *rsa key*, για να μην χρειάζεται να το πληκτρολογούμε κάθε φορά και επειδή κάποια προγράμματα που χρησιμοποιήσαμε στην συνέχεια (pdsh, pdcp) απαιτούν την αφαίρεσή του.

```
root@master:~/.ssh# openssl rsa -in ~/.ssh/id_rsa -out ~/.ssh/id_rsa_new
Enter pass phrase for /root/.ssh/id_rsa:
writing RSA key
root@master:~/.ssh# cp id_rsa id_rsa.backup
root@master:~/.ssh# rm id_rsa
root@master:~/.ssh# mv id_rsa_new id_rsa
root@master:~/.ssh# mv id_rsa_new id_rsa
root@master:~/.ssh# chmod 400 id_rsa
```

Σχήμα 7.0.8 Αφαίρεση passphrase από το rsa key

Κάνουμε *ssh* σε όλους τους κόμβους για να προστεθούν ως *known_hosts*. Σε κάθε κόμβο αλλάζουμε επίσης το όνομα του, τροποποιώντας πάλι το αρχείο */etc/hostname*.

```
>> root@master:~# ssh node1
```

Μετά αντιγράφουμε τον φάκελο *.ssh* σε όλους τους κόμβους.

```
>> root@master:~# scp –r .ssh root@node1:~/
```

Τέλος, εγκαθιστούμε σε όλους τους κόμβους το πρόγραμμα *pdsh*, που περιέχει 2 πολύ χρήσιμες εντολές. Την εντολή *pdsh*, που χρησιμοποιείται για ταυτόχρονη εκτέλεση μιας εντολής σε πολλούς κόμβους, και την εντολή *pdcp*, για ταυτόχρονη αντιγραφή σε κόμβους. Χρησιμοποιώντας την εντολή *pdcp*, αντιγράφουμε το αρχείο */etc/hosts*, που περιέχει τα ονόματα των hosts σε όλους τους κόμβους.

```
>> root@master:~# aptitude install pdsh
>> root@nodes:~# aptitude install pdsh
>> root@master:~# pdcp -R ssh -w node[1-4] /etc/hosts /etc
```

Παράρτημα ΙΙ

Σε αυτό το παράρτημα περιγράφουμε την διαδικασία εγκατάστασης του συστήματος διαχείρισης πόρων. Ως διαχειριστή πόρων επιλέξαμε τον *Torque* και ως χρονοδρομολογητή τον *Maui*.

Αρχικά κατεβάσαμε την τελευταία έκδοση του *Torque* από το επίσημο site *http://www.adaptivecomputing.com/support/download-center/torque-download/* και το μεταφέραμε στον φάκελο */root* του master node. Επίσης, κατεβάσαμε και εγκαταστήσαμε τα απαραίτητα πακέτα με τις παρακάτω εντολές:

```
>> root@master:~# apt-get install gcc
>> root@master:~# apt-get install build-essential
>> root@master:~# apt-get install libssl-dev
>> root@master:~# apt-get install libxml2-dev
```

Αποσυμπιέσαμε το Torque, μπήκαμε στον φάκελό του, θέσαμε το *library path* στον φάκελο όπου θα εγκατασταθούν οι βιβλιοθήκες του Torque και κάναμε το *configure* και το *make* και το *make install*.

```
>> root@master:~# tar -xzvf torque-4.2.2.tar.gz
>> root@master:~# cd torque-4.2.2/
>> root@master:~/torque-4.2.2# echo '/usr/local/lib' \
> > /etc/ld.so.conf.d/torque.conf
>> root@master:~/torque-4.2.2# ldconfig
>> root@master:~/torque-4.2.2# ./configure
>> root@master:~/torque-4.2.2# make
>> root@master:~/torque-4.2.2# make install
```

Από προεπιλογή το *make install* δημιουργεί έναν φάκελο */var/spool/torque*. Αυτός ο φάκελος αναφέρεται ως *TORQUE_HOME*.

Στην συνέχεια φτιάξαμε τα *tpackages* τα οποία θα εγκατασταθούν στους υπολογιστικούς κόμβους. Το output ήταν το εξής:

```
root@master:~/torque-4.2.2# make packages
Building packages from /root/torque-4.2.2/tpackages
rm -rf /root/torque-4.2.2/tpackages
mkdir /root/torque-4.2.2/tpackages
Building ./torque-package-server-linux-x86_64.sh ...
libtool: install: warning: remember to run `libtool --finish /usr/local/lib'
Building ./torque-package-mom-linux-x86_64.sh ...
libtool: install: warning: remember to run `libtool --finish /usr/local/lib'
Building ./torque-package-clients-linux-x86_64.sh ...
libtool: install: warning: remember to run `libtool --finish /usr/local/lib'
Building ./torque-package-devel-linux-x86_64.sh ...
libtool: install: warning: remember to run `libtool --finish /usr/local/lib'
Building ./torque-package-doc-linux-x86_64.sh ...
Done.

The package files are self-extracting packages that can be copied
and executed on your production machines.  Use --help for options.
```

Σχήμα 8.0.1 Δημιουργία tpackages

Στην συνέχεια χρησιμοποιήσαμε τις εντολές *pdcp* και *pdsh* για να κάνουμε αντιγραφή και εγκατάσταση των απαραίτητων *tpackages* στους υπολογιστικούς κόμβους.

```
>> root@master:~# pdcp -R ssh -w node[1-4] \
> ~/torque-4.2.2/torque-package-clients-linux-x86_64.sh ~/
>> root@master:~# pdcp -R ssh -w node[1-4] \
> ~/torque-4.2.2/torque-package-mom-linux-x86_64.sh ~/
>> root@master:~# pdsh -R ssh -w node[1-4] \
> ./torque-package-clients-linux-x86_64.sh --install | dshbak
>> root@master:~# pdsh -R ssh -w node[1-4] \
> ./torque-package-mom-linux-x86_64.sh --install | dshbak
```

