ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Υλοποίηση και Πειραματική Αξιολόγηση Ενός Αλγόριθμου Ατομικών Αντικειμένων Γραφής-Ανάγνωσης

Χάρης Λουκά

Επιβλέπων Καθηγητής
Δρ. Χρύσης Γεωργίου

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Ιούνιος 2008

Ευχαριστίες

Υπεύθυνος καθηγητής της διπλωματικής μου εργασίας ήταν ο κ. Χρύσης Γεωργίου, τον οποίο θα ήθελα να ευχαριστήσω για τη συνεργασία του, την καθοδήγησή του, τις συμβουλές, την υπομονή και το ενδιαφέρον του και γενικά για την πολύτιμη συμβολή του όλους αυτούς τους μήνες. Ο χρόνος που αφιέρωσε, το ενδιαφέρον και οι γνώσεις του ήταν καθοριστικές για την πραγμάτωση της εργασίας μου.

Περίληψη

Η διατήρηση της συνέπειας των δεδομένων σε ένα κατανεμημένο σύστημα ανταλλαγής μηνυμάτων και η διατήρηση της ατομικότητας στο σύστημα αυτό, είναι πολύ σημαντική γιατί ενισχύει την αποτελεσματικότητά στο σύστημα, καθιστά ευκολότερη την υλοποίηση κατανεμημένων εφαρμογών και μπορεί να μειώσει τον χρόνο εκτέλεσης των διεργασιών των συστημάτων αυτών. Ένας αλγόριθμος ατομικών αντικειμένων γραφής-ανάγνωσης έχει σχεδιαστεί και έχει γίνει η αυστηρή απόδειξη της ορθότητας του στο άρθρο [2], αλλά δεν έχει αξιολογηθεί η συμπεριφορά του αλγόριθμου σε πραγματικό περιβάλλον.

Σκοπός αυτής της διπλωματικής εργασίας είναι η υλοποίηση και η πειραματική αξιολόγηση ενός αλγορίθμου Ατομικών Αντικειμένων Γραφής-Ανάγνωσης μέσω του κατανεμημένου συστήματος *Planetlab*. Αυτή η αξιολόγηση είναι πολύ σημαντικό να γίνει, έτσι ώστε να δούμε τη συμπεριφορά του αλγόριθμου σε ένα ρεαλιστικό δίκτυο υπολογιστών και να την συγκρίνουμε με τη θεωρητική του συμπεριφορά και τα αποτελέσματα των προσομοιώσεων που έγιναν στο άρθρο [2]. Αναλύονται επίσης, οι μέθοδοι και οι μηχανισμοί που χρησιμοποιήθηκαν για να επιτευχθεί αυτός ο στόχος.

Η μεθοδολογία που ακολουθήθηκε για την πραγματοποίηση της εργασίας αυτής ήταν αρχικά η μελέτη του αναγκαίου υπόβαθρου για κατανόηση των βασικών εννοιών των κατανεμημένων συστημάτων και κατ' επέκταση του αλγορίθμου που επρόκειτο να υλοποιηθεί. Η υλοποίηση που έγινε χρησιμοποιεί το μοντέλο του πελάτη-εξυπηρετητή, εφαρμόζοντας κανόνες των παράλληλων και κατανεμημένων συστημάτων. Ακολούθησε η μελέτη και εκμάθηση του προγραμματισμού υποδοχών και του *Planetlab*, καθώς ο προγραμματισμός υποδοχών είναι το βασικό εργαλείο προγραμματισμού της γλώσσας *C* για υλοποίηση κατανεμημένων εφαρμογών, και το *Planetlab* ήταν η πλατφόρμα αξιολόγησης μας. Στη συνέχεια, έγινε η υλοποίηση και αποσφαλμάτωση του αλγορίθμου στις μηχανές του Πανεπιστημίου Κύπρου. Τέλος, εξάχθηκαν αποτελέσματα εκτελώντας την εφαρμογή στο *Planetlab*.
Η πειραματική αξιολόγηση που έγινε στο κατανεμημένο δίκτυο υπολογιστών *Planetlab* δείχνει ότι η υλοποίηση του αλγορίθμου ήταν επιτυχημένη καθώς μέσα από τα

αποτελέσματα διαφαίνεται η πρακτικότητα του αλγορίθμου. Επίσης, επαληθεύονται τα αποτελέσματα της προσομοίωσης που έγινε στο άρθρο [2].

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

Στο κεφάλαιο αυτό θα μιλήσουμε για τα κίνητρα που μας ώθησαν στην εκπόνηση της εργασίας αυτής, του σκοπού της, της μεθοδολογίας που ακολουθήθηκε, των πηγών από τις οποίες αντλήθηκαν οι πληροφορίες που χρησιμοποιήθηκαν και τέλος, θα δούμε τι θα ακολουθήσει στα επόμενα κεφάλαια.

1.1 Κίνητρα

Η ατομική μνήμη, δηλαδή η μνήμη στην οποία όταν γίνεται πρόσβαση από πολλές διεργασίες, είτε γραφής είτε ανάγνωσης, η εκτέλεση των διεργασιών αυτών γίνεται σειριακά και όχι παράλληλα, αποτελεί την βασικότερη αρχή αφαιρετικότητας στον παράλληλο υπολογισμό. Η διατήρηση της συνέπειας των δεδομένων της ατομικής μνήμης σε ένα κατανεμημένο σύστημα ανταλλαγής μηνυμάτων και η διατήρηση της ατομικότητας στο σύστημα αυτό, είναι πολύ σημαντική γιατί ενισχύει την αποτελεσματικότητά στο σύστημα, καθιστά ευκολότερη την υλοποίηση κατανεμημένων εφαρμογών και μπορεί να μειώσει τον χρόνο εκτέλεσης των

διεργασιών των συστημάτων αυτών. Έτσι, είναι σημαντικό να έχουμε ρεαλιστικούς αλγόριθμους ατομικών αντικειμένων γραφής-ανάγνωσης.

Ένας αλγόριθμος Ατομικών Αντικειμένων Γραφής-Ανάγνωσης έχει σχεδιαστεί και έχει γίνει η αυστηρή απόδειξη της ορθότητας του στο άρθρο [2], αλλά δεν έχει αξιολογηθεί η συμπεριφορά του αλγόριθμου σε πραγματικό περιβάλλον.

1.2 Σκοπός Διπλωματικής Εργασίας

Σκοπός αυτής της διπλωματικής εργασίας είναι η υλοποίηση και η πειραματική αξιολόγηση ενός αλγορίθμου Ατομικών Αντικειμένων Γραφής-Ανάγνωσης [2] μέσω του κατανεμημένου συστήματος *Planetlab* [11]. Αυτή η αξιολόγηση είναι πολύ σημαντικό να γίνει, έτσι ώστε να δούμε τη συμπεριφορά του αλγόριθμου σε ένα ρεαλιστικό δίκτυο υπολογιστών και να την συγκρίνουμε με τη θεωρητική του συμπεριφορά και τα αποτελέσματα των προσομοιώσεων που έγιναν στο άρθρο [2]. Αναλύονται επίσης, οι μέθοδοι και οι μηχανισμοί που χρησιμοποιήθηκαν για να επιτευχθεί αυτός ο στόχος. Ο αλγόριθμος που υλοποιήθηκε χρησιμοποιεί το μοντέλο του πελάτη-εξυπηρετητή (*client/server model*) [8], εφαρμόζοντας κανόνες των παράλληλων και κατανεμημένων συστημάτων, με τη χρήση προγραμματισμού Υποδοχών Ροής (*Stream Socket Programming*) [6]. Το *Planetlab*, όπως εξηγείται αναλυτικά στο επόμενο κεφάλαιο, είναι ένας γεωγραφικά κατανεμημένος παγκόσμιος ιστός υπολογιστών αποτελούμενος μέχρι στιγμής από πολλούς υπολογιστικούς πόρους, οι οποίοι χρησιμοποιούνται για την ανάπτυξη νέων τεχνολογιών στο χώρο της Πληροφορικής, όπως είναι η κατανεμημένη αποθήκευση (*distributed storage*), η χαρτογράφηση δικτύων (*network mapping*), τα συστήματα *peer-to-peer*, οι κατανεμημένοι πίνακες κατακερματισμού (*distributed hash tables*) και η επεξεργασία επερωτήσεων (*query processing*). Οι υπολογιστικοί αυτοί πόροι βρίσκονται διασκορπισμένοι ανά το παγκόσμιο.

Όπως προαναφέραμε, στόχος μας είναι η υλοποίηση ενός αλγόριθμου Ατομικών Αντικειμένων Γραφής-Ανάγνωσης. Ο αλγόριθμος αυτός υλοποιεί ένα μοντέλο ατομικής μνήμης σε ένα κατανεμημένο σύστημα (*Distributed System*) ανταλλαγής μηνυμάτων, όπου μπορούμε να έχουμε απώλειες μηνυμάτων, ετεροχρονισμένη παραλαβή

μηνυμάτων, ακόμα και απώλεια διεργασιών ή κατάρρευση κόμβων. Σε αυτό το σύστημα ανταλλαγής μηνυμάτων, υλοποιούμε τρεις βασικές διεργασίες, *S* διεργασίες εξυπηρετητών (*Servers*) οι οποίες θα διατηρούν την ατομική μνήμη που αναφέραμε πιο πάνω στην τοπική τους μνήμη, την διεργασία γραφής *w* (*Writer*) η οποία θα ενημερώνει την ατομική μνήμη σε κάθε γύρο επικοινωνίας (*communication round*) και τέλος τις διεργασίες ανάγνωσης *R* (*Readers*) οι οποίες θα έχουν πρόσβαση στην ατομική μνήμη για να διαβάζουν την τελευταία ενημερωμένη τιμή της ατομικής μνήμης. Αναφορικά με τα σφάλματα που αναφέρθηκαν πιο πάνω, ο αλγόριθμος επιτρέπει μέχρι και σε *t* *Servers* να καταρρεύσουν, συνεχίζοντας την ορθή λειτουργία του.

Ένα βασικό στοιχείο των συστημάτων αυτών είναι ότι οι λειτουργίες ανάγνωσης πρέπει να γράφουν. Ο αλγόριθμος αλλάζει αυτή τη λειτουργικότητα υλοποιώντας τις διεργασίες ανάγνωσης με τέτοιο τρόπο που να μην χρειάζεται να γράφουν κατά την εκτέλεση τους. Η ιδιαιτερότητα την οποία εισάγει ο αλγόριθμος είναι η ομαδοποίηση των λειτουργιών ανάγνωσης, έτσι ώστε να μειώσει το ποσοστό των διεργασιών ανάγνωσης που εκτελούν πάνω από ένα γύρο επικοινωνίας για κάθε ξεχωριστή λειτουργία ανάγνωσης. Η ομαδοποίηση έχει άμεσο αντίκτυπο στον αριθμό των μηνυμάτων που ανταλλάζουν οι *Readers* με τους *Servers* για να καθορίσουν την τελευταία ενημερωμένη τιμή της ατομικής μνήμης, καθώς δεν είναι απαραίτητο να περιμένουν αποκρίσεις από όλους τους Servers για να αποφασίσουν αν χρειάζεται δεύτερος γύρος επικοινωνίας ή όχι. Η ανάγκη δεύτερου γύρου επικοινωνίας γίνεται βάσει του αριθμού των ομάδων που έχουν δει την μέγιστη τιμή που βρέθηκε μέσα στις αποκρίσεις των *Servers*, αντί να γίνει βάσει του αριθμού των *Readers* που έχουν δει την τιμή αυτή.

Η αυστηρή απόδειξη ορθότητας του αλγορίθμου που έχει γίνει στο [2], επισημαίνει την ύπαρξη μερικών περιορισμών που πρέπει να ικανοποιούνται για να είναι ορθή η υλοποίηση μας. Περιληπτικά, ο αριθμός *t* που περιγράφει τον αριθμό των *Servers* που δύναται να καταρρεύσουν, φράσσεται από την ανισότητα $t < \frac{S}{2}$, εννοώντας ότι πρέπει να έχουμε ζωντανούς τουλάχιστον πάνω από τους μισούς *Servers* (και κατ' επέκταση αντικείμενα ατομικής μνήμης) για μην παραβιάζεται η ορθότητα. Ακόμα, ο αριθμός *V* των ομάδων που θα δημιουργηθούν ή νοητά αναγνωριστικά (*virtual identifiers*) όπως

αναφέρονται πιο κάτω, δεν πρέπει να υπερβαίνουν την σχέση $V < \frac{S}{t} - 2$. Τέλος, κατά την πραγματοποίηση δεύτερου γύρου επικοινωνίας ένας *Reader* πρέπει να στείλει τουλάχιστον $3t+1$ μηνύματα ενημέρωσης σε αντίστοιχο αριθμό *Servers*.

Συνδυάζοντας πολλές διεργασίες, η εκτέλεση των διεργασιών που αποτελούν τα στοιχεία του αλγόριθμου, θα αποτελείται από πολλά σενάρια και για αυτό το λόγο θα χρειαστούμε αρκετούς υπολογιστικούς πόρους. Τα σενάρια τα οποία αποτελούν την πειραματική αξιολόγηση διαφέρουν σε κάποιες παραμέτρους, όπως αριθμός μηχανών που χρησιμοποιείται, αριθμός *Readers R*, αριθμός των *Servers* που μπορούν να καταρρεύσουν *t*, έτσι ώστε να έχουμε αρκετά αποτελέσματα όσο αφορά την επίδοση του αλγορίθμου και τον τρόπο που ανταποκρίνεται σε κάθε περίπτωση. Τα σενάρια αυτά εκτελέστηκαν για εκατό λειτουργίες ανάγνωσης μετρώντας κάθε φορά τον αριθμό των δεύτερων γύρων επικοινωνίας που εκτέλεσε κάθε διεργασία ξεχωριστά, υπολογίζοντας το μέσο όρο δεύτερων γύρων επικοινωνίας σε κάθε εκτέλεση του εκάστοτε σεναρίου.

1.3 Μεθοδολογία

Για την εργασία αυτή, ακολουθήθηκε η εξής μεθοδολογία. Αρχικά, μελετήθηκε το αναγκαίο υπόβαθρο για την κατανόηση του αλγόριθμου Ατομικών Αντικειμένων Γραφής-Ανάγνωσης που υλοποιήθηκε, έτσι ώστε να γίνει μια πρώτη εισαγωγή στον κατανεμημένο υπολογισμό και στις βασικές αρχές που τον διέπουν. Αυτό βοήθησε στην διαφοροποίηση του τρόπου σκέψης μας και την μετάβαση από τον σειριακό στον παράλληλο υπολογισμό, με κύριο στόχο να κατανοήσουμε πλήρως τη λειτουργικότητα του αλγόριθμου που υλοποιήσαμε. Ακολούθως, μελετήθηκε διεξοδικά ο αλγόριθμος και οι λειτουργίες που υποστηρίζει και οι περιορισμοί που πρέπει να ικανοποιούνται για να λειτουργεί ορθά. Στη συνέχεια, για σκοπούς υλοποίησης αναπτύχθηκε μια μικρή εφαρμογή πελάτη-εξυπηρετητή για εξοικείωση με το μοντέλο αυτό καθώς είναι το βασικό προγραμματιστικό μοντέλο της υλοποίησης. Το επόμενο στάδιο ήταν η υλοποίηση και η απασφαλμάτωση του αλγόριθμου στο τοπικό δίκτυο του Πανεπιστημίου Κύπρου. Στο τελικό στάδιο μελετήθηκε το κατανεμημένο σύστημα υπολογιστών *Planetlab*, το οποίο είναι η βασική πλατφόρμα αξιολόγησης του

αλγόριθμου που υλοποιήσαμε. Μετά την εκμάθηση του *Planetlab*, εκτελέστηκαν διάφορα σενάρια για την συγκομιδή αποτελεσμάτων με σκοπό τη σύγκριση με την προσομοίωση που έγινε στο άρθρο [2].

1.4 Πηγές Άντλησης Πληροφορίας

Οι πηγές για την υλοποίηση αυτής της Ατομικής Διπλωματικής Εργασίας πηγάζουν κυρίως από το άρθρο [2], όπου έγινε πλήρης περιγραφή του αλγορίθμου, αυστηρή απόδειξη της ορθότητας του και προσομοίωση στο προσομοιωτή δικτύων *ns-2* για την πειραματική αξιολόγηση του. Το βιβλίο [5] χρησιμοποιήθηκε για την εξοικείωση με τις βασικές αρχές του παράλληλου προγραμματισμού, καθώς επίσης τα βιβλία [4,6,7,8] βοήθησαν στην κατανόηση της λειτουργικότητας των βασικών αρχών του προγραμματισμού Υποδοχών Ροής και στην γενικότερη λειτουργία των δικτύων υπολογιστών. Η επίσημη ιστοσελίδα του *Planetlab* [11] παρέχει όλες τις πληροφορίες που χρειάζεται κάποιος για να γίνει μέλος και να χρησιμοποιήσει όλες τις υπηρεσίες του. Το λειτουργικό σύστημα *Fedora* [10] κάτω από το οποίο τρέχουν όλες οι μηχανές του *Planetlab* εκτός από τις δικές του ξεχωριστές χρήσεις [12], συμπεριφέρεται όπως όλα τα υπόλοιπα λογισμικά σε περιβάλλον *Unix* [7]. Τόσο το λειτουργικό σύστημα *Fedora* όσο και η γλώσσα προγραμματισμού *C* είναι "ανοικτής πηγής" (*open source*), πράγμα που σημαίνει ότι οποιοσδήποτε μπορεί να χρησιμοποιήσει όλες τις υπηρεσίες τους δωρεάν χωρίς περιορισμούς.

1.5 Τι θα ακολουθήσει

Στο επόμενο κεφάλαιο θα δούμε αναλυτικά τι ακριβώς είναι ο προγραμματισμός Υποδοχών Ροής, μια γενική περιγραφή του *Planetlab* και θα μελετήσουμε μερικούς σημαντικούς όρους για τον αλγόριθμο, όπως είναι η ατομικότητα. Στο Κεφάλαιο 3 θα επεξηγηθεί αναλυτικά ο αλγόριθμος Ατομικών Αντικειμένων Γραφής-Ανάγνωσης που υλοποιήθηκε, η μεθοδολογία που ακολουθήθηκε και ο τρόπος που εκτελείται η εφαρμογή. Στο Κεφάλαιο 4 θα μελετήσουμε αναλυτικά όλα τα πειράματα που έγιναν στο *Planetlab*. Θα παρουσιάσουμε πειραματικά αποτελέσματα από διάφορα σενάρια και θα εισάγουμε όλα τα συμπεράσματα που πηγάζουν από αυτά. Τέλος, στο Κεφάλαιο 5 θα δώσουμε τα τελικά συμπεράσματα που πηγάζουν συγκεντρωτικά από την μελέτη

αυτή, τα προβλήματα που παρουσιάστηκαν και ο τρόπος αντιμετώπισης τους, καθώς και μελλοντικές εργασίες που δύναται να προκύψουν από την εργασία αυτή.

Κεφάλαιο 2

Υπόβαθρο

Στο κεφάλαιο αυτό θα επεξηγηθούν ορισμοί που αφορούν το υπόβαθρο του αλγορίθμου που υλοποιήθηκε, όπως είναι η ατομικότητα, οι γρήγορες και ημιγρήγορες (*fast and semifast*) υλοποιήσεις, το μοντέλο που χρησιμοποιήθηκε και η προηγούμενη εργασία που έχει γίνει στο τομέα αυτό. Ακόμα, θα μιλήσουμε για τις Υποδοχές Ροής (*Stream sockets*) και πως χρησιμοποιούνται και θα γίνει μια περιληπτική αναφορά στο κατανεμημένο δίκτυο υπολογιστών *Planetlab*.

2.1 Πρόλογος

Προτού μιλήσουμε για οτιδήποτε άλλο, είναι αναγκαίο να εξηγήσουμε τους μηχανισμούς πάνω στους οποίους κτίστηκε αυτή η διπλωματική και να επεξηγήσουμε

τους ορισμούς που χρησιμοποιούνται στον αλγόριθμο. Ο λόγος είναι διότι αυτά που θα ειπωθούν σε αυτό το κεφαλαίο είναι καθοριστικά όσο αφορά στην κατανόηση του προβλήματος, στην κατανόηση της εφαρμογής και στα συμπεράσματα που θα ακολουθήσουν. Συγκεκριμένα θα εξηγήσουμε τι ακριβώς είναι η ατομικότητα και οι γρήγορες (*fast*) [1] και ημιγρήγορες (*semifast*) [2] υλοποιήσεις, τι είναι ο προγραμματισμός Υποδοχών Ροής (*Stream socket programming*) [6], με τη βοήθεια του οποίου υλοποιήθηκε αυτή η εφαρμογή και τι είναι το κατανεμημένο σύστημα *Planetlab* [11], στο οποίο έτρεξε ο αλγόριθμος Ατομικών Αντικειμένων Γραφής-Ανάγνωσης.

