ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ
Τμήμα Πληροφορικής

Μεταπτυχιακή Διατριβή

ΥΛΟΠΟΙΗΣΗ ΚΑΙ ΠΕΙΡΑΜΑΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ ΕΡΓΑΣΙΟΚΕΝΤΡΙΚΩΝ ΚΑΤΑΝΕΜΗΜΕΝΩΝ ΑΛΓΟΡΙΘΜΩΝ ΣΤΗΝ ΠΛΑΤΦΟΡΜΑ YALPS

Ανδρέας Ανδρέου

Μάιος, 2012

ΠΕΡΙΛΗΨΗ

Ένα από τα βασικότερα προβλήματα του κατανεμημένου υπολογισμού είναι συνεργασιακή εκτέλεση μεγάλων υπολογιστικών συνόλων εργασιών από κατανεμημένες διεργασίες. Τέτοιου είδους εκτελέσεις, για να είναι αποτελεσματικές, πρέπει το σύστημα να είναι σχεδιασμένο έτσι ώστε να μπορεί να αντιμετωπίσει δυναμικές διαταραχές του μέσου επικοινωνίας ή πιθανόν σφάλματα των διεργασιών. Για το σκοπό αυτό, πολλές μελέτες έχουν αφιερωθεί τις τελευταίες δύο δεκαετίες στην ανάπτυξη αλγοριθμικών λύσεων με ανοχή στα σφάλματα για ποικίλες εκδοχές τέτοιων προβλημάτων συνεργασίας και στην ανάπτυξη κατανεμημένων συστημάτων και εφαρμογών. Πιο πρόσφατη είναι η εργασία των Γεωργίου και Kowalski με τίτλο *«Performing Dynamically Injected Tasks on Processes Prone to Crashes and Restarts»*, με την οποία θα επικεντρωθούμε στη διατριβή αυτή.

Σε αυτή την εργασία περιγράφονται αλγοριθμικές λύσεις οι οποίες επιλύουν μια εκδοχή του προβλήματος εκτέλεσης κατανεμημένων εργασιών όπου οι εργασίες εισάγονται δυναμικά στο σύστημα και οι διεργασίες υπόκεινται σε καταρρεύσεις και επανεκκινήσεις. Η παρούσα διατριβή στοχεύει στην πειραματική αξιολόγηση των αλγορίθμων, που αναφέρονται στην εργασία, η οποία θα υποδείξει κατά πόσο αυτοί οι αλγόριθμοι είναι αποδοτικοί και στην πράξη.

Κατόπιν εις βάθους μελέτης πραγματοποιήθηκε η υλοποίησή των αλγορίθμων σε περιβάλλον Java με τη χρήση της βιβλιοθήκης YALPS και ακολούθησε η εκτέλεσή τους σε περιβάλλον προσομοίωσης με διάφορες παραμέτρους για την εξαγωγή αποτελεσμάτων, τα οποία χρησιμοποιήθηκαν για την πειραματική αξιολόγηση. Η προσομοίωση βοήθησε στην μελέτη της συμπεριφοράς των αλγορίθμων, κυρίως υπό συνδυασμό παραμέτρων που ήταν δύσκολο να αναλυθούν κατά την θεοτική ανάλυση (π.χ. ο χρόνος διεκπεραίωσης σε σχέση με τα σφάλματα).

ΥΛΟΠΟΙΗΣΗ ΚΑΙ ΠΕΙΡΑΜΑΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ ΕΡΓΑΣΙΟΚΕΝΤΡΙΚΩΝ ΚΑΤΑΝΕΜΗΜΕΝΩΝ ΑΛΓΟΡΙΘΜΩΝ ΣΤΗΝ ΠΛΑΤΦΟΡΜΑ YALPS

Ανδρέας Ανδρέου

Η Διατριβή αυτή

Υποβλήθηκε προς Μερική Εκπλήρωση των

Απαιτήσεων για την Απόκτηση

Τίτλου Σπουδών Master

στην Επιστήμη της Πληροφορικής

στο

Πανεπιστήμιο Κύπρου

Συστήνεται προς Αποδοχή

από το Τμήμα Πληροφορικής

Μάιος, 2012

ΣΕΛΙΔΑ ΕΓΚΡΙΣΗΣ

Διατριβή Master

ΥΛΟΠΟΙΗΣΗ ΚΑΙ ΠΕΙΡΑΜΑΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ ΕΡΓΑΣΙΟΚΕΝΤΡΙΚΩΝ ΚΑΤΑΝΕΜΗΜΕΝΩΝ ΑΛΓΟΡΙΘΜΩΝ ΣΤΗΝ ΠΛΑΤΦΟΡΜΑ YALPS

Παρουσιάστηκε από

Ανδρέα Ανδρέου

Ερευνητικός Σύμβουλος _____

Δρ. Γεωργίου Χρύσης

Μέλος Επιτροπής _____

Δρ. Ζεϊναλιπούρ Δημήτρης

Μέλος Επιτροπής _____

Δρ. Νικολάου Νικόλας

Πανεπιστήμιο Κύπρου

Μάιος, 2012

ΕΥΧΑΡΙΣΤΙΕΣ

Αρχικά θέλω να εκφράσω τις θερμές μου ευχαριστίες στον επιβλέποντα καθηγητή μου Δρ. Χρύση Γεωργίου, για τη συνεχή καθοδήγηση και βοήθεια που μου παρείχε σε όλα τα στάδια της εκπόνησης της διατριβής μου. Η συνεχής παρακολούθηση, το αμείωτο ενδιαφέρον, οι συμβουλές, οι οδηγίες και οι κατευθυντήριες γραμμές που έθεσε, συνέβαλαν καταλυτικά στην επιτυχή ολοκλήρωση.

Τέλος θέλω να ευχαριστήσω την οικογένειά μου και τους φίλους μου για την υπομονή τους αλλά και για τη στήριξη που μου πρόσφεραν όλα αυτά τα χρόνια.

Ευχαριστώ!

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΜΕ ΠΙΝΑΚΕΣ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

Κεφάλαιο 1

Εισαγωγή

1.1 Κίνητρο

Ένα από τα βασικότερα προβλήματα του κατανεμημένου υπολογισμού είναι συνεργασιακή εκτέλεση μεγάλων υπολογιστικών συνόλων εργασιών *(Tasks)* από κατανεμημένες διεργασίες *(Nodes)*. Τέτοιου είδους εκτελέσεις για να είναι αποτελεσματικές πρέπει το σύστημα να είναι σχεδιασμένο έτσι ώστε να μπορεί να αντιμετωπίσει δυναμικές διαταραχές του μέσου επικοινωνίας ή πιθανόν σφάλματα των διεργασιών. Για το σκοπό αυτό, πολλές μελέτες έχουν αφιερωθεί τις τελευταίες δύο δεκαετίες στην ανάπτυξη αλγοριθμικών λύσεων με ανοχή στα σφάλματα για ποικίλες εκδοχές τέτοιων προβλημάτων συνεργασίας (π.χ., [1] [2] [3] [4] [5] [6]) και στην ανάπτυξη κατανεμημένων συστημάτων και εφαρμογών (π.χ., [7] [8] [9] [10]). Πιο πρόσφατη είναι η εργασία των Γεωργίου και Kowalski με τίτλο *«Performing Dynamically Injected Tasks on Processes Prone to Crashes and Restarts»* [11], με την οποία θα επικεντρωθούμε στη διατριβή αυτή.

Προκειμένου να προσδιοριστεί η ανταλλαγή μεταξύ της αποδοτικότητας και ανοχή σε σφάλματα στα κατανεμημένα συστήματα, πολλές ερευνητικές εργασίες έχουν μελετήσει το γενικευμένο πρόβλημα, όπου *n* διεργασίες συνεργάζονται για την εκτέλεση *m* ανεξάρτητων εργασιών στην παρουσία σφαλμάτων (π.χ., [12] [3] [13]). Στο πρόβλημα αυτό, γνωστό ως *Do-All*, ο αριθμός των εργασιών *m* είναι προκαθορισμένος και γνωστός εκ των προτέρων σε όλες τις διεργασίες. Αν και υπάρχουν αρκετές εφαρμογές στις οποίες οι εργασίες είναι γνωστές εκ των προτέρων, στις μέρες για υπολογισμούς που πραγματοποιούνται στο Διαδίκτυο οι εργασίες συνήθως δημιουργούνται δυναμικά και οι διεργασίες μεταξύ τους κατέχουν διαφορετική γνώση για τις εν αναμονή εργασίες. Τέτοιου είδους εφαρμογές βρίσκουμε σε Υπολογιστικά Πλέγματα *(Grid Computing)* (π.χ., [8]), στο Σύννεφο *(Cloud*

Services) (π.χ., [7]), η σε συστήματα τύπου Εξυπηρετητή-Εργάτη *(Master-Worker)* (π.χ., [9] [10]). Ως εκ τούτου οι υπολογισμοί γίνονται ο κανόνας και όχι η εξαίρεση καθώς υπάρχει η ανάγκη για ανάπτυξη αποτελεσματικών και ανεκτικών σε σφάλματα αλγοριθμικών λύσεων που θα είναι επίσης σε θέση να αντιμετωπίσουν τη δυναμική εισαγωγή εργασιών.

Στο άρθρο των Γεωργίου και Kowalski [11] περιγράφονται αλγοριθμικές λύσεις οι οποίες επιλύουν μια εκδοχή του προβλήματος εκτέλεσης κατανεμημένων εργασιών όπου οι εργασίες εισάγονται δυναμικά στο σύστημα και οι διεργασίες υπόκεινται σε καταρρεύσεις και επανεκκινήσεις.

Βάσει των πιο πάνω τίθεται το ερώτημα κατά πόσο οι αλγόριθμοι μπορούν να υλοποιηθούν και αν τελικά είναι αποδοτικοί και στην πράξη.

1.2 Στόχος

Η παρούσα διατριβή στοχεύει στην πειραματική αξιολόγηση των αλγορίθμων, που αναφέρονται στην εργασία [11], η οποία θα υποδείξει κατά πόσο αυτοί οι αλγόριθμοι μπορούν να υλοποιηθούν και αν είναι αποδοτικοί και στην πράξη. Συγκεκριμένα έχουν υλοποιηθεί οι αλγόριθμοι που αναφέρονται σε περιβάλλον το οποίο επιτρέπει την πειραματική εκτέλεση σε περιβάλλον προσομοίωσης. Μέσω των εξαγόμενων αποτελεσμάτων είναι εφικτή η περαιτέρω μελέτη της απόδοσης των αλγορίθμων καθώς και η συμπεριφορά τους σε πρακτικό επίπεδο.

Η υλοποίηση των αλγόριθμων πραγματοποιήθηκε σε περιβάλλον JAVA [14] με τη χρήση της βιβλιοθήκης YALPS [15] του ινστιτούτου επιστήμης και τεχνολογίας INRIA [16] στην οποία και θα αναφερθούμε εκτενέστερα στη συνέχεια της Διατριβής.

1.3 Μεθοδολογία

Για την επίτευξη των στόχων της διατριβής αυτής η όλη διαδικασία διαχωρίστηκε σε διακριτά στάδια και ακολουθήθηκε η πιο κάτω μεθοδολογία.

Αρχικά προηγήθηκε η μελέτη του άρθρου αυτού καθώς και άλλων σχετικών άρθρων με σκοπό την ευρύτερη κατανόηση του γενικότερου προβλήματος αλλά και τη μελέτη των προσπαθειών που έχουν προηγηθεί για την επίλυσή του. Περισσότερη έμφαση δόθηκε στην κατανόηση της χρησιμότητας και την ακριβή λειτουργία των αλγοριθμικών λύσεων που προτάθηκαν έτσι ώστε να διευκρινιστούν πιθανές ασάφειες οι οποίες πιθανόν να οδηγούσαν σε σφάλματα και εκ των υστέρων σε λάθος συμπεράσματα.

Στο επόμενο στάδιο ακολούθησε η μελέτη της βιβλιοθήκης YALPS [15]. Για την εξοικείωση με τη βιβλιοθήκη αναπτύχθηκαν διάφορα απλά παραδείγματα με τη χρήση κεντρικού χρονοδρομολογητή και μη, έτσι ώστε να έρθουν στην επιφάνεια, σχετικά νωρίς, τυχόν προβλήματα τα οποία θα έπρεπε να αντιμετωπιστούν. Πρέπει να αναφέρουμε ότι η βιβλιοθήκη YALPS βρίσκεται σε αρχικά στάδια (έκδοση 0.3 Alpha) και η ύπαρξη εγχειριδίου χρήσης σε τυπικά επίπεδα. Ως εκ τούτου έπρεπε να αντιμετωπιστούν ποικίλα προβλήματα τα οποία ήταν αναμενόμενο ότι θα εμφανίζονταν. Η αντιμετώπιση των προβλημάτων αυτών σε γενικές γραμμές άξιζε το χρόνο που σπαταλήθηκε λόγω των πολλών πλεονεκτημάτων της YALPS όπως ο μηχανισμός επικοινωνίας μεταξύ διεργασιών και ο διαχειριστής εργασιών, μηχανισμοί οι οποί παρέχονται από την πλατφόρμαYALPS μεταξύ άλλων.

Ακολούθως υλοποιήθηκαν οι αλγόριθμοι *AlgCS, AlgLI, AlgCSF* και *AlgLIF* τους οποίους θα αναλύσουμε εκτενέστερα στην συνέχεια της Διατριβής.

Τέλος πραγματοποιήθηκε η εκτέλεση των υλοποιήσεων σε περιβάλλον προσομοίωσης το οποίο παρέχεται από την πλατφόρμα YALPS με διάφορες παραμέτρους για την εξαγωγή αποτελεσμάτων τα οποία χρησιμοποιήθηκαν για την πειραματική αξιολόγηση έτσι ώστε να διαφανεί κατά πόσον οι αλγόριθμοι είναι αποδοτικοί και στην πράξη. Για την αξιολόγηση

συγκρίθηκαν οι αλγόριθμοι μεταξύ τους και κατέληξα σε συμπεράσματα σχετικά με τη συμπεριφορά και την απόδοσή τους σε περιβάλλον προσομοίωσης.

1.4 Οργάνωση Εγγράφου

Η διατριβή αυτή αποτελείται από οκτώ κεφάλαια, εξαιρουμένων της βιβλιογραφίας και του παραρτήματος.

Στο *Κεφάλαιο 1*, παρουσιάζεται μια εισαγωγή του γενικότερου προβλήματος αλλά και τα βήματα τα οποία ακολουθήθηκαν.

Το επόμενο κεφάλαιο, *Κεφάλαιο 2,* ασχολείται με το υπόβαθρο της εργασίας, δηλαδή με τη γενική περιγραφή των αλγοριθμικών λύσεων καθώς και τη βιβλιοθήκη YALPS που χρησιμοποιήθηκε για την υλοποίηση και προσομοίωση.

Το *Κεφάλαιο 3* αναφέρεται στο μοντέλο υπολογισμού των αλγοριθμικών λύσεων.

Ακολουθούν το *Κεφάλαιο 4* στο οποίο περιγράφονται οι αλγόριθμοι *AlgCS* και *AlgLI* και το *Κεφάλαιο 5* στο οποίο περιγράφονται οι αλγόριθμοι *AlgCSF* και *AlgLIF*.

Στο *Κεφάλαιο 6* περιγράφεται η διαδικασία υλοποίησης των αλγοριθμικών λύσεων, τα προβλήματα τα οποία παρουσιάστηκαν καθώς και η αντιμετώπισή τους. Ίσως το σημαντικότερο κεφάλαιο της διατριβής μαζί με το *Κεφάλαιο 7*.

Στο *Κεφάλαιο 7* παρουσιάζονται τα αποτελέσματα της πειραματικής αξιολόγησης, συγκρίνονται οι αλγόριθμοι και αναλύονται τα αποτελέσματα.

Τέλος η διατριβή κλείνει με το *Κεφάλαιο 8* όπου περιγράφονται τα γενικά συμπεράσματα και μελλοντική έρευνα στο θέμα αυτό.

Κεφάλαιο 2

Υπόβαθρο

2.1 Προηγούμενες Εργασίες

Το πρόβλημα *Do-All* έχει μελετηθεί σε διάφορα μοντέλα υπολογισμού κάτω από διάφορες προϋποθέσεις σχετικά με το συγχρονισμό και τις αποτυχίες/αστοχίες., μερικά εκ των οποίον:

- Μεταβίβασης μηνυμάτων *(Message-passing)* (π.χ., [12] [3]) όπου οι διεργασίες επικοινωνούν μεταξύ τους με ανταλλαγή μηνυμάτων για την εκτέλεση εργασιών, μοντέλο το οποίο χρησιμοποιήθηκε και στην εργασία που αναφερόμαστε στην Διατριβή.
- Κοινόχρηστης Μνήμης *(Shared-memory)* (π.χ., [17] [13] [18]) όπου οι διεργασίες έχουν πρόσβαση σε μια κοινόχρηστη μνήμη στην οποία διατηρείται η απαιτούμενη γνώση και η επικοινωνία υλοποιείται με ανάγνωση και γραφή στην μνήμη
- Διαμερισμό Δικτύων *(Partitionable networks)* (π.χ., [19]) όπου ομάδες διεργασιών αποσυνδέονται επικοινωνιακά μεταξύ τους και πιθανών επανασυνδέονται κατά των διάρκεια του κατανεμημένου υπολογισμού.
- Στην απουσία επικοινωνίας *(In absence of communication)* (π.χ., [20]) όπου εξετάζεται κατά πόσο μπορεί να υπάρξει συνεργασιακή εκτέλεση εργασιών σε περίπτωση που η επικοινωνία διακοπεί για μεγάλο διάστημα

Όπως έχει ήδη αναφερθεί στις πιο πάνω εργασίες, βασική προϋπόθεση είναι ότι ο αριθμός των εργασιών*(tasks) m* είναι σταθερός, με ένα ανώτατο όριο και γνωστές εκ των προτέρων από όλες τις διεργασίες*(processes)*. Το πρόβλημα *Do-All* θεωρείται λυμένο όταν όλες οι εργασίες εκτελεστούν, υπό την προϋπόθεση ότι τουλάχιστον μία διεργασία θα παραμένει σε λειτουργιά σε ολόκληρη τη διαδικασία εκτέλεσης.

Η απόδοση του αλγόριθμου *Do-All* μετριέται είτε ως ο συνολικός αριθμός των εργασιών που έχουν εκτελεσθεί – *work complexity* [12] ή ο συνολικός αριθμός των διαδικαστικών βημάτων *(available process steps)* [13].

Οι Γεωργίου και συν. [21] θεωρούν μια επαναλαμβανόμενη εκδοχή του προβλήματος, όπου ομάδες *(waves) m* εργασιών πρέπει να εκτελούνται, η μια μετά την άλλη. Όλες οι ομάδες εργασιών υποτίθεται ότι είναι γνωστές εκ των προτέρων από τις διεργασίες. Σαφώς θεωρούν ότι το πρόβλημα σε αυτό το έργο είναι πιο γενικό, οι εργασίες δεν έρχονται κατά ομάδες, δεν είναι γνωστές εκ των προτέρων και ο αριθμός τους ενδέχεται να μην περιορίζεται. Επιπλέον, θεωρούν ότι οι διεργασίες καταρρέουν και επανεκκινούν, σε αντίθεση με προηγούμενη έρευνα [21] που εξετάζει μόνο καταρρεύσεις διεργασιών.

Οι Chlebus et al. [22] στην έρευνά τους εξετάζουν το πρόβλημα *Do-All* στο συγχρονισμένο μοντέλο μεταβίβασης μηνυμάτων με τις διεργασίες να καταρρέουν και να επανεκκινούν. Προκειμένου να επιτευχθεί μια λύση για το πρόβλημα, έκαναν δύο υποθέσεις μοντέλων:

i. Αξιόπιστη Πολυεκπομπή (*Reliable Multicast)*: Εάν μια διαδικασία αποτύχει, ενώ αποστέλλει ένα μήνυμα, τότε είτε το σύνολο των μη ελαττωματικών διεργασιών λαμβάνουν το μήνυμα, ή κανένας.

ii. Υπάρχει τουλάχιστον μία ζωντανή διαδικασία για $k > 1$ συνεχόμενους γύρους.

Στην εργασία αυτή των Γεωργίου και συν., χρειάζεται επίσης αξιόπιστη πολυεκπομπή κατά την εισαγωγή εργασιών στα μοντέλα όπου οι εργασίες εισάγονται τοπικά στην κάθε διεργασία, όπως θα δούμε και αργότερα για τη διασφάλιση ολοκλήρωσης χρειάζεται ένας περιορισμός στην περίοδο όπου η κάθε διαδικασία πρέπει να παραμένει ζωντανή.

Τέλος, στην εργασία [19], παρουσιάζεται μια έκδοση του προβλήματος *Do-All* όπου αλλάζει η τοπολογία του δικτύου δυναμικά και οι διεργασίες αποσυνδέονται από την ομάδα τους. Στην εργασία αυτή και πάλι ο αριθμός των εργασιών είναι σταθερός με ανώτατο όριο και γνωστές εκ των προτέρων σε όλες τις διεργασίες. Για την μέτρηση της απόδοσης ο

αλγόριθμος συγκρίνεται με την αποδοτικότητα ενός βέλτιστου αλγόριθμου ο οποίος γνωρίζει εκ των προτέρων τις αλλαγές στην τοπολογία του δικτύου.

Πρέπει να επισημάνουμε ότι η έννοια της ανταγωνιστικότητας *(competiveness),* δηλαδή, η σύγκριση της απόδοσης ενός συγκεκριμένου αλγόριθμου με την απόδοση του βέλτιστου αλγόριθμου ο οποίος γνωρίζει εκ των προτέρων το σενάριο εκτέλεσης εισήχθη από τους Sleator και Tarján [23].

2.2 Εργασία των Γεωργίου και Kowalski

Στην εργασία των Γεωργίου και Kowalski [11] όπου και θα επικεντρωθούμε στη διατριβή αυτή έγινε μια προσπάθεια να εξισορροπηθούν οι παράγοντες *(α) απόδοση* και *(β) ανοχή σε σφάλματα*, ξεκίνησαν με τη μελέτη της εκτέλεσης εργασιών όπου *n* διεργασίες υπόκεινται σε καταρρεύσεις και επανεκκινήσεις και συνεργάζονται στην εκτέλεση ανεξάρτητων εργασιών που συνεχώς εισάγονται στο σύστημα.

Γύροι επικοινωνίας: Ο υπολογισμός έχει χωριστεί σε συγχρονισμένους γύρους *(rounds),* στους οποίους κάθε διεργασία εισάγονται εργασίες, λαμβάνονται τα μηνύματα που στάλθηκαν (αν στάλθηκαν) στον προηγούμενο γύρο, εκτελεί τοπικά υπολογισμούς (συμπεριλαμβανομένου και την εκτέλεση το πολύ μιας εργασίας), και στέλνει μηνύματα (αν χρειάζεται). Υποθέτουν ότι οι εργασίες είναι ομοιόμορφες, δηλαδή χρειάζονται τον ίδιο χρόνο να εκτελεστούν και ότι η κάθε εργασία χρειάζεται ένα γύρο για να εκτελεστεί.

Αποδοτικότητα Αλγορίθμων: Στη συνέχεια, η αποδοτικότητα του αλγόριθμου μετριέται σε σχέση με το μέγιστο αριθμό των εργασιών σε αναμονή *(max number of pending tasks)* κατά την έναρξη ενός γύρου. Αυτό δίνει τη δυνατότητα να δούμε το πρόβλημα ως online πρόβλημα και να αναλύσουμε τους αλγόριθμους ως προς την ανταγωνιστικότητά τους [23], δηλαδή, να συγκριθεί η αποδοτικότητα ενός συγκεκριμένου αλγόριθμου με την αποδοτικότητα του βέλτιστου αλγόριθμου ο οποίος γνωρίζει εκ των προτέρων για τις καταρρεύσεις, επανεκκινήσεις και εισαγωγές εργασιών.

Εγγυήσεις εκτέλεσης εργασιών: Για τις αναγκαίες εγγυήσεις που δίνονται για την εκτέλεση των εργασιών μελετήθηκαν δύο εκδόσεις του προβλήματος. Η πρώτη, η οποία αποτελεί τη βασική ιδιότητα της *Διασφάλισης της Ορθότητας (basic correctness)*, προϋποθέτει ότι καμία εργασία δε χάνεται, δηλαδή, μια εργασία είτε εκτελείται είτε παραμένει στο σύστημα σε αναμονή. Η δεύτερη και ισχυρότερη ιδιότητα, η *Διασφάλιση της Ολοκλήρωσης (fairness)*, απαιτεί ότι όλες οι εργασίες που εισάγονται στο σύστημα τελικά θα εκτελεστούν. Επίσης παρατηρούμε ότι η πρώτη ιδιότητα δεν εγγυάται ότι κάποια εργασία τελικά θα εκτελεστεί.

Μοντέλα Πληροφόρησης: Οι συγγραφείς έχουν αναπτύξει μια σταδιακή προσέγγιση στη μελέτη του προβλήματος. Πρώτα υπόθεσαν ότι υπάρχει μια κεντρική αρχή, που ονομάζεται «Κεντρικός Χρονοπρογραμματιστής - ΚΧ» *(Central Scheduler),* που στην αρχή του κάθε γύρου ενημερώνει τις διεργασίες σχετικά με τις εργασίες που εκκρεμούν ακόμη να εκτελεστούν, συμπεριλαμβανομένων και τυχόν νέων εργασιών που εισήχθηκαν στο γύρο αυτό.

Υπήρχαν δύο λόγοι για να ξεκινήσουν με αυτήν την υπόθεση: (α) Το γεγονός ότι οι διεργασίες έχουν ίδια πληροφορία σχετικά με τον αριθμό των εκκρεμούντων εργασιών, επιτρέπει την επικέντρωση στον προσδιορισμό των περιορισμών του προβλήματος χωρίς να χρειάζεται η ανταλλαγή πληροφοριών μεταξύ των διαδικασιών. (β) Η μελέτη του προβλήματος σε αυτή την περίπτωση έχει δικό του ανεξάρτητο ενδιαφέρον, καθώς ο κεντρικός χρονοπρογραμματιστής μπορεί να θεωρηθεί ένας παρατηρητής που χρησιμοποιείται για την παρακολούθηση της προόδου και την παροχή ανάδρασης στις διεργασίες σε πραγματικές εφαρμογές όπως στο Πλέγμα *(Grid)* [8] και στον Διαδικτυακό Υπολογισμό όπως στο SETI [9].