Το επόμενο βήμα ήταν η αρχικοποίηση και διαμόρφωση του Torque στον master node.

```
>> root@master:~/torque-4.2.2# ldconfig
>> root@master:~/torque-4.2.2# ./torque.setup root
```

Μετά περάσαμε στον προσδιορισμό των υπολογιστικών κόμβων. Αυτό γίνεται μέσω της διαμόρφωσης του αρχείου *TORQUE_HOME/server_priv/nodes*.

```
>> root@master:~# vi /var/spool/torque/server_priv/nodes
```

Σχήμα 8.0.2 Τροποποίηση του αρχείου server_priv/nodes

Τον αριθμό των επεξεργαστών του κάθε κόμβου τους παίρνει αυτόματα θέτοντας την μεταβλητή *auto_node_np = True* μέσω του *qmgr*.

```
>> root@master:~/torque-4.2.2# qmgr -c \
> 'set server auto_node_np = True'
```

Ακολούθησε η διαμόρφωση του *daemon trqauthd* που επιτρέπει τις συνδέσεις των χρηστών στο *pbs_server*. Ο Torque επικοινωνεί με τον *trqauthd* στην port 15005. Είναι πολυνηματικός και μπορεί να χειριστεί μεγάλο όγκο ταυτόχρονων αιτήσεων.

```
>> root@master:~# cp /
> ~/torque-4.2.2/contrib/init.d/debian.trqauthd /etc/init.d/
>> root@master:~# cd /etc/init.d
>> root@master:/etc/init.d# mv debian.trqauthd trqauthd
```

Τελειώνοντας, πρέπει να εκκινήσουμε το *pbs_mom* στους υπολογιστικούς κόμβους και μετά να επανεκκινήσουμε τον *pbs_server* στον master node.

```
>> root@master:~# pdsh -R ssh -w node[1-4] ldconfig | dshbak
>> root@master:~# pdsh -R ssh -w node[1-4] pbs_mom | dshbak
>> root@master:~# qterm
>> root@master:~# pbs_server
```

Με την εντολή *pbsnodes* μπορούμε να δούμε ότι όλα έγιναν σωστά και ο Torque βλέπει τους υπολογιστικούς κόμβους.

```
>> root@master:/var/spool/torque/server_priv# pbsnodes
```

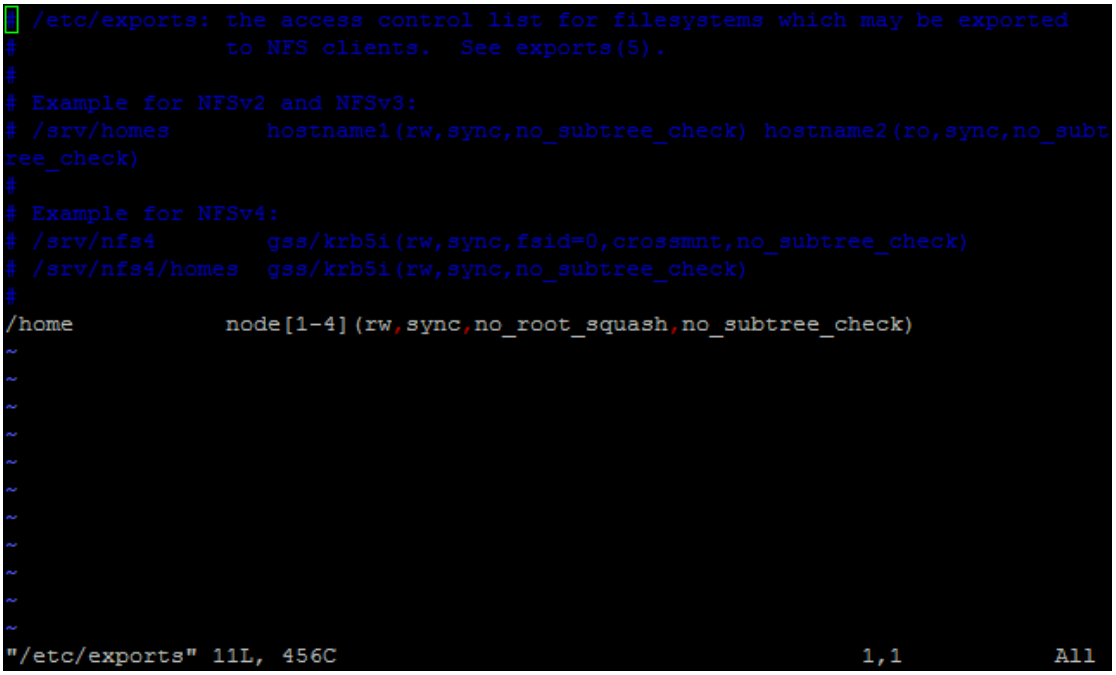
Αναλυτικές πληροφορίες για την εγκατάσταση, διαμόρφωση και χρήση του Torque περιέχονται στο [35].

Στην συνέχεια έπρεπε να εγκαταστήσουμε ένα Network File System (NFS) για την διευκόλυνση της επικοινωνίας μεταξύ του master node και των υπολογιστικών κόμβων. Αυτό είναι απαραίτητο γιατί οι υπολογιστικοί κόμβοι πρέπει να έχουν πρόσβαση και στους φακέλους που είναι το εκτελέσιμο και τα αρχεία εισόδου, αλλά και στον φάκελο που θα αποθηκεύσει τα αρχεία εξόδου και σφαλμάτων.