2.1.1 Ατομικότητα

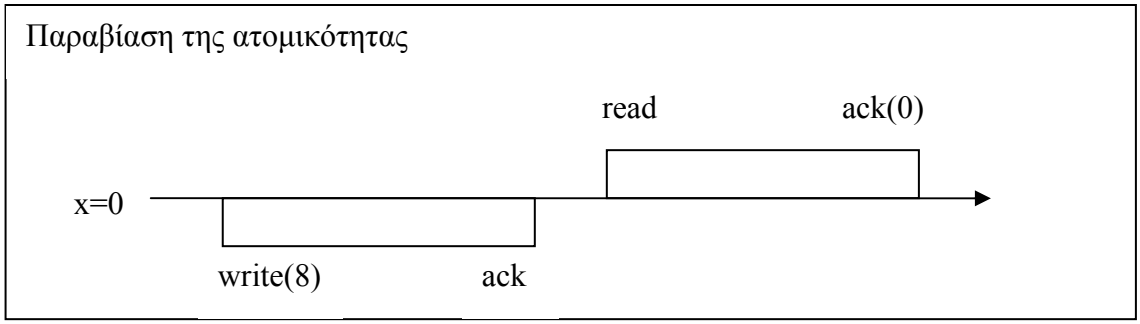
Έχουμε ήδη αναφέρει σε προηγούμενο κεφάλαιο τι είναι η ατομική μνήμη. Απόρροια του ορισμού της είναι τα ατομικά αντικείμενα. Ατομικά αντικείμενα ονομάζονται τα αντικείμενα στα οποία όταν επιτρέπεται πρόσβαση σε αυτά από πολλές διεργασίες είτε γραφής είτε ανάγνωσης, οι διεργασίες αυτές εκτελούνται με μια καθορισμένη σειρά και όχι παράλληλα. Αυτή η συνέπεια στην διατήρηση των πληροφοριών του ατομικού αντικειμένου ονομάζεται ατομικότητα [9].

Ο στόχος στην υλοποίηση του αλγόριθμου είναι η διατήρηση της ατομικότητας (συνέπειας) σε ένα σύστημα ανταλλαγής μηνυμάτων, αντιγράφοντας την τιμή του αντικειμένου ανάμεσα στους εξυπηρετητές (*Servers*) του συστήματος. Κάθε αντίγραφο αποτελείται από δύο τιμές, την τιμή (*value*) και τη χρονική σφραγίδα (*timestamp*) που ενημερώθηκε η τιμή. Κάθε λειτουργία γραφής ή ανάγνωσης αποτελείται από το βήμα ενεργοποίησης (*invocation step*) και το βήμα απόκρισης (*response step*). Μια λειτουργία θεωρείται ημιτελής (*incomplete*) αν το βήμα ενεργοποίησης δεν έχει το αντίστοιχο βήμα απόκρισης, αλλιώς θεωρείται ολοκληρωμένη (*complete*). Υποθέτουμε ότι κάθε διεργασία εκτελεί μία λειτουργία κάθε φορά, δηλαδή περιμένει απόκριση πριν να ενεργοποιήσει μια άλλη λειτουργία. Έτσι, μπορούμε να πούμε ότι μια λειτουργία π_1 προηγείται (*precedes*) μιας λειτουργίας π_2, ή η π_2 έπεται (*succeeds*) της π_1, αν το βήμα απόκρισης της π_1 προηγείται του βήματος ενεργοποίησης της π_2. Δύο λειτουργίες καλούνται ταυτόχρονες (*concurrent*) αν καμία δεν προηγείται της άλλης. Τέλος, η ατομικότητα δεν παραβιάζεται αν ικανοποιούνται οι εξής συνθήκες:

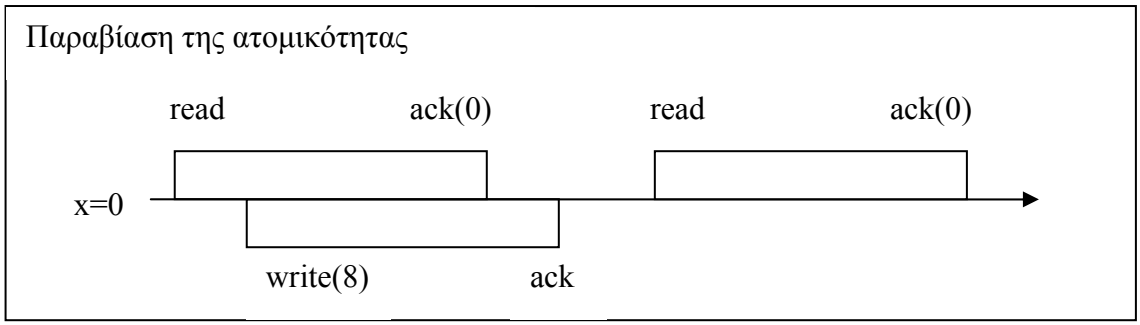
- Η τιμή που θα επιστρέφει μια λειτουργία ανάγνωσης να είναι η τιμή της τελευταίας ολοκληρωμένης γραφής ή η τιμή μιας γραφής που συμβαίνει το ίδιο χρονικό διάστημα (χωρίς να έχει ολοκληρωθεί)

- Η τιμή που θα επιστρέψει η επόμενη λειτουργία ανάγνωσης είτε είναι η ίδια με την προηγούμενη ανάγνωση, είτε η τιμή μιας γραφής μεταγενέστερης της προηγούμενης ανάγνωσης

Στα Σχήματα 2.1 και 2.2 που ακολουθούν παρουσιάζονται δύο παραδείγματα παραβίασης της ατομικότητας. Πιο συγκεκριμένα, στο Σχήμα 2.1 η ατομικότητα παραβιάζεται διότι η διεργασία *read* δεν επιστρέφει την τιμή της τελευταίας διεργασίας γραφής, έτσι παραβιάζεται η πρώτη συνθήκη διατήρησης της ατομικότητας. Στο Σχήμα 2.2 η ατομικότητα παραβιάζεται διότι η δεύτερη διεργασία *read* δεν επιστρέφει την τιμή της μεταγενέστερης διεργασίας γραφής της πρώτης διεργασίας *read* και έτσι παραβιάζεται η δεύτερη συνθήκη διατήρησης της ατομικότητας.



Σχήμα 2.1 Παράδειγμα παραβίασης της ατομικότητας



Σχήμα 2.2 Παράδειγμα παραβίασης της ατομικότητας

2.1.2 Μοντέλο

Εξετάζουμε το μοντέλο *SWMR* (*Single Writer Multiple Readers*). Στην υλοποίηση του μοντέλου αυτού που δίνουμε, υπάρχουν τρία είδη διεργασιών οι οποίες είναι η διεργασία *Server*, η διεργασία *Writer* και η διεργασία *Reader*. Ο *Writer* είναι μια διακεκριμένη διεργασία *w*. Υπάρχουν *R Readers*, οι οποίοι είναι διαδικασίες με μοναδικά αναγνωριστικά (*unique ids*) από το σύνολο $R = \{r_1, \ldots, r_R\}$. Ο *Writer* και οποιοδήποτε υποσύνολο από τους *Readers* μπορεί να καταρρεύσει (*crash*). Το ζεύγος της τιμής (*value*) μαζί με την αντίστοιχη χρονική σφραγίδα (*timestamp*), το οποίο αποτελεί το ατομικό αντικείμενο, είναι αποθηκευμένο στους *S Servers*, οι οποίοι είναι διεργασίες με μοναδικά αναγνωριστικά (*unique ids*) από το σύνολο $S = \{s_1, \ldots, s_S\}$. Οποιοδήποτε *t* υποσύνολο από τους *Servers* μπορεί να καταρρεύσει, έτσι ώστε να ισχύει $t < \frac{S}{2}$. Ένας νοητός κόμβος (*virtual node*) είναι μια αφηρημένη οντότητα που αποτελείται από ένα σύνολο από διεργασίες *Readers*. Κάθε νοητός κόμβος έχει μοναδική ταυτότητα από το σύνολο $V=\{v_1, \ldots, v_V\}$, όπου $V < \frac{S}{t} - 2$. Ένας *Reader* r_i ο οποίος ανήκει σε ένα νοητό κόμβο, διατηρεί την ταυτότητα του r_i, καθώς και την ταυτότητα του νοητού κόμβου στον οποίο ανήκει (*virtual identifier*) τέτοιο ώστε $v(r_i) = v_j$. Μια τέτοια διεργασία αναγνωρίζεται από τη δυάδα $\langle r_i, v_j \rangle$. Όσες διεργασίες έχουν το ίδιο νοητό αναγνωριστικό ονομάζονται διεργασίες "αδέρφια". Σημειώνεται ότι το νοητό αναγνωριστικό σε κάθε *Reader* μπορεί να υπολογιστεί με απλό τοπικό υπολογισμό, απλά με τη γνώση των *S*, *t* και $V = \frac{S}{t} - 3$. Πιο συγκεκριμένα, το νοητό αναγνωριστικό υπολογίζεται ως εξής: $v(r_i) = r_i \bmod V$. Με τη χρήση του τύπου αυτού παρατηρείται μια ομοιόμορφη κατανομή των *Readers* στους νοητούς κόμβους.

2.1.3 Γρήγορες-Ημιγρήγορες Υλοποιήσεις

Στη συνέχεια, ακολουθούν ορισμοί εννοιών οι οποίες είναι πολύ σημαντικές για την κατανόηση του αλγορίθμου.

Πότε μια λειτουργία γραφής-ανάγνωσης ολοκληρώνεται σε ένα γύρο επικοινωνίας: Μια διεργασία p ολοκληρώνει ένα γύρο επικοινωνίας (*communication round*), δηλαδή μια λειτουργία γραφής-ανάγνωσης ολοκληρώνεται σε ένα γύρο επικοινωνίας, όταν κατά τη διάρκεια μιας λειτουργίας π αν ισχύουν τα πιο κάτω:

- Η διεργασία p στέλνει ένα μήνυμα m κατά τη διάρκεια του βήματος ενεργοποίησης της λειτουργίας π ή στο βήμα επικοινωνίας της π, σε ένα υποσύνολο διεργασιών

- Οποιαδήποτε διεργασία p' που παραλαμβάνει το μήνυμα m από την p σε ένα βήμα σ, απαντά στην p με ένα μήνυμα m' κατά τη διάρκεια του ιδίου βήματος

- Όταν η p παραλαμβάνει τουλάχιστον ένα μήνυμα απάντησης m', είτε εκτελεί ένα βήμα απόκρισης της π, είτε εισάγει ένα σύνολο μηνυμάτων m σύνολο μηνυμάτων που πρόκειται να στείλει και εκτελεί ένα βήμα επικοινωνίας

Γρήγορες (*fast*) λειτουργίες γραφής-ανάγνωσης: Μια λειτουργία π η οποία ενεργοποιήθηκε από μια διεργασία p είναι γρήγορη εάν τερματίσει κατά τη διάρκεια του πρώτου γύρου επικοινωνίας ακολουθώντας την ενεργοποίηση της π.

Γρήγορη (*fast*) Υλοποίηση Ατομικού Αντικειμένου Γραφής-Ανάγνωσης: Μια υλοποίηση ενός ατομικού αντικειμένου είναι γρήγορη, αν όλες οι λειτουργίες γραφής-ανάγνωσης είναι γρήγορες σε κάθε εκτέλεση.

Ημιγρήγορη (*Semifast*) Υλοποίηση Ατομικού Αντικειμένου Γραφής-Ανάγνωσης: Απόρροια της γρήγορης υλοποίησης ατομικών αντικειμένων είναι η ημιγρήγορη υλοποίηση ατομικών αντικειμένων. Μια υλοποίηση καλείται ημιγρήγορη (*Semifast*) όταν όλες οι λειτουργίες ανάγνωσης ή όλες οι λειτουργίες γραφής είναι γρήγορες. Στη δική μας περίπτωση θεωρούμε ότι όλες οι λειτουργίες γραφής είναι γρήγορες. Συγκεκριμένα, για να είναι μια υλοποίηση ατομικών αντικειμένων ημιγρήγορη πρέπει να ικανοποιεί τις εξής συνθήκες:

- Σε κάθε εκτέλεση της υλοποίησης, κάθε λειτουργία γραφής είναι γρήγορη

- Σε κάθε εκτέλεση της υλοποίησης, κάθε ολοκληρωμένη λειτουργία ανάγνωσης εκτελεί το πολύ δύο γύρους επικοινωνίας μεταξύ των βημάτων ενεργοποίησης και απόκρισης

- Σε κάθε εκτέλεση της υλοποίησης, αν p_1 είναι μια λειτουργία ανάγνωσης δύο κύκλων επικοινωνίας, τότε οποιαδήποτε λειτουργία ανάγνωσης p_2 η οποία διαβάζει την τιμή που προήλθε από την λειτουργία γραφής που διάβασε και η p_1 θα είναι γρήγορη

- Υπάρχει τουλάχιστον μια εκτέλεση όπου υπάρχουν μια λειτουργία γραφής *w* και μια λειτουργία ανάγνωσης p_1 που είναι ταυτόχρονες, όπου η p_1 διαβάζει την τιμή που έγραψε η *w*, έτσι ώστε όλες οι λειτουργίες ανάγνωσης συμπεριλαμβανομένης και της p_1 να είναι γρήγορες

Τώρα θα εξετάσουμε το σχήμα επικοινωνίας του μοντέλου *SWMR*. Δεδομένου ότι οποιοδήποτε υποσύνολο των *Readers* και ο *Writer* μπορεί να καταρρεύσουν, για να εγγυηθούμε τερματισμό της εφαρμογής, δεν μπορούμε να επιτρέψουμε σε έναν *Reader* ή στον *Writer* να περιμένουν αποκρίσεις από διεργασίες αυτού του είδους. Εφόσον θέλουμε οι λειτουργίες γραφής να είναι γρήγορες, οι *Servers* δεν πρέπει να περιμένουν αποκρίσεις από άλλες διεργασίες για να μπορέσουν να ανταποκριθούν σε μια λειτουργία γραφής. Από την άλλη, οι λειτουργίες ανάγνωσης μπορούν να εκτελούν δύο γύρους επικοινωνίας. Αυτό θα μπορούσε να γίνει με δύο τρόπους: (α) Ο *Reader* θα επικοινωνεί δύο φορές με τους *Servers*, (β) Ο *Reader* θα στέλνει μηνύματα στους *Servers* κατά τη διάρκεια του πρώτου γύρου επικοινωνίας, οι *Servers* θα εκτελούν ένα βήμα επικοινωνίας, θα επικοινωνούν με τους υπόλοιπους *Servers* στο δεύτερο γύρο επικοινωνίας και θα αποκρίνονται στον *Reader* τελειώνοντας έτσι τον πρώτο γύρο επικοινωνίας. Εάν γίνει το (β), εάν δηλαδή οι *Servers* είναι υπεύθυνοι για το δεύτερο γύρο επικοινωνίας και όλες οι λειτουργίες ανάγνωσης χρειάζονται δύο γύρους επικοινωνίας, τότε θα παραβιάζονται οι συνθήκες 3 και 4 της ημιγρήγορης υλοποίησης. Ακόμα χειρότερα, εάν ένας *Server* καταρρεύσει θα εμποδίσει μια λειτουργία ανάγνωσης από το να τερματίσει. Έτσι, είναι αναγκαίο ο δεύτερος γύρος επικοινωνίας

να εκτελείται από τον *Reader* ανάλογα με τις πληροφορίες που συνέλεξε στον πρώτο γύρο επικοινωνίας. Τέλος, υποθέτουμε ότι σε μια ημιγρήγορη υλοποίηση οι *Servers* θα αποκρίνονται αμέσως όταν παραλαμβάνουν ένα μήνυμα γραφής (*WRITE message*), ανάγνωσης (*READ message*) ή ενημέρωσης (*INFORM message*).

2.1.4 Προηγούμενη Εργασία

Στα κατανεμημένα συστήματα (*Distributed Systems*) η κατά συνθήκη λειτουργικότητα των διεργασιών γραφής και ανάγνωσης είναι να εκτελούν δύο γύρων επικοινωνίας έτσι ώστε να λαμβάνουν ή να ενημερώνουν αντίστοιχα την τελευταία τιμή των ατομικών αντικειμένων, διατηρώντας την ατομικότητα. Έχουν γίνει αρκετές προσπάθειες στον τομέα αυτό για να υλοποιηθούν συστήματα όπου οι λειτουργίες αυτές να είναι γρήγορες. Αρχικά, οι συγγραφείς του άρθρου [1] έχουν κάνει υλοποιήσεις για το μοντέλο *SWMR* (*Single Writer Multiple Readers*), όπου οι διεργασίες γραφής και ανάγνωσης είναι εγγυημένο ότι θα τερματίσουν φτάνει να μην καταρρεύσουν περισσότεροι από τους μισούς εξυπηρετητές. Στην υλοποίηση αυτή, οι διεργασίες γραφής ολοκληρώνονται σε ένα γύρο επικοινωνίας και οι διεργασίες ανάγνωσης ολοκληρώνονται σε δύο γύρους επικοινωνίας. Στο ίδιο άρθρο [1] έχει γίνει υλοποίηση για το μοντέλο *MWMR* (*Multiple Writer Multiple Readers*), στην οποία οι διεργασίες γραφής και ανάγνωσης χρησιμοποιούν πρωτόκολλο δύο γύρων επικοινωνίας.

Παρόλα ταύτα, οι συγγραφείς του άρθρου [3] έχουν δείξει ότι είναι δυνατόν να υπάρξουν γρήγορες (*fast*) υλοποιήσεις, όπου δηλαδή οι λειτουργίες γραφής και ανάγνωσης θα εκτελούνται σε ένα γύρο επικοινωνίας, περιορίζοντας όμως τον αριθμό των διεργασιών ανάγνωσης σε $R < \frac{S}{t} - 2$. Αυτός ο περιορισμός είναι απαγορευτικός και καθόλου ρεαλιστικός, καθώς για να μπορέσουμε να αυξήσουμε τον αριθμό των διεργασιών ανάγνωσης στο σύστημα πρέπει να αυξήσουμε και τον αριθμό των αντιγράφων των ατομικών αντικειμένων. Αυτό δεν είναι καθόλου πρακτικό και πολύ ακριβό για ένα ρεαλιστικό κατανεμημένο σύστημα καθώς θα απαιτούσε ένα μεγάλο αριθμό εξυπηρετητών για τη λειτουργία του. Ακόμα, στο άρθρο αυτό έχει αποδειχθεί ότι δεν είναι δυνατόν να υπάρξουν γρήγορες (*Fast*) υλοποιήσεις για το μοντέλο *MWMR* (*Multiple Writers Multiple Readers*).

Τέλος, στο άρθρο [2] παρουσιάζεται μια ημιγρήγορη (*Semifast*) υλοποίηση η οποία έχει ως στόχο να αφαιρέσει τον περιορισμό του αριθμού των διεργασιών ανάγνωσης που μπορούν να υπάρξουν στο σύστημα. Αυτό γίνεται αναθέτοντας σε κάθε διεργασία ανάγνωσης ένα νοητό αναγνωριστικό (*virtual identifier*), εφαρμόζοντας έτσι τον περιορισμό στον αριθμό των νοητών κόμβων (*virtual nodes*) που μπορούν να υπάρξουν στο σύστημα ως εξής $V < \frac{S}{t} - 2$. Για να επιτευχθεί αυτή η αλλαγή του περιορισμού, είμαστε υποχρεωμένοι να ανεχθούμε ένα ποσοστό διεργασιών ανάγνωσης να εκτελούν δεύτερο γύρο επικοινωνίας. Στο ίδιο άρθρο [2] έχει αποδειχθεί ότι δεν είναι δυνατόν να υπάρξουν ημιγρήγορες υλοποιήσεις για το μοντέλο *MWMR* (*Multiple Writers Multiple Readers*).