Ακολούθως περιορίστηκε η πληροφορία που παρεχόταν στις διεργασίες. Μια πιο αδύναμη κεντρική αρχή, που ονομάζεται «Κεντροποιημένη Εισαγωγή Εργασιών - ΚΕΕ» *(Central Injector)*, η οποία ενημερώνει τις διαδικασίες, στην αρχή του κάθε γύρου, μόνο για τις

εργασίες που εισήχθηκαν σε αυτό το γύρο και πληροφορίες σχετικά με τις εργασίες που έχουν πραγματοποιηθεί στον προηγούμενο γύρο μόνο.

Με τις αποκτηθείσες γνώσεις και κατανόηση των πιο πάνω, προχώρησαν σε ένα αλγόριθμο τον «Τοπικό Εισαγωγέα - ΤΕ» *(Local Injector)* όπου οι διεργασίες μπορούν να αποκτήσουν κοινή πληροφορία για το σύνολο των εκκρεμών εργασιών χωρίς τη χρήση μιας κεντρικής αρχής. Υπέθεσαν ότι οι εργασίες εισάγονται τοπικά στις διεργασίες χωρίς να τους παρέχει καμία κοινή πληροφορία.

Ο τοπικός εισαγωγέας εργασιών μπορεί να θεωρηθεί, ως μια τοπική υπηρεσία *(daemon)* μιας κατανεμημένης εφαρμογής που παρέχει τοπικά πληροφορίες στις διεργασίες εκτελείται. Έχει αποδειχτεί ότι λύσεις σε αυτό το γενικότερο πλαίσιο έχουν ελάχιστο κόστος για την ανταγωνιστικότητα, με την προϋπόθεση όμως ότι παρέχεται αξιόπιστη πολυεκπομπή [22].

Συνοπτικά τα αποτελέσματα:

i. Λύσεις που διασφαλίζουν την Ορθότητα: Για το μοντέλο ΚΧ, το κατώτερο όριο των εργασιών σε αναμονή είναι $OPT + n / 3$ σε σχέση με την ανταγωνιστικότητά για κάθε ντετερμινιστικό αλγόριθμο. Ανάπτυξαν στη συνέχεια ένα ντετερμινιστικό, τον σχεδόν βέλτιστο αλγόριθμο *AlgCS*, αλγόριθμος που δεν κάνει οποιαδήποτε χρήση της ανταλλαγής μηνυμάτων μεταξύ διεργασιών και επιτυγχάνει ανταγωνιστικότητα $OPT + 2n$ σε σχέση με τον βέλτιστο αλγόριθμο. Χρησιμοποιώντας ένα απλό μετασχηματισμό παίρνουμε τον αλγόριθμο *GenCI* για την εκδοχή με τον τοπικό εισαγωγέα εργασιών όπου έχει την ίδια την ανταγωνιστικότητα με τον αλγόριθμο *AlgCS*. Τέλος, αναπτύχτηκε ο αλγόριθμος *AlgLI* για την εκδοχή με τις τοπικές εισαγωγές εργασιών και έδειξαν ότι επιτυγχάνει ανταγωνιστικότητα $OPT + 3n$ σε σχέση με τον βέλτιστο αλγόριθμο, με την προϋπόθεση αξιόπιστης πολυεκπομπής.

ii. Λύσεις που διασφαλίζουν την Ολοκλήρωση: Η διασφάλιση της ολοκλήρωσης είναι πολύ πιο σύνθετη απ’ ότι η διασφάλιση ορθότητας. Η «Διασφάλιση Ολοκλήρωσης»

εγγυάται ότι όλες οι εργασίες τελικά κάποια στιγμή θα εκτελεστούν. Έδειξαν ότι είναι αναγκαίο να υποθέσουμε ότι μετά την επανεκκίνηση μιας διεργασίας η διεργασία δεν καταρρέει ξανά μέσα στους επόμενους δύο τουλάχιστον γύρους. Κάτω από αυτόν τον περιορισμό, που ονομάζεται *2-survivability*, έχουν αναπτύξει τους αλγόριθμους *AlgCSF* και *AlgLIF* οι οποίοι διασφαλίζουν την ολοκλήρωση. Οι αλγόριθμοι αυτοί έχει αποδεχτεί ότι επιβαρύνονται ένα επιπλέον *n* στην ανταγωνιστικότητά τους, σε σύγκριση με τους αλγορίθμους που διασφαλίζουν την ορθότητα μόνο. Μια ενδιαφέρουσα παρατήρηση είναι ότι η ολοκλήρωση μπορεί να εγγυηθεί μόνο σε εκτελέσεις μη φραγμένης διάρκειας.

iii. Οριοθέτηση επικοινωνίας: Έχει αποδειχτεί ότι το μοντέλο του ΚΧ και ΤΧ, αν οι διεργασίες δεν στέλνουν τα μηνύματα σε όλες τις άλλες διεργασίες, τότε ορθότητα (και επομένως και η ολοκλήρωση) δεν μπορεί να εγγυηθεί.

iv. Εργασίες μη-μοναδιαίου μήκους: Για τα παραπάνω αποτελέσματα υποθέτουν ότι οι εργασίες είναι μοναδιαίου μήκους, που απαιτούν από κάποια διεργασία ένα γύρο για την εκτέλεση. Η κατάσταση είναι ακόμη πιο περίπλοκη όταν οι εργασίες δεν είναι μοναδιαίου μήκους.

Πρέπει να αναφέρουμε ότι η μελέτη περιορίστηκε σε θεωρητικό επίπεδο και σκοπός της παρούσας Διατριβής είναι η επέκταση της μελέτης σε πρακτικό επίπεδο.

2.3 Βιβλιοθήκη YALPS

Η βιβλιοθήκη YALPS [15] είναι μία ανοικτού κώδικα βιβλιοθήκη γραμμένη στην Java η οποία σχεδιάστηκε για να διευκολύνει το σχεδιασμό, την ανάπτυξη και τον έλεγχο κατανεμημένων εφαρμογών. Εφαρμογές που έχουν αναπτυχθεί χρησιμοποιώντας τη βιβλιοθήκη YALPS μπορούν να τρέξουν σε περιβάλλον προσομοίωσης ή σε πραγματικό περιβάλλον χωρίς καμία αλλαγή στον κώδικα της εφαρμογής. Η όλη προσαρμογή δε

χρειάζεται μεταγλώττιση του πηγαίου κώδικα, πραγματοποιείται με απλά βήματα μέσω του αρχείου ρυθμίσεων. Αυτό είναι ένα πολύ σημαντικό πλεονέκτημα της βιβλιοθήκης γιατί επιτρέπει την ανάπτυξη μιας εφαρμογής και τον έλεγχό της σε περιβάλλον προσομοίωσης πριν την εφαρμογή της σε πραγματικό περιβάλλον.

Οι εφαρμογές της βιβλιοθήκης YALPS είναι οργανωμένες σαν μία συλλογή από εργασίες[1] «Tasks» τις οποίες διαχειρίζεται ο διαχειριστής εργασιών «*TaskManager*». Ο διαχειριστής εργασιών επιτρέπει στις εργασίες να καθορίσουν το χρόνο στον οποίο θα εκτελεστούν. Για παράδειγμα μπορεί να προγραμματιστούν για άμεση εκτέλεση ή μετά από κάποιο προκαθορισμένο χρονικό διάστημα ή μπορούν να εκτελούνται περιοδικά. Επίσης, οι εργασίες μπορούν να ομαδοποιηθούν σε «*TaskGroups*» για να διαχειρίζονται σαν ομάδα αντί σαν ατομικές εργασίες. Αυτό επιτρέπει στον προγραμματιστή να ενεργοποιήσει/απενεργοποιήσει μέρος της εφαρμογής με μία απλή αλλαγή σε μια ομάδα εργασιών. Ακόμη, μέσω των δυο υλοποιήσεων του διαχειριστή εργασιών, *SimulatedTaskManger* και *ExecutorTaskManager*, η ίδια εφαρμογή έχει τη δυνατότητα να τρέξει, είτε σε προσομοίωση είτε σε πραγματικό κατανεμημένο σύστημα αντίστοιχα.

Ένα άλλο στοιχείο της βιβλιοθήκης YALPS είναι ο τρόπος με τον οποίο επιτυγχάνεται η επικοινωνία. Για το σκοπό αυτό υπάρχει το στρώμα επικοινωνίας της YALPS *«YALPS CommunicationLayer»* για το οποίο υπάρχουν επίσης δύο υλοποιήσεις, μία προσομοίωση και μια για πραγματικό περιβάλλον. Για το πραγματικό σύστημα τα μηνύματα της εφαρμογής δρομολογούνται μέσω των πρωτοκόλλων UDP/TCP ενώ για την προσομοίωση της εφαρμογής χρησιμοποιείται μία εσωτερική υποδομή επικοινωνίας της YALPS. Και στις δύο περιπτώσεις, το στρώμα επικοινωνίας της YALPS μας προσφέρει λειτουργίες για έλεγχο και αξιολόγηση των κατανεμημένων εφαρμογών, όπως είναι ο περιορισμός του εξερχόμενου εύρους ζώνης και η επιλογή για απώλεια μηνυμάτων κατά την εκτέλεση εφαρμογής.

[1] Διαφορετική έννοια από της εργασίες του προβλήματος εκτέλεσης εργασιών που μελετούμε

Τέλος, τα μηνύματα που ανταλλάσσονται μέσω των εφαρμογών της YALPS απαιτείται να είναι σε σειριακή μορφή (serialized). Η μετατροπή ενός μηνύματος σε σειριακή μορφή πραγματοποιείται έτσι ώστε να μπορεί να μεταδοθεί μέσω μίας σύνδεσης ενός δικτύου [24]. Σε αυτή την εργασία τα μηνύματα μετατρέπονται αυτόματα μέσω της JAVA σε σειριακή μορφή για να μπορέσουν να μεταδοθούν από τον ένα κόμβο στον άλλο μέσω του δικτύου, μέσω πρωτοκόλλου UDP. Όταν το μήνυμα φτάσει στον παραλήπτη τότε μπορεί εύκολα να επανέλθει στην αρχική του μορφή. Υπάρχουν δύο τρόποι για να επιτευχθεί αυτό σε μία εφαρμογή της YALPS. Είτε χρησιμοποιώντας την πρότυπη μέθοδο σειριοποίησης της Java (κλάση Serializable), είτε χρησιμοποιώντας ένα πιο αποτελεσματικό μηχανισμό ο οποίος μπορεί να υλοποιηθεί μέσω της διεπαφής της βιβλιοθήκης YALPS και με βάσει τις απαιτήσεις της εφαρμογής.

Έτσι, έχοντας τη βιβλιοθήκη YALPS στα χέρια μας η οποία μας δίνει έτοιμα όλα τα απαραίτητα στοιχεία μίας κατανεμημένης εφαρμογής, μπορούμε πιο εύκολα να υλοποιήσουμε κατανεμημένες εφαρμογές, να τις προσομοιώσουμε ή να τις τρέξουμε σε πραγματικά κατανεμημένα συστήματα χωρίς να χρειάζεται να ανησυχούμε για τα πρακτικά θέματα δικτύων ή και για άλλες τεχνικές δυσκολίες.

2.2.1 Διεπαφές της βιβλιοθήκης YALPS

Διεπαφή NodeID

Οι κόμβοι μίας εφαρμογής δηλώνονται ως αντικείμενα της διεπαφής *NodeID*, χωρίς να έχει σημασία αν η εφαρμογή θα τρέξει σε προσομοίωση ή σε πραγματικό σύστημα. Στην παρασκήνιο όμως υπάρχουν δύο κλάσεις που υλοποιούν τη διεπαφή *NodeID*, η κλάση *E_NodeID* για την εκτέλεση της εφαρμογής σε πραγματικό σύστημα και η κλάση *S_NodeID* σε περιβάλλον προσομοίωσης. Το ποια από τις δύο υλοποιήσεις θα επιλεχθεί καθορίζεται από το αρχείο διαμόρφωσης (configuration file) κατά την εκτέλεση της εφαρμογής.

Ένας κόμβος αρχικοποιείται σε μία εφαρμογή της βιβλιοθήκης YALPS με τον εξής τρόπο:

```
NodeID MyNode=cl.getNodeIdFromString(NodeName);
```

Όπου το αντικείμενο *«cl»* αντιστοιχεί στο υπόστρωμα επικοινωνίας το οποίο χρησιμοποιεί η εφαρμογή και η συνάρτηση *«getNodeIdFromString(NodeName)»* επιστρέφει το μοναδικό αναγνωριστικό του κόμβου με το όνομα *«NodeName»*. Το μοναδικό αναγνωριστικό που επιστρέφει η συνάρτηση αποθηκεύεται στη μεταβλητή *«MyNode»*.

Διεπαφή YalpsNode

Η βασική αφαιρετική κλάση της βιβλιοθήκης YALPS είναι η *AbstractYalpsNode* η οποία υλοποιεί τη διεπαφή *YalpsNode*. Η *AbstractYalpsNode* απλοποιεί την κωδικοποίηση συστατικών μιας εφαρμογής τα οποία είναι απαραίτητα για όλες τις εφαρμογές της YALPS. Παραδείγματα τέτοιων συστατικών είναι η δήλωση ενός αντικειμένου *«tm»* τύπου *TaskManager* το οποίο διαχειρίζεται τις εργασίες της εφαρμογής και η δήλωση ενός αντικειμένου *«cl»* τύπου *CommunicationLayer* μέσω του οποίου θα επιτυγχάνεται η επικοινωνία μεταξύ των κόμβων της εφαρμογής.

Συνεπώς, κλάση σε κάθε εφαρμογή της YALPS μπορεί να επεκτείνει την κλάση AbstractYalpsNode με τον εξής τρόπο:

```
public class MyApplication extends AbstractYalpsNode{ }
```

Διεπαφή Task

Οι εργασίες (tasks) αποτελούν το ενεργό μέρος των εφαρμογών της YALPS. Μία εργασία είναι απλά ένα αντικείμενο που υλοποιεί τη διεπαφή *«Task»*. Για να ολοκληρωθεί η υλοποίηση της διεπαφής *Task* πρέπει να προστεθεί κώδικας στη μέθοδο *run()* του

αντικειμένου ο οποίος θα περιγράφει το λειτουργικό μέρος της εργασίας. Το εργαλείο που χρησιμοποιούν οι εφαρμογές YALPS για να προγραμματίσουν τις εργασίες τους είναι το αντικείμενο τύπου «*TaskManager»* που υπάρχει σε κάθε εφαρμογή. Επιπλέον, οι εργασίες μπορούν να ομαδοποιηθούν χρησιμοποιώντας το μηχανισμό ομαδοποίησης εργασιών (*TaskGroup*) για να τις διαχειρίζεται ο «*TaskManager»* σαν ομάδα. Η δημιουργία μίας εργασίας γίνεται με τον εξής τρόπο:

```
class MyTask implements Task{
   public void run(){
      ...
   }
}
```

Υπάρχουν τρία είδη εργασιών, (α) οι εργασίες που εκτελούνται αμέσως μετά την καταχώρησή τους, (β) οι εργασίες που εκτελούνται σε συγκεκριμένο χρονικό διάστημα μετά την καταχώρησή τους και οι εργασίες που εκτελούνται περιοδικά. Η διεπαφή *«TaskManager»* μας προσφέρει διάφορες μεθόδους για τον προγραμματισμό της εκτέλεσης των εργασιών.

i. Άμεση εκτέλεση εργασιών: Η πιο κάτω μέθοδος καταχωρεί την εργασία *t* και τη συσχετίζει με την ομάδα εργασιών *g*. Αυτός ο τρόπος καταχώρησης της εργασίας *t*, συνεπάγεται ότι η εργασία θα εκτελεστεί άμεσα.

   ```
   void registerTask(Task t, TaskGroup g);
   ```

   Η μέθοδος αυτή μπορεί να κληθεί από το διαχειριστή εργασιών *«tm»* της εφαρμογής με τον εξής τρόπο, που αναφέρεται πιο κάτω, όπου *tm.getSystemTaskGroup()* θα επιστρέψει την προεπιλεγμένη ομάδα εργασιών του συστήματος.

   ```
   tm.registerTask(new MyTask(), tm.getSystemTaskGroup());
   ```

ii. Αναβεβλημένη εκτέλεση εργασιών: Αν θέλουμε μία εργασία να μην εκτελεστεί άμεσα αλλά μετά από συγκεκριμένο χρονικό διάστημα αυτό επιτυγχάνεται με τη μέθοδο:

```
void registerTask(Task t, TaskGroup g, long millis);
```

Η μέθοδος αυτή έχει την ίδια λειτουργία με την προηγούμενη, με τη διαφορά ότι έχει την επιπλέον παράμετρο *«long millis»* η οποία προσδιορίζει ότι η εργασία θα εκτελεστεί μετά από χρονικό διάστημα *«millis»* χιλιοστών του δευτερολέπτου.

iii. Περιοδικές εργασίες: Οι εργασίες που εκτελούνται περιοδικά, δηλαδή ανά τακτά χρονικά διαστήματα. Για το σκοπό αυτό υπάρχει μέθοδος η οποία διαφέρει με τις προηγούμενες στο ότι καταχωρεί την *«t»* ως περιοδική εργασία (*PeriodicTask*).

```
void registerPeriodicTask(PeriodicTask t, TaskGroup g, long
millis);
```

Η «*PeriodicTask»* είναι μία διεπαφή που εντάσσεται στη διεπαφή *«Task»* και έχει κάποιες επιπλέον μεθόδους τις οποίες χρειάζονται αυτού του είδους οι εργασίες όπως είναι η μέθοδος *getPeriod()*. Η μέθοδος αυτή επιστρέφει το χρονικό διάστημα στο οποίο η εργασία θα πρέπει να εκτελεστεί ξανά κάτι το οποίο καθορίζεται από την παράμετρο *«long millis».*

2.2.2 Ανταλλαγή Μηνυμάτων

Αποστολή Μηνυμάτων

Η επικοινωνία μεταξύ των κόμβων μίας εφαρμογής YALPS γίνεται μέσω ενός αντικειμένου τύπου *«CommunicationLayer»* το οποίο υπάρχει σε όλες τις εφαρμογές της YALPS. Το αντικείμενο αυτό έχει δύο διαφορετικές μεθόδους για αποστολή μηνυμάτων, χρησιμοποιώντας το πρωτόκολλο TCP ή UDP.

Η πιο κάτω μέθοδος στέλλει το μήνυμα *«msg»* στον κόμβο *«dest»* χρησιμοποιώντας το TCP πρωτόκολλο επικοινωνίας.

```
public void sendTCP(Serializable msg, NodeIDdest) throws
IOException;
```

Για την αποστολή του μηνύματος *«msg»* στον κόμβο *«dest»* χρησιμοποιώντας το UDP πρωτόκολλο επικοινωνίας χρησιμοποιούμε τη μέθοδο:

```
public void sendUDP(Serializable msg, NodeID dest) throws
IOException;
```

Παραλαβή Μηνυμάτων

Ένας κόμβος σε μία εφαρμογή YALPS παραλαμβάνει όλα τα μηνύματα τα οποία προέρχονται από άλλους κόμβους της εφαρμογής και προορίζονται γι' αυτόν. Αυτό επιτυγχάνεται με την υλοποίηση της πιο κάτω μεθόδου που ορίζεται στη διεπαφή *«Receiver»*.

```
public boolean receive(short header, Object msg, NodeID
sender);
```

Η μέθοδος παραλαμβάνει το μήνυμα *«msg»* με επικεφαλίδα *«header»* από τον κόμβο *«sender»*.

2.2.3 Βασικά Συστατικά και Εκτέλεση

Αρχικοποίηση του Κόμβου

Σημαντική μέθοδος της κλάσης «*AbstractYalpsNode»* είναι η μέθοδος που επιτρέπει στον κόμβο να εκτελέσει όλες τις αρχικοποιήσεις που είναι απαραίτητες για την περαιτέρω λειτουργία της εφαρμογής. Οι αρχικοποιήσεις αυτές αφορούν συνήθως προγραμματισμό των

εργασιών του κόμβου ή αποστολή κάποιων μηνυμάτων στους υπόλοιπους κόμβους της εφαρμογής.

```
public void startNode();
```

Η μέθοδος κληρονομείται και πάλι από την κλάση «*AbstractYalpsNode»* και στην υλοποίησή της μπορούν να συμπεριληφθούν όλες οι αρχικοποιήσεις. Ένα παράδειγμα υλοποίησης της μεθόδου είναι:

```
public void startNode(){
   tm.registerTask(new myTask(), tm.getSystemTaskGroup());
}
```

Στην πιο πάνω υλοποίηση της μεθόδου *«stardNode()»* γίνεται προγραμματισμός των εργασιών του κόμβου.

Εκτελεστές μίας Εφαρμογής YALPS

Μετά την υλοποίηση κλάσης διεπαφής «*YalpsNode»* δημιουργείται η εφαρμογή YALPS η οποία μπορεί να εκτελεστεί είτε σε προσομοίωση είτε σε πραγματικό κατανεμημένο σύστημα. Υπάρχουν δύο εκτελεστές της YALPS οι οποίοι χρησιμοποιούνται ανάλογα με τον τρόπο που θέλουμε να τρέξει η εφαρμογή.

Ο πιο κάτω εκτελεστής χρησιμοποιείται για να τρέξει η εφαρμογή σε πραγματικό σύστημα.

```
yalps.launchers.ExecutorNodeStarter
```

Ο εκτελεστής πιο κάτω χρησιμοποιείται για να τρέξει η εφαρμογή σε προσομοίωση.

```
yalps.launchers.SimulatedNodeStarter
```

Και οι δύο εκτελεστές παίρνουν είσοδο ένα αρχείο διαμόρφωσης, το οποίο προσδιορίζει ποιοι κόμβοι της εφαρμογής θα αρχικοποιηθούν και ακολούθως αναλαμβάνουν την εκτέλεσή τους.

Εκτέλεση εφαρμογής βασισμένης βιβλιοθήκης YALPS

Ο εκτελεστής της YALPS που θα επιλεχθεί για να εκτελέσει την εφαρμογή καθορίζει το περιβάλλον στο οποίο θα τρέξει, προσομοίωση ή πραγματικό σύστημα. Στην πρώτη περίπτωση χρησιμοποιείται ο εκτελεστής *«SimulatedNodeStarter»* ενώ στη δεύτερη περίπτωση χρησιμοποιείται ο εκτελεστής *«ExecutorNodeStarter»*.

Παράδειγμα εντολής για προσομοίωση της εφαρμογής:

```
java -cp /yalps.jar launchers.SimulatedNodeStarter myExample-
sim.conf
```

Παράδειγμα εντολής για τρέξιμο της εφαρμογής σε πραγματικό σύστημα:

```
java -cp /yalps.jar launchers.ExecutorNodeStarter myExample-
exec.conf
```

Όπου *myExample-sim.conf* και *myExample-exec.conf* τα αντίστοιχα αρχεία διαμόρφωσης.

2.2.1 Αρχείο Διαμόρφωσης

Το αρχείο διαμόρφωσης (configuration file) της YALPS είναι ουσιαστικά ένα αρχείο ιδιοτήτων της Java το οποίο δηλώνει ένα σύνολο αντικειμένων με τις ιδιότητές τους. Το αρχείο διαμόρφωσης αποτελείται από δύο είδη δηλώσεων, μία για τις αρχικοποιήσεις αντικειμένων και μία για τη ρύθμιση των ιδιοτήτων τους.

i. Δηλώσεις Αρχικοποίησης Αντικειμένων

   Οι δηλώσεις για αρχικοποίηση των αντικειμένων της εφαρμογής έχουν την ακόλουθη μορφή

   ```
   <object name>.CLASS=<fully qualified class name>
   ```

Όπου 'object name' είναι το όνομα του αντικειμένου και μπορεί να είναι οποιοδήποτε έγκυρο αναγνωριστικό της Java ενώ *«fully qualified class name»* είναι το απόλυτο όνομα της κλάσης του αντικειμένου. Με αυτό τον τρόπο αρχικοποιούμε αντικείμενα των οποίων ο κατασκευαστής δεν παίρνει παραμέτρους. Μπορούν όμως να καθοριστούν τυχόν ιδιότητες του αντικειμένου χρησιμοποιώντας δηλώσεις ρύθμισης ιδιοτήτων, όπως αυτές περιγράφονται πιο κάτω.

ii. Δηλώσεις Ρύθμισης Ιδιοτήτων

Οι δηλώσεις για ρύθμιση των ιδιοτήτων ενός αντικειμένου της εφαρμογής έχουν την ακόλουθη μορφή

```
<object name>.<property name>=<value>
```

Όπου «object name» το όνομα του αντικειμένου του οποίου θέλουμε να ρυθμίσουμε την ιδιότητα, *«property name»* το όνομα της ιδιότητας που θα ρυθμιστεί και «value» η τιμή που θα πάρει η ιδιότητα.

Για να μπορεί να υπάρχει μία τέτοια δήλωση σε ένα αρχείο διαμόρφωσης πρέπει η κλάση του αντικειμένου *«object name»* να περιέχει μία μέθοδο

```
void set+'property name'();
```

Όπου με το «+» δηλώνεται η συνένωση των δύο συμβολοσειρών, «set» και «property name». Η παράμετρος της μεθόδου πρέπει να είναι συμβατή με τον τύπο της τιμής *«value»*. Ο εκτελεστής της YALPS περνά αυτόματα την τιμή *«value»* σαν παράμετρο στην πιο πάνω μέθοδο μετατρέποντάς την στον κατάλληλο τύπο.

Ακολουθεί ένα παράδειγμα αρχείου διαμόρφωσης της YALPS:

```
node.CLASS = yalps.example.application.myApplication
node.nodeList=10
```

Για να είναι σωστό το πιο πάνω αρχείο διαμόρφωσης, θα πρέπει στην εφαρμογή *myApplication* να υπάρχει η μέθοδος

```
void setNodeList(Type myParam) { }
```

Όπου *«Type»* μπορεί να είναι οποιοσδήποτε τύπος (π.χ. Integer, String) στον οποίο μπορεί να μετατραπεί η τιμή *«10»*.

Μία εφαρμογή η οποία προορίζεται για να τρέξει σε περιβάλλον προσομοίωσης ή σε πραγματικό σύστημα. Και στις δύο περιπτώσεις το αρχείο διαμόρφωσης θα έχει την ίδια μορφή με διαφορά στο σημείο που δηλώνονται οι κόμβοι της εφαρμογής. Ακολουθούν παραδείγματα και για τις δύο περιπτώσεις.