Master Node

Αρχικά κατεβάσαμε τα απαραίτητα αρχεία και κάναμε τις απαραίτητες τροποποιήσεις στον master node. Ειδικότερα τροποποιήσαμε το αρχείο */etc/exports*, διευκρινίζοντας τους κόμβους που θα μοιράζονται το NFS και τις ρυθμίσεις πρόσβασης στον μοιραζόμενο φάκελο. Με την εντολή *exportfs –a* κάναμε εξαγωγή τον μοιραζόμενο φάκελο στους υπολογιστικούς κόμβους.

```
>> root@master:~# apt-get install nfs-kernel-server portmap
>> root@master:~# vi /etc/exports
```



Σχήμα 8.0.3 Τροποποίηση αρχείου /etc/exports

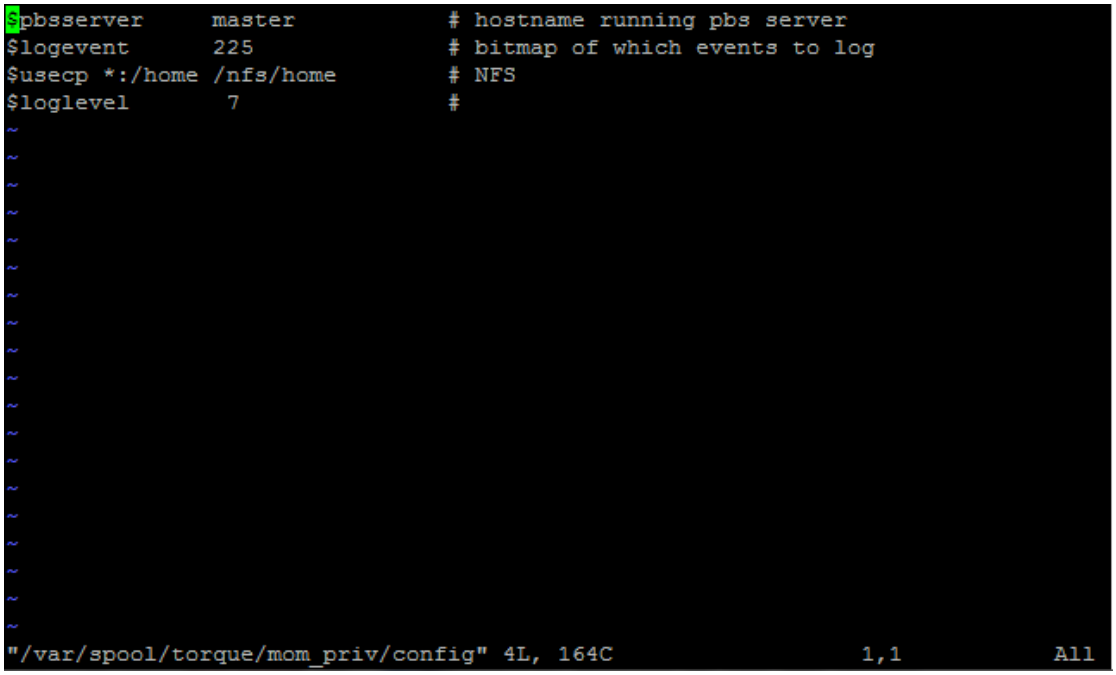
```
>> root@master:~# exportfs –a
```

Με την τροποποίηση αυτή στο */etc/exports* δηλώνουμε ότι ο φάκελος */home* του master node θα είναι μοιραζόμενος με τους κόμβους 1-4 με τις εξής ρυθμίσεις:

- **rw:** Αυτή η επιλογή επιτρέπει στους clients να μπορούν να γράψουν και να διαβάσουν μέσα στον μοιραζόμενο φάκελο.
- **sync:** Η ρύθμιση *sync* επιβεβαιώνει τις αιτήσεις στον μοιραζόμενο φάκελο μόνο όταν οι αλλαγές έχουν ολοκληρωθεί.
- **no_subtree_check:** Αυτή η ρύθμιση αποτρέπει έλεγχο των υποδέντρων. Όταν ένας μοιραζόμενος φάκελος είναι υποφάκελος ενός μεγαλύτερου συστήματος αρχείων, το nfs ελέγχει κάθε φάκελο πάνω από αυτόν, για να επιβεβαιώσει δικαιώματα και λεπτομέρειες. Η απενεργοποίηση του *subtree checking* μπορεί να αυξήσει την αξιοπιστία αλλά μειώνει την ασφάλεια.
- **no_root_squash:** Αυτή η ρύθμιση επιτρέπει στον *root* να συνδεθεί στον καθορισμένο κατάλογο.

Τέλος, θα πρέπει να τροποποιήσουμε και το *TORQUE_HOME/mom_priv/config* για να ενεργοποιήσουμε την ανταλλαγή αρχείων με απλό copy paste, να το στείλουμε στους υπολογιστικούς κόμβους με την εντολή *pdcp* και να κάνουμε επανεκκίνηση στο *mom_priv* στους υπολογιστικούς κόμβους.

```
>> root@master:~# vi /var/spool/torque/mom_priv/config
```



Σχήμα 8.0.4 Τροποποίηση αρχείου mom_priv/config

```
>> root@master:~# pdcp -R ssh -w node[1-4] /
> /var/spool/torque/mom_priv/config /var/spool/torque/mom_priv/config
>> root@master:~# pdsh -R ssh -w node[1-4] pkill pbs_mom | dshbak
>> root@master:~# pdsh -R ssh -w node[1-4] pbs_mom | dshbak
```

Αναλυτικά οι γραμμές που προσθέσαμε στο *config* δηλώνουν:

- **$pbsserver master:** Καθορίζεται ο κόμβος master ως ο κόμβος όπου θα εκτελείται ο *pbs_server*.
- **$logevent 225:** Καθορισμός των γεγονότων που θα καταγράφονται στα log files (bitmap).
- **$usecp *:/home /nfs/home:** Θα γίνεται copy-paste των αρχείων μεταξύ των φακέλων */home* του master και */nfs/home* των υπολογιστικών κόμβων.
- **$loglevel 7:** Θα καταγράφονται γεγονότα στα log files που έχουν βαρύτητα μέχρι και 7.