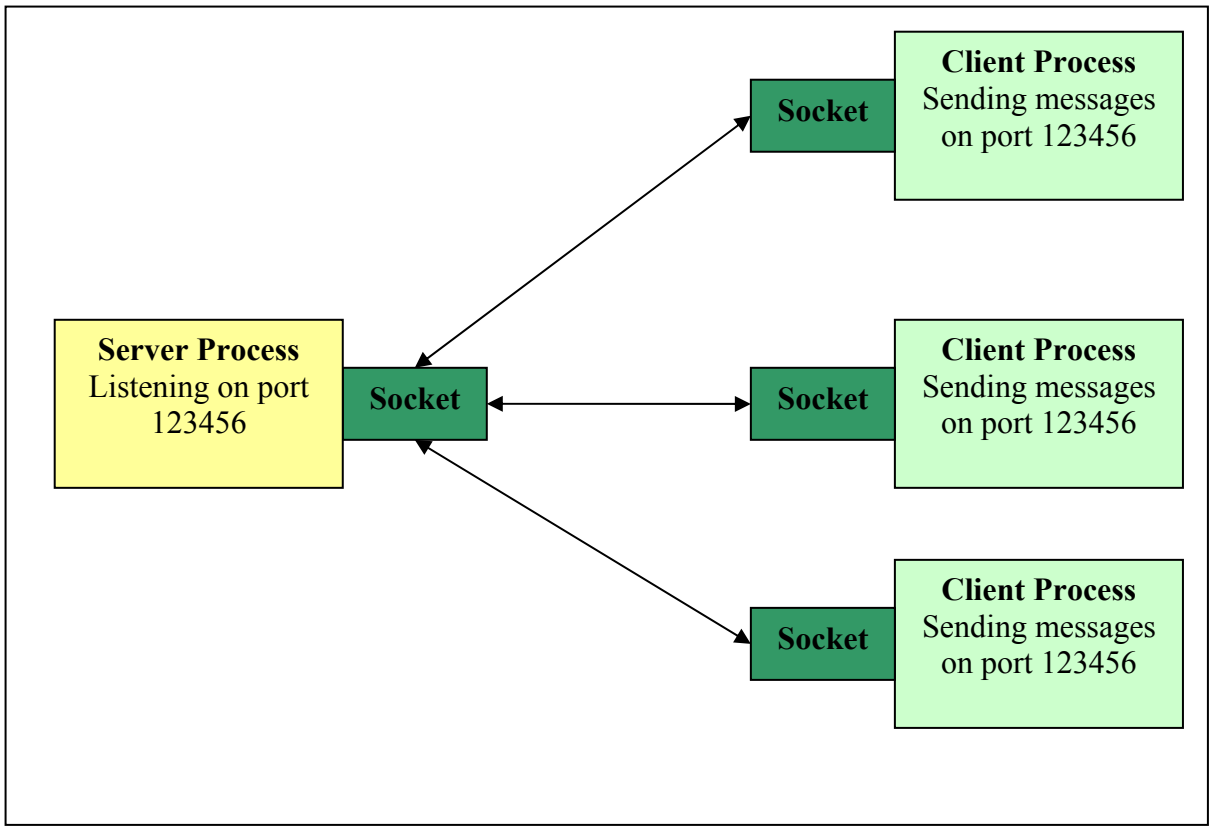
2.2 Προγραμματισμός Υποδοχών Ροής

Το μοντέλο πελάτη-εξυπηρετητή περιγράφει την επικοινωνία μεταξύ δύο διεργασιών (*processes*) κατά τη διάρκεια της οποίας η διεργασία πελάτης (*client*) στέλνει μια αίτηση (*request*) στην διεργασία εξυπηρετητή (*server*) που εκτελεί την αίτηση. Αν και το μοντέλο πελάτη-εξυπηρετητή μπορεί να χρησιμοποιηθεί από τα προγράμματα μέσα σε έναν ενιαίο υπολογιστή, η σημαντικότερη χρήση του είναι μέσα σε ένα δίκτυο. Από τις δύο αυτές χρήσεις του μοντέλου πηγάζει και ο διαχωρισμός των υποδοχών ροής σε δύο πεδία, το πεδίο υποδοχών *Unix* (*Unix domain*) και το πεδίο υποδοχών *Internet* (*Internet domain*). Σε ένα δίκτυο, το μοντέλο πελάτη-εξυπηρετητή παρέχει έναν κατάλληλο τρόπο να διασυνδεθούν αποτελεσματικά τα προγράμματα που βρίσκονται σε διαφορετικές γεωγραφικές τοποθεσίες. Οι συναλλαγές υπολογιστών που χρησιμοποιούν το μοντέλο πελάτη-εξυπηρετητή είναι πολύ κοινές. Οι περισσότερες εφαρμογές διαδικτύου, όπως το ηλεκτρονικό ταχυδρομείο, η πρόσβαση Ιστού και η πρόσβαση βάσεων δεδομένων, είναι βασισμένες στο μοντέλο πελάτη-εξυπηρετητή. Παραδείγματος χάριν, ο φυλλομετρητής ιστοσελίδων (*Web browser*) είναι ένα πρόγραμμα πελάτη στον υπολογιστή του χρήστη που επιθυμεί να έχει πρόσβαση στις πληροφορίες σε οποιοδήποτε κεντρικό υπολογιστή δικτύου στον κόσμο.

Κατά τη δημιουργία των υποδοχών, πρέπει να καθοριστεί το πεδίο που θα χρησιμοποιεί η υποδοχή. Έχουμε επιλέξει τη χρήση του πεδίου *Internet* διότι το *Planetlab* χρησιμοποιεί το συγκεκριμένο πεδίο υποδοχών.

2.2.1 Τι είναι οι Υποδοχές Ροής

Ένας τρόπος επικοινωνίας μεταξύ διεργασιών είναι με τη χρήση Υποδοχών ροής (*Stream Socket*). Υποδοχή ροής (*Stream Socket*) ονομάζεται το κάθε τελικό σημείο ενός αμφίδρομου διαύλου επικοινωνίας (*two-way communication link*) μεταξύ δύο διεργασιών (*processes*) που εκτελούνται σε ένα δίκτυο. Η ταυτοποίηση γίνεται με το συνδυασμό μιας *IP* (*Internet Protocol*) διεύθυνσης και ενός αριθμού θύρας (*port number*). Η υποδοχή συνδέει τον εξυπηρετητή με μια συγκεκριμένη θύρα στη μηχανή στην οποία τρέχει, έτσι οποιοδήποτε πρόγραμμα πελάτη οπουδήποτε στο δίκτυο, μπορεί να επικοινωνήσει με το πρόγραμμα του εξυπηρετητή με μια υποδοχή ροής που συνδέεται με την ίδια θύρα εάν χρησιμοποιείται το ίδιο πεδίο υποδοχής, το οποίο στην περίπτωση μας είναι το πεδίο *Internet* (*Internet domain*).



Σχήμα 2.3 Παράδειγμα επικοινωνίας Εξυπηρετητών-Πελατών

Όπως φαίνεται στο Σχήμα 2.3, ένας εξυπηρετητής παρέχει τους τυπικούς πόρους σε ένα δίκτυο από διεργασίες πελατών. Οι διεργασίες πελατών στέλνουν τα αιτήματα στον εξυπηρετητή και ο εξυπηρετητής ανταποκρίνεται σε αυτά στην αντίστοιχη θύρα, μέσω της υποδοχής ροής που έχουν ήδη δημιουργήσει εξυπηρετητής και πελάτης.

Ένας τρόπος να αντιμετωπιστούν αιτήματα από περισσότερους από έναν πελάτες είναι να γίνει ο εξυπηρετητής πολυνηματικός (*multithreaded*). Ένας πολυνηματικός εξυπηρετητής δημιουργεί ένα νήμα για κάθε αίτημα που δέχεται από έναν πελάτη. Ένα νήμα είναι μια ακολουθία εντολών η οποία τρέχει ανεξάρτητα από τον πελάτη και από οποιαδήποτε άλλα νήματα.

Χρησιμοποιώντας νήματα, ένας πολυνηματικός εξυπηρετητής μπορεί να δεχτεί μια σύνδεση από έναν πελάτη, να ξεκινήσει ένα νήμα (*thread*) για εκείνη την επικοινωνία, και να συνεχίσει να ανταποκρίνεται στα αιτήματα από άλλους πελάτες.

Οι υποδοχές ροής υλοποιούνται χρησιμοποιώντας δύο πρωτόκολλα επικοινωνίας μεταξύ διεργασιών, το *TCP* (*Transmission Control Protocol*) και το *UDP* (*User Datagram Protocol*). Έχουμε επιλέξει το πρωτόκολλο *TCP*. Το πρωτόκολλο αυτό παρέχει αξιόπιστη μεταφορά των μηνυμάτων μεταξύ των επικοινωνούντων διεργασιών με τη βοήθεια μηχανισμών ελέγχου, εγκαθίδρυσης και τερματισμού επικοινωνίας, σε αντίθεση με το πρωτόκολλο UDP το οποίο είναι μεν γρηγορότερο, καθώς δεν υποστηρίζει εγκαθίδρυση επικοινωνίας μεταξύ των διεργασιών, αλλά δεν είναι αξιόπιστο.

Το πρωτόκολλο *TCP* λειτουργεί ως εξής:

Αρχικά, εγκαθιδρύεται η επικοινωνία μεταξύ των διεργασιών μέσω του "*3-way handshake*". "*3-way handshake*" καλείται η ανταλλαγή μηνυμάτων για πληροφορίες που αφορούν την σύνδεση που πρόκειται να εγκαθιδρυθεί. Το "*3-way handshake*" λειτουργεί ως εξής. Αρχικά, η διεργασία πελάτη στέλνει αίτημα σύνδεσης (*connection request*) στη διεργασία εξυπηρετητή. Ακολούθως, η διεργασία εξυπηρετητή ανταποκρίνεται στέλνοντας μήνυμα εγκαθίδρυσης της σύνδεσης στην διεργασία πελάτη το οποίο περιέχει πληροφορίες σχετικά με τη σύνδεση και τις παραμέτρους της. Στη

συνέχεια, οι διεργασίες ξεκινούν τη διαδικασία ανταλλαγής μηνυμάτων μεταξύ τους όπου το πρωτόκολλο *TCP* εφαρμόζει έναν μηχανισμό ελέγχου συμφόρησης (*congestion control*), έτσι ώστε να επιταχύνει την αποστολή μηνυμάτων όταν το δίκτυο το επιτρέπει και την καθυστερεί όταν δίκτυο έχει συμφόρηση. Τέλος, η επικοινωνία τερματίζεται μέσω της λειτουργίας "*connection teardown*", όπου οι διεργασίες ανταλλάζουν μηνύματα με πληροφορίες σχετικά με τον τερματισμό της επικοινωνίας.

2.2.2 Πως δουλεύουν οι Υποδοχές Ροής

Στο υποκεφάλαιο αυτό θα εξετάσουμε τον τρόπο λειτουργίας των υποδοχών ροής. Οτιδήποτε αναφέρεται στο υποκεφάλαιο αυτό, είναι βασισμένο στο τρόπο λειτουργίας των υποδοχών ροής για την γλώσσα προγραμματισμού *C* διότι σε αυτήν έχει βασιστεί η υλοποίηση μας.

Αρχικά, ο εξυπηρετητής δημιουργεί μια υποδοχή την οποία δεσμεύει σε μια συγκεκριμένη θύρα. Αυτό γίνεται με την κλήση των διαδικασιών *socket(AF_INET,SOCK_STREAM,0)* και *bind(socket,serverPointer,serverLength)*. Η διαδικασία socket δημιουργεί μια νέα υποδοχή τύπου *SOCK_STREAM* (Υποδοχή Ροής) στο πεδίο *AF_INET* (πεδίο *Internet*), χωρίς την χρήση επιπρόσθετων πληροφοριών (*0*). Η διαδικασία *bind* δεσμεύει μια υποδοχή (*socket*) με μια διεύθυνση *IP* (*serverPointer*) με μέγεθος *serverLength* και την αντίστοιχη θύρα του εξυπηρετητή.

Στη συνέχεια, περιμένει να "ακούσει" κάποια αίτηση σύνδεσης από πελάτη σε αυτή τη θύρα. Αυτό γίνεται με την κλήση της συνάρτησης *listen(socket,100)*. Η συνάρτηση αυτή καλείται από την πλευρά του εξυπηρετητή ούτως ώστε να περιμένει για αιτήσεις από τους πελάτες στην υποδοχή του (*socket*). Η δεύτερη παράμετρος χρησιμοποιείται για να δηλώσει το μέγιστο αριθμό αιτήσεων που θα περιμένουν στην ουρά του εξυπηρετητή.

Έτσι, ο πελάτης γνωρίζοντας τόσο τη διεύθυνση *IP* του εξυπηρετητή όσο και τον αριθμό της θύρας της υπηρεσίας στην οποία επιθυμεί να αποκτήσει πρόσβαση, αποστέλλει αίτηση σύνδεσης δημιουργώντας αρχικά την υποδοχή και στη συνέχεια προσπαθώντας να ενωθεί με τον εξυπηρετητή καλώντας τις μεθόδους

socket(AF_INET,SOCK_STREAM,0) και *connect(socket,serverPointer,serverLength)*. Η μέθοδος *socket* δημιουργεί μια νέα υποδοχή τύπου *SOCK_STREAM* (Υποδοχή Ροής) στο πεδίο *AF_INET* (πεδίο*Internet*), χωρίς την χρήση επιπρόσθετων πληροφοριών (*0*). Η μέθοδος *connect* συνδέει την υποδοχή του πελάτη (*socket*) με την υποδοχή του εξυπηρετητή (*serverPointer*), η οποία είναι μεγέθους *serverLength*.

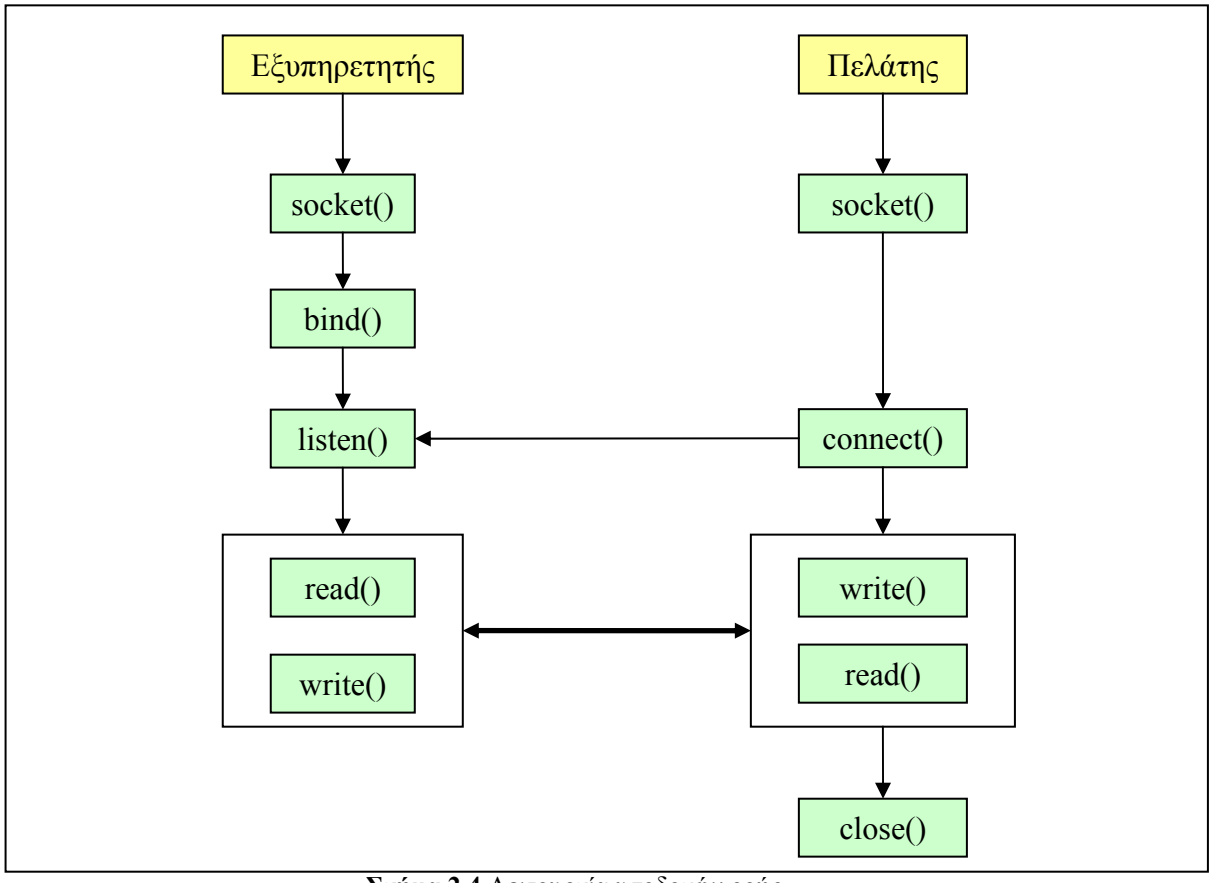
Ακολούθως, ο εξυπηρετητής αποδέχεται την αίτηση του πελάτη και σημειώνει τη διεύθυνση *IP* και τη θύρα του, χρησιμοποιώντας την διαδικασία *accept(socket,clientPointer,clientLength)*. Με τη χρήση της διαδικασίας αυτής ο εξυπηρετητής δέχεται αιτήσεις από τον πελάτη και συνδέει την υποδοχή του (*socket*) με αυτή του πελάτη (*clientPointer*) η οποία είναι μεγέθους *clientLength*.

Για να μπορεί να δέχεται πολλαπλές αιτήσεις σύνδεσης από πολλούς πελάτες, ο εξυπηρετητής πρέπει να δημιουργεί μια καινούργια υποδοχή για κάθε αιτούμενο πελάτη, δηλαδή να είναι πολυνηματικός εξυπηρετητής όπως αναφέραμε πιο πάνω. Με την αποδοχή της αίτησης, ο πελάτης χρησιμοποιεί την υποδοχή που δημιούργησε για να επικοινωνεί με τον εξυπηρετητή. Εφόσον έχει εγκαθιδρυθεί η επικοινωνία μεταξύ εξυπηρετητή και πελατών, οι διεργασίες ανταλλάζουν μηνύματα μέχρις ότου οι διεργασίες να τερματίσουν την επικοινωνία με τον εξυπηρετητή, με τη χρήση των μεθόδων *read(socket,buffer,sizeof(buffer))* και *write(socket,buffer,sizeof(buffer))*, οι οποίες καλούνται και στις δύο πλευρές, του εξυπηρετητή και των πελατών. Η μέθοδος *read* διαβάζει από την υποδοχή είτε του εξυπηρετητή, είτε του πελάτη (*socket*) αναλόγως που θα κληθεί, και αντιγράφει τα *bytes* που αποστέλλονται από την υποδοχή στην τοπική μνήμη (*buffer*) της διεργασίας που παραλαμβάνει το μήνυμα. Η τρίτη παράμετρος περιγράφει το μέγεθος της τοπικής μνήμης της διεργασίας όπου θα αποθηκευτεί το μήνυμα που παραλήφθηκε. Η μέθοδος *write* γράφει στην υποδοχή είτε του εξυπηρετητή, είτε του πελάτη (*socket*) αναλόγως που θα κληθεί, και αντιγράφει τα *bytes* που υπάρχουν στην τοπική μνήμη (*buffer*) στην υποδοχή της διεργασίας που θα παραλαμβάνει το μήνυμα. Η τρίτη παράμετρος περιγράφει το μέγεθος της τοπικής μνήμης της διεργασίας όπου υπάρχει το μήνυμα που θα σταλεί.

Τέλος, η επικοινωνία τερματίζεται με το κλείσιμο των υποδοχών. Η λειτουργία αυτή επιτυγχάνεται καλώντας την πιο συνάρτηση *close(socket)* στην πλευρά των πελατών. Η

συνάρτηση αυτή κλείνει την υποδοχή ροής του πελάτη χρησιμοποιώντας τις λειτουργίες του πρωτοκόλλου *TCP* που χρησιμοποιείται για την υλοποίηση των υποδοχών ροής.

Η επικοινωνία μεταξύ των διεργασιών εξυπηρετητή και πελάτη που επεξηγείτε πιο πάνω απεικονίζεται στο Σχήμα 2.4.



Σχήμα 2.4 Λειτουργία υποδοχών ροής

2.3 Κατανεμημένο Σύστημα Planetlab

Όπως έχουμε προαναφέρει, η πλατφόρμα αξιολόγησης της υλοποίησης μας είναι το *Planetlab*. Στα υποκεφάλαια που ακολουθούν περιγράφεται τι είναι, τι προσφέρει και πως χρησιμοποιείται το *Planetlab*.

2.3.1 Εισαγωγή στο Planetlab

Το *Planetlab* [11] είναι ένα γεωγραφικά κατανεμημένο δίκτυο υπολογιστών παγκόσμιας κλίμακας, στο οποίο λαμβάνουν μέρος αυτή τη στιγμή γύρω στα 300 πανεπιστήμια και οργανισμοί ανά τον κόσμο. Κάθε ένας από αυτούς διαθέτει τουλάχιστο 2 υπολογιστικούς κόμβους, με αποτέλεσμα να υπάρχουν αυτή τη στιγμή 852 κόμβοι οι οποίοι είναι διαθέσιμοι στην κοινότητα του *Planetlab*.

Αυτή τη στιγμή υπάρχουν εκατοντάδες εφαρμογές που τρέχουν στο *Planetlab* οι οποίες δημιουργήθηκαν από διάφορα πανεπιστήμια και άλλους οργανισμούς. Πολλές εταιρίες όπως η *Intel, IBM, HP* έχουν επενδύσει σε αυτό κάνοντας την ανάπτυξη του πιο ενδιαφέρουσα και πιο γρήγορη. Υπάρχουν όμως και απλές εφαρμογές που γίνονται από φοιτητές και μικρές ομάδες ερευνητών, οι οποίες μπορούν να χρησιμοποιηθούν από οποιονδήποτε, για ερευνητικό και μη κερδοσκοπικό σκοπό.