Παράδειγμα δήλωσης των κόμβων της εφαρμογής που θα τρέξει σε προσομοίωση:

```
node.nodeList=1;2;3;4;5;6;7;8;9;10
```

Παράδειγμα δήλωσης των κόμβων της εφαρμογής που θα τρέξει σε πραγματικό σύστημα:

```
node.nodeList=hostname:port;hostname2:port2;....;
```

Όπως είναι φυσικό οι κόμβοι στην προσομοίωση δεν είναι πραγματικοί άρα δε χρειάζεται να δηλωθεί πραγματική διεύθυνση δικτύου και πραγματικός αριθμός θύρας για τον κάθε κόμβο.

Κεφάλαιο 3

Μοντέλο Υπολογισμού

3.1 Κατανεμημένο Περιβάλλον

Θεωρούμε ότι το κατανεμημένο σύστημα αποτελείται από *n* συγχρονισμένες διεργασίες/κόμβους, επιρρεπείς σε σφάλματα/καταρρεύσεις, με μοναδικά αναγνωριστικά στο σύνολο *[n] = {1,2,...,n}*. Υποθέτουμε ότι οι διεργασίες έχουν πρόσβαση σε ένα κοινό ρολόι χρονισμού.

Υποθέτουμε επίσης ένα πλήρως συνδεδεμένο μέσω επικοινωνίας (δηλαδή, κάθε διεργασία μπορεί να επικοινωνεί απευθείας με κάθε άλλη διεργασία) όπου τα μηνύματα δε χάνονται ή αλλοιώνονται κατά τη μεταφορά.

3.2 Γύροι

Για την απλότητα των αλγορίθμων, υποθέτουμε ότι ένας γύρος (round) χωρίζεται σε τέσσερα διαδοχικά στάδια: *(α) Λήψη Μηνυμάτων*, στο οποίο μια διεργασία λαμβάνει τα μηνύματα που αποστέλλονται σε αυτήν στον προηγούμενο γύρο. *(β) Εισαγωγή Εργασιών*, όπου οι νέες εργασίες δημιουργούνται και εισάγονται στο σύστημα. *(γ) Στάδιο Υπολογισμού*, όπου η κάθε διεργασία εκτελεί το πολύ μια εργασία. *(δ) Αποστολή Μηνυμάτων*, στο οποίο μια διεργασία στέλνει μηνύματα προς άλλες διεργασίες.

3.3 Εργασίες

Κάθε εργασία (task) *τ* αποτελείται από *(id, τ, ρ)*, όπου: *τ.id* είναι ένας μοναδικός θετικός ακέραιος που αποτελεί το αναγνωριστικό της εργασίας *τ*, *τ.ρ* αντιστοιχεί στον αριθμό του

γύρου που εισήχθηκε η εργασία *τ* στο σύστημα και *τ.code* αντιστοιχεί στον υπολογισμό που πρέπει να εκτελεστεί ώστε η εργασία θεωρείται ολοκληρωμένη (δηλαδή, το υπολογιστικό μέρος της εργασίας).

Οι εργασίες υποθέτουμε ότι είναι όμοιες, ανεξάρτητες και ταυτοδύναμες. Με την ομοιότητα σημαίνει ότι οι υπολογισμοί για κάθε εργασία καταναλώνουν την ίδια υπολογιστική ισχύ. Με την ανεξαρτησία εννοούμε ότι η ολοκλήρωση των εργασιών δεν επηρεάζει οποιαδήποτε άλλη εργασία, καθώς και κάθε εργασία μπορεί να γίνει ταυτόχρονα με οποιαδήποτε άλλη εργασία. Με την ταυτοδυναμία εννοούμε ότι κάθε εργασία μπορεί να εκτελεστεί μία ή περισσότερες φορές παράγοντας το ίδιο αποτέλεσμα. Τέλος, θεωρούμε ότι ο αριθμός των εργασιών είναι πολυωνυμικού μεγέθους n.

3.4 Αντίπαλος

Υποθέτουμε την ύπαρξη ενός αντίπαλου/εχθρού *(adversary)* που μπορεί να προκαλέσει καταρρεύσεις και επανεκκινήσεις διεργασιών *(process crashes and restarts),* καθώς και εισαγωγές εργασιών *(task injections)*. Σε ένα εχθρικό μοτίβο Α, το *crash(r, i)* δηλώνει ότι η διεργασία *i* θα καταρρεύσει στο γύρο *r*. Το *restart(r, i)* δηλώνει το γεγονός ότι η διεργασία *i* θα επανεκκινήσει στο γύρο *r*. Εξυπακούεται ότι δε θα υπάρξει επανεκκίνηση αν δεν υπάρχει προηγούμενη κατάρρευση. Τέλος η δήλωση *inject(r, i, τ)* δηλώνει ότι η εργασία *τ* θα εισαχτεί στην διεργασία *τ* στο γύρο *r*.

Θεωρούμε ότι μια διεργασία *i* είναι «ζωντανή» στο γύρο *r*, εάν η διεργασία λειτουργεί κατά την έναρξη του γύρου και δε θα καταρρεύσει πριν από το τέλος του γύρου. Επίσης υποθέτουμε ότι όταν ο αντίπαλος διοχετεύει εργασίες σε ένα γύρο, διοχετεύει ένα αριθμό πεπερασμένων εργασιών.

3.5 Επανεκκίνηση των Διεργασιών

Υποθέτουμε ότι διεργασία η οποία έχει επανεκκινήσει έχει γνώση μόνο του αλγορίθμου και τα αναγνωριστικά των άλλων διεργασιών του συστήματος και καμία πληροφορία σχετικά με ποιές διεργασίες που είναι «ζωντανές». Αλγοριθμικά μιλώντας, η κάθε διεργασία μετά την επανεκκίνηση, περιμένει να λάβει μηνύματα ή να εισαγάγει εργασίες. Στη συνέχεια, γνωρίζει ότι ένας νέος γύρος έχει ήδη αρχίσει και ως εκ τούτου, μπορεί να ξεκινήσει ομαλά να συμμετέχει ενεργά στον αλγόριθμο. Υποθέτουμε επίσης ότι η επανεκκίνηση της διεργασίας γίνεται στην αρχή του γύρου *r* και λαμβάνει τα μηνύματα που αποστέλλονται σε αυτόν (αν υπάρχουν) στο τέλος του γύρου *r - 1*.

Ένα εχθρικό μοτίβο είναι αποδεχτό, εάν:

i. σε κάθε γύρο υπάρχει τουλάχιστον μια «ζωντανή» διεργασία

ii. μια εργασία που εισάγεται σε ένα συγκεκριμένο γύρο εισάγεται σε τουλάχιστον μία ζωντανή διαδικασία. Με λίγα λόγια, δεν μας ενδιαφέρουν οι εργασίες που εισάγονται σε διεργασίες που θα καταρρεύσουν πριν το τέλος του γύρου.

3.6 Μοντέλα Πληροφόρησης

Σε σχέση με την εισαγωγή εργασιών, όπως αναφέραμε πιο πάνω, μελετάμε τις δυο εκδοχές:

(i) Κεντρικού Χρονοπρογραμματιστή (ΚΧ): Στην αρχή κάθε γύρου παρέχει σε όλες τις ζωντανές διεργασίες το τρέχον σύνολο των σε αναμονή εργασιών.

(ii) Τοπικού Εισαγωγέα (ΤΕ): Στην αρχή του κάθε γύρου η κάθε διεργασία ενημερώνει τις υπόλοιπες διεργασίες με (α) τις εργασίες που εισήχθηκαν στον προηγούμενο γύρο σε αυτή τη διεργασία, (β) όλες τις εργασίες που εκτελέστηκαν στον προηγούμενο γύρο και (γ) όλες τες εν αναμονή εργασίες.

3.7 Ορθότητα και Ολοκλήρωση

Οι δύο πιο σημαντικές ιδιότητες/εγγυήσεις που παρέχονται ανάλογα με τον αλγόριθμο:

i. Ορθότητα: Ένας αλγόριθμος είναι σωστός εάν για κάθε εκτέλεση του αλγορίθμου σε αποδεκτό εχθρικό μοτίβο, για κάθε εισαγωγή εργασιών και κάθε γύρο μετά την εισαγωγή, υπάρχει μια διεργασία ζωντανή η οποία να γνωρίζει την εργασία, εκτός και εάν η εργασία έχει ήδη εκτελεστεί. Με λίγα λόγια μια εργασία δεν χάνεται από το σύστημα εκτός και αν εκτελεστεί

ii. Ολοκλήρωση: Άπειρη εκτέλεση ενός αλγορίθμου κάτω από ένα εχθρικό μοτίβο, όπου εργασία τελικά θα πραγματοποιηθεί.

3.8 Αποτίμηση Πολυπλοκότητας

Ορίζουμε ως εν αναμονή εργασία την εργασία η οποία εισήχθη στο σύστημα αλλά δεν έχει ακόμη εκτελεστεί από οποιαδήποτε διεργασία.

Πολυπλοκότητα βάσει εν αναμονή εργασιών ανά γύρο είναι το σύνολο των εν αναμονή εργασιών κατά την έναρξη κάθε γύρου r. Στην εργασία αυτή χρησιμοποιούμε πολυπλοκότητα βάσει εν αναμονή εργασιών ανά γύρο.

Κεφάλαιο 4

Αλγόριθμοι Διασφάλισης Ορθότητας

Στο κεφάλαιο αυτό παρουσιάζεται το πρόβλημα και ειδικότερα η ανάπτυξη λύσεων οι οποίες διασφαλίζουν την Ορθότητα, αλλά όχι κατ 'ανάγκην και την Ολοκλήρωση (η ιδιότητα αυτή παρουσιάζεται στο επόμενο κεφάλαιο). Θεωρούμε ότι οι εργασίες είναι μοναδιαίου μήκους (ενός γύρου) και δεν υπάρχει κανένας περιορισμός ως προς το αριθμό μηνυμάτων που μπορούν να αποσταλούν σε κάθε γύρο. Για παράδειγμα, μια διεργασία θα μπορούσε να στείλει ένα μήνυμα σε κάθε άλλη διαδικασία, σε κάθε γύρο.

4.1 Αλγόριθμος Κεντρικού Χρονοπρογραμματισμού

Ο αλγόριθμος Κεντρικού Χρονοδρομολογητή, ο αλγόριθμος με την πιο περιορισμένη επικοινωνία ο οποίος δεν απαιτεί την αποστολή μηνυμάτων μεταξύ των διεργασιών αποτελείται στην ουσία από τρία απλά βήματα. Παραθέτουμε τον αλγόριθμο *AlgCS* για μια διεργασία *i* και γύρο *r*.

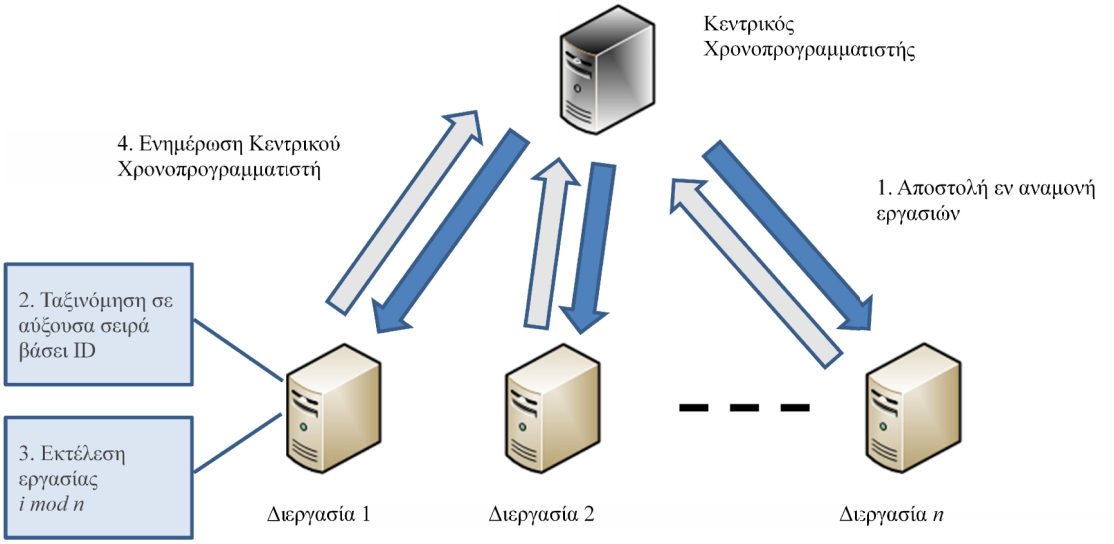
Αλγόριθμος AlgCS*(i,r)*
• Παραλαβή συνόλου των εν αναμονή εργασιών από τον χρονοπρογραμματιστή. • Ταξινόμηση εργασιών σε αύξουσα σειρά βάσει του id της κάθε εργασίας (τ.id για κάθε τ). • Εκτέλεση εργασίας με κατάταξη *i mod n.*

Όπως ήδη αποδείχτηκε ο κάθε αλγόριθμος έχει ανταγωνιστική πολυπλοκότητα βάσει των εν αναμονή εργασιών τουλάχιστον $k + n / 3$ σε εχθρικό μοτίβο k το οποίο ικανοποιεί το *t-survivability*, για κάθε μη αρνητικό ακέραιο *k, t*.

Επίσης όπως αναφέραμε στο υποκεφάλαιο 2.2 αποδείχτηκε ότι ο αλγόριθμος *AlgCS* επιτυγχάνει ανταγωνιστική πολυπλοκότητα βάσει των εν αναμονή εργασιών το πολύ *OPT* + *2n* έναντι οποιουδήποτε αποδεκτού εχθρικού μοτίβου.

Παρατηρήθηκε επίσης ότι κανείς δεν μπορεί να επιτύχει καλύτερη ανταγωνιστικότητα από τον αλγόριθμο *AlgCS* (ή κάποιο άλλο αλγόριθμο) που χρησιμοποιεί την πλήρη επικοινωνία.

Στο πιο κάτω σχήμα (*Σχήμα **1***) περιγράφεται σε βήματα ένας ολοκληρωμένος γύρος μεταξύ κεντρικού χρονοδρομολογητή και «Διεργασίας 1». Ένας ολοκληρωμένος γύρος αποτελείται από τέσσερα βήματα. Στο πρώτο βήμα αποστέλλονται οι εν αναμονή εργασίες από τον κεντρικό χρονοδρομολογητή στις διεργασίες. Μετά την παραλαβή τους από τις διεργασίες ακολουθεί η ταξινόμηση τους και η εκτέλεση εργασιών. Τέλος οι διεργασίες ενημερώνουν τον κεντρικό χρονοδρομολογητή για την εργασία που εκτέλεσαν.



Σχήμα 1: Περιγραφή λειτουργίας αλγορίθμου AlgCS

4.2 Αλγόριθμος Τοπικού Εισαγωγέα

Σε αυτό το μέρος περιγράφεται ο αλγόριθμος Τοπικού Εισαγωγέα *AlgLI* όπου δεν γίνεται χρήση οιασδήποτε κεντρικής αρχής, όπως καθορίζεται και πιο κάτω, για διεργασία *i* και ο γύρο *r*.

Ενδιάμεσος Αλγόριθμος AlgLI*(i,r)*
• Εισαγωγή συνόλου με τις νέες εργασίες από τον τοπικό εισαγωγέα εργασιών και διατήρηση στο σύνολο *new* (προηγείται διαγραφή οιασδήποτε παλιάς πληροφορίας) . Παραλαβή μηνυμάτων από κάθε άλλη διεργασία που αποστάληκε στον προηγούμενο γύρο *r - 1*. • Ενημέρωση συνόλου *old* βάσει των ληφθέντων μηνυμάτων. Το νέο σύνολο *old* θα περιέχει την ένωση (*union*) όλων των ληφθέντων πακέτων *old* και *new* μείον τις ήδη εκτελεσμένες εργασίες που έχουν αναφερθεί. • Εκτέλεση εργασίας βάσει της πιο κάτω πολιτικής: αν το σύνολο $old \neq 0$ ταξινόμηση του συνόλου *old* βάσει των *id* και εκτέλεση της εργασίας $i \bmod |old|$. Αλλιώς εάν το σύνολο $new \neq 0$ εκτέλεση της εργασίας με τη μικρότερη κατάταξη στο σύνολο *new*. • Αποστολή συνόλων *new*, *old* και του *id* της εργασίας που εκτελέστηκε σε αυτό το γύρο σε όλες τις διεργασίες.

Σε κάθε γύρο r, κάθε διεργασία *i* διατηρεί δύο σύνολα, *new* και *old*. Το σύνολο «*new*» περιέχει όλες τις νέες εργασίες που εισήχθηκαν σε αυτό το γύρο. Το σύνολο «*old*» περιέχει όλες τις εργασίες που έχουν εισαχθεί στο σύστημα σε προηγούμενους γύρους (όχι απαραίτητα από αυτή τη διεργασία), αλλά ακόμη δεν έχουν επιβεβαιωθεί ότι έχουν εκτελεστεί.

Σε κάθε εκτέλεση του αλγορίθμου *AlgLI*, εργασίες οι οποίες εισάγονται δυναμικά στο γύρο γνωστοποιούνται σε όλες τις διεργασίες που είναι ζωντανές κατά την έναρξη του γύρου $r + 1$, κάτω από οποιαδήποτε εχθρικό μοντέλο.

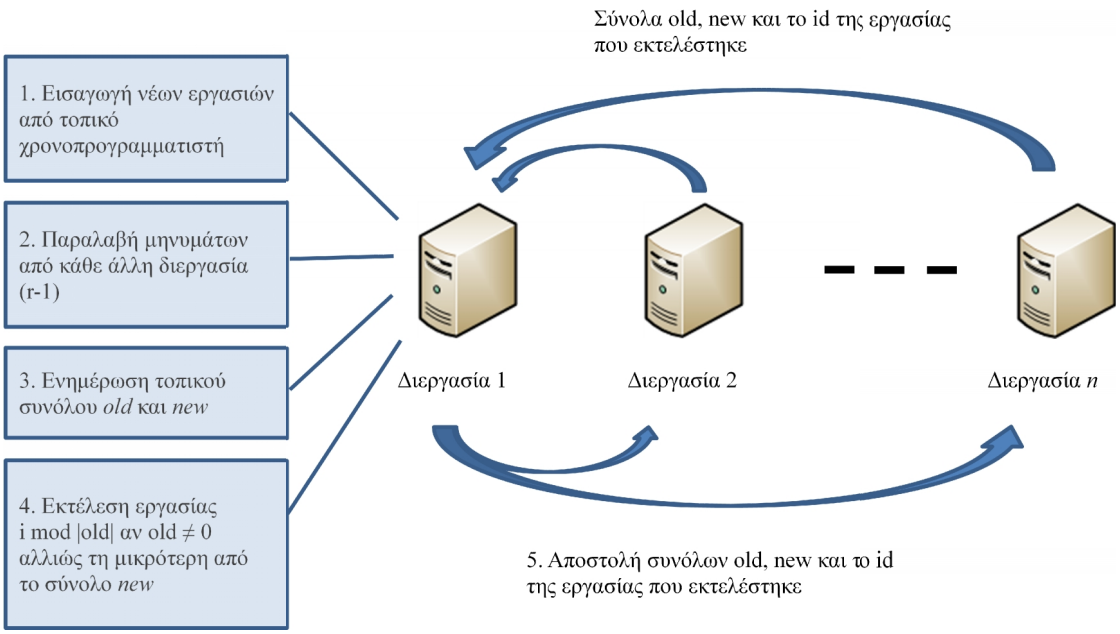
Στην αρχή του κάθε γύρου οι διεργασίες κατέχουν κοινή πληροφορία για το σύνολο των εν αναμονή εργασιών. Αυτό όμως προϋποθέτει ότι υπάρχει αξιόπιστη πολυεκπομπή [22]. Εάν

μια διεργασία καταρρεύσει, ενώ αποστέλλει ένα μήνυμα, τότε πρέπει ή όλες οι διεργασίες (που είναι ζωντανές) να λάβουν το μήνυμα ή καμία.

Λόγο της υπόθεσης ότι υπάρχει αξιόπιστη πολυεκπομπή, έχουμε ως αποτέλεσμα, οι διεργασίες που είναι ζωντανές στην αρχή του κάθε γύρου *r*, έχουν την ίδια πληροφορία σχετικά με το σύνολο των εργασιών που εκκρεμούν. Έτσι καμία εργασία δε θα χαθεί και η κάθε εργασία θα παραμείνει στο σύστημα μέχρι να εκτελεστεί.

Τέλος ο αλγόριθμος *AlgLI*, όπως αποδείχτηκε επιτυγχάνει ανταγωνιστικότητα *OPT + 3n* έναντι του βέλτιστου αλγορίθμου σε οποιουδήποτε εχθρικό μοτίβο.

Στο πιο κάτω σχήμα (*Σχήμα **2***) περιγράφεται σε βήματα ένας ολοκληρωμένος γύρος για τη «Διεργασία 1». Ένας ολοκληρωμένος γύρος αποτελείται από πέντε βήματα. Ο γύρος αρχίζει με την εισαγωγή των νέων εργασιών από τον τοπικό εισαγωγέα και την παραλαβή των μηνυμάτων που στάλθηκαν στον προηγούμενο γύρο από τις άλλες διεργασίες. Ακολουθεί η ενημέρωση των τοπικών συνόλων και η εκτέλεση των εργασιών βάσει της πολιτικής εκτελέσεων. Τέλος η κάθε διεργασία ενημερώνει όλες τις άλλες διεργασίες για τις νέες εργασίες που εισήχθησαν στον γύρο αυτό καθώς και για την εργασία που εκτέλεσε.



Σχήμα 2: Περιγραφή λειτουργίας αλγορίθμου AlgLI

Κεφάλαιο 5

Αλγόριθμοι Διασφάλισης Ολοκλήρωσης

Στο κεφάλαιο αυτό θα εστιάσουμε την προσοχή μας στο πρόβλημα της Διασφάλισης της Ολοκλήρωσης. Βάσει των προηγουμένων που ειπώθηκαν διασφαλίζεται η Ολοκλήρωση μόνο σε εκτέλεση μη φραγμένης διάρκειας.

5.1 Αλγόριθμος Κεντρικού Χρονοπρογραμματισμού

Σε γενικές γραμμές, το πρόβλημα της Διασφάλισης Ολοκλήρωσης είναι πολύ πιο περίπλοκο από το πρόβλημα της Διασφάλισης Ορθότητας. Εξετάζοντας τον απλό αλγόριθμο διασφάλισης ολοκλήρωσης *LIS* (Longest-In-The-System), κάθε διεργασία εκτελεί τη μεγαλύτερη σε ηλικία εργασία που βρίσκεται στο σύστημα. Ο αλγόριθμος *LIS* μπορεί να έχει απεριόριστες εργασίες σε αναμονή κάτω από οποιοδήποτε εχθρικό μοτίβο.

Σύμφωνα με την εργασία [11] ένας αλγόριθμος διασφάλισης ολοκλήρωσης δεν πρέπει μόνο να προβλέπει ότι η κάθε εργασία τελικά θα εκτελεστεί, αλλά πρέπει επίσης να εξασφαλίζει και πρόοδο, όταν μεγάλος αριθμός εργασιών εκκρεμούν. Όπως αποδείχτηκε διασφαλίζεται η ολοκλήρωση εάν οι διεργασίες μετά από κάθε επανεκκίνηση παραμείνουν ζωντανές για τουλάχιστο δύο γύρους.

Εξετάζοντας τον αλγόριθμο *AlgCSF* όπως ορίζεται και πιο κάτω για διεργασία *i* και *r* γύρο. Κάθε διεργασία *i* διατηρεί μια μεταβλητή με την ηλικία της (ο συνολικός αριθμός των γύρων που είναι ζωντανή η διεργασία από την τελευταία επανεκκίνηση). Κατά την επανεκκίνηση η διεργασία έχει ηλικία = 0, και αυξάνεται κατά ένα στο τέλος κάθε γύρου. Επίσης οι διεργασίες ανταλλάζουν μεταξύ τους αυτή την πληροφορία.

Αλγόριθμος AlgCSF*(i,r)*

- Παραλαβή συνόλου των εν αναμονή εργασιών από το χρονοπρογραμματιστή εάν υπάρχουν στον γύρο *r - 1*.
- Ταξινόμηση εργασιών λεξικογραφικά, πρώτα βάσει των γύρων που βρίσκονται σε αναμονή(με πρώτη την πιο παλιά) και ακολούθως βάσει του *id* της κάθε εργασίας (σε αύξουσα σειρά).
- Βάσει των ληφθέντων μηνυμάτων, δημιουργία λίστας *ASure* με περιεχόμενο όλες τις διεργασίες με ηλικία $age_r(j) = 1$. Αν $age = 1$ τότε η διεργασία *i* προσθέτει και τον εαυτό της στη λίστα.
- Αν ο αριθμός των εν αναμονή εργασιών είναι μεγαλύτερος από *2n.*
    - Αν $ASure \neq 0$ τότε
        - Αν $age \neq 1$ τότε εκτέλεση εργασίας $n + i$.
        - Αλλιώς ταξινόμηση *ASure* βάσει των *ids* και εκτέλεση εργασίας $rank(i)_{ASure}$ ($i^{η}$ εργασία στο σύνολο *ASure*) .
    - Αλλιώς αν $ASure = 0$ τότε
        - Αν $age \neq 0$ δημιουργία λίστας Recved που περιέχει όλες τις διεργασίες από τις οποίες έχει ληφθεί μήνυμα στην αρχή του γύρου. Η διεργασία *i* προσθέτει τον εαυτό της στη λίστα. Ακολούθως ταξινομείται η λίστα *Recved* λεξικογραφικά πρώτα βάσει ηλικίας και μετά βάσει του *id* (σε αύξουσα σειρά). Αν $rank(i)_{Recved} = 1$ τότε εκτελείται η πρώτη σε σειρά εργασία αλλιώς η εργασία $i + 1$.
- Αλλιώς εκτελείται η πρώτη σε κατάταξη εργασία (πιο παλιά).
- Αύξηση ηλικίας $age = age + 1$.
- Αποστολή μηνύματος με την ηλικία $age_{r+1}(i)$ σε όλες τις διεργασίες.