Υπολογιστικοί κόμβοι

Στην πλευρά των υπολογιστικών κόμβων θα πρέπει να κατεβάσουμε επίσης τα απαραίτητα αρχεία. Στην συνέχεια φτιάχνουμε τον φάκελο */nfs/home*, που θα είναι ο φάκελος που θα βλέπει στον φάκελο */home* του *master*. Τέλος, κάνουμε *mount* τον φάκελο */home* του master. Με την εντολή *df –h* μπορούμε να δούμε αν το mount πέτυχε και οι υπολογιστικοί κόμβοι βλέπουν τον φάκελο */home* του *master*.

```
>> root@nodes:~# apt-get install nfs-common portmap
>> root@master:~# pdsh -R ssh -w node[1-4] \
> mkdir -p /nfs/home | dshbak
>> root@master:~# pdsh -R ssh -w node[1-4] \
> mount master:/home /nfs/home | dshbak
>> root@master:~# pdsh -R ssh -w node[1-4] df -h | dshbak
```

```
root@master:~# pdsh -R ssh -w node[1-3] df -h | dshbak
----------------
node1
----------------
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1       2.0G  895M 1011M  47% /
udev            242M  8.0K  242M   1% /dev
tmpfs            99M  232K   99M   1% /run
none            5.0M     0  5.0M   0% /run/lock
none            246M     0  246M   0% /run/shm
/dev/vda2       197G  1.1G  186G   1% /opt
master:/home    2.0G  1.3G  594M  69% /nfs/home
----------------
node2
----------------
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1       2.0G  895M 1011M  47% /
udev            242M  8.0K  242M   1% /dev
tmpfs            99M  232K   99M   1% /run
none            5.0M     0  5.0M   0% /run/lock
none            246M     0  246M   0% /run/shm
/dev/vda2       197G  1.1G  186G   1% /opt
master:/home    2.0G  1.3G  594M  69% /nfs/home
```

Σχήμα 8.0.5 Output εντολής df -h

Τέλος, για την πραγματοποίηση των πειραμάτων ήταν απαραίτητη η εγκατάσταση του *MPI* σε όλους τους κόμβους, ώστε να μπορούμε να τρέχουμε εργασίες παράλληλα σε περισσότερους από έναν κόμβους. Κατεβάσαμε την τελευταία έκδοση του *MPI* από το επίσημο site *http://www.mpich.org/*, το αποσυμπίεσαμε, φτιάξαμε έναν καινούριο φάκελο *mpich-install* στον μοιραζόμενο φάκελο */home* του master, το κάναμε εγκατάσταση εκεί και κάναμε *export* το *PATH* των εκτελέσιμων.

```
>> root@master:~# gunzip mpich-3.0.3.tar.gz
>> root@master:~# tar xf mpich-3.0.3.tar
>> root@master:~# cd mpich-3.0.3
>> root@master:~/mpich-3.0.3#  mkdir /home/mpich-install/
>> root@master:~/mpich-3.0.3# ./configure --disable-f77\ >
> --disable-fc --prefix=/home/mpich-install 2>&1 | tee c.txt
>> root@master:~/mpich-3.0.3#  make 2>&1 | tee m.txt
>> root@master:~/mpich-3.0.3# export \
> PATH=$PATH:/home/mpich-install/bin
```

Παράρτημα ΙΙΙ

Σε αυτό το παράρτημα περιγράφουμε την διαδικασία κατασκευής του κώδικα που χρησιμοποιήσαμε στα πειράματα.

Εισαγωγή πεδίου MemIntensity

Το 1ο βήμα που πρέπει να κάνουμε είναι η εισαγωγή του πεδίου *MemIntensity* στο *struct mqos_t*. Έτσι, στην 1041 του *msched.h* τροποποιούμε το struct *mqos_t* ως εξής:

```
typedef struct {
  char          Name[MAX_MNAME];
  int           Index;
  long          MTime;     /* time record was last updated */
  long          QTWeight;
  long          XFWeight;
  long          QTTarget;
  double        XFTarget;  /* Targeted XFactor for QOS */
  int           MemIntensity;
  unsigned long Flags;     /* Flags/Exemptions Associated with QOS */
  char          ResName[16][MAX_MNAME];
  mfs_t         F;
  mcredl_t      L;
  must_t        Stat;      /* Usage Statistics */
  } mqos_t;
```

Με έντονα γράμματα σημειώνουμε την αλλαγή που κάναμε. Επίσης, στο ίδιο αρχείο προσθέτουμε την default τιμή του πεδίου *MemIntensity*, στη σειρά 398, όπου ορίζονται οι default τιμές για τα πεδία του *mqos_t*.

```
#define DEFAULT_QOSMEMINTENSITY     0
```

Στην συνέχεια θα προσθέσουμε την μεταβλητή *MemIntCapacity* στο *struct mnode_t* που θα δηλώνει το άθροισμα των τιμών *MemIntensity* των εργασιών που εκτελούνται ή έχουν κάνει κράτηση σε έναν κόμβο μια συγκεκριμένη χρονική στιγμή. Άρα, στην γραμμή 1770 όπου γίνεται ο ορισμός της παραπάνω δομής προσθέτουμε την γραμμή:

```
int     MemIntCapacity;
```