Οι δημιουργοί του *Planetlab*, οι οποίοι το αποκαλούν ως την "Νέα Γενεά του Διαδικτύου", οραματίζονται πως κάποια μελλοντική αναβάθμιση του συστήματος, θα αντικαταστήσει το διαδίκτυο. Τα πειραματικά αποτελέσματα που πηγάζουν από τη μελέτη αυτή οδηγούν σε συμπεράσματα τα οποία θα μας βοηθήσουν να δούμε κατά πόσο το όραμα αυτό μπορεί να είναι εφικτό στο άμεσο μέλλον.

Στο Σχήμα 2.5 που ακολουθεί, παρουσιάζονται μερικά από τα εκατοντάδες *sites* του *Planetlab*. Κάθε *site* αποτελεί και παραπομπή στην ιστοσελίδα του συγκεκριμένου *site*.

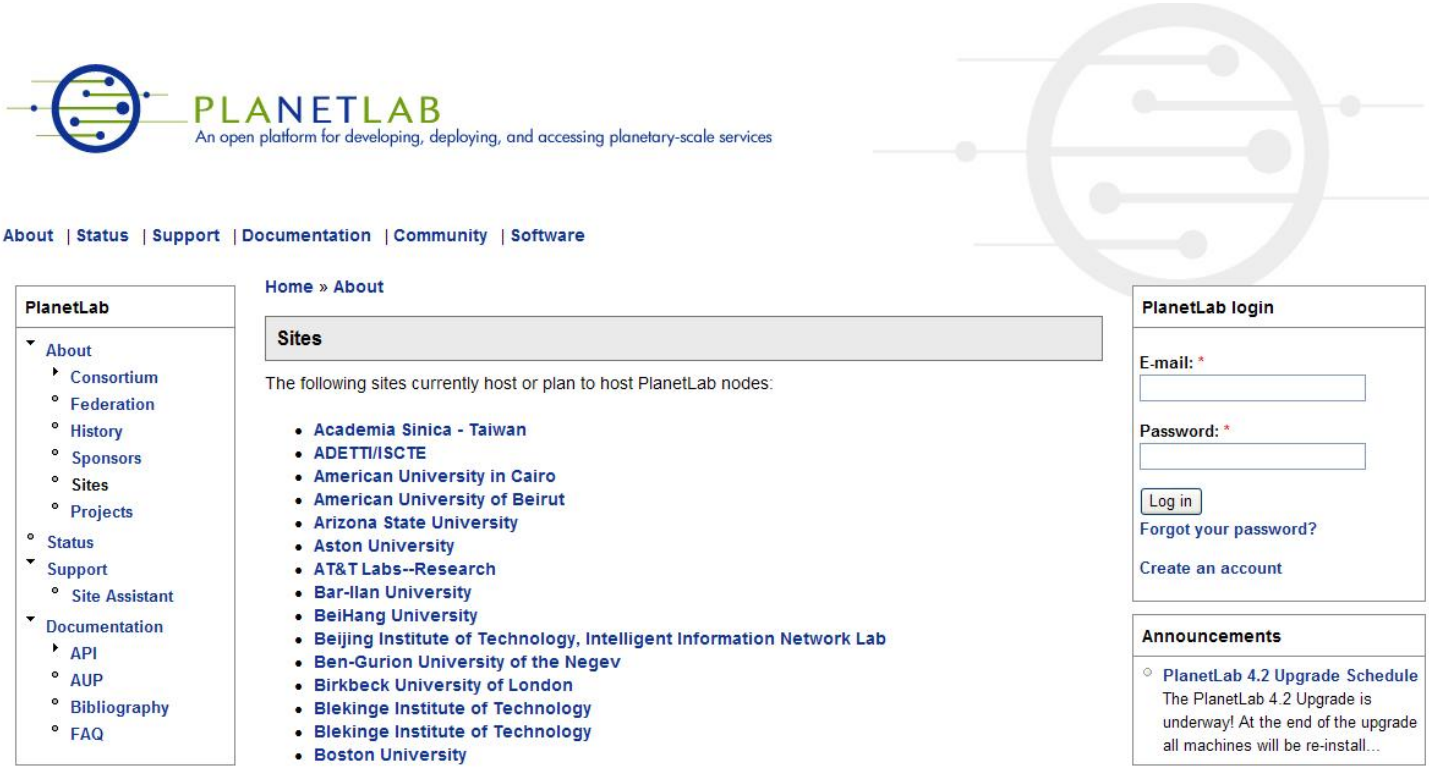
2.3.2 Τι προσφέρει το Planetlab

Το *Planetlab* επιτρέπει στους χρήστες του, είτε είναι υψηλής κλίμακας, είτε χαμηλής να βιώσουν τα πιο κάτω :

- Ένα μεγάλο αριθμό από παράλληλες μηχανές σε παγκόσμιο γεωγραφικό επίπεδο. Οι χρήστες μπορούν να επιλέξουν μέσα από όλες τις μηχανές που διαθέτει το *Planetlab*, όποιες επιθυμούν σε οποιεσδήποτε τοποθεσίες,

δημιουργώντας έτσι το δικό τους παγκόσμιας κλίμακας δίκτυο και να εκτελέσουν τις εφαρμογές τους σε αυτό.

- Ένα ξεχωριστό, ρεαλιστικό δίκτυο το οποίο έχει όλες τις συμπεριφορές ενός κανονικού δικτύου, καθώς επίσης και των προβλημάτων, που πηγάζουν από αυτό. Οι χρήστες μέσα από το *Planetlab* θα μπορούν να βιώσουν όλες τις συμπεριφορές που μπορούν να βιώσουν και από το διαδίκτυο, παρόλο που είναι ξεχωριστά. Είναι ρεαλιστικό λόγω των του ότι αντιμετωπίζει προβλήματα όπως υπερφόρτωση δικτύου και συμπεριφέρεται όπως ένα φυσιολογικό δίκτυο.



Σχήμα 2.5 Μερικά από τα εκατοντάδες *sites* του *Planetlab*

2.3.3 Πως χρησιμοποιείται το Planetlab

Σε κάθε οργανισμό (*site*), μέλος του *Planetlab*, ο οποίος διαθέτει το λιγότερο 2 μηχανές σε αυτό, έχει την εξουσιοδότηση να εγγράφει μέλη-χρήστες σε αυτό. Υπεύθυνοι για κάθε οργανισμό είναι οι ερευνητικοί υπεύθυνοι, αλλιώς *PI* (*Principal Investigators*), οι οποίοι μπορούν να εγκρίνουν την εγγραφή κάποιου που επιθυμεί να γίνει χρήστης και να του δώσουν πρόσβαση στο *Planetlab*.

Η χρήση του *Planetlab* σαν ιδέα είναι απλή. Εφόσον υπάρχει εξουσιοδότηση από κάποιον, έχει στην διάθεση του τις μηχανές και μπορεί κάτω από κάποιους περιορισμούς να τρέξει τις εφαρμογές του. Υπάρχει μια διαδικασία για να ενωθεί κάποιος με τον εξυπηρετητή για λόγους ελέγχου και ασφάλειας και ακολούθως μέσα στο δικό σου χώρο μπορείς να δημιουργήσεις τις εφαρμογές σου και να τις τρέξεις. Τα πάντα τρέχουν σε περιβάλλον *Unix*.

Το *Planetlab* χρησιμοποιεί την έννοια του "μερίσματος" (*slice*) για να εκφράσει τον τρόπο που εκτελούνται οι διάφορες εφαρμογές σε αυτό. Κάθε οργανισμός μπορεί να έχει μέχρι 10 "μερίσματα". Το "μέρισμα" είναι ένα σύνολο από εικονικές μηχανές του *Planetlab* (*Planetlab virtual machines*), όπου είναι διατιθέμενες για μια υπηρεσία (*service*). Ένα "μέρισμα" δεσμεύει κάποιο ποσό επεξεργασίας (*processing*), μνήμης αποθήκευσης (*storage memory*), δικτυακών πόρων (*network resources*) και εύρους ζώνης (*bandwidth*), μέσα από ένα σύνολο μηχανών του *Planetlab*, οι οποίες είναι κατανεμημένες στο διαδίκτυο. Τα πιο πάνω μέτρα είναι περιορισμένα σύμφωνα με την υπάρχουσα κατάσταση του δικτύου, για κάθε χρήστη. Ο κάθε χρήστης μπορεί να δουλεύει προσωπικά και ανενόχλητα σε κάθε μηχανή μέσω του "μερίσματος" του χωρίς να επηρεάζει ή να επηρεάζεται από τους άλλους χρήστες που χρησιμοποιούν την ίδια μηχανή. Επηρεάζεται όμως η κατάσταση του δικτύου και της μηχανής σε θέματα απόδοσης. Ένα από τα αρνητικά στοιχεία του "μερίσματος" είναι ότι παραδίδεται σε κάθε χρήστη εντελώς άδεια. Περιέχει μόνο τα βασικά του *Unix* και κάποια ξεχωριστά στοιχεία του *Fedora*, αλλά δεν παρέχει βασικά προγράμματα που χρειάζονται, όπως μεταγλωττιστής *C* και *Java*, επεξεργαστές κειμένων και άλλα πολλά που απαιτούνται από ένα προγραμματιστή. Ο λόγος που συμβαίνει αυτό είναι διότι ο κάθε χρήστης χρειάζεται το "μέρισμα" του για τους δικούς του λόγους και είναι κάτω από δική του ευθύνη να εγκαταστήσει σε αυτό ότι του είναι αναγκαίο. Αφού κάποιος έχει πάρει πρόσβαση ως απλός χρήστης τότε μπορεί να χρησιμοποιήσει το *Planetlab*.

Για λόγους ασφάλειας και ελέγχου πρόσβασης για να ενωθεί κάποιος σε κάποια μηχανή του *Planetlab* θα πρέπει να χρησιμοποιήσει την ακόλουθη μέθοδο [12]. Ο χρήστης θα πρέπει να δημιουργήσει ένα κρυπτογραφημένο κλειδί (*encryption key*) το οποίο θα ανεβάσει στον δημόσιο εξυπηρετητή του *Planetlab*. Το ανέβασμα του κρυπτογραφημένου κλειδιού γίνεται από την επίσημη ιστοσελίδα του Planetlab [11],

μέσω της διαχειριστικής πλατφόρμας (*control panel*). Το κλειδί αυτό θα είναι κρυπτογραφημένο σύμφωνα με τον κωδικό που θα δώσει ο χρήστης. Η εντολή που ακολουθεί θα δημιουργήσει δύο αρχεία. Θα πρέπει να εκτελεστεί σε *Unix* και θα δημιουργήσει τα δύο κρυπτογραφημένα αρχεία που χρειάζεται. Το ένα θα μείνει στον τοπικό χώρο του χρήστη και το άλλο θα πρέπει να ανεβεί όπως είπαμε προηγουμένως στο *Planetlab*. Αυτό γίνεται με την πιο κάτω εντολή:

```
> ssh-keygen -t rsa -f ~/.ssh/id_rsa
```

Κάθε φορά που ο χρήστης επιθυμεί να ενωθεί σε μια μηχανή θα πρέπει να εκτελέσει την πιο κάτω εντολή. Στην πιο κάτω εντολή το "*slice_name*" είναι το όνομα της "μερίσματος" του χρήστη το οποίο του έχει ανατεθεί από τον *PI*, το "*id_rsa*" είναι το ένα από τα δύο κρυπτογραφημένα αρχεία το οποίο βρίσκεται στον τοπικό χώρο του χρήστη, το "*node_address*" είναι η διεύθυνση της μηχανής που θέλουμε να ενωθούμε. Όλες οι διευθύνσεις των μηχανών βρίσκονται στην ιστοσελίδα του *Planetlab*. Η σύνδεση σε μια μηχανή του *Planetlab* γίνεται με την πιο κάτω εντολή:

```
> ssh -l slice_name -i ~/.ssh/id_rsa node_address
```

Αφού εισαγάγει τον κωδικό του, ο μηχανισμός αυτός θα συγκρίνει τα δύο κλειδιά και θα ελέγχει αν είναι σωστά σε σχέση με τον κωδικό που δόθηκε. Υπάρχουν πολλές άλλες εντολές που μπορεί να χρησιμοποιήσει ο χρήστης για να κάνει λειτουργίες στο *Planetlab*, όπως αποστολή αρχείων, κτλ. Αφού κάποιος ενωθεί σε κάποια μηχανή του *Planetlab* μεταφέρεται αυτόματα στο "μέρισμα" του και μπορεί να ξεκινήσει να χρησιμοποιεί την μηχανή.

Τέλος, μπορεί κάποιος να μεταφέρει είτε τα αρχεία του στο "μέρισμα" του, είτε να μεταφέρει απλά το εκτελέσιμο αρχείο του στις μηχανές του *Planetlab* και να ξεκινήσει να τρέχει την εφαρμογή του. Αυτό μπορεί να γίνει με την πιο κάτω εντολή:

```
> scp -i ~/.ssh/id_rsa -r app_name slice_name@node_name:
```

Η πιο πάνω εντολή θα αντιγράψει το αρχείο ή φάκελο "*app_name*" στον αρχικό φάκελο του χρήστη στον οποίο ανήκει το "μέρισμα" "*slice_name*", στην μηχανή "*node_name*".

Τα βήματα που πρέπει να κάνει κάποιος για να χρησιμοποιήσει το *Planetlab* βρίσκονται πιο αναλυτικά στα εγχειρίδια χρήσης του *Planetlab* [12].

Ο κάθε χρήστης έχει τη δική του διαχειριστική πλατφόρμα η οποία βρίσκεται στην επίσημη ιστοσελίδα του *Planetlab*, όπου υπάρχει γραφικό και πολύ κατανοητό περιβάλλον. Η πλατφόρμα του *Planetlab* φαίνεται στο Σχήμα 2.6. Μέσω αυτής ο χρήστης μπορεί να κάνει τα εξής:

- Να δει όλες τις μηχανές που εμπλέκονται στο *Planetlab*. Μπορεί να βρει μέσω μηχανής αναζήτησης όλες τις μηχανές που του έχουν ανατεθεί στο Planetlab από όλους τους οργανισμούς. Επίσης, κάθε μηχανή είναι παραπομπή στην αντίστοιχη σελίδα της μηχανής

- Να βρει άλλους χρήστες που εμπλέκονται στο *Planetlab*. Η μηχανή αναζήτησης επιστρέφει το ηλεκτρονικό ταχυδρομείο του χρήστη, σχετικά με το όνομα που δόθηκε στο πεδίο

- Να ανεβάσει, όπως είπαμε πριν το κρυπτογραφημένο κλειδί. Το κλειδί βρίσκεται στο χώρο του χρήστη και από εδώ πρέπει να το ανεβάσει στο *Planetlab*, έτσι ώστε να έχει πρόσβαση στις μηχανές. Όπως φαίνεται στην ιστοσελίδα το κλειδί αυτό είναι κρυπτογραφημένο

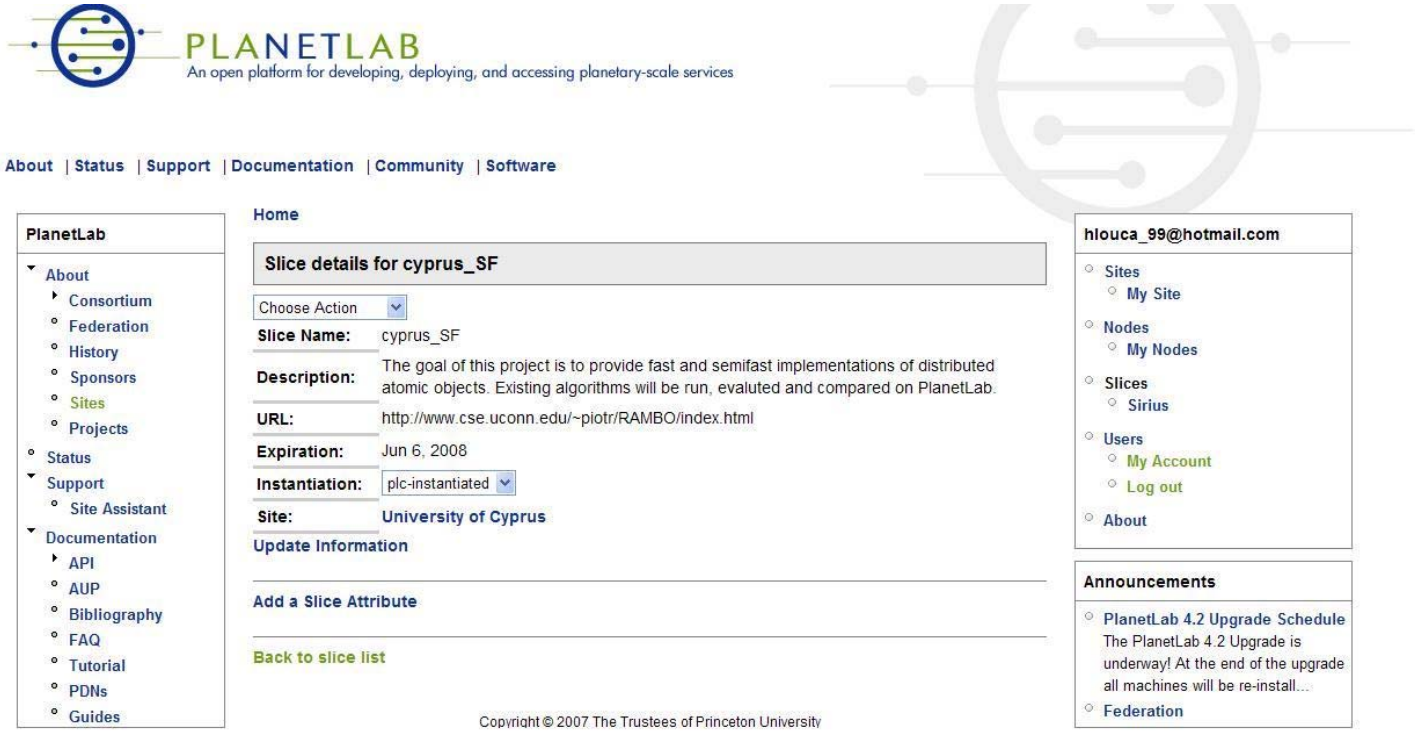
- Να δημοσιεύσει τα προσωπικά του στοιχεία

- Να διαχειριστεί τα "μερίσματα" του, αναθέτοντας μηχανές σε αυτά. Ο χρήστης μπορεί να αναθέσει μηχανές στα "μερίσματα" του δημιουργώντας το προσωπικό του δίκτυο για την εφαρμογή του. Από την στιγμή που ανατίθεται μια μηχανή από το *Planetlab* χρειάζεται κάποιος χρόνος για να εγκριθεί από τον οργανισμό στον οποίο ανήκει η μηχανή, προτού ο χρήστης να μπορεί να την διαχειριστεί. Ο χρήστης δεν μπορεί να επιλέξει μηχανές οι οποίες ανήκουν στον οργανισμό του (π.χ. Πανεπιστήμιο Κύπρου για την περίπτωση μας). Ο χρήστης μπορεί να αναθέσει μέχρι 32 μηχανές σε κάθε "μέρισμα"

- Να δημοσιεύσει πληροφορίες για το τι ακριβώς κάνει η εφαρμογή του στο *Planetlab*, συμπεριλαμβανομένου και της προσωπικής του ιστοσελίδας σχετικά με την εφαρμογή. Οι πληροφορίες πρέπει να είναι αληθείς, αλλιώς υπάρχει κίνδυνος να διαγραφεί το "μέρισμα" από τους υπεύθυνους

- Να ανανεώνει το "μέρισμα" του το πολύ για τη διάρκεια ενός μήνα, όσες φορές επιθυμεί. Το "μέρισμα" έχει ημερομηνία λήξης για να αποφευχθεί το συμμάζεμα αχρησιμοποίητων και μη ενεργών "μερισμάτων" από χρήστες που δεν τις χρησιμοποιούν πλέον. Για αυτό αν ο χρήστης επιθυμεί να συνεχίσει να χρησιμοποιεί το "μέρισμα" του θα πρέπει απλά να το ανανεώνει

Στο Σχήμα 2.6 που ακολουθεί παρουσιάζεται η πλατφόρμα του *Planetlab*, δείχνοντας ένα παράδειγμα "μερίσματος" (*slice*) του *Planetlab*.



Σχήμα 2.6 Πλατφόρμα του *Planetlab*

Κεφάλαιο 3

Αλγόριθμος Ατομικών Αντικειμένων Γραφής-Ανάγνωσης

Στο κεφάλαιο αυτό θα παρουσιάσουμε τη μεθοδολογία που ακολουθήθηκε για την υλοποίηση της εργασίας αυτής, μια σύντομη περιγραφή του αλγορίθμου ατομικών αντικειμένων που υλοποιήθηκε, τον τρόπο υλοποίησης του αλγορίθμου στη γλώσσα προγραμματισμού *C* χρησιμοποιώντας παραδείγματα από τον πηγαίο κώδικα της εφαρμογής και επεξήγηση του τρόπου με τον οποίο τρέχει η εφαρμογή.