Ο αλγόριθμος *AlgCSF* διασφαλίζει την ολοκλήρωση, υπό την προϋπόθεση του *2-survivability* και έχει αποδεικτή ότι ισχύουν:

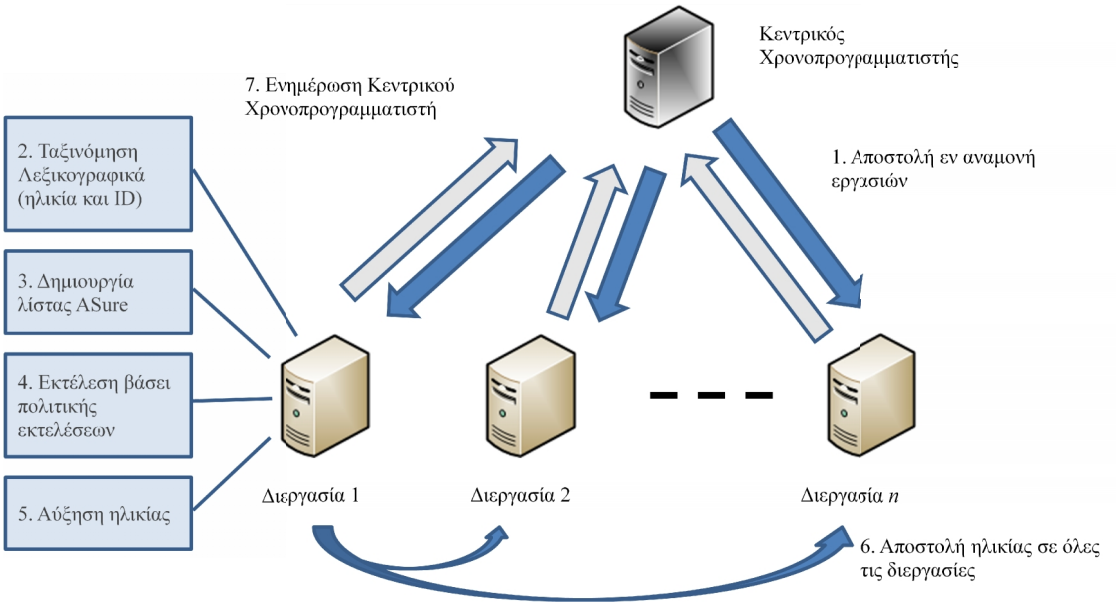
i. Σε γύρο *r,* t_{old} η παλαιότερη εργασία σε αναμονή στο σύστημα (*rank 1*) και τουλάχιστον μία διεργασία με ηλικία $age_r = 1$, τότε η t_{old} θα εκτελεστεί στο γύρο *r*. Αν σε γύρο *r* δεν υπάρχει διεργασία με $age = 1$, αλλά υπάρχει τουλάχιστον μια με $age = 0$, τότε ακόμα και αν δεν εκτελεστεί στο γύρο *r*, θα πραγματοποιηθεί στο γύρο $r + 1$.

ii. Αν σε γύρο *r* όλες οι ζωντανές διεργασίες είναι ηλικίας *age > 1 (ASure = 0)* και τ_{old} η παλαιότερη εργασία του συστήματος, τότε θα πρέπει να εκτελεστούν το αργότερο μέχρι το γύρο *r + 2n*.

Από τα πιο πάνω συμπεραίνουμε ότι ο αλγόριθμος *AlgCSF* διασφαλίζει την ολοκλήρωση κάτω από οποιοδήποτε εχθρικό μοτίβο με τον περιορισμό *2-survivability*.

Επίσης ο αλγόριθμος *AlgCSF* βάσει των εν αναμονή εργασιών επιτυγχάνει πολυπλοκότητα σε σχέση με το βέλτιστο, το πολύ *OPT + 3n* κάτω από οποιοδήποτε εχθρικό μοτίβο.

Στο πιο κάτω σχήμα (*Σχήμα 3*) περιγράφεται σε βήματα ένας ολοκληρωμένος γύρος μεταξύ κεντρικού χρονοδρομολογητή και «Διεργασίας 1». Ένας ολοκληρωμένος γύρος αποτελείται από έξι βήματα. Στο πρώτο βήμα αποστέλλονται οι εν αναμονή εργασίες από τον κεντρικό χρονοδρομολογητή στις διεργασίες. Μετά την παραλαβή των εργασιών ακολουθεί η λεξικογραφική τους ταξινόμηση και η δημιουργία λίστας ASure με τις διεργασίες που είναι η ηλικία τους είναι ίση με 1. Ακολουθεί η εκτέλεση των εργασιών βάσει της πολιτικής εκτελέσεων. Τέλος η διεργασία αυξάνει την ηλικία της και ενημερώνει τον κεντρικό χρονοδρομολογητή για την εργασία που εκτέλεσε και τις άλλες διεργασίες για την ηλικία της.



Σχήμα 3: Περιγραφή λειτουργίας αλγορίθμου AlgCSF

5.2 Αλγόριθμος Τοπικού Εισαγωγέα

Ο αλγόριθμος Τοπικού Εισαγωγέα συνδυάζει το μηχανισμό που χρησιμοποιήθηκε στον αλγόριθμο *AlgLI* για την εισαγωγή εργασιών με ένα γύρο καθυστέρηση και την πολιτική χρονοπρογραμματισμού του αλγόριθμου *AlgCSF* έτσι ώστε να διασφαλιστεί η ολοκλήρωση. Το αξιόπιστη πολυεκπομπή προϋποτίθεται και πάλι για τη διασφάλιση, ότι οι διαδικασίες που διατηρούν ίδια σύνολα για τις εν αναμονή εργασίες.

Η ανταγωνιστικότητά του είναι η ίδια με την ανταγωνιστικότητα του *AlgCSF* συν έναν πρόσθετο παράγοντα *n* που προέρχεται από την καθυστέρηση της διάδοσης των νέων εργασιών κατά ένα γύρο. Συγκεκριμένα, ο αλγόριθμος *AlgLIF*, θεωρώντας αξιόπιστο *multicast*, διασφαλίζει την ολοκλήρωση και επιτυγχάνει πολυπλοκότητα σε σχέση με τον ανταγωνισμό το πολύ *OPT + 4n* εναντίον οποιουδήποτε εχθρικού μοτίβου με *2-survivability*. Με την ίδια λογική όπως και στους υπόλοιπους αλγόριθμους, συνάγεται ότι ο *AlgLIF* είναι σχεδόν βέλτιστος.

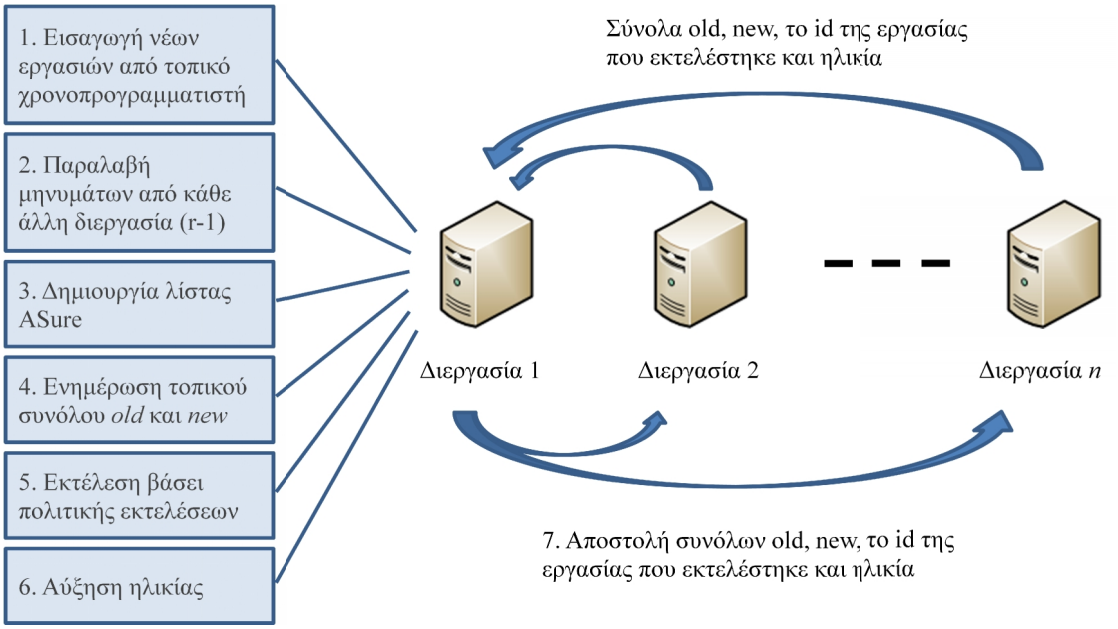
Ενδιάμεσος Αλγόριθμος AlgLIF*(i,r)*

- Εισαγωγή συνόλου με τις νέες εργασίες από τον τοπικό εισαγωγέα εργασιών και διατήρηση στο σύνολο *new* (προηγείται διαγραφή οιασδήποτε παλιάς πληροφορίας) .
  Παραλαβή μηνυμάτων από τον προηγούμενο γύρο *r - 1*. (Για κάθε διεργασία *j* η διεργασία *i* παραλαμβάνει τα σύνολα *old*, *new*, την εργασία t_j που εκτελέστηκε και την ηλικία της διεργασίας $age_r(j)$)
- Βάσει των ληφθέντων μηνυμάτων, δημιουργία λίστας *ASure* με περιεχόμενο όλες τις διεργασίες με ηλικία $age_r(j) = 1$. Αν *age = 1* τότε η διεργασία *i* προσθέτει και τον εαυτό της στη λίστα.
- Ενημέρωση συνόλου *old* βάσει των ληφθέντων μηνυμάτων. Το νέο σύνολο *old* θα περιέχει την ένωση (*union*) όλων των ληφθέντων πακέτων *old* και *new* μείον τις ήδη εκτελεσμένες εργασίες που έχουν αναφερθεί.
- Ταξινόμηση του συνόλου old λεξικογραφικά, πρώτα βάσει της περιόδου που βρίσκονται οι εργασίες σε αναμονή (με την πιο παλιά στην πρώτη θέση) και μετά βάσει των ids(σε αύξουσα σειρά).
- Αν ο αριθμός των εργασιών στο σύνολο old είναι μεγαλύτερος από *2n*.

- Αν *ASure* ≠ *0* τότε
  - Αν *age* ≠ *1* τότε εκτέλεση εργασίας *n* + *i*.
  - Αλλιώς ταξινόμηση *ASure* βάσει των *ids* και εκτέλεση εργασίας $rank(i)_{ASure}$.
- Αλλιώς αν *ASure* = *0* τότε
  - Αν *age* ≠ *0* δημιουργία λίστας Recved που περιέχει όλες τις διεργασίες από τις οποίες έχει ληφθεί μήνυμα στην αρχή του γύρου. Η διεργασία *i* προσθέτει τον εαυτό της στη λίστα. Ακολούθως ταξινομείται η λίστα *Recved* λεξικογραφικά πρώτα βάσει ηλικίας και μετά βάσει του *id* (σε αύξουσα σειρά). Αν $rank(i)_{Recved} = 1$ τότε εκτελείται η πρώτη σε σειρά εργασία στο σύνολο *old* αλλιώς η εργασία *i* + *1*.

- Αλλιώς εάν ο αριθμός των εργασιών στο σύνολο old είναι μικρότερος από *2n.*
  - Εάν *old* ≠ *0* και *new* ≠ *0* τότε από το ταξινομημένο σε αύξουσα σειρά σύνολο *new* εκτελείται η εργασία με τη μικρότερη κατάταξη εργασία.
  - Αλλιώς εκτελείται από το σύνολο *old* η εργασία *i* + *1*.
- Αύξηση ηλικίας *age* = *age* + *1*.
- Αποστολή συνόλων *new*, *old*, του *id* της εργασίας που εκτελέστηκε σε αυτό το γύρο και ή ηλικία *age* της διεργασίας σε όλες τις άλλες διεργασίες.

Στο πιο κάτω σχήμα (*Σχήμα 4Σχήμα **3***) περιγράφεται σε βήματα ένας ολοκληρωμένος γύρος για τη «Διεργασία 1». Ένας ολοκληρωμένος γύρος αποτελείται από επτά βήματα. Ο γύρος αρχίζει με την εισαγωγή των νέων εργασιών από τον τοπικό εισαγωγέα και την παραλαβή των μηνυμάτων που στάλθηκαν στον προηγούμενο γύρο από τις άλλες διεργασίες. Μετά την παραλαβή των εργασιών και την ενημέρωση των τοπικών συνόλων ακολουθεί η λεξικογραφική τους ταξινόμηση και η δημιουργία λίστας ASure με τις διεργασίες που είναι η ηλικία τους είναι ίση με 1. Ακλουθεί η εκτέλεση εργασιών βάσει την πολιτικής εκτελέσεων. Τέλος η διεργασία αυξάνει την ηλικία της και ενημερώνει όλες τις άλλες διεργασίες για τις νέες εργασίες που εισήχθησαν στον γύρο αυτό καθώς και για την εργασία που εκτέλεσε και την ηλικία της.



Σχήμα 4: Περιγραφή λειτουργίας αλγορίθμου AlgLIF

Κεφάλαιο 6

Υλοποίηση

Στο κεφάλαιο αυτό περιγράφονται αρχικά όλες κλάσεις οι οποίες είναι απαραίτητες για την ορθή και βέλτιστη λειτουργία των αλγορίθμων. Όλες οι κλάσεις υλοποιήθηκαν βάσει των προδιαγραφών των αλγορίθμων και δεν παρέκλιναν από αυτές για κανένα λόγο. Ιδιαίτερη έμφαση δόθηκε στις συναρτήσεις ταξινόμησης όπου ο στόχος κύριος στόχος ήταν η ελαχιστοποίηση υπολογιστικού κόστους.

Ακολούθως περιγράφονται οι υλοποιήσεις των ίδιων των αλγορίθμων καθώς και μηχανισμοί οι οποίοι υλοποιήθηκαν για την ορθή λειτουργία τους όπως (α) η αποστολή και παραλαβή μηνυμάτων και (β) η επιλογή και εκτέλεση εργασιών.

6.1 Βασικά Κλάσεις

Στην ενότητα αυτή περιγράφονται τα διάφορες βασικές κλάσεις, απαραίτητες για τη λειτουργία της όλης εφαρμογής.

Όλες οι κλάσεις ακλουθούν τις προδιαγραφές όπως αναφέρονται και στο σχετικό άρθρο για την αποφυγή αποκλίσεων από την επιθυμητή λύση.

Κλάση DemoTask

Η κλάση *DemoTask* προσομοιώνει τις εργασίες. Οι εργασίες δημιουργούνται δυναμικά και αποστέλλονται σε πακέτα τύπου *«Package»* στις διεργασίες (ή αλλιώς nodes όπως αναφέρονται και στην εφαρμογή).

Η κλάση υλοποίει τη διεπαφή *serializable* η οποία προσφέρει τη δυνατότητα σειριοποίησης των δεδομένων. Απαραίτητο χαρακτηριστικό για την αποστολή των αντικειμένων μέσω δικτύου.

Επίσης η κλάση υλοποιεί τη διεπαφή *comparable* έτσι ώστε να υπάρχει δυνατότητα επανεγγραφής της πολιτικής ταξινόμησης όπου χρησιμοποιήθηκαν *hashes* των αντικειμένων. Πρέπει να αναφέρουμε ότι σημειώθηκε σημαντική βελτίωση κατά την ταξινόμηση ειδικά σε περιπτώσεις όπου η λίστα των εν αναμονή εργασιών ήταν σχετικά μεγάλη.

```
@Override
public int hashCode() {
   int hash = 7;
   hash = 37 * hash + (this.injector != null ?
   this.injector.hashCode() : 0);
   hash = 37 * hash + this.seqNum;
   return hash;
}

public int compareTo(Object obj) {

    if (obj instanceof DemoTask) {
        DemoTask demoTask = (DemoTask) obj;
        if (this.hashCode() == demoTask.hashCode()) {
            return 0;
        } else if (this.hashCode() > demoTask.hashCode()) {
            return 1;
        } else if (this.hashCode() < demoTask.hashCode()) {
            return -1;
        }
    }
    return 0;
}
```

Το κάθε αντικείμενο τύπου *DemoTask* διατηρεί ένα μοναδικό αριθμό αναγνώρισης (*Sequence Number*), το όνομα της διεργασίας (*injector*) που το εισήγαγε στο σύστημα και τέλος ένα σηματοφόρο (*IsFinished*) ο οποίος υποδεικνύει αν η εργασία έχει εκτελεστεί ή όχι.

Κλάση NodeList

Η κλάση *NodeList* η οποία επεκτείνει την κλάση *«LinkedList»* (Συνδεδεμένες Λίστες) χρησιμοποιείται για τη διατήρηση λίστας τοπικά στην κάθε διεργασία με όλες τις διεργασίες.

Επίσης αντικείμενο τύπου *NodeList* χρησιμοποιείται για τη δημιουργία των λιστών *ASure* στους αλγορίθμους *AlgCSF* και *AlgLIF*.

Κλάση Package

Τα αντικείμενα τύπου *Package* είναι στην ουσία τα πακέτα τα οποία αποστέλλονται μεταξύ διεργασιών. Η κλάση υλοποιεί τη διεπαφή *serializable* η οποία προσφέρει τη δυνατότητα σειριοποίησης των δεδομένων. Απαραίτητο χαρακτηριστικό για την αποστολή των αντικειμένων μέσω δικτύου.

Η κλάση Package η επεκτείνει την κλάση *«LinkedList»* έτσι ώστε τα πακέτα να κληρονομούν όλα τα πλεονεκτήματα των συνδεδεμένων λιστών και να τυχαίνουν μεταχείρισης σαν να είναι λίστες.

Επίσης τα πακέτα κατά τη δημιουργία τους σηματοδοτούνται με τον τύπο της πληροφορίας που θα περιέχουν όπως *New*, *Pending*, *Completed*, *Old* και *Info*. Κατά την εκτέλεση διατηρούν ένα μοναδικό αριθμό αναγνώρισης.

Τέλος στην κλάση Package υλοποιήθηκαν συναρτήσεις για τομή (*intersection*) και ένωση (*union*) όπως προϋποθέτουν οι αλγόριθμοι.

```
public void union(Package o) {
    for (DemoTask dt : o) {
        if (!this.contains(dt)) {
            this.add(dt);
        }
    }
}
```

```
public void intersection(Package o) {
    for (DemoTask dt : o) {
        if (this.contains(dt)) {
            this.remove(dt);
        }
    }
}
```

Κλάση Recved

Η κλάση *Recved* είναι μια παραλλαγή της κλάσης *NodeList* με επιπρόσθετα χαρακτηριστικά.

Η διαφορά της κλάσης Recved βρίσκεται στην υλοποίηση της διεπαφής *Comparable* που όπως προϋποθέτει ο αλγόριθμος *AlgLIF* πρέπει να λαμβάνεται υπόψη η ηλικία. Πραγματοποιείται ταξινόμηση κατά ηλικία και μετά λεξικογραφική βάσει ονόματος.

```
public int compareTo(Object o) {
    Recved temp = (Recved) o;
    Integer age1 = this.getAge();
    Integer age2 = temp.getAge();
    int ageComp = age1.compareTo(age2);
    if (ageComp != 0) {
        return ageComp;
    } else {
        String node1 = this.getNode();
        String node2 = temp.getNode();
        return node1.compareTo(node2);
    }
}
```

6.2 Αλγόριθμοι Διασφάλισης Ορθότητας

Στο υποκεφάλαιο αυτό περιγράφονται οι υλοποιήσεις των αλγορίθμων AlgCS και AlgLI οι οποί διασφαλίζουν την ορθότητα.

6.2.1 Αλγόριθμος Κεντρικού Χρονοπρογραμματισμού

Ο αλγόριθμος κεντρικού χρονοπρογραμματισμού *AlgCS*, είναι ο πιο απλός σε γενικές γραμμές βάσει του τρόπου λειτουργίας του. Στο στάδιο της υλοποίησης όμως θέτει τις βάσεις για την υλοποίηση και των υπόλοιπων αλγορίθμων.

Λόγω φιλοσοφίας της YALPS έπρεπε μια διεργασία να έχει το ρόλο του κεντρικού χρονοπρογραμματιστή. Έτσι μια προκαθορισμένη διεργασία (*Node1*) εκτελεί το μέρος του χρονοπρογραμματιστή σε αντίθεση με τις υπόλοιπες.

Για λόγους ευκολίας χρησιμοποιήθηκαν δύο περιοδικοί γύροι της YALPS για να αναπαραστήσουν ένα γύρο του *AlgCS*, ένα γύρο για τον κεντρικό χρονοπρογραμματιστή και ένα για τις υπόλοιπες διεργασίες που αναλαμβάνουν την εκτέλεση των διεργασιών όπως φαίνεται και πιο κάτω.

```
class SchedulerTask extends BasicPeriodicTask {

   @Override
   protected void periodicRun() {
      stepNum++;
      if ((stepNum % 2) == 1) {
         roundNum++;
         ...
      }
   }
}

class SimpleNodeTask extends BasicPeriodicTask {

   @Override
```

```
    protected void periodicRun() {
      stepNum++;
      if ((stepNum % 2) == 0) {
        roundNum++;
        ...
      }
    }
  }
```

Το επόμενο κύριο μέρος του αλγορίθμου *AlgCS* είναι η συνάρτηση receiver όπου κατά την εκτέλεση σε ανεξάρτητο νήμα *(thread)* σε κάθε διεργασία και αναλαμβάνει την παραλαβή μηνυμάτων. Και σε αυτό το μέρος ήταν αναγκαία η διαφορετική υλοποίηση για το χρονοπρογραμματιστή όπου παραλαμβάνει πακέτα από τις υπόλοιπες διεργασίες που περιέχουν πληροφορίες σχετικά με τις εργασίες που εκτέλεσαν. Σε αντίθεση με το χρονοπρογραμματιστή οι υπόλοιπες διεργασίες παραλαμβάνουν πακέτα με τις εν αναμονή εργασίες όπως φαίνεται πιο κάτω

```
  @Override
  public boolean receive(short header, Object msg, NodeID sender){
    Package m = new Package();
    m = (Package) msg;

    // Scheduler receive package
    if (getNodeStr().equals(schedulerStr) == true) {
      ...
    }
    // Node receive package
    If((getNodeStr().equals(schedulerStr) == false) &&
    (sender.toString().equals(schedulerStr) == true)) {
      ...
    }
    return true;
  }
```

Ο κεντρικός χρονοπρογραμματιστής στέλνει στις διεργασίες ένα πακέτο τύπου *Pending* αν υπάρχουν εργασίες σε αναμονή και οι διεργασίες απαντούν με ένα πακέτο που περιέχει το μοναδικό αριθμό αναγνώρισης της εργασίας που εκτέλεσαν, αν εκτέλεσαν.

6.2.2 Αλγόριθμος Τοπικού Εισαγωγέα

Ο αλγόριθμος τοπικού εισαγωγέα *AlgLI*, με το πιο χαλαρό μοντέλο πληροφόρησης σε σχέση με τον προηγούμενο παρουσιάζει μεγάλες διαφορές στη δομή λόγω της απουσίας του κεντρικού χρονοπρογραμματιστή.

Ο χρονοπρογραμματιστής και ο εκτελεστής εργασιών βρίσκονται και οι δύο στην ίδια κλάση αλλά εκτελούνται εναλλάξ με τον ίδιο μηχανισμό όπως και στους προηγούμενους αλγορίθμους όπως φαίνεται πιο κάτω.

```
class SimpleNodeTask extends BasicPeriodicTask {

  @Override
  protected void periodicRun() {
    stepNum++;

    // Local Scheduler
    if ((stepNum % 2) == 1) {
      ...
    }

    // Performer
    if ((stepNum % 2) == 0) {
      if (oldTasks.size() > 0) {
        ...
        int taskIdForProcessing = ((tempNodeId - 1) %
        oldTasks.size());
        ...
      } else if (newTasks.size() > 0) {
        DemoTask taskForProcessing = newTasks.getFirst();
        ...
    }
    ...
  }
}
```

Το επόμενο κύριο μέρος του αλγορίθμου *AlgLI*, είναι η συνάρτηση receiver, ο οποίος αναλαμβάνει την παραλαβή μηνυμάτων από τις υπόλοιπες διεργασίες. Ο *AlgLI*, σε γενικές

γραμμές έχει τον πιο απλό *«Receiver»* καθώς οι διεργασίες παραλαμβάνουν τριών ειδών πακέτα *New*, *Old* και *Completed* τα οποία περιέχουν τις νέες εργασίες, τις παλιές και της εργασίες που εκτελέστηκαν στον προηγούμενο γύρο αντίστοιχα. Τα τρία αυτά σύνολα διατηρούνται από τις διεργασίες και επεξεργάζονται αναλόγως σύμφωνα με την προδιαγραφή του αλγόριθμου.

```
@Override
public boolean receive(short header,Object msg,NodeID sender){

    Package m = new Package();
    m = (Package) msg;

    if (nodeIsDown == false){
       if (m.size() > 0){

          if (m.getPackageType() == PackageType.NEW){
             ...
          } else if (m.getPackageType() == PackageType.OLD){
             ...
          } else if (m.getPackageType()== PackageType.COMPLETED){
             ...
          }
       }
    }
    return true;
}
```

6.3 Αλγόριθμοι Διασφάλισης Ολοκλήρωσης

Οι αλγόριθμοι οι οποίοι διασφαλίζουν και την ολοκλήρωση στην ουσία, από άποψη υλοποίησης, δε διαφέρουν κατά πολύ στην λογική από τους αλγόριθμους που διασφαλίζουν μόνο την ορθότητα. Όπως περιγράφεται πιο κάτω οφείλουν το χαρακτηριστικό τους αυτό στην πολιτική επιλογής και εκτέλεσης των εργασιών.