Για να προσθέσουμε το όνομα του καινούριο αυτού πεδίου ώστε να μπορούμε να το εμφανίζουμε στην οθόνη, πρέπει να τροποποιήσουμε τον πίνακα *MQOSAttr* που ορίζεται στο αρχείο *MConst.c*. Οπότε, στην σειρά 670 που ορίζεται ο πίνακας κάνουμε την εξής τροποποίηση:

```
const char *MQOSAttr[] = {
  NONE,
  "PRIORITY",
  "MAXJOB",
  "MAXPROC",
  "MAXNODE",
  "XFWEIGHT",
  "QTWEIGHT",
  "XFTARGET",
  "QTTARGET",
```

```
  "QFLAGS",
  "FSTARGET",
  "MEMINTENSITY",
  NULL };
```

Στην συνέχεια πρέπει να προσθέσουμε αυτό το πεδίο και στον πίνακα που περιέχει τα configuration data και ορίζεται στο ίδιο αρχείο στην γραμμή 1389. Άρα, στον πίνακα *MCfg* προσθέτουμε την γραμμή:

```
 { "QOSMEMINTENSITY", pQOSMemIntensity, mdfInt, mxoQOS,NULL },
```

Την παράμετρο *pQOSMemIntensity* την δηλώνουμε στο enum που ορίζεται στο αρχείο *msched-common.h* στην γραμμή 519 και δηλώνει διάφορες πολιτικές. Οπότε, απλά προσθέτουμε στο enum στην γραμμή 813 που γίνεται η αναφορά στα πεδία της δομής *mqos_t* την έντονη γραμμή:

```
pQOSName,
pQOSXFWeight,
pQOSTargetXF,
pQOSQTWeight,
pQOSTargetQT,
pQOSMemIntensity,
pQOSPriority,
pQOSFlags,
```

Τέλος, προσθέτουμε αυτό το πεδίο και στο enum που περιέχει τις ιδιότητες του QoS. Έτσι, στο αρχείο *moab.h* στην γραμμή 928 κάνουμε την εξής αλλαγή:

```
enum MQOSAttrType {
  mqaNONE = 0,
  mqaPriority,
  mqaMaxJob,
  mqaMaxProc,
  mqaMaxNode,
  mqaXFWeight,
  mqaQTWeight,
  mqaXFTarget,
  mqaQTTarget,
  mqaFlags,
  mqaFSTarget,
  mqaMemIntensity };
```

Εισαγωγή σημαίας MEMINTENSIVE

Αρχικά προσθέτουμε την σημαία *MEMINTENSIVE* στον πίνακα *MQOSFlags* που ορίζεται στο αρχείο *MConst.c* στην γραμμή 990:

```
const char *MQOSFlags[] = {
  "IGNJOBPERUSER",
  "IGNPROCPERUSER",
  "IGNNODEPERUSER",
  "IGNPSPERUSER",
  "IGNJOBQUEUEDPERUSER",
  "IGNSYSMAXPROC",
  "IGNSYSMAXTIME",
```

```
  "IGNSYSMAXPS",
  "IGNSRMAXTIME",
  "IGNUSER",
  "IGNSYSTEM",
  "IGNALL",
  "PREEMPTOR",
  "PREEMPTEE",
  "DEDICATED",
  "RESERVEALWAYS",
  "USERESERVED",
  "NOBF",
  "NORESERVATION",
  "RESTARTPREEMPT",
  "NTR",
  "RUNNOW",
  "PREEMPTSPV",
  "IGNHOSTLIST",
  "MEMINTENSIVE",
  NULL };
```

Προσθέτουμε επίσης στο enum *MQOSFlagEnum* που ορίζεται στο αρχείο *moab.h* την *qos flag mqfmemintensive*.

```
enum MQOSFlagEnum {
  mqfignJobPerUser = 0,
  mqfignProcPerUser,
  mqfignNodePerUser,
  mqfignPSPerUser,
  mqfignJobQueuedPerUser,
  mqfignMaxProc,
  mqfignMaxTime,
  mqfignMaxPS,
  mqfignSRMaxTime,
  mqfignUser,
  mqfignSystem,
  mqfignAll,
  mqfpreemptor,
  mqfpreemptee,
  mqfdedicated,
  mqfreserved,
  mqfusereservation,
  mqfnobf,
  mqfnoreservation,
  mqfrestartpreempt,
  mqfNTR,
  mqfRunNow,
  mqfPreemptSPV,
  mqfignHostList,
  mqfmemintensive  };
```

Προσθέτουμε και την *job flag mjfmemintensive* στο enum που ορίζεται στην γραμμή 1155 του αρχείου *moab.h*.

```
enum {
  mjfNONE = 0,
  mjfAllocLocal,
  mjfBackfill,
  mjfSpan,
  mjfAdvReservation,
  mjfNoQueue,
```

```
  mjfHostList,
  mjfResMap,
  mjfSharedResource,
  mjfByName,
  mjfBestEffort,
  mjfRestartable,
  mjfPreemptee,
  mjfPreemptor,
  mjfNASingleJob,
  mjfPreload,
  mjfRemote,
  mjfNASingleTask,
  mjfSPViolation,
  mjfIgnNodePolicies,
  mjfNoRMStart,
  mjfGlobalQueue,
  mjfIsExiting,
  mjfmemintensive };
```

Τέλος, προσθέτουμε και μια καινούρια πολιτική κατανομής κόμβων τροποποιώντας το enum *MNodeAccessPolicyEnum* που ορίζεται στην γραμμή 391 του *moab.h*.

```
enum MNodeAccessPolicyEnum {
  mnacNONE = 0,   /* DEFAULT:  shared */
  mnacShared,     /* any combination of workload allowed */
  mnacSingleJob,  /* peer tasks from a single job may utilize node */
  mnacSingleTask, /* only a single task may utilize node */
  mnacSingleUser, /* any number of tasks from the same user may utilize
node */
  mnacMemIntensive
  };
```