Η βιωσιμότητα των δεδομένων στον αλγόριθμο διασφαλίζεται μέσω αναπαραγωγής των ατομικών αντικειμένων. Έτσι, η πρόκληση είναι να διατηρείται η συνέπεια των δεδομένων σε κάθε αντίγραφο. Αυτό μας οδηγεί στην έννοια της ατομικότητας που εξηγήσαμε στο κεφάλαιο 2 και η διατήρηση της οποίας είναι η βασικός στόχος του αλγορίθμου.

3.1 Μεθοδολογία

Όπως αναφέρθηκε και στην εισαγωγή, σκοπός μας είναι η πειραματική αξιολόγηση του αλγορίθμου Ατομικών Αντικειμένων Γραφής-Ανάγνωσης στο κατανεμημένο δίκτυο υπολογιστών *Planetlab*. Για να επιτύχουμε αυτό το σκοπό χρησιμοποιήσαμε την πιο κάτω μεθοδολογία:

1. Κατανόηση Αλγόριθμου Ατομικών Αντικειμένων Γραφής-Ανάγνωσης και αναγκαίου υποβάθρου

Στη πρώτη φάση της παρούσας εργασίας, μελετήσαμε όλους τους αναγκαίους ορισμούς που θα μας βοηθούσαν να κατανοήσουμε τη λειτουργικότητα του αλγορίθμου. Οι ορισμοί αυτοί περιγράφονται εκτεταμένα στο κεφάλαιο 2.

2. Εκμάθηση και χρήση Προγραμματισμού Υποδοχών Ροής

Στο επόμενο στάδιο, θεωρήθηκε απαραίτητη η εκμάθηση του προγραμματισμού Υποδοχών Ροής, καθώς είναι το βασικό εργαλείο της γλώσσας προγραμματισμού *C* για υλοποίηση κατανεμημένων εφαρμογών. Ο προγραμματισμός υποδοχών μελετήθηκε στα βιβλία [4,5,6,7], τα οποία βοήθησαν στην κατανόηση του τρόπου λειτουργίας των υποδοχών και του τρόπου σκέψης που χρειάστηκε να υιοθετήσουμε για την πραγμάτωση των στόχων μας. Έτσι, αρχικά υλοποιήθηκε μια απλή εφαρμογή που θα υποστήριζε το μοντέλο εξυπηρετητή-πελάτη (*client-server model*) όπου εξυπηρετητής και πελάτης αντάλλαζαν μερικά μηνύματα και ακολούθως τερμάτιζαν. Αυτό είχε ως σκοπό την εξοικείωση με αυτό το είδος προγραμματισμού, την εξοικείωση με το τρόπο λειτουργίας του μοντέλου εξυπηρετητή-πελάτη και την εκμάθηση του τρόπου λειτουργίας και προγραμματισμού των υποδοχών ροής.

3. Μελέτη κατανεμημένου συστήματος *Planetlab*

Η εκμάθηση του *Planetlab* έπρεπε να γίνει για γνωριμία του τρόπου με τον οποίο λειτουργεί το κατανεμημένο σύστημα, τι δυνατότητες έχουμε και τι μπορεί να μας προσφέρει. Σκοπός ήταν η εξοικείωση μου με το περιβάλλον *Fedora* και των εντολών *Unix* στο *Planetlab*. Αυτό που επιτεύχθηκε μετά από τη μελέτη του *Planetlab* ήταν τα πιο κάτω:

- Εγράφη μέλους στο *Planetlab*
- Βασικές αρχές και ιστορία του *Planetlab*
- Πως διαχειρίζεται το *Planetlab* τους πόρους και τους χρήστες
- Τι προσφέρει το *Planetlab*

- Πως ενώνεται κάποιος σε αυτό και πώς χρησιμοποιείται

4. Υλοποίηση Αλγορίθμου και αποσφαλμάτωση σε τοπικό δίκτυο

Το επόμενο στάδιο ήταν η υλοποίηση του αλγόριθμου Ατομικών Αντικειμένων Γραφής-Ανάγνωσης και η αποσφαλμάτωση του στο τοπικό δίκτυο του Πανεπιστημίου Κύπρου. Μετά την υλοποίηση του αλγορίθμου, η εφαρμογή έτρεξε στο τοπικό δίκτυο του Πανεπιστημίου μέσα από διάφορα σενάρια με σκοπό την εξασφάλιση της ορθής λειτουργίας του αλγόριθμου.

5. Πειραματική αξιολόγηση του αλγόριθμου στο *Planetlab*

Το τελευταίο στάδιο ήταν, κατόπιν εκτέλεσης της εφαρμογής μέσα από διάφορα σενάρια, να πάρουμε κάποια αποτελέσματα, να τα συγκρίνουμε και να προσπαθήσουμε να οδηγηθούμε σε κάποια συμπεράσματα. Για σκοπούς αξιολόγησης, εισάγαμε στον αλγόριθμο και συγκεκριμένα στις διεργασίες γραφής και ανάγνωσης ένα μηχανισμό αναμονής μετά από το τέλος κάθε λειτουργίας γραφής και ανάγνωσης αντιστοίχως. Ο σκοπός της εισαγωγής του μηχανισμού αυτού ήταν να δημιουργήσουμε δύο σενάρια όπου στο πρώτο οι διεργασίες θα περίμεναν για τυχαίο χρονικό διάστημα (*random time interval*) και στο δεύτερο θα περίμεναν σταθερό χρονικό διάστημα (*fixed time interval*) πριν να εκτελέσουν την επόμενη λειτουργία γραφής ή ανάγνωσης.

3.2 Σύντομη περιγραφή αλγορίθμου

Αρχικά θα προσεγγίσουμε τον αλγόριθμο σε γενικές γραμμές και θα εμβαθύνουμε σε μεταγενέστερο σημείο. Εν συντομία, η λειτουργία γραφής περιλαμβάνει την αύξηση της χρονικής σφραγίδας και την μετάδοση του σε όλους τους *Servers*. Η λειτουργία γραφής τερματίζει όταν λάβει αποκρίσεις $S-t$ από τους *Servers*.

Η λειτουργία ανάγνωσης είναι πιο πολύπλοκη καθώς ο *Reader* στέλνει αιτήματα ανάγνωσης (*read request*) σε όλους τους *Servers*, περιμένει μέχρι να παραλάβει $S-t$ αποκρίσεις από τους *Servers* και ελέγχει την εγκυρότητα μιας συνθήκης (*predicate*) έτσι ώστε να αποφασίσει αν θα εκτελέσει έναν ή δύο γύρους επικοινωνίας. Η συνθήκη

ελέγχει τα εξής: (α) τη μέγιστη χρονική σφραγίδα που βρέθηκε ανάμεσα στα μηνύματα απόκρισης, (β) τον αριθμό των *Servers* που αποκρίθηκαν με τη συγκεκριμένη χρονική σφραγίδα, (γ) τον αριθμό των νοητών κόμβων των οποίων τα μέλη έχουν δει τη συγκεκριμένη χρονική σφραγίδα διαμέσου αυτών των *Servers*. Εάν η συνθήκη ισχύει, τότε η λειτουργία ανάγνωσης επιστρέφει τη μέγιστη χρονική σφραγίδα που βρήκε μεταξύ των αποκρίσεων, αλλιώς επιστρέφει τη μέγιστη χρονική σφραγίδα πλην ένα. Στην περίπτωση που θα πραγματοποιηθεί δεύτερος γύρος επικοινωνίας, η διεργασία πρέπει να ενημερώσει τουλάχιστον $3t+1$ *Servers*.

Κάθε διεργασία *Server* διατηρεί ένα αντίγραφο της δυάδας της τιμής και της χρονικής σφραγίδας, του ατομικού αντικειμένου δηλαδή, τις οποίες ενημερώνει όταν παραλάβει ένα μήνυμα που περιέχει χρονική σφραγίδα μεγαλύτερη από αυτή που διατηρεί στην τοπική του μνήμη μέχρι εκείνη τη στιγμή. Επιπρόσθετα, η διεργασία *Server* διατηρεί και ένα σύνολο με τα νοητά αναγνωριστικά των διεργασιών που έχουν στείλει αίτηση είτε ανάγνωσης είτε γραφής με τη πιο πρόσφατη χρονική σφραγίδα, και αποκρίνεται αποστέλλοντας τη τιμή, τη χρονική σφραγίδα και το σύνολο αναγνωριστικών που διατηρεί. Αυτά τα στοιχεία είναι πολύ σημαντικά για τη διεργασία *Reader* έτσι ώστε να μπορέσει να καθορίσει την εγκυρότητα της συνθήκης.

Οι περιορισμοί που απορρέουν από την αυστηρή απόδειξη που έγινε στο [2], είναι οι εξισώσεις $3t+1$, $V<\frac{S}{t}-2$ και $t<\frac{S}{2}$. Οι περιορισμοί αυτοί αναφέρονται: α) στον ελάχιστο αριθμό Servers που πρέπει να ενημερώσει κάθε *Reader* κατά την εκτέλεση δεύτερου γύρου επικοινωνίας, β) στον μέγιστο αριθμό νοητών αναγνωριστικών (*virtual identifiers*) που μπορούν να υπάρξουν στο σύστημα και γ) στον μέγιστο αριθμό *Servers* που μπορούν να καταρρεύσουν, αντίστοιχα. Η μη ικανοποίηση τους οδηγεί σε παραβίαση της ατομικότητας και τα δεδομένα δεν διατηρούνται με συνέπεια στους εξυπηρετητές (*Servers*) του συστήματος. Ακόμα, έχει αποδειχθεί στο [2] ότι ο αλγόριθμος διατηρεί την ατομικότητα σε κάθε εκτέλεση του ακόμα και αν υποστεί t απώλειες διεργασιών τύπου *Server*, αν καταρρεύσει οποιοδήποτε υποσύνολο των *Readers* ή αν καταρρεύσει ο *Writer*.

Τέλος, στο άρθρο [2] έχει γίνει προσομοίωση του αλγορίθμου στον προσομοιωτή δικτύων *ns-2*. Τα αποτελέσματα της προσομοίωσης αυτής δείχνουν ότι οι λειτουργίες γραφής είναι όλες γρήγορες, ενώ μόνο ένα μικρό ποσοστό των λειτουργιών ανάγνωσης, περίπου 7% στην χειρότερη περίπτωση όπου έχουμε πολλές ταυτόχρονες λειτουργίες ανάγνωσης και γραφής, εκτελεί δεύτερο γύρο επικοινωνίας.

3.3 Υλοποίηση Αλγόριθμου

Η υλοποίηση έγινε στη γλώσσα προγραμματισμού *C*, χρησιμοποιεί το μοντέλο πελάτη-εξυπηρετητή και βασίζεται στη δημιουργία τριών διεργασιών όπως ακριβώς αναφέρονται στην περιγραφή του αλγόριθμου, δηλαδή στην υλοποίηση των διεργασιών *Server*, *Writer* και *Reader*. Θα επεξηγηθεί ο τρόπος που δουλεύει η εφαρμογή με τη παρουσίαση κομματιών κώδικα από τις διεργασίες που υλοποιήθηκαν. Η υλοποίηση του αλγόριθμου Ατομικών Αντικειμένων Γραφής-Ανάγνωσης υποστηρίζει την αυθαίρετη ύπαρξη πολλών *Readers*. Όπως αναφέραμε, υποθέτουμε ότι ο αριθμός των νοητών αναγνωριστικών V είναι τέτοιος ώστε $V < \frac{S}{t} - 2$.

Διεργασία *Writer*: Αρχικά, η διεργασία *Writer* εγκαθιδρύει την επικοινωνία με τους *Servers*. Αυτό γίνεται με τη δημιουργία μιας υποδοχής ροής για κάθε *Server*, διαβάζοντας ένα αρχείο το οποίο δίνεται στην διεργασία από τις παραμέτρους της γραμμής εντολών (*command line arguments*). Από το αρχείο αυτό, διαβάζονται σειριακά οι παράμετροι που χρειάζονται για κάθε υποδοχή, όπως είναι το όνομα του εκάστοτε *Server* και ο αριθμός θύρας. Αυτό γίνεται στο κομμάτι κώδικα που ακολουθεί:

```
i=0;
while(!feof(fp)){
	// Διαβάζει το όνομα του Server και τον αριθμό θύρας
	fscanf(fp,"%s\t%d\n",&serverSTR,&port);

	// Δημιουργεί νέα υποδοχή
	sock[i]=socket(AF_INET,SOCK_STREAM,0);

	// Μετατρέπει το όνομα του Server στην IP διεύθυνση του
	rem=gethostbyname(serverSTR);

	// Αναθέτει διάφορες παραμέτρους στην υποδοχή που δημιούργησε
	server[i].sin_family=AF_INET;
```

```
        bcopy((char*)rem->h_addr,&server[i].sin_addr,rem->h_length);
        server[i].sin_port=htons(port);
        serverptr=(struct sockaddr *) &server[i];
        serverlen=sizeof(server[i]);

        // Δεσμεύει την υποδοχή με την IP διεύθυνση του Server
        connect(sock[i],serverptr,serverlen);

        // Αυξάνει τον αριθμό των Servers με τους οποίους ενώθηκε
        i++;
}
```

Η διεργασία *Writer* διατηρεί τη χρονική σφραγίδα και την τιμή και εκτελεί ένα γύρο επικοινωνίας ως ακολούθως. Στέλνει σε όλους τους *Servers* αίτηση γραφής (*write request*) όπου περιλαμβάνει τη τρέχουσα χρονική σφραγίδα και την αντίστοιχη τιμή. Ένα μήνυμα αίτησης γραφής έχει την ακόλουθη μορφή: "*WRITE, χρονική_σφραγίδα, τιμή, μετρητής_λειτουργίας, 0, 0*", όπου το *WRITE* περιγράφει το είδος του μηνύματος, η *χρονική_σφραγίδα* και η *τιμή* περιέχουν τη δυάδα της τιμής με την αντίστοιχη χρονική σφραγίδα, ο *μετρητής_λειτουργίας* περιγράφει τον αριθμό των λειτουργιών γραφής που έγιναν μέχρι τώρα, το πρώτο μηδενικό περιγράφει το νοητό αναγνωριστικό του *Writer* και τέλος το δεύτερο μηδενικό περιγράφει το μοναδικό αναγνωριστικό του *Writer*. Η δημιουργία των μηνυμάτων φαίνεται πιο κάτω:

```
// Δημιουργία μηνύματος με τις απαραίτητες παραμέτρους
sprintf(buf,"WRITE,%d,%d,%d,0,0#",ts,value,wCounter);
for(i=0;i<numberOfServers;i++){
        // Αποστολή του μηνύματος στον i-οστό Server
        write(sock[i],buf,sizeof(buf));
}
```

Τα είδη των μηνυμάτων που υπάρχουν στο σύστημα είναι τα εξής:

- *WRITE*, για λειτουργίες γραφής
- *WRITEACK*, για αποκρίσεις λειτουργιών γραφής
- *READ*, για λειτουργίες ανάγνωσης
- *READACK*, για αποκρίσεις λειτουργιών ανάγνωσης
- *INFORM*, που υποδηλεί δεύτερο γύρο επικοινωνίας
- *INFORMACK*, που σημαίνει απόκριση δεύτερου γύρου επικοινωνίας

Εφόσον *t Servers* μπορούν να καταρρεύσουν, περιμένει για $S-t$ αποκρίσεις από τους *Servers*. Όταν τις παραλάβει, αυξάνει τη χρονική σφραγίδα, υπολογίζει με τυχαίο τρόπο

τη νέα τιμή για την αντίστοιχη χρονική σφραγίδα και τερματίζει τη λειτουργία της γραφής. Η χρονική σφραγίδα περιγράφει μια φυσική σειρά στη διαδικασία των λειτουργιών γραφής καθώς υπάρχει μόνο ένας *Writer* στην υλοποίηση μας. Στο κομμάτι κώδικα που ακολουθεί παρουσιάζεται η αναμονή για τις αποκρίσεις των *Servers*, η αύξηση της χρονικής σφραγίδας και ο τυχαίος υπολογισμός της νέας τιμής.

```
// Η λειτουργία γραφής περιμένει μέχρι να παραλάβει S-t αποκρίσεις από
// τους Servers
while(i<(numberOfServers-t)){
	bzero(buf2,sizeof(buf2));

	// Διαβάζει από την υποδοχή την απόκριση του i-οστού Server
	read(sock[j],buf2,sizeof(buf2))
	temp=strtok(buf2,",");

	// Εάν η απόκριση που παίρνει είναι όντως απόκριση για
	// λειτουργία γραφής, αυξάνει το μετρητή των αποκρίσεων
	if(strcmp(temp,"WRITEACK")==0){
		i++;
	}

	// Βοηθητικός μετρητής που αυξάνεται συνεχώς μέχρι να γίνει ίσος
	//με S, έτσι ώστε να περιμένει απόκριση από όλους τους Servers
	if(j==numberOfServers-t){
		j=0;
	}
	else{
		j++;
	}
}

// Αύξηση της τρέχουσας χρονικής σφραγίδας
ts++;

// Τυχαίος υπολογισμός νέας τιμής
value=rand()/100000000;
```

Διεργασία *Server*: Κάθε διεργασία *Server* διατηρεί την χρονική σφραγίδα και την τιμή. Επιπρόσθετα, διατηρεί κάποιες τοπικές μεταβλητές: (1) *ts*, όπου αποθηκεύεται η μέγιστη χρονική σφραγίδα που παραλήφθηκε από τον *Writer*, (2) *seenSet*, το οποίο είναι το σύνολο των νοητών αναγνωριστικών που συνάντησε ο *Server* κατά τη διάρκεια της τελευταίας χρονικής σφραγίδας, (3) *counter*, έναν πίνακα από ακέραιους που θα αποθηκεύουν τον αριθμό λειτουργίας γραφής ή ανάγνωσης για κάθε διεργασία, ο οποίος χρησιμεύει για να ξεχωρίζει ο *Server* τα παλιά από τα καινούργια μηνύματα λόγω του ότι το δίκτυο είναι ασύγχρονο και τα μηνύματα μπορεί να φτάνουν με λανθασμένη σειρά, (4) *postit*, η οποία χρησιμοποιείται για να ενημερώνουν οι *Readers*, αν χρειάζεται, τους υπόλοιπους για την χρονική σφραγίδα που πρόκειται να

επιστρέψουν. Αρχικά, η διεργασία *Server* αρχικοποιεί τις τοπικές μεταβλητές του ως εξής: $ts = 0$, $seenSet = \varnothing$, $counter[0...V+1] = 0$, $postit = 0$. Στη συνέχεια, δημιουργεί την υποδοχή ροής του και αρχικοποιεί τις απαραίτητες παραμέτρους όπως φαίνεται στον κώδικα που ακολουθεί:

```
// Εισαγωγή του αριθμού θύρας από τις παραμέτρους της γραμμής εντολών
port=atoi(argv[1]);

// Δημιουργία της υποδοχής και αρχικοποίηση απαραίτητων παραμέτρων
sock=socket(AF_INET,SOCK_STREAM,0);
server.sin_family=AF_INET;
server.sin_addr.s_addr=htonl(INADDR_ANY);
server.sin_port=htons(port);
serverptr=(struct sockaddr *) &server;
serverlen=sizeof(server);

// Δέσμευση της υποδοχής με τη θύρα και αναμονή για αιτήσεις
bind(sock,serverptr,serverlen);
listen(sock,100)<0);
```

Ακολούθως, ο *Server* περιμένει μέχρι να δεχθεί κάποια αίτηση από κάποιον πελάτη. Όταν γίνει αυτό, δημιουργεί νέα υποδοχή και νέο νήμα για την συγκεκριμένη αίτηση. Αυτό γίνεται με τη χρήση της εντολής *fork()* και φαίνεται πιο κάτω:

```
// Δημιουργία νέας υποδοχής για τη συγκεκριμένη αίτηση
newsock=accept(sock,clientptr,&clientlen);

switch(fork()){
      case -1: // Η δημιουργία του νήματος απέτυχε. Τερματισμός
            exit(-1);
      case 0: // Εκτέλεση του γύρου επικοινωνίας
            ...
}
```