6.3.1 Αλγόριθμος Κεντρικού Χρονοπρογραμματισμού

Ο αλγόριθμος κεντρικού χρονοπρογραμματισμού *AlgCSF*, ο οποίος διασφαλίζει και την ολοκλήρωση αποτελεί μια επέκταση του *AlgCS* που αναφέραμε πιο πάνω. Για να είναι εφικτή η διασφάλιση της ολοκλήρωσης είναι αναγκαία η επικοινωνία μεταξύ διεργασιών για ανταλλαγή πληροφοριών. Η ανταλλαγή πληροφοριών πραγματοποιείται με την αποστολή ενός πακέτου τύπου *Info*.

```
for (NodeID neighbor : nodes) {
  try {
    if (neighbor.toString().equals(getNodeStr()) == false) {
      if (neighbor.toString().equals(schedulerStr) == true) {
        getCommLayer().sendUDP(tasksPackage, neighbor);
        ...
      } else {
        getCommLayer().sendUDP(infoPackage, neighbor);
        ...
      }
    }
  } catch (IOException e) {
    e.printStackTrace();
  }
}
```

Οι διεργασίες με τη σειρά τους βάσει της πληροφορίας που κατέχουν στον κάθε γύρο δημιουργούν δύο λίστες, την ASure που περιέχει όλες τις διεργασίες που η ηλικία τους είναι ίση με 1 και την λίστα *Recved* που περιέχει όλες τις διεργασίες από τις οποίες έχει παραλάβει μήνυμα στην αρχή του τρέχοντος γύρου.

```
// NodeList LinkedList – Asure
private NodeList lstASure = new NodeList();

private LinkedList<Recved> lstRecved = new
LinkedList<Recved>();
```

Όπως προαναφέρθηκε το χαρακτηριστικό της διασφάλισης της ολοκλήρωσης οφείλεται στην πολιτική επιλογής και εκτέλεσης των εργασιών. Όπως φαίνεται και πιο κάτω, πρώτα

δημιουργούνται οι λίστες *ASure* και *Recved*, αναγκαίες για την εκτέλεση. Ακολούθως ταξινομούνται οι διεργασίες στη λίστα *ASure* και οι εν αναμονή εργασίες. Η λίστα *Recved* δεν ταξινομείται προς το παρόν καθώς υπάρχει πιθανότητα να μη χρησιμοποιηθεί.

```
if (infoPackage.getNodeAge() == 1) {
  lstASure.add(getNodeStr());
}
Recved nodeObjInRecved = new Recved(getNodeStr(),
infoPackage.getNodeAge());
lstRecved.add(nodeObjInRecved);

Collections.sort(lstASure);
Collections.sort(tasksPackage);
...
```

Στην συνέχεια ακολουθείται πιστά η πολιτική επιλογής εργασίας όπως περιγράφεται στο σχετικό άρθρο [11] και τέλος η εκτέλεσή της.

```
// If the number of pending tasks is larger than 2n then
if (tasksPackage.size() > (nodes.size() * 2)) {
  if (lstASure.size() != 0) { // If ASure != 0 then
    if (infoPackage.getNodeAge() != 1) { // If age != 1
      // Perform task with rank n + i
      taskIdForProcessing = (nodes.size() - 1) + tempNodeId;

    } else {
      // Else rank processes in ASure based on their ids and
      perform task
      // with rank rank(i)ASure (i.e., ith task in set ASure)
      taskIdForProcessing = lstASure.indexOf(getNodeStr());

    }
  } else if (lstASure.size() == 0) { // Else [ASure = 0]
    if (infoPackage.getNodeAge() != 0) { // If age != 0

      Collections.sort(lstRecved); // Construct set Recved
      // If rank(i) = 1 then perform task with rank 1
      taskIdForProcessing = lstRecved.indexOf(nodeObjInRecved);
```

```
        if (taskIdForProcessing > 0) {
          // Otherwise perform task with rank i + 1
            taskIdForProcessing++;
        }
      } else { // Else perform task with rank i + 1
        taskIdForProcessing = tempNodeId + 1;
      }
    }
  } else { // Else perform task with rank 1
    taskIdForProcessing = 0;
  }
  ...
```

Στο τέλος, μετά την επιλογή, ακολουθεί η εκτέλεση της επιλεγμένης εργασίας από το σύνολο *old όπου* διατηρούνται όλες οι εν αναμονή εργασίες.

6.3.2 Αλγόριθμος Τοπικού Εισαγωγέα

Στον αλγόριθμο τοπικού εισαγωγέα *AlgLIF*, ο οποίος διασφαλίζει και την ολοκλήρωση, όπως και στον *AlgCSF*, αποτελεί μια επέκταση του *AlgLI* που αναφέραμε. Η απουσία του κεντρικού χρονοπρογραμματιστή και η προϋπάρχουσα επικοινωνία μεταξύ διεργασιών από τον *AlgLI* κάνουν τη μετατροπή πιο εύκολη. Για να είναι εφικτή η διασφάλιση της ολοκλήρωσης κατά την επικοινωνία μεταξύ διεργασιών η ανταλλαγή πακέτων *New, Old, Completed* και *Info* είναι απαιτητή.

```
for (NodeID neighbor : nodes) {
  try {
    if (neighbor.toString().equals(getNodeStr()) == false) {
      if (newTasks.size() > 0) {
        getCommLayer().sendUDP(newTasks, neighbor);
      }
      if (oldTasks.size() > 0) {
        getCommLayer().sendUDP(oldTasks, neighbor);
      }
      if (completedTasks.size() > 0) {
        getCommLayer().sendUDP(completedTasks, neighbor);
```

```
        }
        getCommLayer().sendUDP(infoPackage, neighbor);
      }
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
```

Όπως και στον *AlgCSF* οι λίστες *ASure* και *Recved* είναι αναγκαίες για την εκτέλεση. Η διαδικασία δημιουργίας και ταξινόμησης είναι ή ίδια όπως και στον *AlgCSF* όπως παρουσιάζεται και πιο κάτω.

```
if (infoPackage.getNodeAge() == 1) {
  lstASure.add(getNodeStr());
}
Recved nodeObjInRecved = new Recved(getNodeStr(),
infoPackage.getNodeAge());
lstRecved.add(nodeObjInRecved);

Collections.sort(lstASure);

...
```

Η διαδικασία επιλογής εργασίας αν και πανομοιότυπη, η προς εκτέλεση εργασία δεν είναι πάντα από το σύνολο *old* όπως στον αλγόριθμο *AlgCSF*. Η διαδικασία εκτέλεσης εμπεριέχεται στη διαδικασία επιλογής για την αντιμετώπιση του προβλήματος αυτού.

```
if (oldTasks.size() > (nodes.size() * 2)) {
  if (lstASure.size() != 0) { // If ASure != 0 then
    if (infoPackage.getNodeAge() != 1) {
      //If age != 1 then perform task in old with rank n + i
      taskIdForProcessing = (nodes.size() - 1) + tempNodeId;

    } else {
      // Else rank processes in ASure based on their ids and
      perform
      // task in old with rank rank(i)ASure
      taskIdForProcessing = lstASure.indexOf(getNodeStr());
```

```
      }
    } else if (lstASure.size() == 0) { // Else [ASure = 0]
      if (infoPackage.getNodeAge() != 0) { // If age != 0

        // Then rank processes in set Recved lexicographically,
        Collections.sort(lstRecved);

        //If rank(i)Recved=1 then perform task in old with rank 1
        taskIdForProcessing = lstRecved.indexOf(nodeObjInRecved);
        // Otherwise perform task in old with rank i + 1.
        if (taskIdForProcessing > 0) {
          taskIdForProcessing++;
        }

      } else { //Else perform task in old with rank i + 1
        taskIdForProcessing = tempNodeId + 1;
      }
    }
    if (taskIdForProcessing >= 0) {
      DemoTask taskForProcessing =
      oldTasks.get(taskIdForProcessing);
      //PROCESSING STEP
      ...
    }
  } else { // Else [old has fewer than 2n tasks]
    // if old = 0 and if new != 0
    if ((oldTasks.size() == 0) && (newTasks.size() > 0)) {
      DemoTask taskForProcessing = newTasks.getFirst();
      //PROCESSING STEP
      ...
    // Else perform task in old with rank 1.
    } else if (oldTasks.size() > 0) {
      DemoTask taskForProcessing = oldTasks.getFirst();
      //PROCESSING STEP
      ...
    }
  }
```

Κεφάλαιο 7

Πειραματική Αξιολόγηση

Στο κεφάλαιο της Πειραματική Αξιολόγησης παρουσιάζεται η όλη διαδικασία που ακολουθήθηκε για την εξαγωγή των αποτελεσμάτων. Γίνετε αναλυτική περιγραφή όλων των διαδικαστικών βημάτων από την δημιουργία των σεναρίων εκτέλεσης, την παρουσίαση των αποτελεσμάτων και τέλος την σύγκριση μεταξύ των αλγορίθμων.

7.1 Πλατφόρμα Πειραματισμού

Η συγγραφή του κώδικα πραγματοποιήθηκε στην Java 1.6.0_31 και χρησιμοποιήθηκε η βιβλιοθήκη YALPS με έκδοση 0.3 Alpha σε λειτουργικό σύστημα Windows 7 x86.

Η πειραματική διαδικασία εκτελέστηκε σε μηχανή Intel P45 και επεξεργαστή Intel Core 2 Quad Q9550 Yorkfield με ταχύτητα ρολογιού στα 2.83GHz και μνήμη 4GB RAM με ταχύτητα χρονισμού στα 1333MHz.

7.2 Μετρικές Αξιολόγησης Αλγορίθμων

Μια τέτοια πειραματική αξιολόγηση δεν μπορεί να δικαιολογήσει κανένα συμπέρασμα αν πρώτα δεν συγκρίνει τους αλγόριθμους μεταξύ τους. Για να είναι αυτό εφικτό πρέπει πρώτα να τεθεί ένα μέτρο σύγκρισης. Στο υποκεφάλαιο παρουσιάζονται οι μετρικές αξιολόγησης τις οποίες και χρησιμοποιήσαμε για τη σύγκριση και την εξαγωγή συμπερασμάτων στη συνέχεια του κεφαλαίου.

Οι μετρικές αξιολόγησης που χρησιμοποιήθηκαν είναι οι εξής πιο κάτω:

- Συμπληρωμένοι γύροι: Ο συνολικός αριθμός γύρων όπου διεργασίες βρίσκονταν εν ζωή και μπορούσαν να εκτελέσουν επιτυχώς εργασίες (αν υπήρχαν).

- Συμπληρωμένες εργασίες: Ο συνολικός αριθμός των εργασιών που έχουν εκτελεστεί επιτυχώς από όλες τις διεργασίες κατά τη διάρκεια της εκτέλεσης.

- Γύροι σε αδράνεια: Ο συνολικός αριθμός των γύρων όπου διεργασίες δεν βρίσκονταν εν ζωή κατά τη διάρκεια της εκτέλεσης, μπορούμε να τους χαρακτηρίσουμε και «νεκρούς γύρους». Εάν σε γύρο *r* πέντε από τις είκοσι διεργασίες δεν βρίσκονται εν ζωή τότε έχουμε πέντε αδρανείς γύρους.

- Χρόνος Διεκπεραίωσης: Ο συνολικός χρόνος σε δευτερόλεπτα *(sec)* που χρειάστηκε για την εκτέλεση του κάθε σεναρίου, εξαιρουμένου του χρόνου που χρειάζεται η πλατφόρμα YALPS για τη δημιουργία του περιβάλλοντος προσομοίωσης.

- Αριθμός μηνυμάτων: Ο συνολικός αριθμός μηνυμάτων που κινήθηκε στο δίκτυο από και προς τις διεργασίες.

7.3 Σενάρια

Κατά την πειραματική αξιολόγηση των αλγοριθμικών λύσεων, στόχος ήταν να δοκιμαστούν οι αλγόριθμοι σε περιβάλλον προσομοίωσης με τη χρήση «ρεαλιστικών» σεναρίων, παρέχοντας στο τέλος αποτελέσματα τα οποία να έχουν αντίκρισμα στην πραγματικότητα. Για το λόγο αυτό έχουν αποφευχθεί σενάρια στα οποία οι αλγόριθμοι θα υπολειτουργούσαν τον περισσότερο χρόνο ή θα ήταν αδύνατο λόγω υπερφόρτωσης να αντεπεξέλθουν.

Για την εκτέλεση των αλγορίθμων και τη δημιουργία των σεναρίων ορίστηκαν τέσσερις παράμετροι:

i. Ελάχιστος αριθμός εισαγωγής εργασιών ανά γύρο

ii. Μέγιστος αριθμός εισαγωγής εργασιών ανά γύρο

iii. Πιθανότητα ο κόμβος να παραμείνει ζωντανός

iv. Πιθανότητα να επανέρθει πίσω ένας μη ενεργός κόμβος

Σημαντικό είναι να αναφέρουμε ότι οι εργασίες που εισάγονταν στο σύστημα ήταν κενές εργασίες *(dummy task)* έτσι ώστε η «εκτέλεσή» τους να μην έχει αντίκτυπο στα αποτελέσματα.

Επίσης σημαντικός παράγοντας είναι η διάρκεια εκτέλεσης των αλγορίθμων στο περιβάλλον προσομοίωσης. Οι αλγόριθμοι μεταξύ τους διαφέρουν κατά πολύ στο χρόνο που χρειάζονται για την εκτέλεση ενός γύρου. Για τη δίκαιη σύγκριση των αλγορίθμων δεν τέθηκε χρονικό περιθώριο αλλά αριθμός γύρων. Μετά από αρκετές προκαταρτικές εκτελέσεις αποφασίστηκε ότι οι γύροι θα είναι πεντακόσιοι (500). Σημαντικό είναι ότι οι αλγόριθμοι μετά από ένα μικρό αριθμό γύρων όπως φαίνεται και πιο κάτω στης γραφικές παραστάσεις αποκτούν μια σταθερή κατάσταση.

Σενάριο	*Ελάχιστος αριθμός νέων εργασιών ανά γύρο*	*Μέγιστος αριθμός νέων εργασιών ανά γύρο*	*Πιθανότητα ο κόμβος να παραμείνει ζωντανός*	*Πιθανότητα να επανέρθει πίσω ένας κόμβος*
1	5	25	50%	50%
2	5	25	70%	50%
3	5	25	90%	50%
4	5	25	95%	50%
5	10	25	50%	50%
6	10	25	70%	50%
7	10	25	90%	50%
8	10	25	95%	50%
9	15	25	50%	50%
10	15	25	70%	50%
11	15	25	90%	50%
12	15	25	95%	50%

Πίνακας 1: Σενάρια εκτέλεσης αλγορίθμων AlgCS και AlgCSF

Για τους αλγορίθμους AlgCS και AlgCSF με κεντρικό χρονοδρομολογητή χρησιμοποιήθηκαν εικοσιένα (21) κόμβοι/διεργασίες όπου η μια διεργασία κατείχε το ρόλο του χρονοδρομολογητή και ο κάθε αλγόριθμος εκτελέστηκε δώδεκα φορές όπως παρουσιάζονται στον πιο πάνω πίνακα (*Πίνακας 1*).

Ο μέγιστος αριθμός των νέων εργασιών παρέμεινε σταθερός στους εικοσιπέντε (25) καθώς θεωρήθηκε ένας καλός αριθμός ανώτατου ορίου ο οποίος δεν υπερφορτώνει το σύστημα κρατώντας το σε λογικά πλαίσια.

Για τους αλγορίθμους AlgLI και AlgLIF χρησιμοποιήθηκαν είκοσι (20) κόμβοι λόγω απουσίας κεντρικού χρονοδρομολογητή. Επίσης λόγω των τοπικών εισαγωγέων η διακύμανση του ελάχιστου και μέγιστου αριθμού νέων εργασιών ανά γύρο ήταν περιορισμένη. Ο κάθε αλγόριθμος εκτελέστηκε τρεις φορές όπως παρουσιάζονται στον πιο κάτω πίνακα. Επίσης στις εκτελέσεις των αλγορίθμων διασφάλισης ολοκλήρωσης δε συγκαταλέχθηκε η περίπτωση με πιθανότητα κατάρρευσης 50% δια τον λόγο ότι ήταν μια ακραία και μη ρεαλιστική τιμή την οποία χρησιμοποιήσαμε στους αλγόριθμους κεντρικού χρονοδρομολογητή για να πιέσουμε το σύστημα στα άκρα. Επίσης σε προκαταρτικές εκτελέσεις παρατηρήσαμε ότι με τέτοια πιθανότητα κατάρρευσης οι υπολογιστές δεν μπορούσαν να αντεπεξέλθουν στις απαιτήσεις μνήμης.

Σενάριο	*Ελάχιστος αριθμός νέων εργασιών ανά γύρο ανά διεργασία*	*Μέγιστος αριθμός νέων εργασιών ανά γύρο ανά διεργασία*	*Πιθανότητα ο κόμβος να παραμείνει ζωντανός*	*Πιθανότητα να επανέρθει πίσω ένας κόμβος*
1	0	3	70%	50%
2	0	3	90%	50%
3	0	3	95%	50%

Πίνακας 2: Σενάρια εκτέλεσης αλγορίθμων AlgLI και AlgLIF

Στο επόμενο υποκεφάλαιο παρουσιάζονται τα αποτελέσματα των πιο πάνω σεναρίων και γίνεται σύγκριση των αλγορίθμων βάσει των αποτελεσμάτων που πρόεκυψαν.

Ο μηχανισμός που χρησιμοποιήθηκε για τη δυναμική εισαγωγή εργασιών στο σύστημα είναι σχετικά απλός καθώς αποφασίζεται τυχαία ένας *x* αριθμός νέων εργασιών στο πεδίο [0,3] και ακολούθως εισάγονται στο σύστημα. Πιο κάτω το τμήμα κώδικα από τον εισαγωγέα εργασιών του αλγόριθμου *AlgLIF*.

```
// Create random number of dummy Tasks
randomNum = minNumOfTasks + (int) (Math.random() *
(maxNumOfTasks - minNumOfTasks));
tasksDynamicInjected = tasksDynamicInjected + randomNum;

// Append these tasks to the current queue
for (int count = 0; count < randomNum; count++) {
   DemoTask dt = new DemoTask(getNodeStr(), getNewSeqNum());
   newTasks.add(dt);
}
```

Στην ίδια λογική βασίζεται και ο μηχανισμός ο οποίος αποφασίζει ποια διεργασία θα καταρρεύσει ή θα επανεκκινήσει εάν έχει ήδη καταρρεύσει. Πιο κάτω φαίνεται το τμήμα κώδικα από το μηχανισμό κατάρρευσης και επανεκκίνησης διεργασιών του αλγόριθμου *AlgLIF*. Στο σημείο αυτό διασφαλίζεται και ο περιορισμός του 2-survivability όπου ή κάθε διεργασία πρέπει μετά από κάθε επανεκκίνηση να παραμείνει για τουλάχιστο δύο γύρους ζωντανή βάσει προδιαγραφών. Επίσης μέσω του ίδιου μηχανισμού η εφαρμογή διασφαλίζει ότι σε κάθε γύρο τουλάχιστο μια διεργασία θα παραμείνει ζωντανή. Η εφαρμογή διατηρεί των αριθμό των εν ζωή εργασιών και δεν επιτρέπει στην τελευταία ζωντανή διεργασία να καταρρεύσει.

```
if ((nodeIsDown == false) && (stayUpChance >= chanceToStayUp) &&
(infoPackage.getNodeAge() > 1) && nodeCanGoDown()) {
   // happens X% of the time...
   nodeIsDown = true;
   setNodeDown();
   ...
```

```
} else if ((nodeIsDown == true) && (RestartChance <
chanceToRestart)) {
   nodeIsDown = false;
   setNodeUp();
   restartNode();
   ...
}
```

7.4 Ανάλυση Αποτελεσμάτων

Στο παρόν υποκεφάλαιο παρουσιάζονται τα αποτελέσματα της πειραματικής εκτέλεσης των αλγορίθμων και σχολιάζονται για τον κάθε αλγόριθμο που υλοποιήθηκε ξεχωριστά.

7.4.1 Αλγόριθμος AlgCS

Ο αλγόριθμος *AlgCS*, ο πιο απλός σε λειτουργία, εκτελέστηκε δώδεκα φορές με διαφορές στον αριθμό των νέων εργασιών που εισάγονταν ανά γύρο στο σύστημα και στην πιθανότητα κατάρρευσης διεργασιών όπως περιγράφονται και στο προηγούμενο υποκεφάλαιο (

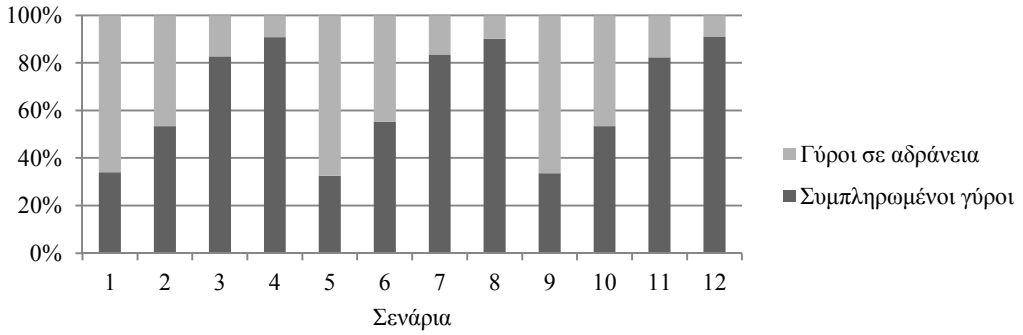
*Πίνακας **1***).

Σενάριο	*Συμπ. γύροι*	*Συμπ. εργασίες*	*Γύροι σε αδράνεια*	*Μηνύματα προς χρονοδρομολογητή*	*Μηνύματα από χρονοδρομολογητή*	*Χρόνος διεκπεραίωσης*
1	3396	3396	6604	3396	6731	134
2	5335	5335	4665	5335	7682	78
3	8276	8276	1724	8276	9185	7
4	9075	9075	925	9075	9537	6
5	3250	3250	6750	3250	6541	184
6	5530	5530	4470	5530	7776	122
7	8360	8360	1640	8360	9220	19
8	9007	9007	993	9007	9528	8
9	3364	3364	6636	3364	6701	233
10	5344	5344	4656	5344	7660	186
11	8237	8237	1763	8237	9121	91
12	9113	9113	887	9113	9562	41

Πίνακας 3: Αποτελέσματα πειραματικών εκτελέσεων σεναρίων αλγόριθμου AlgCS

Πιο πάνω (*Πίνακας **3***) εμφανίζονται τα αποτελέσματα των σεναρίων όπως έχουν εξαχθεί από τα αρχεία ελέγχου. Εύκολα γίνεται αντιληπτό ότι κατά τις εκτελέσεις των σεναρίων 1, 5 και 9 όπου η πιθανότητα να παραμείνει ζωντανή μια διεργασία ήταν σχετικά μικρή (50%) ανά γύρο, το σύστημα δεν απόδιδε ικανοποιητικά καθώς για μεγάλο ποσοστό γύρων συνολικά (66%), οι διεργασίες δεν ήταν ζωντανές.

Στις εκτελέσεις σεναρίων 3, 7 και 11 με πιθανότητα να παραμείνει ζωντανή μια διεργασία στο 90% και στις εκτελέσεις σεναρίων 4, 8 και 12 με πιθανότητα 95%, σενάρια τα οποία μπορούν να θεωρηθούν ρεαλιστικά, παρατηρούμε ότι τα αποτελέσματα βρίσκονται σε πολύ καλά επίπεδα. Με 83% και 90% ολοκληρωμένους γύρους και αντίστοιχες ολοκληρωμένες εργασίες ο αλγόριθμος φαίνεται ότι μπορεί να αντεπεξέλθει κατά τις καταρρεύσεις και επανεκκινήσεις των διεργασιών. Αντιθέτως ο αλγόριθμος AlgCS σε σενάρια τα οποία οι διεργασίες έχουν μεγάλη πιθανότητα κατάρρευσης (50%, 30%) ο αλγόριθμος δεν μπορεί να αντεπεξέλθει.

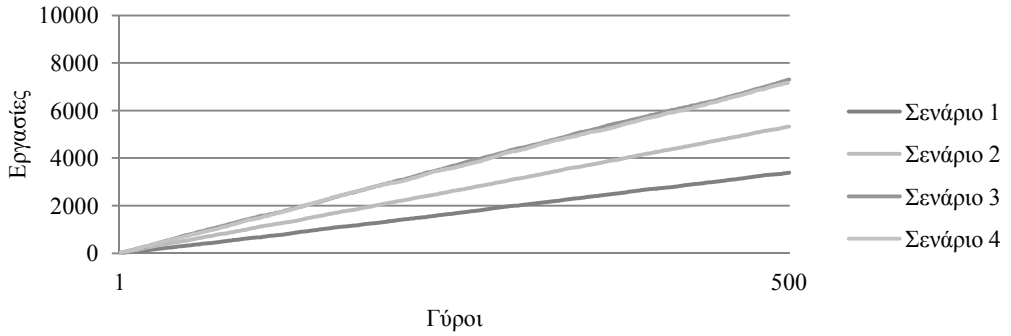


Σχήμα 5: Ποσοστό Συμπληρωμένων Γύρων

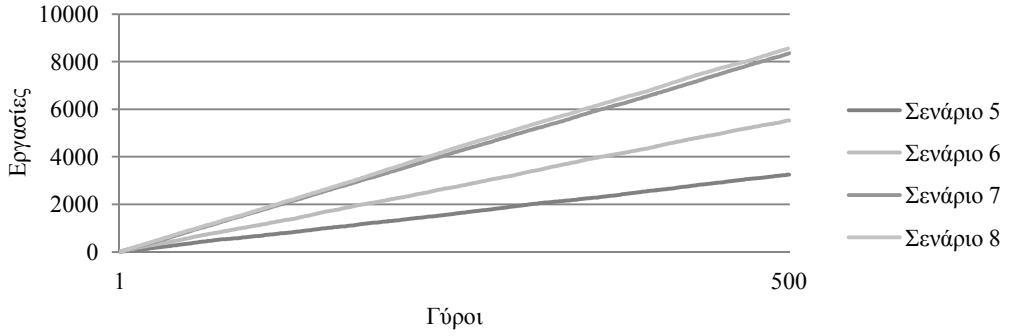
Η καλή, σχεδόν βέλτιστη, απόδοση του αλγόριθμου *AlgCS* βασίζεται στο αυστηρό μοντέλο του κεντρικού χρονοδρομολογητή. Επίσης λόγω του ότι οι διεργασίες δεν επικοινωνούν μεταξύ τους ο αριθμός τον μηνυμάτων που ανταλλάζονται και μεταφέρονται μέσω δικτύου είναι σχετικά μικρός.

Στις πιο κάτω γραφικές παραστάσεις (*Σχήμα 6*, *Σχήμα 7*, *Σχήμα **8***) οι οποίες παρουσιάζουν τις συμπληρωμένες εργασίες ανά γύρο για τους 500 γύρους του κάθε σεναρίου φαίνεται όπως

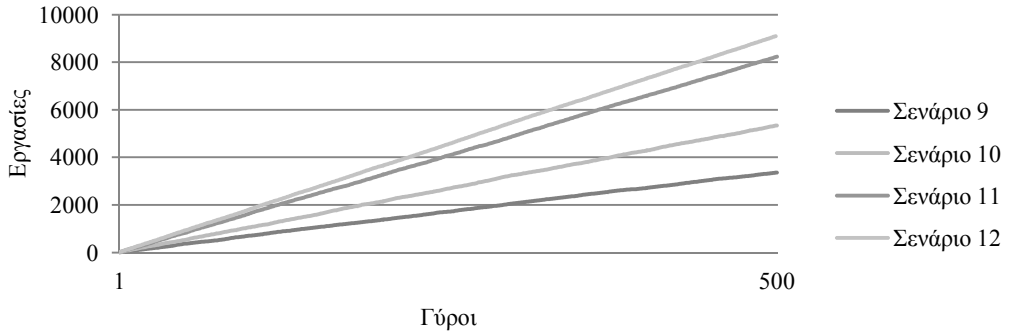
ήταν αναμενόμενο ότι το σύστημα έχει μια σταθερότητα ως προς τις εργασίες που εκτελεί σε κάθε γύρο. Αυτό συμπεραίνεται από τη γραμμική αύξηση των συμπληρωμένων εργασιών. Στις ίδιες γραφικές παραστάσεις επίσης φαίνεται η μειονεκτική θέση του συστήματος στις περιπτώσεις με μεγάλη πιθανότητα κατάρρευσης όπου η απόδοση πέφτει στο 30%.