Τροποποίηση αρχείου MQOS.c

Στην συνάρτηση *MQOSInitialize*, που αρχικοποιεί τα QoS, προσθέτουμε στην γραμμή 401 που ορίζονται οι default τιμές του QoS την default του πεδίου *MemIntensity*:

```
Q->MemIntensity = DEFAULT_QOSMEMINTENSITY;
```

Στην συνάρτηση *MQOSProcessConfig*, όπου λαμβάνονται οι τιμές του QoS από το maui.cfg, προσθέτουμε και την περίπτωση *mqaMemIntensity*. Έτσι, στην γραμμή 512 προσθέτουμε το εξής:

```
case mqaMemIntensity:

        Q->MemIntensity = strtod(ValLine,NULL);

        break;
```

Στην συνάρτηση *MQOSShow*, όπου βλέπουμε τα στοιχεία των QoS με την εντολή *diagnose –Q*, τροποποιούμε την γραμμή 1086 έτσι ώστε να εμφανίζεται και η τιμή του πεδίου *MemIntensity*. Αυξάνουμε και το σύνολο των γραμμάτων που μπορούν να περιέχουν τα πεδία *QFlags* και *JobFlags*, ώστε να μην χαλάει η στοίχιση με την εισαγωγή της σημαίας *MemIntensive*:

```
MUSNPrintF(&BPtr,&BSpace,"%-20s%c %8s %8s %8s %8s %8s %12s %12s %12s
%s\n\n",
    "Name",
    '*',
    "Priority",
    "QTWeight",
    "QTTarget",
    "XFWeight",
    "XFTarget",
    "MemIntensity",
    "QFlags",
    "JobFlags",
    "Limits");
```

Τροποποιήσαμε και την γραμμή 1102 που ελέγχει αν περιέχουν τιμές τα πεδία του QoS:

```
if ((Q->Name[0] == '\0') &&
        (Q->Flags == 0) &&
        (Q->F.Priority == 0) &&
        (Q->QTWeight == 0) &&
        (Q->XFWeight == 0) &&
        (Q->XFTarget == 0.0) &&
        (Q->QTTarget == 0) &&
        (Q->MemIntensity == 0))
      {
      continue;
      }
```

Στην συνέχεια κάναμε αντίστοιχες τροποποιήσεις και στο σημείο που εμφανίζονται οι τιμές των πεδίων στην γραμμή 1134:

```
MUSNPrintF(&BPtr,&BSpace,"%-20s%c %8ld %8ld %8ld %8ld %8.2lf %12d %12s
%12s %s\n",
      (Q->Name[0] != '\0') ? Q->Name : NONE,
      QALChar,
      Q->F.Priority,
      Q->QTWeight,
      Q->QTTarget,
      Q->XFWeight,
      Q->XFTarget,
      Q->MemIntensity,
      (Line[0] != '\0') ? Line : NONE,
      JFlags,
      ((lptr == NULL) || (*lptr == '\0')) ? NONE : lptr);
```

Στην συνέχεια τροποποιήσαμε και την συνάρτηση *MQOSConfigShow*, που δείχνει τα στοιχεία του αρχείου maui.cfg όταν πραγματοποιηθεί η εντολή *showconfig*, προσθέτοντας στην γραμμή 1292:

```
 if ((Q->MemIntensity != 0) || (VFlag || (PIndex == -1)
      || (PIndex == pQOSMemIntensity)))
    strcat(Buffer,MUShowIArray(MParam[pQOSMemIntensity],Q->Index,
            Q->MemIntensity));
```

Στην συνάρτηση *MQOSProcessOConfig* στην γραμμή 1456 προσθέτουμε τα παρακάτω:

```
case pQOSMemIntensity:

      Q->MemIntensity = IVal;

      break;
```

Στην συνάρτηση *MQOSFlagsFromString*, που διαβάζει τιμές από τα strings του αρχείου maui.cfg και ανανεώνει τις σημαίες, προσθέτουμε στην γραμμή 1665 την περίπτωση της *mqfmemintensive* πολιτικής:

```
case mqfmemintensive:

      Q->Flags |= (1 << vindex);

      break;
```

Τέλος, στην συνάρτηση *MQOSFlagsToString* που κάνει το αντίστροφο προσθέτουμε στην γραμμή 1879 το πεδίο *mqfmemintensive* στον πίνακα FList:

```
int FList[] = {
    mqfpreemptor,
    mqfpreemptee,
    mqfdedicated,
    mqfreserved,
    mqfRunNow,
    mqfnobf,
    mqfPreemptSPV,
    mqfignHostList,
    mqfmemintensive,
    -1 };
```

Τροποποίηση αρχείου MJob.c

Στην γραμμή 1308 του αρχείου ενεργοποιούμε την σημαία *mjfmemintensive* αν είναι ενεργοποιημένη η σημαία *mqfmemintensive*. Αυτό γίνεται προσθέτοντας τις παρακάτω γραμμές κώδικα:

```
if (RQ != NULL)
      {
      if (J->Cred.Q->Flags & (1 << mqfdedicated))
        {
        J->Flags |= (1 << mjfNASingleJob);
        }
      else if (J->Cred.Q->Flags & (1 << mqfmemintensive))
        {
        J->Flags |= (1 << mjfmemintensive);
        }
      }
```