Εφόσον έχει δημιουργηθεί το νέο νήμα (*thread*), ο *Server* πρέπει να διαβάσει από προσωρινά αρχεία όλα τα στοιχεία που περιγράφονται πιο πάνω. Αυτό οφείλεται στο γεγονός ότι κάθε νέο νήμα που δημιουργείται είναι μια ξεχωριστή διεργασία, με δική της τοπική μνήμη και δεν γνωρίζει τα δεδομένα που έχουν τα νήματα από τις προϋπάρχουσες αιτήσεις. Έτσι, είναι απαραίτητο στην αρχή κάθε γύρου επικοινωνίας ο *Server* να ενημερώνει τις τοπικές μεταβλητές του από τα προσωρινά αρχεία που διατηρεί. Η εισαγωγή των δεδομένων από τα προσωρινά αρχεία φαίνεται πιο κάτω:

```
// Δημιουργία ονόματος προσωρινού αρχείου βάσει του αριθμού θύρας
sprintf(tempFile,"server%dseen.txt",port);

// Έλεγχος αν το αρχείο υπάρχει ήδη. Αν ναι, τότε εισάγονται τα
// δεδομένα από το αρχείο, αλλιώς το δημιουργούμε εφόσον είναι η πρώτη
// εκτέλεση γύρου επικοινωνίας
if((fp=fopen(tempFile,"r"))==NULL){
	seen=NULL;
}
else{
	while(!feof(fp)){
		fscanf(fp,"%d\n",&num);push(&seen,num);
	}
	fclose(fp);
}

// Παρόμοια λειτουργικότητα με πιο πάνω και
// δεν χρειάζεται να επεξηγηθεί
sprintf(tempFile,"server%dcounter.txt",port);
if((fp=fopen(tempFile,"r"))==NULL){
	fp=fopen(tempFile,"w");
	for(i=0;i<=numOfReaders;i++){
		counter[i]=0;
		fprintf(fp,"0\n");
	}
}
else{
	for(i=0;!feof(fp);i++){
		fscanf(fp,"%d\n",&counter[i]);
	}
	fclose(fp);
}

sprintf(tempFile,"server%d.txt",port);
if((fp=fopen(tempFile,"r"))==NULL){
	fp=fopen(tempFile,"w");
	fprintf(fp,"0\t0\t0\n");
	ts=0;postit=0;value=-1;
}
else{
	fscanf(fp,"%d\t%d\t%d\n",&ts,&postit,&value);
	fclose(fp);
}
```

Το επόμενο βήμα του γύρου επικοινωνίας είναι η κατάτμηση του μηνύματος στα συστατικά του. Καθώς το μήνυμα είναι της μορφής: "*τυπος_μηνύματος, χρονική_σφραγίδα, τιμή, μετρητής_λειτουργίας, νοητό_αναγνωριστικό, μοναδικό_αναγωριστικό*", τότε η κατάτμηση γίνεται στις αντίστοιχες τοπικές μεταβλητές:

```
// Κατάτμηση μηνύματος σε συστατικά
msgType=strtok(buf,",");
tempTs=atoi(strtok(NULL,","));
tempValue=atoi(strtok(NULL,","));
rCounter=atoi(strtok(NULL,","));
uid=atoi(strtok(NULL,","));
id=atoi(strtok(NULL,"#"));
```

Ο ειδικός χαρακτήρας "#" χρησιμοποιείται για να δείξει το τέλος του μηνύματος. Όταν τελειώσει η κατάτμηση, ελέγχεται η χρονική σφραγίδα. Εάν είναι μεγαλύτερη από την ήδη υπάρχουσα στην τοπική μνήμη ενημερώνεται με την καινούργια χρονική σφραγίδα και η νέα τιμή της καινούργιας χρονικής σφραγίδας ανατίθεται στην παλιά. Επιπλέον, ενημερώνεται το σύνολο *seenSet*. Εάν η χρονική σφραγίδα που λήφθηκε είναι μεγαλύτερη, τότε το σύνολο *seenSet* εξισώνεται με το κενό σύνολο ($seenSet = \varnothing$), αλλιώς προστίθεται το νοητό αναγνωριστικό του ληφθέντος μηνύματος στο ήδη υπάρχον *seenSet* ($seenSet = seenSet \bigcup v_j$). Προσθέτοντας το νοητό αναγνωριστικό αντί του μοναδικού αναγνωριστικού διατηρούμε τον αριθμό των στοιχείων του συνόλου *seenSet* μικρότερο από $\frac{S}{t} - 2$ χωρίς να χρειάζεται να περιορίσουμε τον αριθμό των *Readers*, το οποίο είναι απαραίτητη προϋπόθεση για την ορθότητα του αλγορίθμου.

Τέλος, ο *Server* στέλνει μήνυμα απόκρισης δεχόμενος την αίτηση. Το μήνυμα απόκρισης ανάλογα με την αίτηση θα έχει τύπο *WRITEACK*, *READACK* ή *INFORMACK*. Όταν ο *Server* δεχτεί μια αίτηση με τύπο μηνύματος *INFORM* σημαίνει ότι κάποια διεργασία *Reader* θέλει να ενημερώσει τις υπόλοιπες για τη χρονική σφραγίδα που πρόκειται να επιστρέψει. Έτσι, ο *Server* προτού να ανταποκριθεί σε μια τέτοια αίτηση ενημερώνει το *postit* του. Εάν η καινούργια χρονική σφραγίδα είναι μεγαλύτερη από το *postit* τότε αυτή ανατίθεται στο *postit*, αλλιώς παραμένει το ίδιο. Η μορφή του μηνύματος είναι η εξής: "*τύπος_μηνύματος, χρονική_σφραγίδα, τιμή, μετρητής_λειτουργίας, postit, seenSet*". Όταν σταλεί το μήνυμα απόκρισης, ο *Server* αποθηκεύει τις τοπικές μεταβλητές του σε προσωρινά αρχεία όπως έχει ήδη αναφερθεί πιο πάνω. Όλα τα πιο πάνω περιγράφονται στο κομμάτι κώδικα που ακολουθεί:

```
// Έλεγχος αν το μήνυμα είναι καινούργιο ή παλιό
if(rCounter>=counter[id]){
	// Έλεγχος εάν η καινούργια χρονική σφραγίδα είναι μεγαλύτερη
	// από την παλιά
	if(tempTs>ts){
		// Ενημέρωση χρονικής σφραγίδας και seenSet
		ts=tempTs;
		seen=NULL;
	}
	// Προσθήκη νοητού αναγνωριστικού στο σύνολο seenSet
	push(&seen,uid);

	// Διαδικασία αποστολής μηνύματος απόκρισης ανάλογα με την
	// αίτηση
	if(strcmp(msgType,"READ")==0){
		bzero(buf2,sizeof(buf2));
		sprintf(buf2,"READACK,%d,%d,%d,%d,%s",ts,value,rCounter,po
		stit,seenString);
		write(newsock,buf2,sizeof(buf2))
	}
	else if(strcmp(msgType,"WRITE")==0){
		...
	// Η διαδικασία αποστολής για αίτηση γραφής ή ενημέρωσης είναι
	// η ίδια με την αίτηση ανάγνωσης και δεν υπάρχει λόγος να
	// ξαναγραφεί
	}
	else if(strcmp(msgType,"INFORM")==0){
		if(postit<tempTs){postit=tempTs;}
		...
}

// Δημιουργία ονόματος προσωρινού αρχείου βάσει του αριθμού θύρας
sprintf(tempFile,"server%dseen.txt",port);

// Αποθήκευση τοπικών μεταβλητών σε προσωρινά αρχεία
if((fp=fopen(tempFile,"w"))==NULL){
	exit(-1);
}
else{
	for(temp2=seen;temp2!=NULL;temp2=temp2->next){
		fprintf(fp,"%d\n",temp2->id);
	}
	fclose(fp);
}

// Παρόμοια λειτουργικότητα με πιο πάνω και δεν χρειάζεται να
επεξηγηθεί
sprintf(tempFile,"server%dcounter.txt",port);
if((fp=fopen(tempFile,"w"))==NULL){
	exit(-1);
}
else{
	for(i=0;i<=numOfReaders;i++){
		fprintf(fp,"%d\n",counter[i]);
	}
	fclose(fp);
}

sprintf(tempFile,"server%d.txt",port);
if((fp=fopen(tempFile,"w"))==NULL){
```

```
        exit(-1);
}
else{
        fprintf(fp,"%d\t%d\t%d\n",ts,postit,value);
        fclose(fp);
}
```

Διεργασία *Reader*: Η διεργασία *Reader* διατηρεί τις εξής τοπικές μεταβλητές: (1) *maxTS*, η οποία διατηρεί τη μέγιστη χρονική σφραγίδα που βρήκε ο *Reader* ανάμεσα στα μηνύματα απόκρισης από τους *Servers* στην προηγούμενη λειτουργία ανάγνωσης, (2) *rCounter*, η οποία διατηρεί τον αριθμό των λειτουργιών ανάγνωσης που έχουν γίνει μέχρι τώρα και χρησιμοποιείται για να ξεχωρίζει τα παλιά από τα καινούργια μηνύματα που παραλαμβάνει από τους *Servers*, (3) *maxPs*, η οποία διατηρεί το μέγιστο *postit* που βρήκε ο *Reader* ανάμεσα στα μηνύματα απόκρισης από τους *Servers* στην προηγούμενη λειτουργία ανάγνωσης.

Η διεργασία *Reader* εκτελεί μια λειτουργία ανάγνωσης ως εξής. Αρχικά, εγκαθιδρύει την επικοινωνία με τους *Servers*. Αυτό γίνεται με τη δημιουργία μιας υποδοχής ροής για κάθε *Server*, διαβάζοντας ένα αρχείο το οποίο δίνεται στην διεργασία από τις παραμέτρους της γραμμής εντολών (*command line arguments*). Από το αρχείο αυτό, διαβάζονται σειριακά οι παράμετροι που χρειάζονται για κάθε υποδοχή, όπως είναι το όνομα του εκάστοτε *Server* και ο αριθμός θύρας. Αυτό γίνεται στο κομμάτι κώδικα που ακολουθεί:

```
for(i=0;!feof(fp);i++){
        // Διαβάζει το όνομα του Server και τον αριθμό θύρας
        fscanf(fp,"%s\t%d\n",&serverSTR,&port);

        // Δημιουργεί νέα υποδοχή
        sock[i]=socket(AF_INET,SOCK_STREAM,0);

        // Μετατρέπει το όνομα του Server στην IP διεύθυνση του
        rem=gethostbyname(serverSTR);

        // Αναθέτει διάφορες παραμέτρους στην υποδοχή που δημιούργησε
        server[i].sin_family=AF_INET;
        bcopy((char*)rem->h_addr,&server[i].sin_addr,rem->h_length);
        server[i].sin_port=htons(port);
        serverptr=(struct sockaddr *) &server[i];
        serverlen=sizeof(server[i]);

        // Δεσμεύει την υποδοχή με την IP διεύθυνση του Server
        connect(sock[i],serverptr,serverlen);

}
```

Ακολούθως, στέλνει σε όλους τους *Servers* αίτηση ανάγνωσης (*read request*) όπου περιλαμβάνει τη τρέχουσα χρονική σφραγίδα και την αντίστοιχη τιμή. Ένα μήνυμα αίτησης ανάγνωσης έχει την ακόλουθη μορφή: "*READ, χρονική_σφραγίδα, τιμή, μετρητής_λειτουργίας, νοητό_αναγνωριστικό, μοναδικό_αναγνωριστικό*", όπου το *READ* περιγράφει το είδος του μηνύματος, η *χρονική_σφραγίδα* και η *τιμή* περιέχουν τη δυάδα της τιμής με την αντίστοιχη χρονική σφραγίδα, ο *μετρητής_λειτουργίας* περιγράφει τον αριθμό των λειτουργιών ανάγνωσης που έγιναν μέχρι τώρα, το *νοητό_αναγνωριστικό* περιγράφει το νοητό αναγνωριστικό του *Reader* και τέλος το *μοναδικό_αναγνωριστικό* περιγράφει το μοναδικό αναγνωριστικό του Reader. Η δημιουργία των μηνυμάτων φαίνεται πιο κάτω:

```
// Δημιουργία μηνύματος με τις απαραίτητες παραμέτρους
sprintf(buf,"READ,%d,%d,%d,%d,%d#",ts,value,rCounter,uid,id);
for(i=0;i<numberOfServers;i++){
      // Αποστολή του μηνύματος στον i-οστό Server
      write(sock[i],buf,sizeof(buf));
}
```

Εφόσον *t Servers* μπορούν να καταρρεύσουν, περιμένει για $S-t$ αποκρίσεις από τους *Servers*. Ακολούθως, ξεχωρίζει τα καινούργια από τα παλιά μηνύματα. Διαχωρίζει τα καινούργια μηνύματα που παραλαμβάνει από τους *Servers* στα συστατικά τους και υπολογίζει τη μέγιστη χρονική σφραγίδα και το μέγιστο *postit* ανάμεσα στις αποκρίσεις. Ακόμα, αποθηκεύει τα μηνύματα και το *seenSet* κάθε μηνύματος που παραλαμβάνει. Βασιζόμενη στις πληροφορίες που συνέλεξε, η διεργασία δημιουργεί το σύνολο μηνυμάτων *maxTSmsg*, το οποίο περιέχει τα μηνύματα που έχουν σαν *ts* το *maxTS*. Στη συνέχεια, υπολογίζεται η συνθήκη για να γνωρίζει η διεργασία αν χρειάζεται να εκτελεστεί δεύτερος γύρος επικοινωνίας και να αποφασιστεί η τιμή της χρονικής σφραγίδας που θα επιστρέψει η διεργασία. Η διεργασία προσπαθεί να υπολογίσει το ελάχιστο $a \in [1, V+1]$, τέτοιο ώστε να υπάρχει ένα σύνολο *MS* όπου $MS \subseteq maxTSmsg$, με αριθμό στοιχείων $|MS| \geq S-at$ και ο αριθμός στοιχείων της τομής των συνόλων *seenSet* των μηνυμάτων του *MS* είναι $|m \in MS] \; m.seenSet| \geq a$. Εάν υπάρχει ένα τέτοιο *α*, τότε η λειτουργία ανάγνωσης επιστρέφει το *maxTS*, αλλιώς επιστρέφει *maxTS -1*. Με άλλα λόγια, ο έλεγχος που κάνει η συνθήκη είναι εάν έχουν δει αρκετοί *Readers* τη μέγιστη χρονική στιγμή *maxTS*. Δεύτερος γύρος επικοινωνίας είναι απαραίτητος αν ισχύει ότι $|m \in MS] \; m.seenSet| = a$. Κατά τη διάρκεια του

δεύτερου γύρου επικοινωνίας, ο *Reader* ενημερώνει $3t+1$ Servers για τη χρονική σφραγίδα που πρόκειται να επιστρέψουν, στέλνοντας τους αιτήσεις ενημέρωσης (*inform requests*). Η λειτουργία τερματίζει όταν παραλάβει $2t+1$ αποκρίσεις από τους *Servers* και επιστρέφει *maxTS*. Αν δεν ισχύει η συνθήκη, τότε ελέγχεται κατά πόσο έχει βρεθεί *postit* ίσο με το *maxTS* ανάμεσα στα μηνύματα απόκρισης από τους *Servers*. Εάν έχει βρεθεί ένα τέτοιο *postit*, σημαίνει ότι κάποια διεργασία, προηγουμένως ή ταυτόχρονα με τον συγκεκριμένο *Reader*, έχει επιστρέψει ή προτίθεται να επιστρέψει το *maxTS*. Αν δεν έχει βρεθεί ένα τέτοιο *postit* τότε η λειτουργία επιστρέφει *maxTS-1* πραγματοποιώντας μόνο ένα γύρο επικοινωνίας. Εάν ο *Reader* παραλάβει περισσότερα από $t+1$ μηνύματα απόκρισης που περιέχουν αυτό το *postit*, επιστρέφει *maxTS* χωρίς να εκτελέσει δεύτερο γύρο επικοινωνίας. Τέλος, αν δεν ισχύει ούτε αυτό, τότε ο *Reader* πραγματοποιεί δεύτερο γύρο επικοινωνίας έτσι ώστε να εγγυηθεί ότι οποιαδήποτε λειτουργία ανάγνωσης θα παραλάβει το ίδιο *postit*. Ακολουθεί ο κώδικας που υπολογίζει τη συνθήκη:

```
// Μέθοδος η οποία ελέγχει αν ισχύει η συνθήκη. Χρησιμοποιεί τα σύνολα
// MS και maxTSmsg, υπολογίζει τη τομή των συνόλων των μηνυμάτων του
// maxTSmsg και επιστρέφει αναλόγως 0,1 ή 2. Επιστρέφει 0 αν δεν
// ισχύει η συνθήκη, 1 αν η συνθήκη είναι μεγαλύτερη από α και 2 αν η
// συνθήκη είναι ίση με α.
validation=isConditionValid(maxTSsize);

if(validation>0){
      if((validation==2)&&((maxPS<maxTS)||(maxPSsize<(t+1)))){
            // Δεύτερος γύρος επικοινωνίας
            //Αποστολή μηνυμάτων ενημέρωσης σε 3t+1 Servers
            for(i=0;i<((3*t)+1);i++){
                  sprintf(buf,"INFORM,%d,%d,%d,%d,%d#",maxTS,value,rCo
                  unter,uid,id);
                  write(sock[i],buf,sizeof(buf))
            }
            // Αναμονή απόκρισης από 2t+1 Servers
            for(i=0;i<((2*t)+1);i++){
                  bzero(buf2,sizeof(buf2));
                  read(sock[j],buf2,sizeof(buf2));
            }
            if(j==numberOfServers-t){
                  j=0;
            }
            else{
                  j++;
            }
      }
      else{
            // Επιστρέφεται maxTS, ένας γύρος επικοινωνίας
            return maxTS;
      }
```

```
else if(maxPS==maxTS){
      if(maxPSsize<(t+1)){
            // Δεύτερος γύρος επικοινωνίας, ακριβώς όπως πιο πάνω
            ...
      }
      else{
            // Επιστρέφεται maxTS, ένας γύρος επικοινωνίας
            return maxTS;
      }
else{
            // Επιστρέφεται maxTS-1, ένας γύρος επικοινωνίας
            return maxTS-1;
}
```

3.4 Πως τρέχει η εφαρμογή

Το πρώτο πράγμα που πρέπει να κάνουμε είναι η εκκίνηση των εξυπηρετητών που θα χρησιμοποιηθούν από τον πελάτη. Όπως σε όλες τις εφαρμογές εξυπηρετητή-πελάτη έτσι και σε αυτή, η εφαρμογή στην πλευρά του πελάτη δεν μπορεί να χρησιμοποιηθεί αν δεν είναι ενεργοποιημένη η αντίστοιχη εφαρμογή στην πλευρά του εξυπηρετητή. Αφού ενωθούμε με πρωτόκολλο *ssh* στον κάθε εξυπηρετητή, ακολούθως πρέπει ξεκινήσουμε την διεργασία του *Server* με την πιο κάτω εντολή:

```
> server αριθμός_θύρας αριθμός_Readers
```

Η πρώτη παράμετρος περιγράφει τον αριθμό θύρας στην οποία θα δέχεται αιτήσεις ο *Server* και η δεύτερη παράμετρος γνωστοποιεί στον *Server* τον αριθμό των *Readers* που θα υπάρχουν στο σενάριο έτσι ώστε να γνωρίζει πόση μνήμη πρέπει να δεσμεύσει.

Την πιο πάνω διαδικασία θα πρέπει να κάνουμε σε όλους τους εξυπηρετητές που ενδέχεται να χρησιμοποιηθούν από τους πελάτες μας. Θα μπορούσαμε να γράψουμε ένα πρόγραμμα (*script*) που θα αυτοματοποιούσε την διαδικασία αυτή, η οποία όταν γίνεται από τον χρήστη είναι πολύ χρονοβόρα.

Στην πλευρά του πελάτη η εφαρμογή τρέχει όπως κάθε άλλη εφαρμογή. Όταν ξεκινήσουμε το πρόγραμμα στην πλευρά του πελάτη, τότε αμέσως δημιουργείται επικοινωνία με όλους τους εξυπηρετητές. Αυτό γίνεται εκτελώντας τις πιο κάτω εντολές για τον *Writer* και για τον *Reader* αντίστοιχα:

```
> writer αρχείο_Servers.txt
> reader αρχείο_Servers.txt μοναδικό_αναγνωριστικό_Reader
```

Η πρώτη παράμετρος περιγράφει το αρχείο που περιέχει τον αριθμό των *Servers*, τον αριθμό *t* και το όνομα κάθε *Server* μαζί με τον αντίστοιχο αριθμό θύρας για να εγκαθιδρυθεί η επικοινωνία, ενώ η δεύτερη παράμετρος (στην περίπτωση που ο πελάτης είναι *Reader*) είναι το μοναδικό αναγνωριστικό του εκάστοτε *Reader*. Δεν χρησιμοποιείται στην περίπτωση που ο πελάτης είναι ο *Writer* διότι το δικό του μοναδικό αναγνωριστικό είναι κατά συνθήκη το μηδέν.

Ένα ανάλογο πρόγραμμα (*script*) θα μπορούσε να γραφεί και για την εκκίνηση των διεργασιών *Readers* και *Writer*, καθώς και αυτή η διαδικασία είναι σχετικά χρονοβόρα όταν γίνεται από το χρήστη. Κατά τη διάρκεια της εκτέλεσης της υλοποίησης, όλες οι διεργασίες, τόσο οι εξυπηρετητές, όσο και οι πελάτες, παρουσιάζουν στην οθόνη τα μηνύματα που στέλνουν και παραλαμβάνουν.