Σχήμα 6: Σύνολο Συμπληρωμένων Εργασιών ανά Γύρο (Σεν. 1,2,3 και 4)

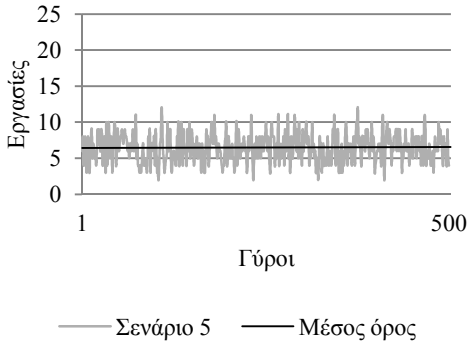


Σχήμα 7: Σύνολο Συμπληρωμένων Εργασιών ανά Γύρο (Σεν. 5,6,7 και 8)

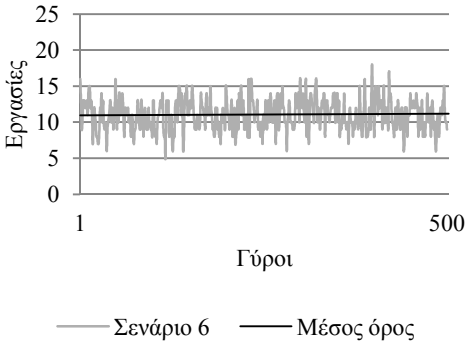


Σχήμα 8: Σύνολο Συμπληρωμένων Εργασιών ανά Γύρο (Σεν. 9,10,11 και 12)

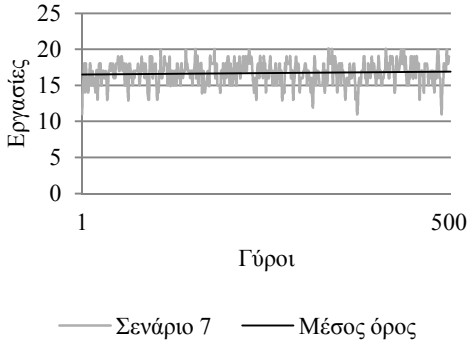
Πιο κάτω στις γραφικές παραστάσεις *(Σχήμα 9, Σχήμα 10, Σχήμα **11**, Σχήμα **12**)* βλέπουμε τον αριθμό των συμπληρωμένων εργασιών ανά γύρο και τον μέσο όρο. Παρατηρούμε ότι όσο η πιθανότητα κατάρρευσης μειώνεται, η διακύμανση είναι μικρότερη και ο αριθμός των συμπληρωμένων εργασιών μεγαλύτερος. Ως εκ τούτου όπως φαίνεται και στον *Πίνακας 4* ο μέσος όρος των συμπληρωμένων εργασιών αυξάνεται δραματικά κατά της περιπτώσεις όπου η πιθανότητα κατάρρευσης βρίσκεται στο 5% και 10%. Αυξάνεται σε τέτοιο βαθμό που η απόδοση του θεωρείται σχεδόν βέλτιστη.



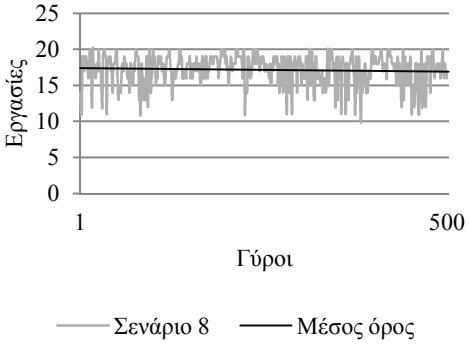
Σχήμα 9: Συμπληρωμένες Εργασίες ανά Γύρο (Πιθανότητα κατάρρευσης 50%)



Σχήμα 10: Συμπληρωμένες Εργασίες ανά Γύρο (Πιθανότητα κατάρρευσης 30%)



Σχήμα 11: Συμπληρωμένες Εργασίες ανά Γύρο (Πιθανότητα κατάρρευσης 10%)



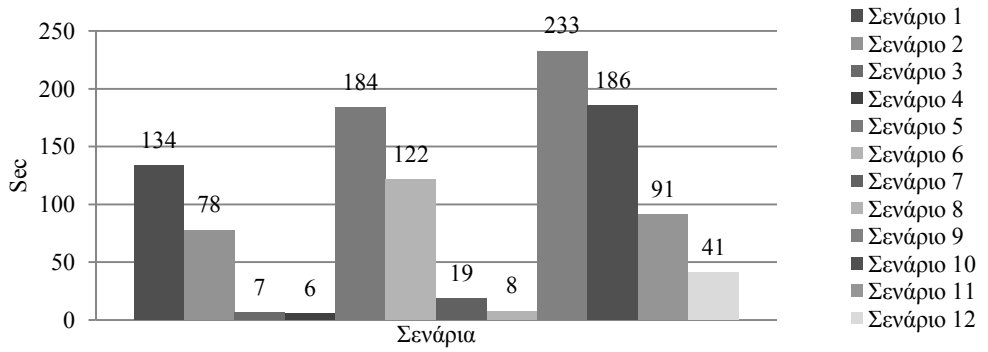
Σχήμα 12: Συμπληρωμένες Εργασίες ανά Γύρο (Πιθανότητα κατάρρευσης 5%)

	Μέγιστος Αριθμός	*Ελάχιστος Αριθμός*	*Μέσος Όρος*
Σενάριο 5	12	2	6,5
Σενάριο 6	18	5	11
Σενάριο 7	20	11	16,72

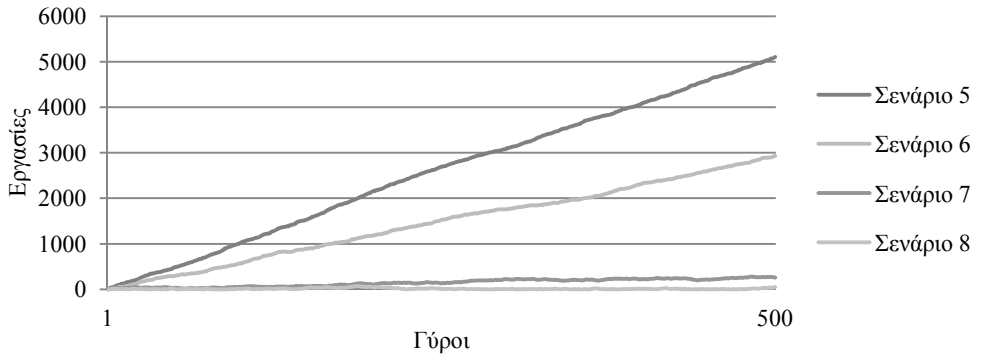
Σενάριο 8	20	10	17,16

Πίνακας 4: Τοπικά ελάχιστα, μέγιστα και μέσοι όροι εκτελέσεων

Ο χρόνος διεκπεραίωσης αυξάνεται σημαντικά όπως φαίνονται στο *Σχήμα 13*, στα σεναρίων με μεγάλη πιθανότητα κατάρρευσης διεργασιών. Επίσης βλέπουμε στο *Σχήμα 14* στις περιπτώσεις αυτές υπάρχει μεγάλος αριθμός εργασιών σε αναμονή. Η καθυστέρηση αυτή δεν εμφανίζεται μόνο λόγω των μεγάλων πακέτων που αποστέλλονται στις διεργασίες από τον κεντρικό χρονοπρογραμματιστή στις περιπτώσεις αυτές αλλά και λόγω του ότι το σύστημα εκτελέστηκε σε ένα υπολογιστή σε περιβάλλον προσομοίωσης όπου η μνήμη αλλά και η επεξεργαστική ισχύ ήταν περιορισμένη. Στον αλγόριθμο *AlgCS* τα μηνύματα που ανταλλάζονταν περιείχαν μέχρι και 6392 εργασίες.



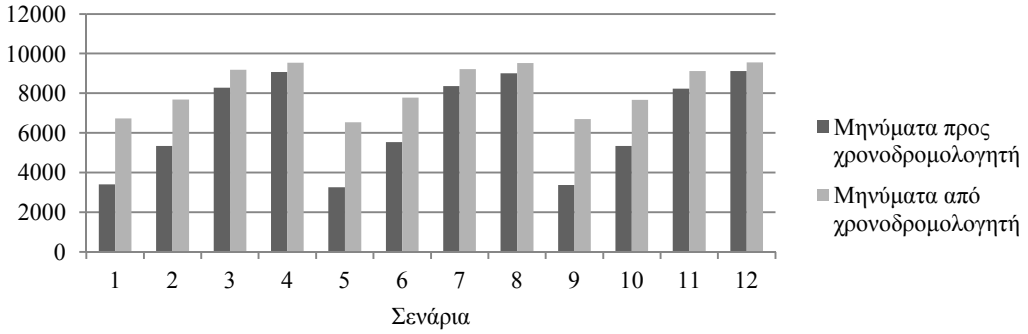
Σχήμα 13: Χρόνος Διεκπεραίωσης



Σχήμα 14: Εργασίες σε Αναμονή

Ο αριθμός των μηνυμάτων που κινηθήκαν στο σύστημα ήταν ο αναμενόμενος καθώς ο μόνο ο κεντρικός χρονοπρογραμματιστής επικοινωνεί με τις διεργασίες όπου σε κάθε γύρο στέλνει

σε κάθε διεργασία όλες τις εν αναμονή εργασίες. Στο *Σχήμα 15* βλέπουμε ότι κατά τις εκτελέσεις των σεναρίων με μεγάλη πιθανότητα κατάρρευσης πολλά μηνύματα δεν παρελήφθησαν από τις διεργασίες λόγω του ότι δεν ήταν «ζωντανές». Αντίθετα στα σενάρια με ελάχιστη πιθανότητα κατάρρευσης σχεδόν όλα τα μηνύματα παρελήφθησαν από τις διεργασίες επιτυχώς.



Σχήμα 15: Κίνηση Μηνυμάτων στους Κόμβους

7.4.2 Αλγόριθμος AlgCSF

Ο αλγόριθμος *AlgCSF*, ο οποίος παρέχει και διασφάλιση ολοκλήρωσης, εκτελέστηκε δώδεκα φορές (

*Πίνακας **1***) όπως και ο αλγόριθμος *AlgCS*.

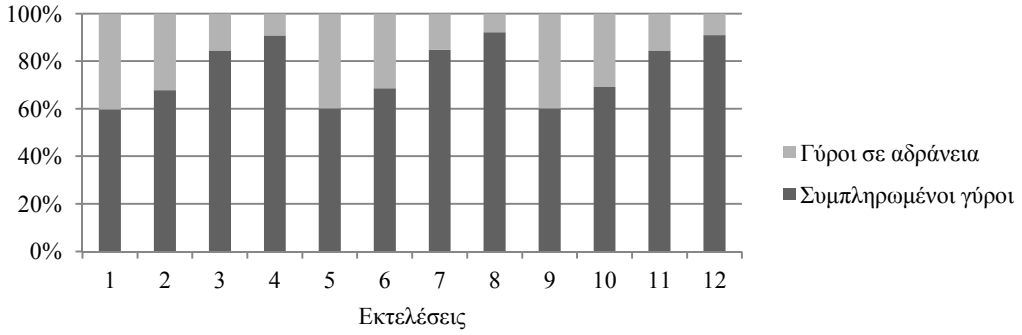
Σενάριο	*Συμπ. γύροι*	*Συμπ. εργασίες*	*Γύροι σε αδράνεια*	*Μηνύματα προς χρο/τή*	*Μηνύματα από χρο/τή*	*Αποστολές μηνυμάτων σε κόμβους*	*Ληφθέντα μηνύματα από κόμβους*	*Χρόνος διεκπ.*
1	5976	5976	4024	5976	7993	113544	68174	74
2	6780	6780	3220	6780	8350	128820	87392	51
3	8446	8446	1554	8446	9238	160474	135400	28
4	9084	9084	916	9084	9548	172596	156800	29
5	6010	6010	3990	6010	7993	114190	68836	125
6	6853	6853	3147	6853	8398	130207	89116	97
7	8478	8478	1522	8478	9263	161082	136696	36
8	9210	9210	790	9210	9608	174990	161110	30
9	6006	6006	3994	6006	7990	114114	68628	178
10	6909	6909	3091	6909	8500	131271	90696	157
11	8448	8448	1552	8448	9207	160512	135484	96

12	9094	9094	906	9094	9544	172786	157114	66

Πίνακας 5: Αποτελέσματα πειραματικών εκτελέσεων σεναρίων αλγόριθμου AlgCSF

Πιο πάνω (*Πίνακας 5*) εμφανίζονται τα αποτελέσματα των εκτελέσεων των σεναρίων όπως έχουν εξαχθεί από τα αρχεία ελέγχου. Εύκολα γίνεται αντιληπτό ότι κατά τις εκτελέσεις σεναρίων 1, 5 και 9 όπου η πιθανότητα να παραμείνει ζωντανή μια διεργασία ήταν ελάχιστη (50%) το σύστημα δεν υπολειτουργεί όπως στον *AlgCS*. Μόνο το 30% των γύρων οι διεργασίες ήταν εκτός λειτουργίας και δεν εκτελούσαν εργασίες.

Στις εκτελέσεις σεναρίων 3, 7 και 11 με πιθανότητα να παραμείνει ζωντανή μια διεργασία στο 90% και στις εκτελέσεις σεναρίων 4, 8 και 12 με πιθανότητα 95%, σενάρια τα οποία μπορούν να θεωρηθούν ρεαλιστικά, παρατηρούμε ότι τα αποτελέσματα βρίσκονται σε πολύ καλά επίπεδα. Με 85% και 91% ολοκληρωμένους γύρους, ελάχιστα περισσότερους από τον *AlgCS*, και αντίστοιχες ολοκληρωμένες εργασίες ο αλγόριθμος φαίνεται ότι μπορεί να αντεπεξέλθει κατά τις καταρρεύσεις και επανεκκινήσεις των διεργασιών.

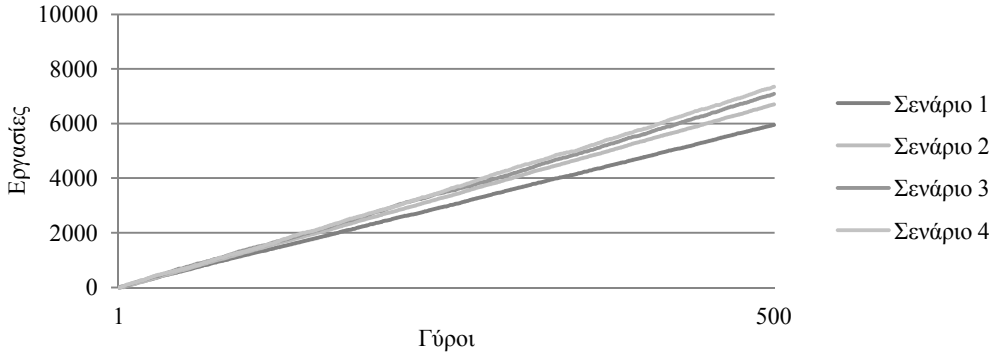


Σχήμα 16: Ποσοστό Συμπληρωμένων Γύρων

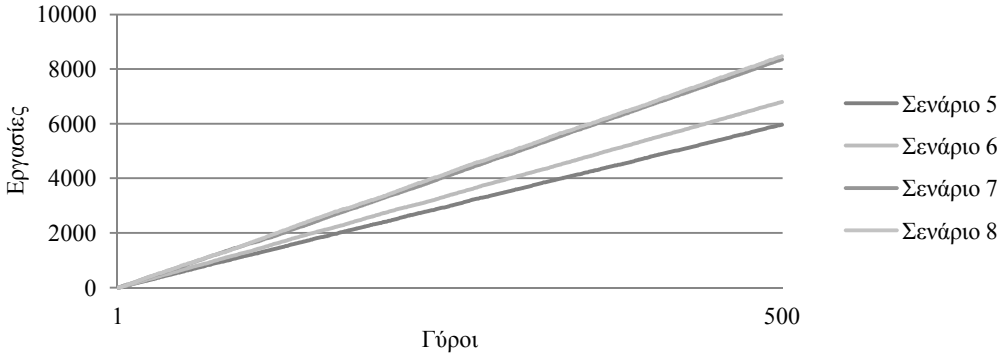
Η καλή, σχεδόν βέλτιστη, απόδοση του αλγόριθμου *AlgCSF* βασίζεται στο αυστηρό μοντέλο του κεντρικού χρονοδρομολογητή αλλά και της επιπρόσθετης πληροφορίας που μεταφέρεται μεταξύ διεργασιών στον αλγόριθμο αυτό, πληροφορία η οποία προσφέρει στην αποτελεσματικότητα του αλγορίθμου. Επίσης λόγω του ότι οι διεργασίες επικοινωνούν

μεταξύ τους ο αριθμός των μηνυμάτων που ανταλλάζονται και μεταφέρονται μέσω δικτύου είναι αυξάνεται σε σχέση με τον απλό *AlgCS*.

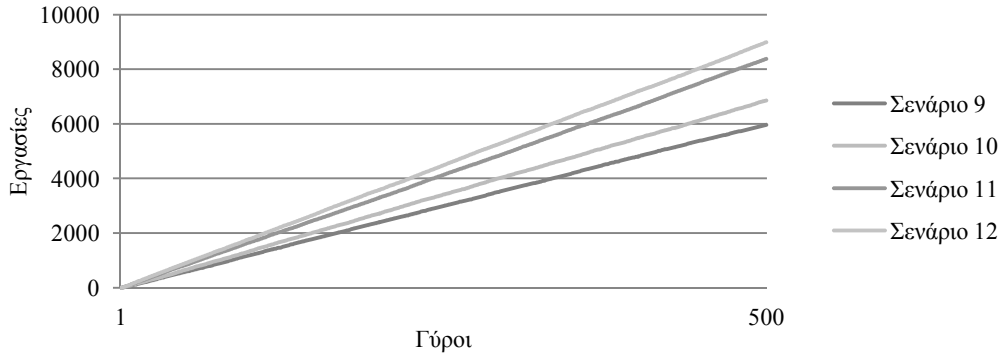
Στις πιο κάτω γραφικές παραστάσεις (*Σχήμα 17, Σχήμα 18, Σχήμα 19*), οι οποίες παρουσιάζουν τις συμπληρωμένες εργασίες ανά γύρο, φαίνεται όπως και στον *AlgCS* η σταθερότητα του συστήματος.



Σχήμα 17: Σύνολο Συμπληρωμένων Εργασιών ανά Γύρο (Σεν. 1,2,3 και 4)

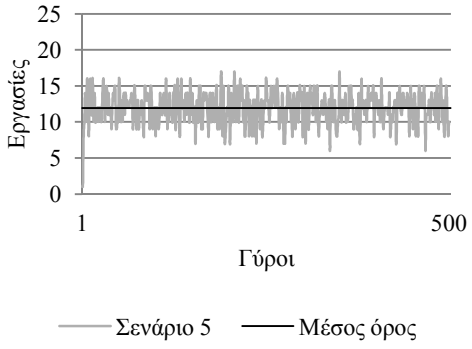


Σχήμα 18: Σύνολο Συμπληρωμένων Εργασιών ανά Γύρο (Σεν. 5,6,7 και 8)



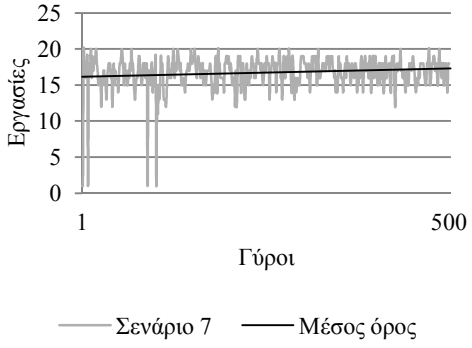
Σχήμα 19: Σύνολο Συμπληρωμένων Εργασιών ανά Γύρο (Σεν. 9,10,11 και 12)

Πιο κάτω στις γραφικές παραστάσεις *(Σχήμα 20, Σχήμα 21, Σχήμα **22**, Σχήμα **23**)* βλέπουμε τον αριθμό των συμπληρωμένων εργασιών ανά γύρο και τον μέσο όρο. Παρατηρούμε ότι όσο η πιθανότητα κατάρρευσης μειώνεται, η διακύμανση είναι μικρότερη και ο αριθμός των συμπληρωμένων εργασιών μεγαλύτερος. Ως εκ τούτου όπως φαίνεται και στον *Πίνακας **6*** ο μέσος όρος των συμπληρωμένων εργασιών αυξάνεται σε μικρό βαθμό. Στις πιο κάτω παραστάσεις φαίνεται η επίδραση του περιορισμού *2-survivability,* περιορισμός ο οποίος φαίνεται ότι μειώνει τις συνολικές καταρρεύσεις με αποτέλεσμα να εκτελούνται περισσότερες εργασίες. Επίσης παρατηρούμε στο *Σχήμα 22* και *Σχήμα 23* τις περιπτώσεις όπου οι εργασίες είναι λιγότερες από *2n* βάσει αλγόριθμου εκτελείται μόνο μια εργασία αυτή με *Rank 1.*

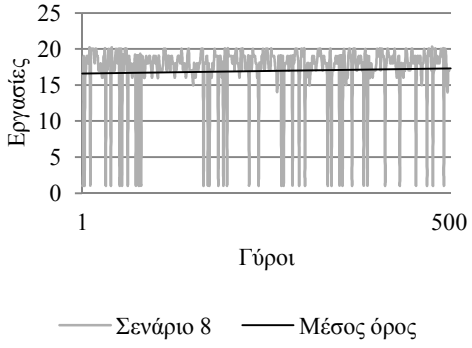


Σχήμα 20: Συμπληρωμένες Εργασίες ανά Γύρο (Πιθανότητα κατάρρευσης 50%)

Σχήμα 21: Συμπληρωμένες Εργασίες ανά Γύρο (Πιθανότητα κατάρρευσης 30%)



Σχήμα 22: Συμπληρωμένες Εργασίες ανά Γύρο (Πιθανότητα κατάρρευσης 10%)



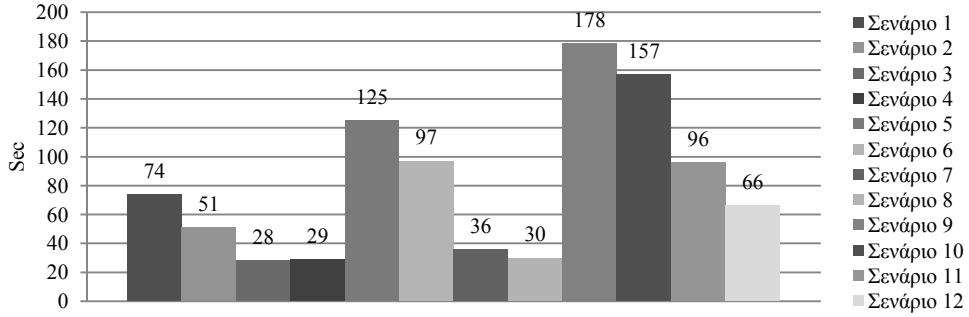
Σχήμα 23: Συμπληρωμένες Εργασίες ανά Γύρο (Πιθανότητα κατάρρευσης 5%)

	Μέγιστος Αριθμός	*Ελάχιστος Αριθμός*	*Μέσος Όρος*
Σενάριο 5	17	1	11,93
Σενάριο 6	19	1	13,60

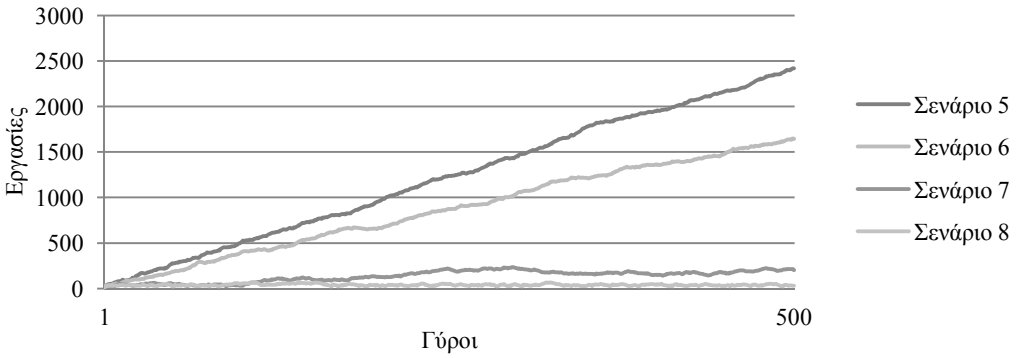
Σενάριο 7	20	1	16,72
Σενάριο 8	20	1	16,94

Πίνακας 6: Τοπικά ελάχιστα, μέγιστα και μέσοι όροι εκτελέσεων

Ο χρόνος διεκπεραίωσης αυξάνεται δραματικά όπως φαίνονται στο *Σχήμα 24*, στα σενάρια με μεγάλη πιθανότητα κατάρρευσης διεργασιών. Επίσης βλέπουμε στο *Σχήμα 25* στις περιπτώσεις αυτές υπάρχει μεγάλος αριθμός εργασιών σε αναμονή. Η καθυστέρηση αυτή εμφανίζεται λόγω των μεγάλων πακέτων που αποστέλλονται στις διεργασίες από τον κεντρικό χρονοπρογραμματιστή, τον μεγάλο αριθμό μηνυμάτων που ανταλλάζονται μεταξύ των διεργασιών αλλά και όπως προαναφέραμε ότι το σύστημα εκτελέστηκε σε ένα υπολογιστή σε περιβάλλον προσομοίωσης όπου η μνήμη αλλά και η επεξεργαστική ισχύ ήταν περιορισμένη. Στον αλγόριθμο *AlgCSF* τα μηνύματα που ανταλλάζονταν περιείχαν μέχρι και 3745 εργασίες.