Στην συνέχεια ενεργοποιούμε και την αντίστοιχη πολιτική παραχώρησης κόμβων τροποποιώντας τον κώδικα στην γραμμή 1328:

```
 if (RQ != NULL)
    {
    if (J->Flags & (1 << mjfNASingleJob))
      {
      RQ->NAccessPolicy = mnacSingleJob;
      }
    else if (J->Flags & (1 << mjfNASingleTask))
      {
      RQ->NAccessPolicy = mnacSingleTask;
      }
    else if (J->Flags & (1 << mjfmemintensive))
      {
      RQ->NAccessPolicy = mnacMemIntensive;
      }
    else
      {
      RQ->NAccessPolicy = MSched.DefaultNAccessPolicy;
      }
    }    /* END if (RQ != NULL) */
```

Στην συνάρτηση *MJobProcessExtensionString*, προσθέτουμε και την περίπτωση της *mjfmemintensive*. Αυτό το πραγματοποιήσαμε προσθέτοντας στην γραμμή 3575 τα παρακάτω:

```
case mjfmemintensive:

  RQ->NAccessPolicy = mnacMemIntensive;
  J->SpecFlags |= (1 << mjfmemintensive);
  break;
```

Επίσης, στην γραμμή 3772 προσθέτουμε και τον έλεγχο της πολιτικής *mnacMemIntensive*:

```
case mxaNAccessPolicy:

        /* get node access mode */

        RQ->NAccessPolicy = MUGetIndex(Value,MNAccessPolicy,FALSE,
                                       MSched.DefaultNAccessPolicy);

        if (RQ->NAccessPolicy == mnacSingleJob)
          {
          J->SpecFlags |= (1 << mjfNASingleJob);
          }
        else if (RQ->NAccessPolicy == mnacSingleTask)
          {
          J->SpecFlags |= (1 << mjfNASingleTask);
          }
        else if (RQ->NAccessPolicy == mnacMemIntensive)
          {
          J->SpecFlags |= (1 << mjfmemintensive);
          }

        DBG(5,fCONFIG) DPrint("INFO:     job node access policy set to
%s\n", MNAccessPolicy[RQ->NAccessPolicy]);

        break;
```

Στην συνάρτηση *MJobGetSNRange* που ελέγχει την διαθεσιμότητα των πόρων των κόμβων προσθέτουμε την τοπική μεταβλητή *MemIntOccupied* που δείχνει αν ο κόμβος μπορεί να δεχτεί επιπλέον memory intensive εργασίες. Αν η τιμή της είναι 0 σημαίνει ότι ο κόμβος μπορεί να δεχτεί και άλλες memory intensive εργασίες, ενώ σε αντίθετη περίπτωση όχι. Άρα, στην γραμμή 9966 προσθέτουμε τα εξής:

```
int    MemIntOccupied;

if (N->MemIntCapacity >= 10)
  MemIntOccupied = TRUE;
else
  MemIntOccupied = FALSE;
```

Στην συνέχεια, στην γραμμή 10410 αφαιρούμε τους διαθέσιμους πόρους στην αρχή μιας κράτησης και στην συνέχεια τους προσθέτουμε στο τέλος της στην γραμμή 10473. Έτσι, κάνουμε τις εξής τροποποιήσεις στον κώδικα:

Γραμμή 10410

```
ResourcesRemoved = TRUE;

if ((J->Req[0]->NAccessPolicy == mnacSingleUser) &&
            (tmpJ != NULL) &&
            (tmpJ->Cred.U != NULL) &&
            (J->Cred.U != NULL) &&
            (tmpJ->Cred.U != J->Cred.U) &&
            (strcmp(J->Cred.U->Name,"[ALL]")))
  {
  /* user dedicated resources removed */

  MCResRemove(&AvlRes,&N->CRes,&N->CRes,RC,FALSE);

  if (UseDed == TRUE)
    MCResAdd(&DedRes,&N->CRes,&N->CRes,RC,FALSE);
  }
else if ((J->Req[0]->NAccessPolicy != mnacMemIntensive))
  {
  MCResRemove(&AvlRes,&N->CRes,BR,RC,FALSE);

  if (UseDed == TRUE)
    MCResAdd(&DedRes,&N->CRes,BR,RC,FALSE);
  }

DBG(7,fSCHED)
  {
  DPrint("INFO:     removed resources: %s (x%d)\n",
          MUCResToString(BR,0,0,NULL), RC);

  DPrint("INFO:     resulting resources: %s\n",
            MUCResToString(&AvlRes,0,0,NULL));
  }

if (N->MemIntCapacity + J->Cred.Q->MemIntensity > 10)
  {
  MemIntOccupied = TRUE;
  DBG(7,fSCHED)   DPrint("BILL:          Because N->MemIntCapacity + J-
>Cred.Q->MemIntensity = %d MemIntOccupied became TRUE\n",
                         N->MemIntCapacity + J->Cred.Q->MemIntensity);
```

```
  }
if (BRes != NULL)
  strcpy(BRes,N->R[N->RE[eindex].Index]->Name);
```

Γραμμή 10473

```
ResourcesAdded = TRUE;

if ((J->Req[0]->NAccessPolicy == mnacSingleUser) &&
    (tmpJ != NULL) &&
    (tmpJ->Cred.U != J->Cred.U))
  {
  /* user dedicated resources added */

  MCResAdd(&AvlRes,&N->CRes,&N->CRes,RC,FALSE);

  if (UseDed == TRUE)
    MCResRemove(&DedRes,&N->CRes,&N->CRes,RC,FALSE);
  }
else if ((J->Req[0]->NAccessPolicy != mnacMemIntensive))
  {
  MCResAdd(&AvlRes,&N->CRes,BR,RC,FALSE);

  if (UseDed == TRUE)
    MCResRemove(&DedRes,&N->CRes,BR,RC,FALSE);
  }

DBG(7,fSCHED)
  {
  DPrint("INFO:     added resources: %s (x%d)\n",
         MUCResToString(BR,0,0,NULL), RC);

  DPrint("INFO:     resulting resources: %s\n",
         MUCResToString(&AvlRes,0,0,NULL));
  }

if (N->MemIntCapacity + J->Cred.Q->MemIntensity < 10)
  {
  MemIntOccupied = FALSE;
  DBG(7,fSCHED)   DPrint("BILL:          Because N->MemIntCapacity + J-
>Cred.Q->MemIntensity = %d MemIntOccupied became FALSE\n",
                         N->MemIntCapacity + J->Cred.Q->MemIntensity);
  }
```