Θα εξηγήσουμε ένα παράδειγμα εκτέλεσης όπως φαίνεται στην Εικόνα 3.1. Το παράδειγμα χρησιμοποιεί 9 μηχανές, 4 ως *Servers*, 4 ως *Readers* και 1 ως *Writer*, οι οποίες τρέχουν σε διαφορετικές μηχανές. Αρχικά, ο χρήστης θα πρέπει να κάνει την διαδικασία που είπαμε στο προηγούμενο κεφάλαιο για να στείλει τα αρχεία του στο "μέρισμα" του στις μηχανές του *Planetlab*, χρησιμοποιώντας την εντολή που αναφέρθηκε στο κεφάλαιο 2 και φαίνεται πιο κάτω:

```
> scp -i ~/.ssh/id_rsa -r app_name slice_name@node_name:
```

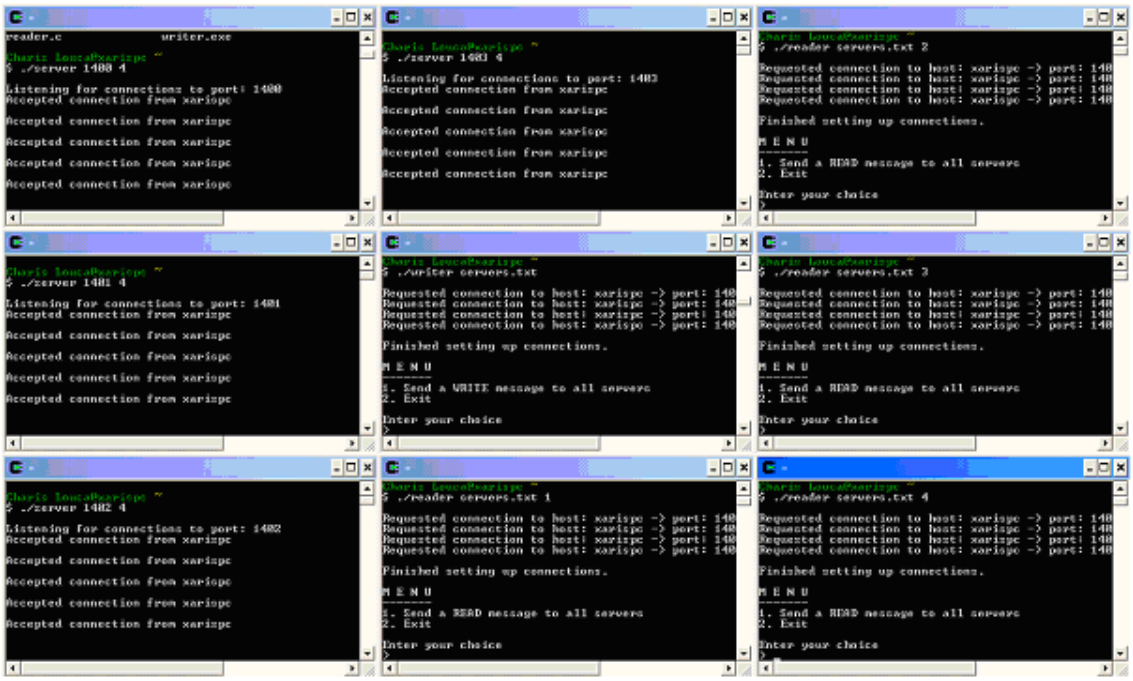
Τα αρχεία που πρέπει να στείλει είναι τα εξής:

- *server* στις μηχανές που θα τρέξουν διεργασίες *Server*
- *reader* και *servers.txt* στις μηχανές που θα τρέξουν διεργασίες *Reader*
- *writer* και *servers.txt* στη μηχανή που θα τρέξει τη διεργασία *Writer*

Ακολούθως, αφού βρεθεί μέσα στο "μέρισμα" του θα πρέπει να βρίσκεται στον φάκελο όπου είναι η εφαρμογή, όπου θα πρέπει να ξεκινήσει πρώτα όλους τους *Servers* και

μετά να ξεκινήσει τις διεργασίες των πελατών. Έτσι, ενωνόμαστε στις μηχανές του *Planetlab* όπου θα τρέξει η εφαρμογή μας, χρησιμοποιώντας το πρωτόκολλο *ssh* με την εντολή:

```
> ssh -l Cyprus_SF -i ~/.ssh/id_rsa όνομα_κόμβου
```

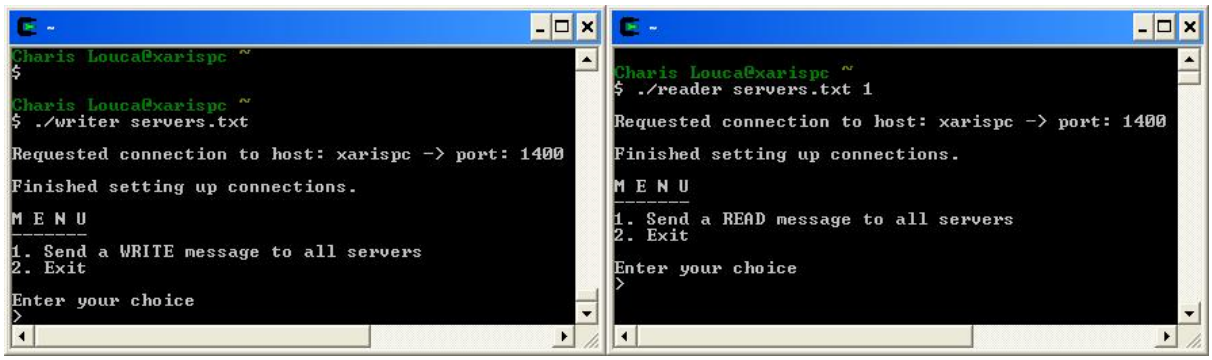


Εικόνα 3.1 Παράδειγμα εκτέλεσης (5 *Servers*, 4 *Readers*, 1 *Writer*)

Στη συνέχεια, τρέχουμε την διεργασία *Server* σε κάθε μηχανή που ενωθήκαμε αλλάζοντας μόνο τον αριθμό θύρας ακριβώς όπως τους έχουμε σημειώσει στο αρχείο μας, και τρέχουμε την εντολή όπως πιο πάνω:

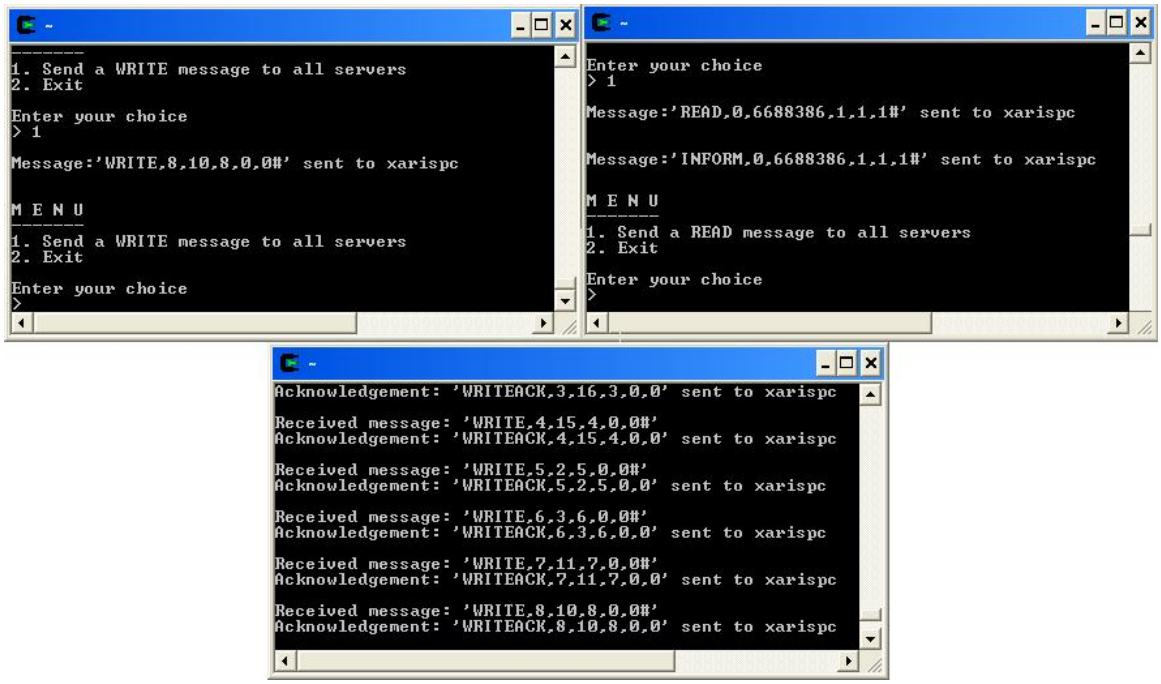
```
> server 1400 4
```

Στο παράδειγμα μας έχουμε χρησιμοποιήσει αριθμό θύρας 1400. Μετά, ξεκινούμε τις διεργασίες *Writer* και *Readers* όπου και εμφανίζεται το μενού επιλογής, όπως φαίνεται στην Εικόνα 3.2.

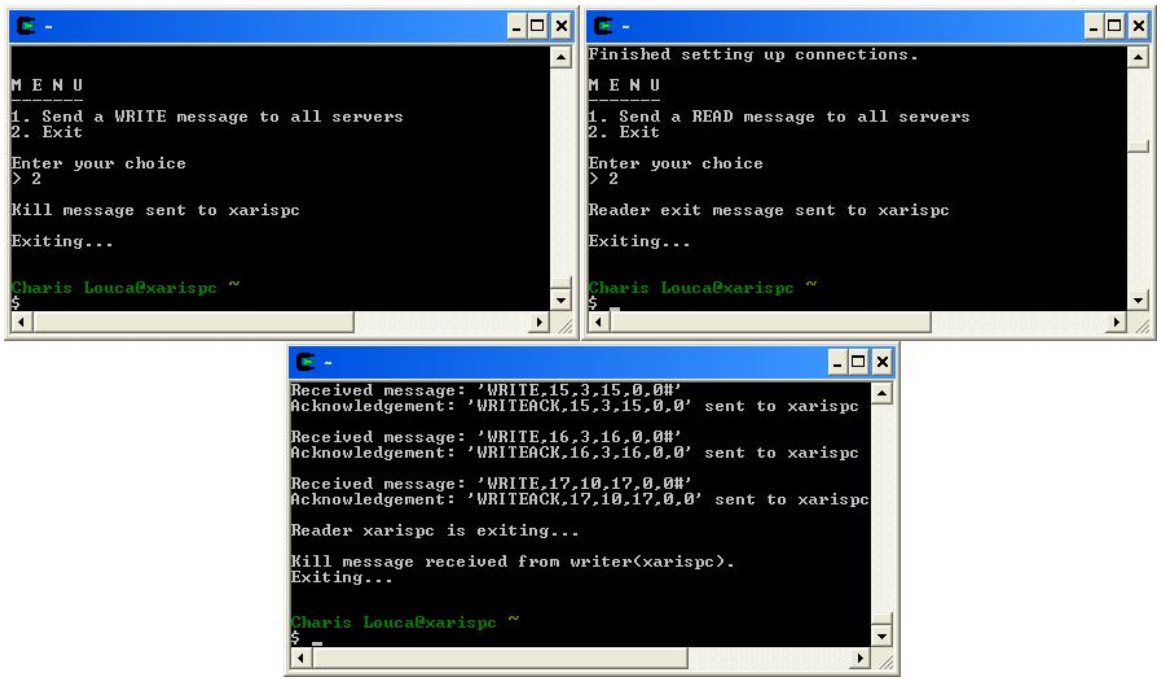


Εικόνα 3.2 Μενού επιλογών *Writer* και *Reader* αντίστοιχα

Όπως φαίνεται και πιο πάνω, ο χρήστης έχει δύο επιλογές και στις δύο διεργασίες. Στην περίπτωση του *Reader*, οι επιλογές του χρήστη είναι είτε να στείλει αιτήσεις ανάγνωσης στους *Servers* και να εκτελέσει λειτουργίες ανάγνωσης, είτε να τερματίσει τη διεργασία του *Reader*. Στην περίπτωση του *Writer*, ο χρήστης έχει και πάλι δύο επιλογές. Η πρώτη είναι πανομοιότυπη με αυτή του *Reader*, με τη διαφορά ότι οι αιτήσεις θα είναι αιτήσεις γραφής αντί ανάγνωσης. Η δεύτερη επιλογή είναι να τερματίσει την διεργασία του *Writer*, αλλά ταυτόχρονα να στείλει "*kill*" μηνύματα στους *Servers* με σκοπό να τερματίσουν και αυτοί τη λειτουργία τους. Παραδείγματα των εκτελέσεων των δύο επιλογών και για τις δύο διεργασίες φαίνονται στις Εικόνες 3.3 και 3.4 που ακολουθούν.



Εικόνα 3.3 Παράδειγμα επιλογής 1 στο μενού των διεργασιών *Writer* και *Reader*



Εικόνα 3.4 Παράδειγμα επιλογής 2 στο μενού των διεργασιών *Writer* και *Reader*

Κεφάλαιο 4

Ανάλυση Πειραματικών Αποτελεσμάτων

4.1 Προκαταρκτικά

Για να πετύχουμε τους στόχους με τους οποίους ξεκίνησε αυτή η εργασία, δημιουργήθηκαν πολλά σενάρια με πολλές αλλαγές έτσι ώστε να έχουμε όσο πιο πολλά αποτελέσματα με διαφορετικές παραμέτρους, για να μπορέσουμε να οδηγηθούμε σε χρήσιμα συμπεράσματα. Τα διαφορά σενάρια στα οποία τρέχει η εφαρμογή επιλέχθηκαν σύμφωνα με τους πόρους που είχαμε και σύμφωνα με τις συγκεκριμένες παραμέτρους που θέλαμε να αξιολογήσουμε. Οι παράμετροι που αλλάζουν σε κάθε σενάριο είναι οι εξής:

Αριθμός των *Readers*: Αλλάζουμε τον αριθμό των διεργασιών *Readers* που κάνουν αιτήσεις στην υλοποίηση μας, για να δούμε κατά πόσο βελτιώνεται ή χειροτερεύει η απόδοση ανάλογα με τον αριθμό των αιτήσεων που γίνονται. Με τον όρο απόδοση εννοούμε τον μέσο χρόνο εκτέλεσης μιας λειτουργίας, ανάγνωσης ή γραφής, και το ποσοστό των λειτουργιών ανάγνωσης που εκτελούν δεύτερο γύρο επικοινωνίας. Μεγαλώνουμε τον αριθμό των *Readers* που ζητούν εξυπηρέτηση την ίδια ώρα από τους *Servers*, για να δούμε πως επηρεάζεται η απόδοση του αλγορίθμου και κατά πόσο μπορούν να αντεπεξέλθουν οι *Servers* στον αυξημένο φόρτο αιτήσεων ανάγνωσης.

Αριθμός *t* των *Servers* που μπορούν να καταρρεύσουν: Αυξάνουμε κάθε φορά τον αριθμό των *Servers* που μπορούν να καταρρεύσουν, μέχρι να φτάσουμε στο επιτρεπόμενο όριο ($t < \frac{S}{2}$). Η αύξηση αυτή γίνεται έτσι ώστε να δούμε πως επηρεάζεται η απόδοση και κατά πόσο μπορούν να αντεπεξέλθουν οι *Servers*.

Έχουν γίνει πολλά πειράματα με βάση τις πιο πάνω παραμέτρους. Αρχικά, θα παρουσιάσουμε μερικά που δίνουν ενδιαφέροντα αποτελέσματα και συμπεράσματα και ακολούθως θα τα κατηγοριοποιήσουμε για να έχουμε μια πιο πλήρη και συγκεντρωτική εικόνα στο τι ακριβώς συμβαίνει.

4.2 Σενάρια

Για την αξιολόγηση της υλοποίησης μας, έχουμε τρέξει την εφαρμογή μας στο κατανεμημένο σύστημα υπολογιστών *Planetlab*. Το κύριο αποτέλεσμα που επιζητούμε είναι το ποσοστό των λειτουργιών ανάγνωσης που εκτελούν δεύτερο γύρο επικοινωνίας σε σχέση με τον αριθμό των διεργασιών *Readers* που τρέχουν σε κάθε σενάριο και τον αριθμό *t* των *Servers* που μπορούν να καταρρεύσουν. Μετρούμε επίσης και το χρόνο εκτέλεσης κάθε λειτουργίας γραφής και ανάγνωσης που εκτελείται.

Η εκτέλεση της εφαρμογής μας θα περιέχει είκοσι *Servers* ($S = 20$). Για να εγγυηθούμε τον τερματισμό της εφαρμογής πρέπει να περιορίσουμε τον αριθμό *t* των *Servers* που μπορούν να καταρρεύσουν, έτσι ώστε $V < \frac{S}{t} - 2$ ή $V \le \frac{S}{t} - 3$. Συνεπώς $t \le \frac{S}{V+3}$. Για να μπορέσουμε να διατηρήσουμε τουλάχιστον έναν νοητό κόμβο ($V = 1$), το *t* δεν πρέπει να ξεπερνά τις πέντε απώλειες. Απόρροια αυτού είναι να μην επιτρέπουμε σε πάνω από πέντε *Servers* να καταρρεύσουν σε τυχαίες χρονικές στιγμές. Έτσι, εκτελούμε το κάθε σενάριο για τιμές του *t* από ένα μέχρι πέντε. Η κατάρρευση δεν έχει υλοποιηθεί στον αλγόριθμο αλλά μπορεί εύκολα να προστεθεί στον πηγαίο κώδικα της υλοποίησης μας σαν μια επιπρόσθετη παράμετρος *terminate* σαν ένας ακέραιος αριθμός από τη γραμμή εντολών (*command line arguments*). Η παράμετρος *terminate* θα χρησιμοποιείται είτε για να τερματίζει τη διεργασία *Server* σε *terminate* δευτερόλεπτα από την αρχή της διεργασίας, είτε να τερματίζει όταν η χρονική σφραγίδα (*timestamp*)

γίνει ίση με *terminate*. Τέλος, διαφοροποιούμε τον αριθμό των διεργασιών των *Readers* από δέκα μέχρι τριάντα.

Για την υλοποίηση των σεναρίων, θεωρούμε δύο μεταβλητές χρόνου, τις *wInt* και *rInt* (και οι δύο είναι μεγαλύτερες από ένα δευτερόλεπτο). Οι μεταβλητές αυτές χρησιμοποιούνται για την μοντελοποίηση των χρονικών διαστημάτων μεταξύ δύο συνεχόμενων λειτουργιών γραφής και ανάγνωσης αντίστοιχα.

Ακολούθως, έχουμε διαχωρίσει τρεις περιπτώσεις ανάλογα με τα χρονικά διαστήματα *wInt* και *rInt*:

1. $rInt < wInt$: Μοντελοποίηση με συχνότερες λειτουργίες ανάγνωσης απ' ότι λειτουργίες γραφής
2. $rInt = wInt$: Μοντελοποίηση με λειτουργίες ανάγνωσης και γραφής με την ίδια συχνότητα
3. $rInt > wInt$: Μοντελοποίηση με συχνότερες λειτουργίες γραφής απ' ότι λειτουργίες ανάγνωσης

Η πιο πάνω μοντελοποίηση είναι δύσκολο να επιτευχθεί σε ένα ρεαλιστικό δίκτυο υπολογιστών που είναι κτισμένο πάνω από το διαδίκτυο όπως είναι το *Planetlab*, καθώς η συμπεριφορά των διεργασιών εξαρτάται και από την συμπεριφορά του διαδικτύου. Εξωτερικοί παράγοντες επηρεάζουν την εφαρμογή μας και ο πιο σημαντικός από αυτούς είναι τυχόν συμφόρηση που μπορεί να υπάρξει στο διαδίκτυο την συγκεκριμένη στιγμή που θα τρέξουμε τα σενάρια. Έτσι, σε κάποιες λειτουργίες δεν θα ισχύουν οι πιο πάνω υποθέσεις με άμεσο αντίκτυπο την αλλοίωση των αποτελεσμάτων μας. Πιο κάτω καθορίζουμε τα χρονικά διαστήματα *rInt* και *wInt*.

Βάσει των πιο πάνω υποθέσεων, θέτουμε τα διαστήματα ως εξής για τις τρεις περιπτώσεις:

1. $rInt < wInt$: $wInt = 4.3$ δευτερόλεπτα, $rInt = 2.3$ δευτερόλεπτα
2. $rInt = wInt$: $wInt = 4.3$ δευτερόλεπτα, $rInt = 4.3$ δευτερόλεπτα
3. $rInt > wInt$: $wInt = 4.3$ δευτερόλεπτα, $rInt = 6.3$ δευτερόλεπτα

Υλοποιήσαμε τις διεργασίες με τέτοιο τρόπο ώστε τα μόνα μηνύματα που μεταδίδονται στο μοντέλο μας είναι οι αιτήσεις από τους πελάτες της υλοποίησης και οι αποκρίσεις από τους εξυπηρετητές. Επίσης, η υλοποίηση δεν επιτρέπει στις διεργασίες να επικοινωνούν μεταξύ τους, ούτε στους εξυπηρετητές να έχουν την δυνατότητα να ανταλλάζουν μηνύματα μεταξύ τους.