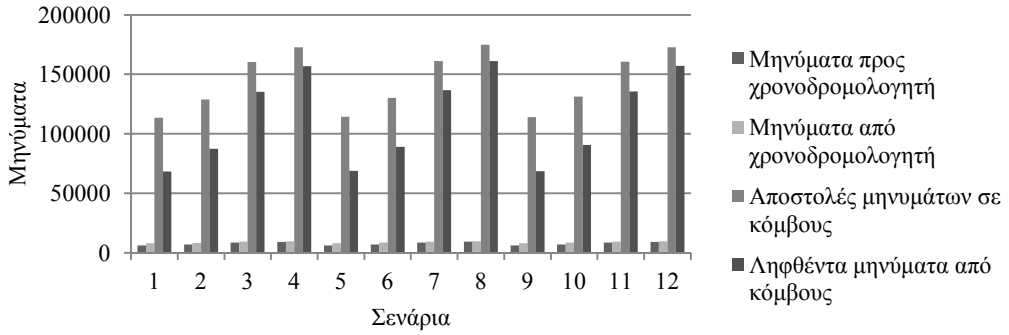


Σχήμα 24: Χρόνος Διεκπεραίωσης



Σχήμα 25: Εργασίες σε Αναμονή

Ο αριθμός των μηνυμάτων (*Σχήμα 26*) που κινήθηκαν στο σύστημα ήταν ο αναμενόμενος. Στην περίπτωση του *AlgCSF* ο αριθμός των μηνυμάτων που ανταλλάζονται μεταξύ διεργασιών είναι μεγάλος αλλά αμελητέος. Καθώς ο κεντρικός χρονοπρογραμματιστής σε κάθε γύρο στέλνει σε κάθε διεργασία όλες τις εν αναμονή εργασίες, μηνύματα με μεγάλο μέγεθος. Σε αντίθεση, τα μηνύματα που ανταλλάζονται μεταξύ διεργασιών είναι μόνο πληροφοριακά με μικρό μέγεθος και δεν επηρεάζουν την απόδοση του συστήματος, ούτε λόγω ποσότητας.



Σχήμα 26: Κίνηση Μηνυμάτων στους Κόμβους

7.4.3 Αλγόριθμος AlgLI

Ο αλγόριθμος *AlgLI*, εκτελέστηκε τρεις φορές με διαφορές στην πιθανότητα κατάρρευσης διεργασιών όπως περιγράφονται και στο προηγούμενο υποκεφάλαιο (*Πίνακας **2***).

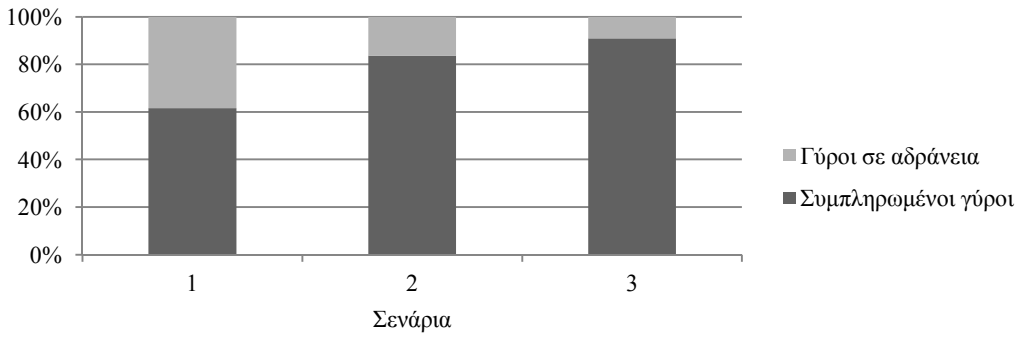
Πιο κάτω (

Πίνακας 7) εμφανίζονται τα αποτελέσματα των εκτελέσεων των σεναρίων όπως έχουν εξαχθεί από τα αρχεία ελέγχου. Εύκολα γίνεται αντιληπτό ότι κατά τις εκτελέσεις σεναρίων 1, 2 και 3 τα αποτελέσματα σε γενικές γραμμές δε διαφέρει από τον αλγόριθμο AlgCS. Ο αλγόριθμος *AlgLI* αν και δεν έχει κεντρικό χρονοδρομολογητή κερδίζει «χαμένο έδαφος» λόγω του ότι ο εισαγωγέας εργασιών βρίσκεται τοπικά και μετά από κάθε επανεκκίνηση εκτελούνται άμεσα εργασίες (αν υπάρχουν).

Σενάριο	*Συμπληρωμένοι γύροι*	*Συμπληρωμένες εργασίες*	*Γύροι σε αδράνεια*	*Αποστολές μηνυμάτων*	*Ληφθέντα μηνύματα*	*Χρόνος διεκπεραίωσης*
1	6159	5544	3841	253859	156492	214
2	8367	8091	1633	398221	332861	201
3	9091	8945	909	446082	405521	141

Πίνακας 7: Αποτελέσματα πειραματικών εκτελέσεων σεναρίων αλγόριθμου AlgLI

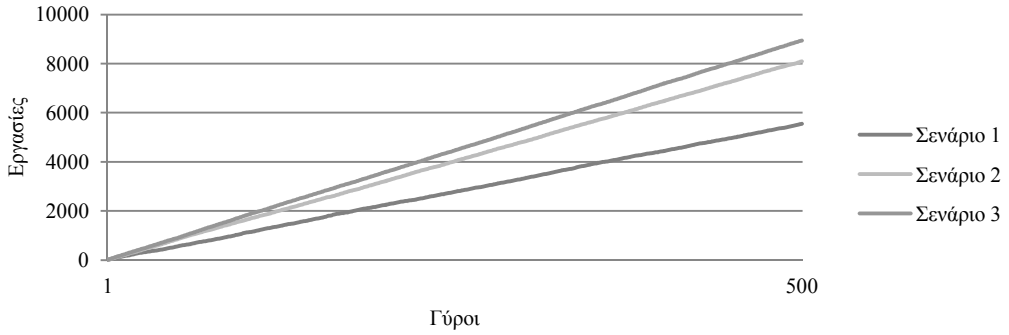
Στις εκτελέσεις, παρατηρούμε ότι τα αποτελέσματα βρίσκονται σε πολύ καλά επίπεδα. Με 62% 84% και 91% ολοκληρωμένους γύρους, και 55%, 81% και 90% ολοκληρωμένες εργασίες αντίστοιχα ο αλγόριθμος φαίνεται ότι μπορεί να αντεπεξέλθει κατά τις καταρρεύσεις και επανεκκινήσεις τον διεργασιών.



Σχήμα 27: Ποσοστό Συμπληρωμένων Γύρων

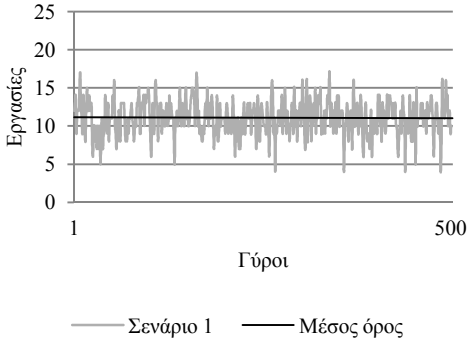
Η απόδοση του αλγορίθμου *AlgLI* βασίζεται στο πιο χαλαρό μοντέλο άμεσης επικοινωνίας μεταξύ διεργασιών και της απουσίας κεντρικού χρονοδρομολογητή. Επίσης λόγω του μοντέλου αυτού οι διεργασίες επικοινωνούν μεταξύ τους και ο αριθμός των μηνυμάτων που ανταλλάζονται και μεταφέρονται μέσω δικτύου είναι εξαιρετικά μεγάλος.

Στην πιο κάτω γραφική παράσταση (*Σχήμα 28*), παρουσιάζει τις συμπληρωμένες εργασίες ανά γύρο για τους 500 γύρους της κάθε εκτέλεσης. Στη γραφική αυτή παράσταση φαίνεται η σταθερότητα του συστήματος. Αυτό συμπεραίνετε από τη γραμμική αύξηση των συμπληρωμένων εργασιών.

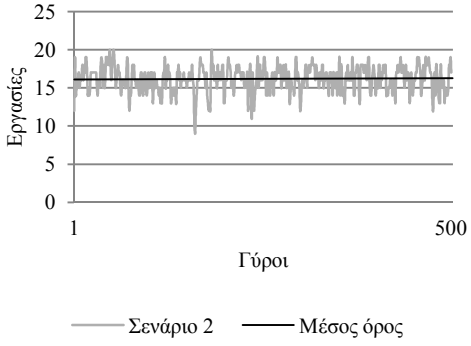


Σχήμα 28: Σύνολο Συμπληρωμένων Εργασιών ανά Γύρο(Σεν. 1,2 και 3)

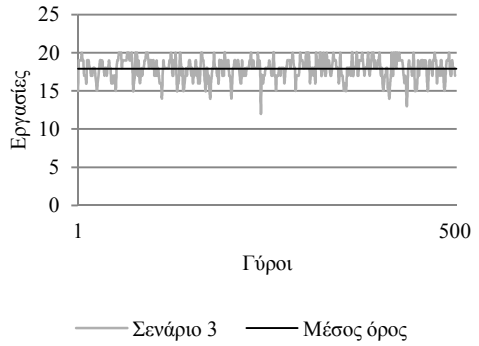
Πιο κάτω στις γραφικές παραστάσεις *(Σχήμα 29, Σχήμα **30**, Σχήμα **31**)* βλέπουμε τον αριθμό των συμπληρωμένων εργασιών ανά γύρο και τον μέσο όρο. Παρατηρούμε ότι όσο η πιθανότητα κατάρρευσης μειώνεται, η διακύμανση είναι μικρότερη και ο αριθμός των συμπληρωμένων εργασιών μεγαλύτερος όπως και στους προηγούμενους αλγόριθμους. Ως εκ τούτου όπως φαίνεται και στον *Πίνακας 6* ο μέσος όρος των συμπληρωμένων εργασιών αυξάνεται σε μικρό βαθμό από εκτέλεση σε εκτέλεση με σταθερό ρυθμό.



Σχήμα 29: Συμπληρωμένες Εργασίες ανά Γύρο (Πιθανότητα κατάρρευσης 30%)



Σχήμα 30: Συμπληρωμένες Εργασίες ανά Γύρο (Πιθανότητα κατάρρευσης 10%)

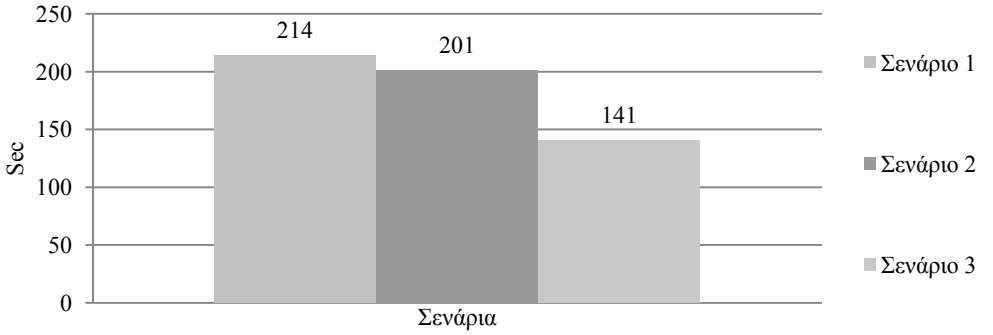


Σχήμα 31: Συμπληρωμένες Εργασίες ανά Γύρο (Πιθανότητα κατάρρευσης 5%)

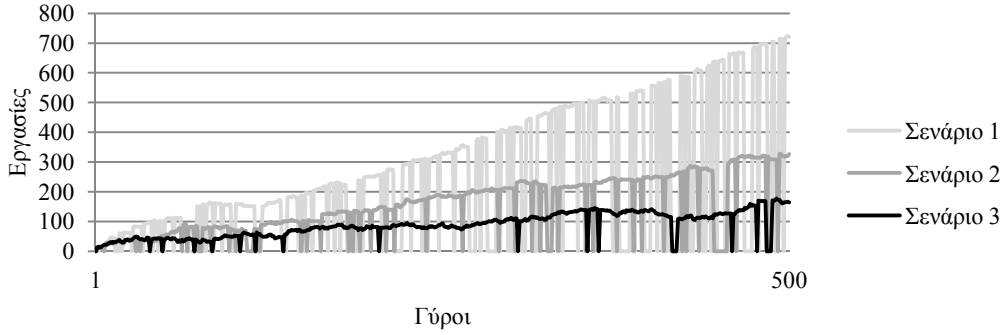
	Μέγιστος Αριθμός	*Ελάχιστος Αριθμός*	*Μέσος Όρος*
Σενάριο 1	17	4	11,08
Σενάριο 2	20	9	16,18
Σενάριο 3	20	12	17,89

Πίνακας 8: Τοπικά ελάχιστα, μέγιστα και μέσοι όροι εκτελέσεων

Ο χρόνος διεκπεραίωσης αυξάνεται, όχι σε μεγάλο βαθμό, όπως φαίνονται στο *Σχήμα 32*, στα σενάρια με μεγάλη πιθανότητα κατάρρευσης διεργασιών. Επίσης βλέπουμε στο *Σχήμα 33* στις περιπτώσεις αυτές υπάρχει μεγάλος αριθμός εργασιών σε αναμονή. Η καθυστέρηση αυτή εμφανίζεται λόγω του μεγάλου αριθμό μηνυμάτων που ανταλλάζονται μεταξύ των διεργασιών αλλά και όπως προαναφέραμε ότι το σύστημα εκτελέστηκε σε ένα υπολογιστή σε περιβάλλον προσομοίωσης όπου η μνήμη αλλά και η επεξεργαστική ισχύ ήταν περιορισμένη. Στον αλγόριθμο *AlgLI* τα μηνύματα που ανταλλάζονταν περιείχαν μέχρι και 723 εργασίες. Οι μηδενικές τιμές στο *Σχήμα 33* οφείλονται στις περιπτώσεις όπου η διεργασία η οποία αναλάμβανε τη διαδικασία καταγραφής κατέρρεε.

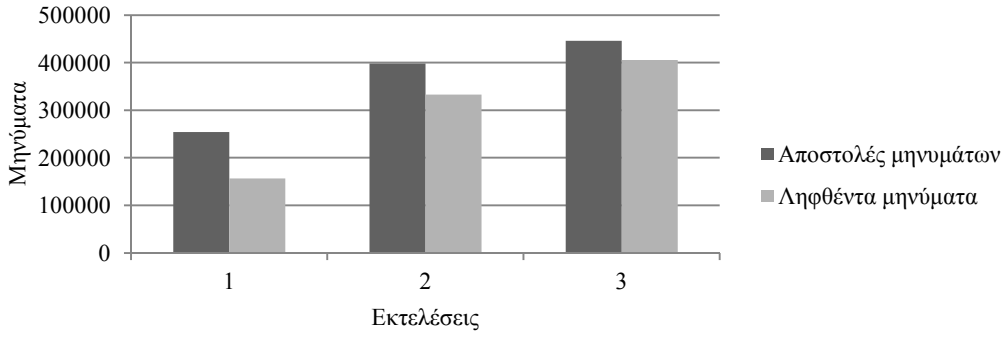


Σχήμα 32: Χρόνος Διεκπεραίωσης



Σχήμα 33: Εργασίες σε Αναμονή (set Old) βάσει του τελευταίου Node

Ο αριθμός των μηνυμάτων που κινήθηκαν στο σύστημα ήταν ο αναμενόμενος. Στην περίπτωση του *AlgLI* ο αριθμός των μηνυμάτων που ανταλλάζονται μεταξύ διεργασιών είναι εξαιρετικά μεγάλος. Τα μηνύματα που ανταλλάζονται μεταξύ διεργασιών περιέχουν τις νέες και τις εν αναμονή εργασίες με μεγάλο μέγεθος αλλά και άλλες πληροφορίες με μικρό μέγεθος.



Σχήμα 34: Κίνηση Μηνυμάτων στους Κόμβους

7.4.4 Αλγόριθμος AlgLIF

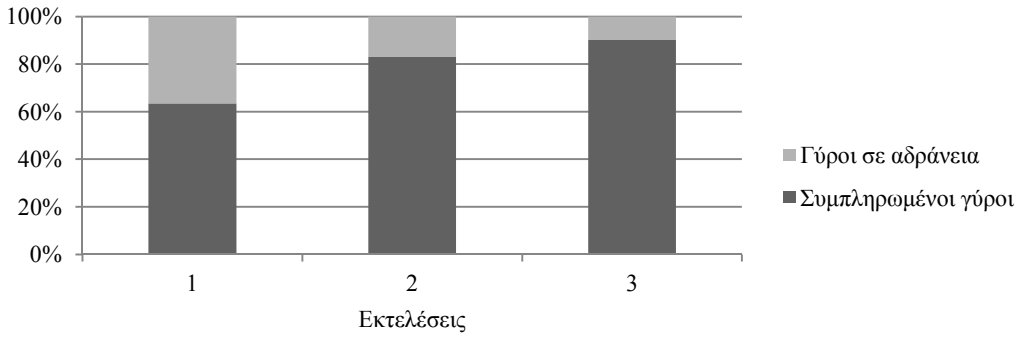
Ο αλγόριθμος *AlgLIF*, εκτελέστηκε τρεις φορές όπως και ο *AlgLI*, με διαφορές στην πιθανότητα κατάρρευσης διεργασιών όπως περιγράφονται και στο προηγούμενο υποκεφάλαιο (*Πίνακας 2*).

Πιο κάτω (*Πίνακας 9*) εμφανίζονται τα αποτελέσματα των εκτελέσεων των σεναρίων όπως έχουν εξαχθεί από τα αρχεία ελέγχου. Πάλι γίνεται αντιληπτό ότι κατά τις εκτελέσεις σεναρίων 1, 2 και 3 τα αποτελέσματα σε γενικές γραμμές δε διαφέρουν από τους αλγορίθμους με κεντρικό χρονοπρογραμματιστή. Ο αλγόριθμος *AlgLIF* όπως και ο *AlgLI* αν και δεν έχει κεντρικό χρονοδρομολογητή κερδίζει «χαμένο έδαφος» λόγω του ότι ο εισαγωγέας εργασιών βρίσκεται τοπικά και μετά από κάθε επανεκκίνηση εκτελούνται άμεσα εργασίες (αν υπάρχουν), επιπλέον στην βελτίωση της απόδοσης συμβάλει και ο περιορισμός *2-survivability*.

Σενάριο	*Συμπληρωμένοι γύροι*	*Συμπληρωμένες εργασίες*	*Γύροι σε αδράνεια*	*Αποστολές μηνυμάτων*	*Ληφθέντα μηνύματα*	*Χρόνος διεκπεραίωσης*
1	6351	5759	3649	383382	244213	233
2	8300	7995	1700	549271	455526	178
3	9012	8828	988	610147	549493	174

Πίνακας 9: Αποτελέσματα πειραματικών εκτελέσεων σεναρίων αλγόριθμου AlgLIF

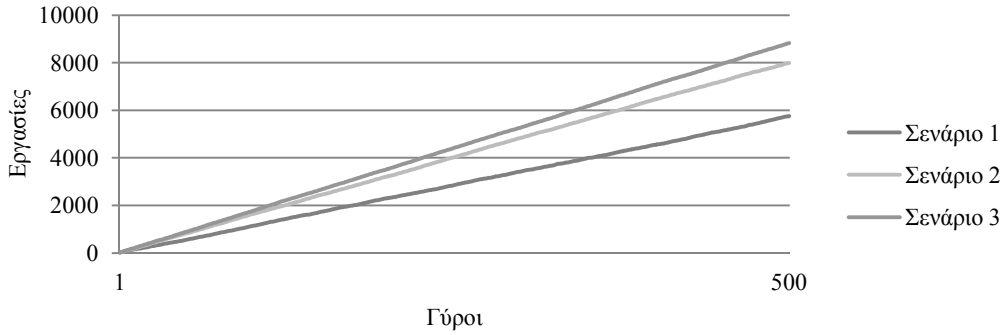
Στις εκτελέσεις, παρατηρούμε ότι τα αποτελέσματα βρίσκονται σε πολύ καλά επίπεδα. Με 63% 83% και 90% ολοκληρωμένους γύρους, και 57%, 80% και 88% ολοκληρωμένες εργασίες αντίστοιχα, όμοια αποτελέσματα με τον *AlgLI*, ο αλγόριθμος φαίνεται ότι μπορεί να αντεπεξέλθει κατά τις καταρρεύσεις και επανεκκινήσεις τον διεργασιών.



Σχήμα 35: Ποσοστό Συμπληρωμένων Γύρων

Η απόδοση του αλγορίθμου *AlgLIF* βασίζεται στο πιο χαλαρό μοντέλο άμεσης επικοινωνίας μεταξύ διεργασιών. Επίσης λόγω του μοντέλου αυτού οι διεργασίες επικοινωνούν μεταξύ τους και ο αριθμός τον μηνυμάτων που ανταλλάζονται και μεταφέρονται μέσω δικτύου είναι εξαιρετικά μεγάλος. Ο αριθμός αυτός είναι ακόμη μεγαλύτερος στον *AlgLIF* από τον *AlgLI* λόγω της επιπρόσθετης πληροφορίας η οποία ανταλλάζεται.

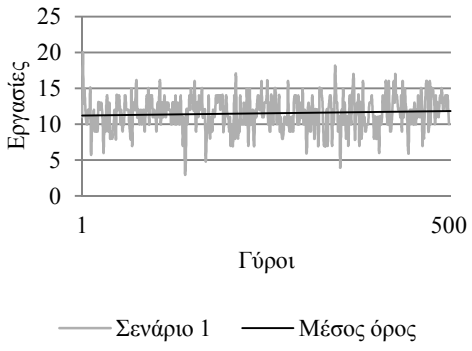
Η πιο κάτω γραφική παράσταση (*Σχήμα 28*)**,** παρουσιάζει τις συμπληρωμένες εργασίες ανά γύρο για τους 500 γύρους της κάθε εκτέλεσης. Στη γραφική αυτή παράσταση φαίνεται η σταθερότητα του συστήματος. Αυτό συμπεραίνεται από τη γραμμική αύξηση των συμπληρωμένων εργασιών.



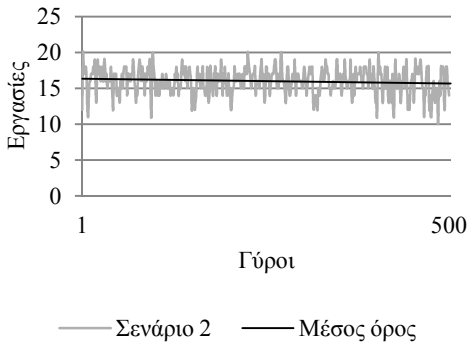
Σχήμα 36: Σύνολο Συμπληρωμένων Εργασιών ανά Γύρο(Σεν. 1,2 και 3)

Πιο κάτω στις γραφικές παραστάσεις *(Σχήμα 37, Σχήμα **38**, Σχήμα 39)* βλέπουμε τον αριθμό των συμπληρωμένων εργασιών ανά γύρο και το μέσο όρο. Παρατηρούμε και στον *AlgLIF*

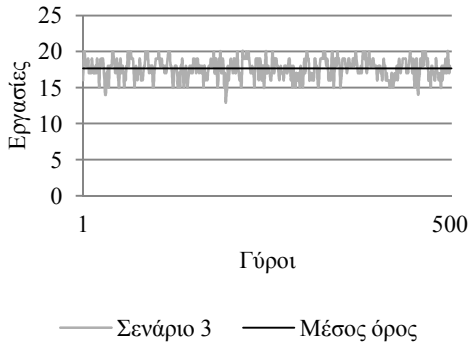
όπως και στους προηγουμένους αλγόριθμους ότι όσο η πιθανότητα κατάρρευσης μειώνεται, η διακύμανση είναι μικρότερη και ο αριθμός των συμπληρωμένων εργασιών μεγαλύτερος, όπως και στους προηγούμενους αλγόριθμους. Ως εκ τούτου όπως φαίνεται και στον *Πίνακας 10* ο μέσος όρος των συμπληρωμένων εργασιών παραμένει στα ίδια πλαίσια που κυμαίνονταν τα αποτελέσματα του *AlgLI*.



Σχήμα 37: Συμπληρωμένες Εργασίες ανά Γύρο (Πιθανότητα κατάρρευσης 30%)



Σχήμα 38: Συμπληρωμένες Εργασίες ανά Γύρο (Πιθανότητα κατάρρευσης 10%)

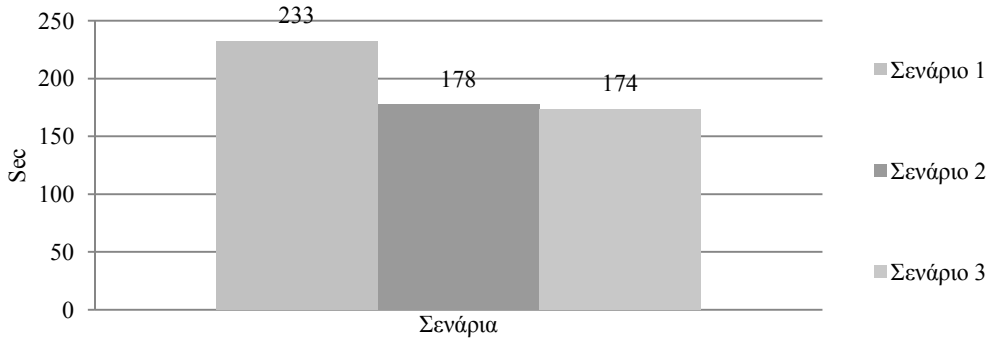


Σχήμα 39: Συμπληρωμένες Εργασίες ανά Γύρο (Πιθανότητα κατάρρευσης 5%)

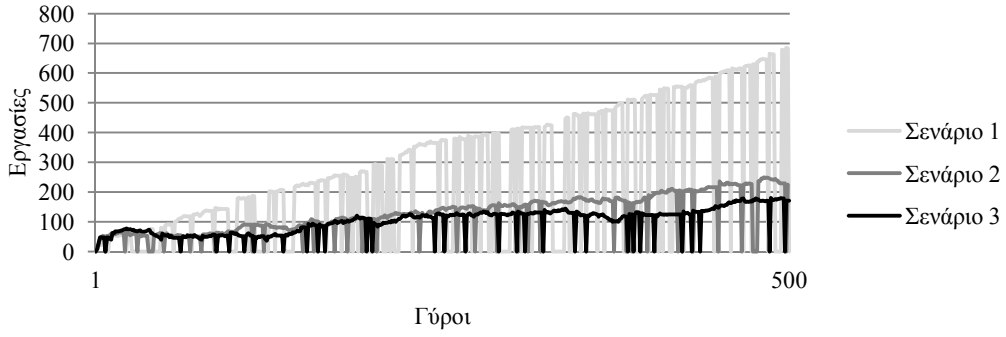
	Μέγιστος Αριθμός	*Ελάχιστος Αριθμός*	*Μέσος Όρος*
Σενάριο 1	20	3	11,52
Σενάριο 2	20	10	15,99
Σενάριο 3	20	13	17,66

Πίνακας 10: Τοπικά ελάχιστα, μέγιστα και μέσοι όροι εκτελέσεων

Ο χρόνος διεκπεραίωσης των εκτελέσεων αυξάνεται, όχι σε μεγάλο βαθμό, όπως φαίνονται στο *Σχήμα 40*, στα σενάρια με μεγάλη πιθανότητα κατάρρευσης διεργασιών. Επίσης βλέπουμε στο *Σχήμα 41* στις περιπτώσεις αυτές υπάρχει μεγάλος αριθμός εργασιών σε αναμονή. Η καθυστέρηση αυτή εμφανίζεται λόγω του μεγάλου αριθμού μηνυμάτων που ανταλλάζονται μεταξύ των διεργασιών, αλλά και όπως προαναφέραμε ότι το σύστημα εκτελέστηκε σε ένα υπολογιστή σε περιβάλλον προσομοίωσης όπου η μνήμη αλλά και η επεξεργαστική ισχύ ήταν περιορισμένη. Στον αλγόριθμο *AlgLIF* τα μηνύματα που ανταλλάζονταν περιείχαν μέχρι και 684 εργασίες. Οι μηδενικές τιμές στο *Σχήμα 41* οφείλονται στις περιπτώσεις όπου η διεργασία η οποία αναλάμβανε τη διαδικασία καταγραφής των αποτελεσμάτων κατέρρεε.