Τέλος, προσθέτουμε τις κατάλληλες συνθήκες που θα καταστήσουν τον κόμβο μη διαθέσιμο όταν η μεταβλητή *MemIntCapacity* είναι μεγαλύτερη ή ίση του 10. Αυτό γίνεται τροποποιώντας τις παρακάτω γραμμές:

Γραμμή 10277

```
if ((MUCResIsNeg(&AvlRes) == SUCCESS) ||
    (PBlock == TRUE) ||
    ((J->Flags & (1 << mjfAdvReservation)) &&
     (IResCount <= 0)) ||
    (MemIntOccupied == TRUE))
```

Γραμμή 10534

```
if (MUCResIsNeg(&AvlRes) == SUCCESS || (MemIntOccupied == TRUE))
```

Γραμμή 10783

```
if (((MUCResIsNeg(&AvlRes) == FAILURE) &&
     (!(J->Flags & (1 << mjfAdvReservation)) ||
     (IResCount > 0)) &&
    (PBlock == FALSE)) &&
    (MemIntOccupied == FALSE))
```

Τροποποίηση αρχείου MNode.c

Στην συνάρτηση *MNodeInitialize* αρχικοποιούμε την μεταβλητή *MemIntCapacity* στο 0. Αυτό γίνεται προσθέτοντας στην γραμμή 4496 την παρακάτω γραμμή κώδικα:

```
N->MemIntCapacity = 0;
```

Τροποποίηση αρχείου MSched.c

Στην συνάρτηση *MJobSelectMNL* πραγματοποιούμε την μετατροπή, ώστε σε μια MPI εργασία με ενεργοποιημένη την σημαία *MemIntensive* να μπορούν να παραχωρηθούν επεξεργαστές από διαφορετικούς κόμβους. Οπότε στην γραμμή 1487 προσθέτουμε τα εξής:

```
if (J->Cred.Q->Flags & (1 << mqfmemintensive))
  {
  DBG(7,fSCHED) DPrint("BILL:      %s job %s is memory
                        intensive\n",FName,J->Name);

  for (index = 0;index < MAX_MNODE; index++)
    {
    if (FNL != NULL)
      {
      if (FNL[index].N == NULL)
          break;
      }
      else
        {
        N = MNode[index];
        if ((N == NULL) || (N->Name[0] == '\0'))
          break;

        }

    }

    DBG(7,fSCHED) DPrint("BILL:      %d feasible nodes\n",index);
    int num_nodes = index, new_jobnodes;
    long memint = J->Cred.Q->MemIntensity;
    long new_memint = memint;
```

```
    for (index = 2; index <= num_nodes && new_memint >10; index++)
      {
      new_memint = memint / index;
      if (memint % index != 0)
        new_memint++;
      }

    new_jobnodes = index - 1;

    if (new_jobnodes > J->Req[0]->NodeCount)
      {
      if (J->Req[0]->TaskCount % new_jobnodes == 0)
        J->Req[0]->TasksPerNode = J->Req[0]->TaskCount / new_jobnodes;
      else
        J->Req[0]->TasksPerNode = (J->Req[0]->TaskCount / new_jobnodes)
+ 1;
      }

    J->Cred.Q->MemIntensity = new_memint;
    J->Req[0]->NodeCount = new_jobnodes;
    DBG(7,fSCHED) DPrint("BILL:      J->Req[0]  NodeCount = %d\n,
                          TaskCount = %d, TasksPerNode = %d,
                          MemIntensity = %ld\n",
                          J->Req[0]->NodeCount,J->Req[0]->TaskCount,
                          J->Req[0]->TasksPerNode,
                          J->Cred.Q->MemIntensity);

  }
```

Επίσης, στην συνάρτηση *MJobDistributeTasks* αυξάνουμε την τιμή *MemIntCapacity* των κόμβων που παραχωρούνται στην εργασία. Για αυτό προσθέτουμε στην γραμμή 6772 τα εξής:

```
tmpNodeList[nindex].N->MemIntCapacity += J->Cred.Q->MemIntensity;
DBG(4,fSCHED) DPrint("BILL:    Node %s memint capacity increased by
                      %d to total %d\n",
                      tmpNodeList[nindex].N->Name,
                      J->Cred.Q->MemIntensity,
                      tmpNodeList[nindex].N->MemIntCapacity);
```

Τροποποίηση αρχείου MRes.c

Τέλος, στην συνάρτηση *MResDeallocateResources* που περιέχεται σε αυτό το αρχείο μειώνουμε την τιμή *MemIntCapacity* των κόμβων που είχαν παραχωρηθεί στην εργασία, όταν αυτή ολοκληρωθεί. Έτσι, στην γραμμή 5322 προσθέτουμε τα παρακάτω:

```
if (((mjob_t *) R->J)->State == mjsCompleted)
  {
  N->MemIntCapacity -= ((mjob_t *) R->J)->Cred.Q->MemIntensity;
  DBG(4,fSCHED) DPrint("BILL:    Node %s memint capacity decreased by
                        %d to total %d\n",
                        N->Name,
                        ((mjob_t *) R->J)->Cred.Q->MemIntensity,
                        N->MemIntCapacity);
  }
```