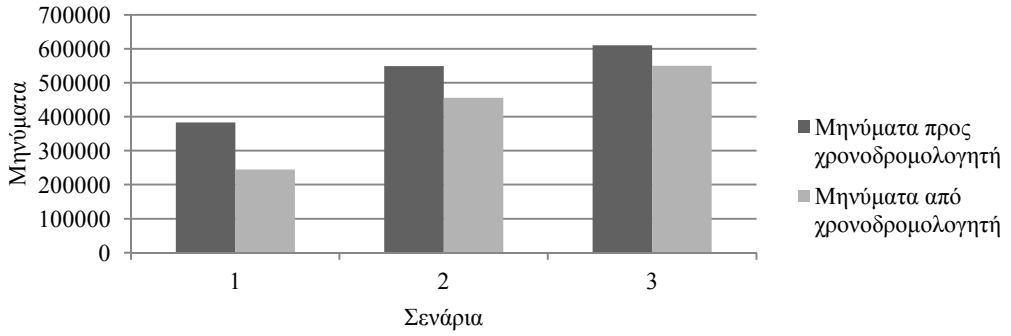


Σχήμα 40: Χρόνος Διεκπεραίωσης



Σχήμα 41:Εργασίες σε Αναμονή (set Old) βάσει του τελευταίου Node

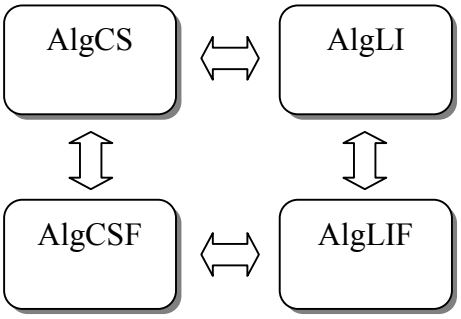
Ο αριθμός των μηνυμάτων που κινήθηκαν στο σύστημα ήταν ο αναμενόμενος. Στην περίπτωση του *AlgLIF* ο αριθμός των μηνυμάτων που ανταλλάζονται μεταξύ διεργασιών είναι εξαιρετικά μεγάλος. Τα μηνύματα που ανταλλάζονται μεταξύ διεργασιών περιέχουν τις νέες και τις εν αναμονή εργασίες με μεγάλο μέγεθος αλλά και άλλες πληροφορίες με μικρό μέγεθος.



Σχήμα 42: Κίνηση Μηνυμάτων στους Κόμβους

7.5 Σύγκριση Αλγορίθμων

Στο υποκεφάλαιο αυτό γίνεται σύγκριση των αλγορίθμων βάσει των αποτελεσμάτων που παρουσιάστηκαν στο προηγούμενο υποκεφάλαιο. Πιο συγκεκριμένα πραγματοποιείται οριζόντια και κάθετη σύγκριση μεταξύ τους .



Σχήμα 43: Οριζόντια και κάθετη σύγκριση αλγορίθμων

7.5.1 Σύγκριση Αλγορίθμων AlgCS και AlgCSF

Παρατηρώντας τα αποτελέσματα των αλγορίθμων AlgCS και AlgCSF από το προηγούμενο υποκεφάλαιο (*Πίνακας 3*, *Πίνακας **5***) παρατηρούμε ότι η απόδοση ως προς τον αριθμό συμπληρωμένων εργασιών του AlgCSF υπερτερεί εναντίων του AlgCS στα σενάρια όπου η πιθανότητα κατάρρευσης μιας διεργασίας ήταν μεγάλη, 50% και 30%, σε αντίθεση με τα σενάρια που η πιθανότητα ήταν μικρή. Στα σενάρια με πιθανότητα κατάρρευσης 5% και 10% ο AlgCS είχε ελαφρός καλύτερη απόδοση.

Τα πιο πάνω αποτελέσματα ήταν αναμενόμενα. Σε πρώτο στάδιο ο AlgCSF απέδιδε καλύτερα στα σενάρια με μεγάλη πιθανότητα κατάρρευσης λόγω του περιορισμού 2-survivability ο οποίος «ανάγκαζε» την κάθε διεργασία να παραμείνει ζωντανή για δύο γύρους μετά από κάθε επανεκκίνηση. Στη συνέχεια στα σενάρια με μικρή πιθανότητα λόγω της πολιτικής επιλογής και εκτέλεσης διεργασιών που ακολουθεί ο AlgCSF πολλοί γύροι παραμένουν «αναξιοποίητοι».

Σε γενικές γραμμές ο AlgCSF εν συγκρίσει με τον AlgCS, επιπλέων του εγγυάται την ολοκλήρωση, προσφέρει περισσότερη αξιοπιστία και ανοχή στα σφάλματα, πράγμα το οποίο συμπεραίνεται και από της γραφικές παραστάσεις που παρουσιάζουν τον αριθμό συμπληρωμένων γύρων *(Σχήμα 5, Σχήμα **16**)*. Την άποψη αυτή έρχονται να ενισχύσουν οι γραφικές παραστάσεις *(Σχήμα 6, Σχήμα 7, Σχήμα **8**, Σχήμα 17, Σχήμα 18, Σχήμα 19)* με το σύνολο των συμπληρωμένων εργασιών ανά γύρο να κυμαίνεται πάντοτε πάνω από το 60% κάτω από οποιοδήποτε σενάριο.

Όλα τα πιο πάνω επιβεβαιώνονται από τα αποτελέσματα στους *Πίνακας 4* και *Πίνακας 6* όπου φαίνεται η διαφορά στους μέσους όρους αλλά και οι επιδράσεις της πολιτικής επιλογής και εκτέλεσης διεργασιών που εφαρμόζει ο AlgCSF με ελάχιστο αριθμό εργασιών τη μια εργασία.

Η αξιοπιστία και η ανοχή στα σφάλματα και ακολούθως η διασφάλιση της ολοκλήρωσης του AlgCSF έχουν το αντίτιμό τους. Ο αριθμός των επιπρόσθετων μηνυμάτων που χρειάζονται

για την ενημέρωση των διεργασιών είναι μεγάλος όπως παρουσιάζονται στις γραφικές παραστάσεις *Σχήμα 15* και *Σχήμα 26*. Θετικό είναι ότι ο μεγάλος αριθμός μηνυμάτων δεν φαίνεται να επηρεάζουν το όλο σύστημα λόγω του μικρού μεγέθους τους σε αντίθεση με τα μηνύματα τα οποία περιέχουν συλλογές εργασιών.

7.5.2 Σύγκριση Αλγορίθμων AlgLI και AlgLIF

Οι δύο αυτοί αλγόριθμοι σε γενικές γραμμές δεν έχουν διαφορές στην απόδοση (*Σχήμα 27, Σχήμα **35***) αλλά ο AlgLIF με την απόδοση αυτή διασφαλίζει και την ολοκλήρωση σε αντίθεση με τον AlgLI. Το αποτέλεσμα αυτό δεν ήταν το αναμενόμενο, αναμέναμε ο AlgLIF να έχει ελαφρός καλύτερη απόδοση λόγο του 2-survivability.

Και οι δύο αλγόριθμοι κάτω από κάθε σενάριο αντιδρούν με πανομοιότυπο τρόπο (*Πίνακας 7*, *Πίνακας **9***). Ο ρυθμός αύξησης των συμπληρωμένων εργασιών (*Σχήμα 28*, *Σχήμα **36***) και ο αριθμός των συμπληρωμένων γύρων και εργασιών ανά γύρο κυμαίνεται στα ίδια επίπεδα, με αποτέλεσμα οι μέσοι όροι (**Error! Reference source not found.**, *Πίνακας 10*) να έχουν ελάχιστη διαφορά.

Η μόνη διαφορά η οποία κάτω από άλλες περιστάσεις και κριτήρια θα ήταν καταλυτική ως προς την απόδοση του AlgLIF είναι η κίνηση των μηνυμάτων. Αν και δε φαίνεται ο μεγάλος αριθμός μηνυμάτων να επηρεάζει τους χρόνους εκτέλεσης δεν μπορεί να μη ληφθεί υπόψη. Δεν επηρεάζει τους χρόνους λόγω του ότι οι εκτελέσεις πραγματοποιούνται σε περιβάλλον προσομοίωσης, σε πραγματικό περιβάλλον κατά πάσα πιθανότητα θα προκαλούσαν συμφόρηση στο δίκτυο και μείωση της απόδοσης.

7.5.3 Σύγκριση Αλγορίθμων AlgCS και AlgLI

Οι δύο αυτοί αλγόριθμοι οι οποίοι διασφαλίζουν μόνο την ορθότητα διαφέρουν σε πολλά βασικά σημεία με πιο βασικό τη χρήση κεντρικού χρονοπρογραμματιστή στον αλγόριθμο

AlgCS. Λόγω της βασικής αυτής διαφοράς δεν μπορεί να πραγματοποιηθεί άμεση σύγκριση μεταξύ των αλγορίθμων παρά μόνο της απόδοσης βάσει των συμπληρωμένων εργασιών του κάθε ενός από αυτούς (*Πίνακας 3* και *Σχήμα 5*, *Πίνακας 7* και *Σχήμα **27***).

Ο AlgLI όπως και ο AlgCSF που συγκρίναμε πιο πάνω έχει ελάχιστα καλύτερη απόδοση από τον AlgCS ειδικότερα στα σενάρια όπου η πιθανότητα κατάρρευσης μίας διεργασίας είναι μεγάλη. Η καλή απόδοση του AlgLI σε μεγάλο βαθμό οφείλεται στο ότι κανένας γύρος δεν πάει χαμένος καθώς κατά την επανεκκίνηση η κάθε διεργασία εκτελεί εργασία από το σύνολο των νέων διεργασιών που μόλις εισήχθησαν στο σύστημα από τον τοπικό εισαγωγέα.

Μεγάλο μειονέκτημα του AlgLI έναντι του AlgCS είναι τα πολλά και μεγάλα πακέτα πληροφοριών που μεταφέρονται μεταξύ διεργασιών. Επακόλουθο αυτού είναι η επεξεργαστική ισχύς που χρειάζεται για την επεξεργασία των πακέτων αυτών με αντίκτυπο στο χρόνο εκτέλεσης ο οποίος αυξήθηκε αισθητά.

7.5.4 Σύγκριση Αλγορίθμων AlgCSF και AlgLIF

Οι αλγόριθμοι διασφάλισης ολοκλήρωσης παρουσιάζουν ελάχιστες διαφορές (*Πίνακας 5*, *Πίνακας 9*). Η απόδοση των δύο δε διαφέρει ιδιαίτερα με των AlgCSF να είναι ελάχιστα πιο αποδοτικός από τον AlgLIF.

Οι δύο αλγόριθμοι με σχεδόν όμοια απόδοση, βάσει των συμπληρωμένων εργασιών, διαφέρουν κατά πολύ στο φόρτο εργασίας και στον αριθμό και το μέγεθος των μηνυμάτων (*Σχήμα 26*, *Σχήμα **42***) με αποτέλεσμα να διαφέρουν κατά πολύ οι χρόνοι εκτέλεσης (*Σχήμα 24*, *Σχήμα **40***).

Όπως και στη σύγκριση των αλγορίθμων διασφάλισης της ορθότητας αλλά και των αλγορίθμων με τοπικό εισαγωγέα, κάτω από άλλες περιστάσεις και κριτήρια τα χαρακτηριστικά αυτά θα είχαν καταλυτική δράση ως προς την απόδοση των αλγορίθμων.

7.6 Συμπεράσματα

Μέσω των αναλύσεων των πειραματικών εκτελέσεων και ακολούθως της σύγκρισης των αλγορίθμων μεταξύ τους, παρατηρούμε ότι όλοι οι αλγόριθμοι και ιδιαίτερα οι αλγόριθμοι AlgCSF και AlgLIF μπορούν να αποδίδουν κάτω από οποιοδήποτε σενάριο προσφέροντας ταυτόχρονα τη διασφάλιση της ορθότητας και της ολοκλήρωσης.

Φυσικά όπως προαναφέρθηκε, τα πιο πάνω και επιπλέον όπως παρατηρήσαμε η αξιοπιστία και η ανοχή σε σφάλματα, προϋποθέτει μεγάλο υπολογιστικό κόστος και μεγάλο αριθμό μηνυμάτων μεταξύ διεργασιών για κάθε είδους ενημέρωση. Εύκολα συμπεραίνουμε ότι η ύπαρξη κάθε είδους κεντρικού χρονοπρογραμματιστή, μειώνει κατά πολύ το υπολογιστικό κόστος και το κόστος μεταφοράς πληροφοριών. Δεν πρέπει όμως να παραλείψουμε το ότι ο κεντρικός χρονοπρογραμματιστής αποτελεί μοναδικό σημείο αποτυχίας *(Single Point Of Failure)*.

Επίσης είναι σημαντικό να αναφέρουμε ότι μέσω της πειραματικής αξιολόγησης στόχος δεν είναι μόνο η εύρεση του αποδοτικότερου αλγορίθμου αλλά και να παρατηρήσουμε και να επιβεβαιώσουμε την ορθή λειτουργία και τη συμπεριφορά του κάθε αλγόριθμου σε περιβάλλον προσομοίωσης.

Κεφάλαιο 8

Επίλογος

8.1 Γενικά Συμπεράσματα

Εκτός της γενικής αντίληψης γύρω από τους αλγορίθμους που αποκτήθηκε και της κατανόησης του γενικότερου προβλήματος, στη διατριβή αυτή εκπληρώθηκε ο στόχος της εργασίας, ο οποίος ήταν να υποδείξει κατά πόσο αυτοί οι αλγόριθμοι μπορούν να υλοποιηθούν και αν είναι αποδοτικοί και στην πράξη. Επιπλέων η πειραματική αξιολόγηση μας έδωσε την δυνατότητα να μελετήσουμε την συμπεριφορά των αλγορίθμων κάτω από παραμέτρους που δεν ήταν εφικτό να μελετηθούν με την θεωρητική ανάλυση.

Επίσης καταλήξαμε στο συμπέρασμα ότι οι αλγόριθμοι συμπεριφέρθηκαν όπως αναμενόταν. Αυτό ίσως είναι το σημαντικότερο συμπέρασμα, καθώς ο κάθε αλγόριθμος επιλέγεται πάντα ανάλογα με τις προδιαγραφές και τους περιορισμούς του συστήματος, που θα εφαρμοστεί. Κανένας αλγόριθμος δεν μπορεί να προσφέρει τα πάντα χωρίς το ανάλογο κόστος. Πάντα θα υπάρχει μια εξισορρόπηση μεταξύ παραμέτρων για την επιλογή του κατάλληλου αλγόριθμου.

Με λίγα λόγια, η πειραματική αξιολόγηση αναδεικνύει τα ανταλλάγματα μεταξύ απόδοσης βάσει διαφόρων μετρικών και σφαλμάτων, που επιτρέπουν την επιλογή του καταλλήλου αλγορίθμου σε συγκεκριμένη εφαρμογή.

8.2 Υλοποίηση – Προβλήματα και Αντιμετώπιση

Κατά την υλοποίηση των αλγορίθμων, ποικίλα προβλήματα προέκυψαν ιδιαίτερα λόγω της αρχικής έκδοσης που βρίσκεται η βιβλιοθήκη YALPS, με κυριότερα (α) την κακή διαχείριση μνήμης και (β) την αδυναμία της προς το παρόν να παρακολουθεί και να καταγράφει την κίνηση στο δίκτυο.

Για την αντιμετώπιση της κακής διαχείρισης της μνήμης προσπαθήσαμε να είμαστε φειδωλοί και ανά πάσα στιγμή δεν ήταν κατοχυρωμένο οτιδήποτε μη χρήσιμο. Επίσης μέσω της JAVA κατά την εκκίνηση των πειραματικών εκτελέσεων κατοχυρωνόταν μεγάλη ποσότητα μνήμης για αποκλειστική χρήση από την εφαρμογή μας.

Για την αδυναμία της βιβλιοθήκης YALPS να καταγράφει την κίνηση στο δίκτυο έγιναν κάποιες προσπάθειες, για μερική επιδιόρθωση της βιβλιοθήκης χωρίς κάποιο ουσιαστικό αποτέλεσμα.

Ένα σημαντικό πρόβλημα, το οποίο στην τελική είχε αντίκτυπο και στα αποτελέσματα, ήταν η αποστολή των μηνυμάτων. Στα αρχικά στάδια της υλοποίησης δημιουργήθηκε ένα πρωτόκολλο επικοινωνίας, το οποίο θα εσώκλειε σε ένα μήνυμα οποιαδήποτε δομή και πληροφορία θα χρειάζονταν οι αλγόριθμοι κατά την υλοποίησή τους. Δυστυχώς λόγω μεγέθους όλοι οι αλγόριθμοι σημείωναν μεγάλους χρόνους εκτέλεσης ή και αδυναμία να εκτελεστούν. Αντί αυτού προτιμήθηκε η διάσπαση σε πολλά μηνύματα τα οποία αποστέλλονται εάν είναι αναγκαίο, πάντα βάσει προδιαγραφών των αλγοριθμικών λύσεων.

Τέλος να αναφέρουμε τη στοιχειώδη τεκμηρίωση, η οποία υπάρχει για τη βιβλιοθήκη YALPS, μέρη της οποίας υπάρχουν και στη διατριβή αυτή. Τεκμηρίωση η οποία αναρτήθηκε κάτω από ιστοσελίδα της βιβλιοθήκης [15] μετά από αίτημα φοιτήτριας του τμήματος Πληροφορικής του Πανεπιστήμιου Κύπρου. Η βιβλιοθήκη YALPS περιέχει μεγάλο αριθμό εργαλείων τα οποία δίνουν τη δυνατότητα δημιουργίας κάθε ολοκληρωμένου κατανεμημένου συστήματος, αλλά λόγω έλλειψης τεκμηρίωσης είναι πολύ δύσκολο να εφαρμοστούν.

8.3 Μελλοντική Εργασία

Κλείνοντας να αναφέρουμε ότι οι αλγόριθμοι που υλοποιήθηκαν εκτελέστηκαν μόνο κάτω από ελεγχόμενο περιβάλλον προσημείωσης. Θα μπορούσαν μελλοντικά να εκτελεστούν σε πραγματικό περιβάλλον με λιγότερους ή χωρίς περιορισμούς για περαιτέρω μελέτη. Ένας

όμως περιορισμός που πρέπει να ληφθεί υπόψη είναι ο συγχρονισμός των γύρων, γεγονός που περιπλέκει την υλοποίηση σε πραγματικό περιβάλλον.

Στους υφιστάμενους αλγόριθμους θα μπορούσε σε μεταγενέστερες εκδόσεις της βιβλιοθήκης YALPS να προστεθούν λειτουργίες για παρακολούθηση (α) της κίνησης στο δίκτυο και (β) του εύρους ζώνης για περεταίρω ανάλυση του όγκου πληροφορίας που μεταφέρεται μέσω δικτύου.

Επίσης θα μπορούσε να αναπτυχθεί πρωτόκολλο επικοινωνίας για τη βελτιστοποίηση της μεταξύ διεργασιών επικοινωνίας. Πρωτόκολλο το οποίο θα έχει στόχο τη μείωση των μηνυμάτων και μεταφορά της απαραίτητης πληροφορίας.

Τέλος θα μπορούσε να αναπτυχθεί υβριδικός αλγόριθμος, ο οποίος θα συνδύαζε τα πλεονεκτήματα των αλγορίθμων AlgCSF και AlgLIF σε μία αλγοριθμική λύση, όπου οι διεργασίες θα έχουν τοπικό εισαγωγέα, ελεγχόμενες από περισσότερους του ενός κεντρικούς κόμβους με απώτερο σκοπό τη μερική αποκεντροποίηση και τη μείωση των μηνυμάτων.

Βιβλιογραφία

[1] G. Cordasco, G. Malewicz and A. L. Rosenberg, "Extending IC-scheduling via the Sweep Algorithm," *J. Parallel Distrib. Comput.,* vol. 70, no. 3, pp. 201-211, 2010.

[2] Y. Emek, M. M. Halldorsson, Y. Mansour, B. Patt-Shamir, J. Radhakrishnan and D. Rawitz, "Online set packing and competitive scheduling of multi-part tasks," New York, NY, USA, 2010.

[3] C. Georgiou and A. Shvartsman, Do-All Computing in Distributed Systems: Cooperation in the Presence of Adversity, Springer, 2008.

[4] L. Hui, H. Yu and L. Xiaoming, "A lightweight execution framework for massive independent tasks," 2008.

[5] G. Malewicz, A. L. Rosenberg and M. Yurkewych, "Toward a Theory for Scheduling Dags in Internet-Based Computing," *IEEE Trans. Comput.,* vol. 55, no. 6, pp. 757-768, 2006.

[6] W. Shi and B. Hong, "Resource Allocation with a Budget Constraint for Computing Independent Tasks in the Cloud," 2010.

[7] "Amazon Elastic Compute Cloud," Amazon, [Online]. Available: http://aws.amazon.com/ec2.

[8] "Enabling Grids for E-sciencE (EGEE)," [Online]. Available: http://www.eu-egee.org.

[9] D. Werthimer, J. Cobb, M. Lebofsky, D. Anderson and E. Korpela, "SETI@HOME: massively distributed computing for SETI," *Computing in Science and Engg.,* vol. 3, no. 1, pp. 78-83, 2001.

[10] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser and G. Czajkowski, "Pregel: a system for large-scale graph processing," New York, NY, USA, 2010.

[11] C. Georgiou and D. R. Kowalski, "Performing dynamically injected tasks on processes prone to crashes and restarts," Berlin, Heidelberg, 2011.

[12] C. Dwork, J. Y. Halpern and O. Waarts, "Performing Work Efficiently in the Presence of Faults," *SIAM J. Comput.,* vol. 27, no. 5, pp. 1457-1491, 1998.

[13] P. C. Kanellakis and A. A. Shvartsman, Fault-Tolerant Parallel Computation, Norwell, MA, USA: Kluwer Academic Publishers, 1997.

[14] "Java Website," Oracle, [Online]. Available: http://www.java.com.

[15] "YALPS: Your Applications Like Planetlab & Simulation," INRIA, [Online]. Available: http://yalps.gforge.inria.fr/.

[16] "INRIA - Inventors for the digital world," [Online]. Available: http://www.inria.fr.

[17] R. J. Anderson and H. Woll, "Algorithms for the Certified Write-All Problem," *SIAM J. Comput.,* vol. 26, no. 5, pp. 1277-1283, 1997.

[18] G. Malewicz, "A Work-Optimal Deterministic Algorithm for the Certified Write-All Problem with a Nontrivial Number of Asynchronous Processors," *SIAM J. Comput.,* vol.

34, no. 4, pp. 993-1024, 2005.

[19] C. Georgiou, A. Russell and A. A. Shvartsman, "Work-Competitive Scheduling for Cooperative Computing with Dynamic Groups," *SIAM J. Comput.,* vol. 34, no. 4, pp. 848-862, 2005.

[20] G. Malewicz, A. Russell and A. Shvartsman, "Distributed scheduling for disconnected cooperation," *Distributed Computing,* vol. 18, pp. 409-420, 2006.

[21] C. Georgiou, A. Russell and A. A. Shvartsman, "The complexity of synchronous iterative Do-All with crashes," *Distrib. Comput.,* vol. 17, no. 1, pp. 47-63, 2004.

[22] B. S. Chlebus, R. D. Prisco and A. A. Shvartsman, "Performing tasks on synchronous restartable message-passing processors," *Distrib. Comput.,* vol. 14, no. 1, pp. 49-64, 2001.

[23] D. D. Sleator and R. E. Tarjan, "Amortized efficiency of list update and paging rules," *Commun. ACM,* vol. 28, no. 2, pp. 202-208, 1985.

[24] B. Awerbuch, S. Kutten and D. Peleg, "Competitive distributed job scheduling (extended abstract)," New York, NY, USA, 1992.

[25] C. Georgiou, S. Gilbert and D. R. Kowalski, "Meeting the deadline: on the complexity of fault-tolerant continuous gossip," New York, NY, USA, 2010.

[26] C. Georgiou, D. R. Kowalski and A. A. Shvartsman, "Efficient gossip and robust distributed computation," *Theor. Comput. Sci.,* vol. 347, no. 1-2, pp. 130-166, 2005.

[27] C. Georgiou and A. A. Shvartsman, Do-All Computing in Distributed Systems, Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.

Παραρτήματα

Παράρτημα Α. Πηγαίος κώδικας

Τα αρχεία του πηγαίου κώδικα βρίσκονται στον ψηφιακό δίσκο που συνοδεύει την διατριβή λόγω μεγέθους κάτω από τον φάκελο «Source Code».

Επίσης οι εκτελεστές(executors) και τα αρχεία διαμόρφωσης(configuration files) βρίσκονται κάτω από τους φακέλους «Executors» και «Configuration Files» αντίστοιχα

Παράρτημα Β. Αποτελέσματα Εκτελέσεων

Τα αποτελέσματα των εκτελέσεων και τα αρχεία ελέγχου βρίσκονται στον ψηφιακό δίσκο που συνοδεύει την διατριβή κάτω από τους φακέλους «Results» και «Debug Files» αντίστοιχα.

Επίσης κάτω από τον φάκελο «Analysis» βρίσκονται αρχεία τύπου Excel τα οποία χρησιμοποιήθηκαν για την ανάλυση των αποτελεσμάτων.