Οι μηχανές του *Planetlab* που χρησιμοποιήθηκαν για την εκτέλεση των σεναρίων αναφέρονται στον Πίνακα 4.1 που ακολουθεί.

Αύξων Αριθμός Διεργασίας	**Όνομα μηχανής (*hostname*)**	**Γεωγραφική τοποθεσία**
Writer	planet1.jaist.ac.jp	Ιαπωνία
Reader 1, Reader 11, Reader 21	planetlab1.di.unito.it	Ιταλία
Reader 2, Reader 12, Reader 22	planetlab2.di.unito.it	Ιταλία
Reader 3, Reader 13, Reader 23	planetlab1.hiit.fi	Φιλανδία
Reader 4, Reader 14, Reader 24	planetlab2.hiit.fi	Φιλανδία
Reader 5, Reader 15, Reader 25	planetlab1.tamu.edu	ΗΠΑ
Reader 6, Reader 16, Reader 26	planetlab1.elet.polimi.it	Ιταλία
Reader 7, Reader 17, Reader 27	planetlab2.elet.polimi.it	Ιταλία
Reader 8, Reader 18, Reader 28	planetlab1.itwm.fhg.de	Γερμανία
Reader 9, Reader 19, Reader 29	planetlab2.itwm.fhg.de	Γερμανία
Reader 10, Reader 20, Reader 30	planetlab2.csail.mit.edu	ΗΠΑ
Server 1, 2, 3, 4	planetlab1.csail.mit.edu	ΗΠΑ
Server 5, 6, 7, 8	planetlab2.inf.ethz.ch	Ελβετία
Server 9, 10, 11, 12	planetlab3.inf.ethz.ch	Ελβετία
Server 13, 14, 15. 16	planetlab4.inf.ethz.ch	Ελβετία
Server 17, 18, 19, 20	planetlab5.csail.mit.edu	ΗΠΑ

Πίνακας 4.1 Μηχανές του *Plantelab* που χρησιμοποιήθηκαν

Ακολούθως, θεωρούμε τα εξής σενάρια:

1ο Σενάριο: Εκτέλεση με τυχαία χρονικά διαστήματα αναμονής μεταξύ λειτουργιών

Στο σενάριο αυτό παρουσιάζουμε μια εκτέλεση της υλοποίησης μας όπου οι διεργασίες όταν τελειώσουν μια λειτουργία ανάγνωσης ή γραφής, περιμένουν ένα τυχαίο χρονικό διάστημα μεταξύ ενός δευτερολέπτου και *rInt* (και *wInt* αντίστοιχα), αναλόγως της διεργασίας που εκτελεί την λειτουργία.

2ο Σενάριο: Εκτέλεση με σταθερά χρονικά διαστήματα αναμονής μεταξύ λειτουργιών

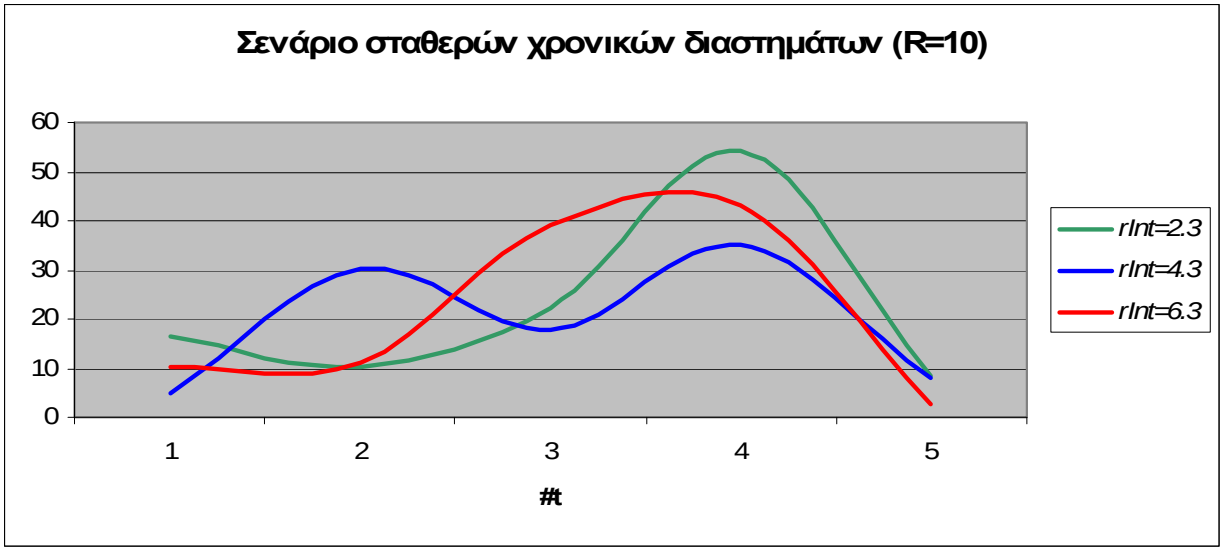
Στο σενάριο αυτό παρουσιάζουμε μια εκτέλεση της υλοποίησης μας όπου οι διεργασίες όταν τελειώσουν μια λειτουργία ανάγνωσης ή γραφής, περιμένουν ένα σταθερό χρονικό διάστημα *rInt* (και *wInt* αντίστοιχα), αναλόγως της διεργασίας που εκτελεί την λειτουργία.

4.3 Αποτελέσματα και Συμπεράσματα Σεναρίων

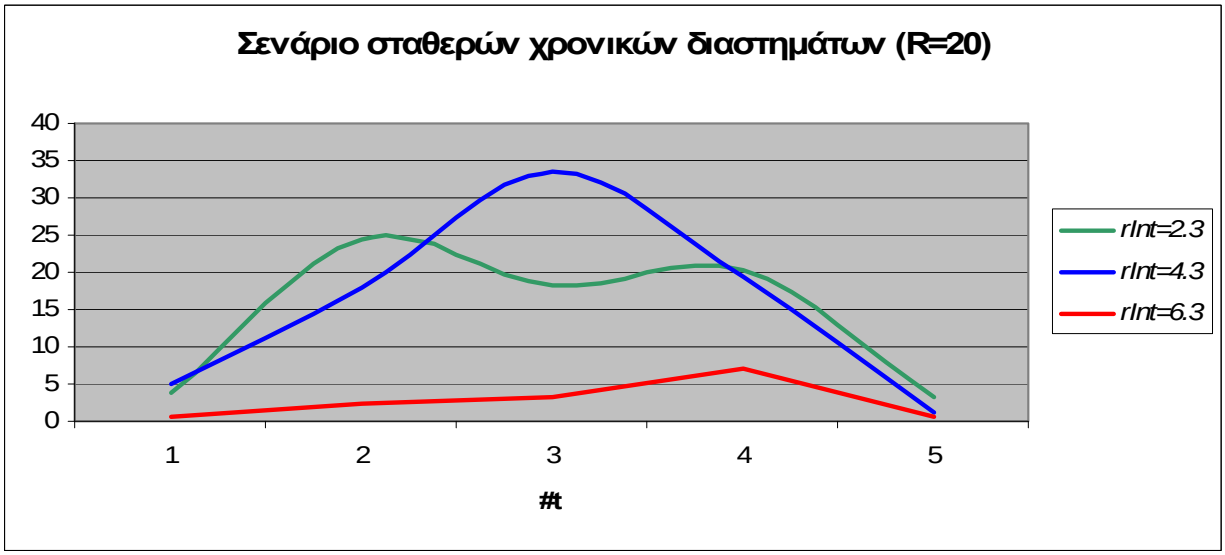
Τα αποτελέσματα που ακολουθούν, έχουν παρθεί με την ακόλουθη μέθοδο: Εκτελέσαμε την εφαρμογή με 100 λειτουργίες ανάγνωσης για κάθε διεργασία *Reader* και ακολούθως προσθέσαμε το σύνολο των λειτουργιών που χρειάστηκαν δεύτερο γύρο επικοινωνίας και διαιρέσαμε το αποτέλεσμα με το συνολικό αριθμό των λειτουργιών ανάγνωσης που εκτελέστηκαν. Έτσι, προέκυψαν τα ποσοστά που φαίνονται στους Πίνακες Α.1 και Α.2 στο Παράρτημα Α. Τέλος, στους Πίνακες Α.3-Α.8 του Παραρτήματος Α φαίνονται τα αποτελέσματα ξεχωριστά για κάθε διεργασία *Reader* της εκτέλεσης των Σεναρίων.

Σενάριο σταθερών χρονικών διαστημάτων: Από τους πίνακες Α.2, Α.5, Α.6 και Α.8 του Παραρτήματος Α, αλλά και από τα Γραφήματα 4.2 και 4.3 που ακολουθούν, μπορούμε να συμπεράνουμε ότι τα καλύτερα αποτελέσματα παρουσιάστηκαν στο σενάριο των σταθερών χρονικών διαστημάτων λόγω του ότι το ποσοστό των διεργασιών που εκτελούσαν δεύτερο γύρο επικοινωνίας ήταν κατά κανόνα μικρότερο από την αντίστοιχη εκτέλεση του σεναρίου των τυχαίων χρονικών διαστημάτων. Αυτό οφείλεται στο γεγονός ότι όταν οι διεργασίες περίμεναν σταθερό χρόνο πριν να εκτελέσουν την επόμενη λειτουργία γραφής (ή ανάγνωσης), ήταν δύσκολο να έχουμε

ταυτόχρονες λειτουργίες γραφής και ανάγνωσης, εκτός από την περίπτωση που είχαμε ίδιο χρονικό διάστημα μεταξύ λειτουργιών γραφής και ανάγνωσης (*rInt=wInt=4.3*).



Γράφημα 4.2 Αποτελέσματα Σεναρίου Σταθερών Χρονικών Διαστημάτων (R=10)



Γράφημα 4.3 Αποτελέσματα Σεναρίου Σταθερών Χρονικών Διαστημάτων (R=20)

Στα Γραφήματα 4.2 και 4.3 που φαίνονται πιο πάνω, παρουσιάζονται δυο γραφικές παραστάσεις των αποτελεσμάτων του σεναρίου σταθερών χρονικών διαστημάτων, ανάλογα με την τιμή του χρονικού διαστήματος *rInt*. Στον οριζόντιο άξονα των *x*, παρουσιάζεται ο αριθμός των *Servers* που μπορούν να καταρρεύσουν *t* και στον κατακόρυφο άξονα των *y* παρουσιάζεται το ποσοστό των λειτουργιών που εκτελούν δεύτερο γύρο επικοινωνίας.

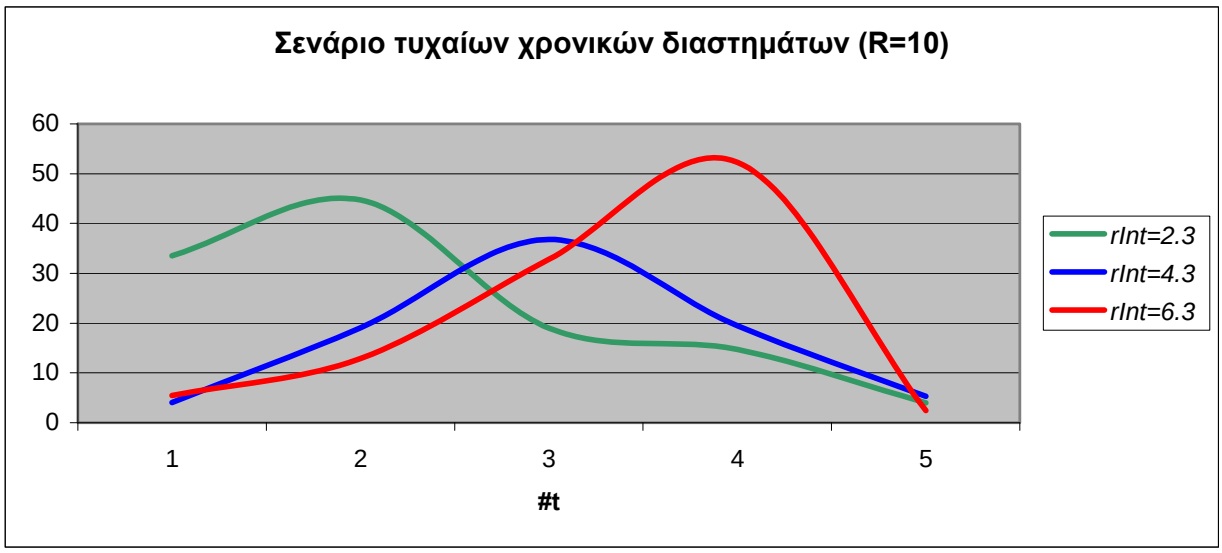
Αυτό που συμπεραίνουμε συγκρίνοντας τις δυο γραφικές παραστάσεις είναι ότι όσο μεγαλώνει ο αριθμός των *Readers* στο σύστημα, τόσο μειώνεται το ποσοστό των λειτουργιών που εκτελούν δεύτερο γύρο επικοινωνίας. Το φαινόμενο αυτό οφείλεται στο γεγονός ότι οι νοητοί κόμβοι έχουν περισσότερα μέλη σε κάθε εκτέλεση του αντίστοιχου σεναρίου, έτσι η συνθήκη ικανοποιείται γρηγορότερα και λιγότεροι *Readers* είναι αναγκασμένοι να εκτελέσουν δεύτερο γύρο επικοινωνίας. Ακόμα, το ίδιο φαινόμενο παρατηρείται όταν μειώνεται ο αριθμός των *Servers* που μπορούν να καταρρεύσουν, καθώς μεγαλώνει ο αριθμός των νοητών κόμβων (*virtual nodes*) του συστήματος.

Στην περίπτωση όπου είχαμε ίσο χρονικό διάστημα καθυστέρησης των λειτουργιών, παρατηρούμε ότι το ποσοστό των δεύτερων κύκλων επικοινωνίας είναι αυξημένο σε σχέση με τις υπόλοιπες εκτελέσεις του σεναρίου όταν ο αριθμός των *Readers* είναι ίσος με είκοσι, γεγονός που οφείλεται στην ύπαρξη ταυτόχρονων λειτουργιών γραφής και ανάγνωσης, αναγκάζοντας έτσι τις λειτουργίες ανάγνωσης να ενημερώνουν τις υπόλοιπες διεργασίες για το ζεύγος τιμής και χρονικής σφραγίδας που πρόκειται να επιστρέψουν, εκτελώντας δεύτερο γύρο επικοινωνίας. Τα ακριβή ποσοστά δεύτερων κύκλων επικοινωνίας παρουσιάζονται τόσο στους πίνακες του Παραρτήματος Α που αναφέρονται πιο πάνω.

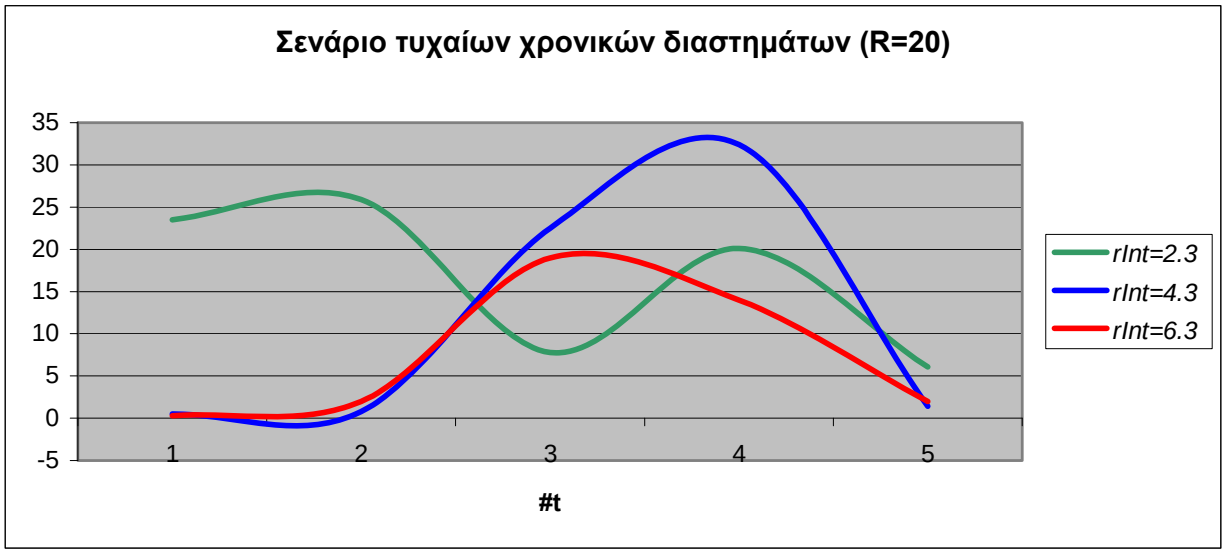
Ακολούθως, πάρθηκαν μετρήσεις για τον κάθε Reader ξεχωριστά για τον αριθμό των δεύτερων γύρων επικοινωνίας που εκτέλεσε ο καθένας και παρατηρήθηκε ότι κάποιες διεργασίες και ειδικότερα όταν $t=3$ ή $t=4$, εκτελούσαν δεύτερους γύρους επικοινωνίας με ανομοιόμορφο τρόπο. Λέγοντας ότι εκτελούσαν δεύτερους γύρους επικοινωνίας με ανομοιόμορφο τρόπο, εννοούμε ότι κάποιες λειτουργίες εκτελούσαν πάρα πολλούς δεύτερους γύρους επικοινωνίας (>60), ενώ κάποιες άλλες δεν εκτελούσαν σχεδόν καθόλου (<5 μέχρι και 0). Το φαινόμενο αυτό οφείλεται στο ότι οι διεργασίες που εκτελούσαν υπερβολικό αριθμό δεύτερων γύρων επικοινωνίας, ήταν τα πρώτα μέλη του νοητού κόμβου στον οποίο ανήκαν που εκτελούσαν την πρώτη λειτουργία ανάγνωσης για κάθε νέα χρονική σφραγίδα. Έτσι, δεν θα μπορούσαν να επιστρέψουν τη χρονική σφραγίδα που έχουν παραλάβει από τους *Servers*, καθώς δεν υπήρχαν άλλες διεργασίες που να έχουν δει αυτή τη χρονική σφραγίδα και έτσι να ικανοποιείται η συνθήκη ελέγχου της συγκεκριμένης διεργασίας *Reader*.

Σενάριο τυχαίων χρονικών διαστημάτων: Όπως έχουμε προαναφέρει το σενάριο τυχαίων διαστημάτων έχει παρουσιάσει χειρότερα αποτελέσματα από το σενάριο σταθερών διαστημάτων, λόγω της παρουσίας της συχνότερης ταυτόχρονης εκτέλεσης των λειτουργιών στο σενάριο αυτό. Στο σενάριο τυχαίων διαστημάτων, όπως και στο σενάριο σταθερών διαστημάτων, η αύξηση του αριθμού των *Readers* επιδρά θετικά στην απόδοση του αλγορίθμου, καθώς το ποσοστό των δεύτερων κύκλων επικοινωνίας μειώνεται όταν αυξηθεί ο αριθμός των *Readers* στο δίκτυο. Από την άλλη, το ποσοστό των δεύτερων γύρων επικοινωνίας αυξάνεται όταν μειώνεται ο αριθμός των *Servers* που μπορούν να καταρρεύσουν, καθώς μεγαλώνει ο αριθμός των νοητών κόμβων (*virtual nodes*) του συστήματος.

.



Γράφημα 4.4 Αποτελέσματα Σεναρίου Τυχαίων Χρονικών Διαστημάτων (R=10)



Γράφημα 4.5 Αποτελέσματα Σεναρίου Τυχαίων Χρονικών Διαστημάτων (R=20)

Στα Γραφήματα 4.4 και 4.5 που φαίνονται πιο πάνω, παρουσιάζονται δυο γραφικές παραστάσεις των αποτελεσμάτων του σεναρίου τυχαίων χρονικών διαστημάτων, ανάλογα με την τιμή του χρονικού διαστήματος *rInt*. Στον οριζόντιο άξονα των *x*, παρουσιάζεται ο αριθμός των *Servers* που μπορούν να καταρρεύσουν *t* και στον κατακόρυφο άξονα των *y* παρουσιάζεται το ποσοστό των λειτουργιών που εκτελούν δεύτερο γύρο επικοινωνίας.

Όσον αφορά τις μετρήσεις που πάρθηκαν για τους Readers ξεχωριστά για το σενάριο αυτό, έχει παρατηρηθεί και σε αυτό το σενάριο το ίδιο φαινόμενο που επεξηγήθηκε πιο πάνω, ότι δηλαδή κάποιες διεργασίες εκτελούσαν υπερβολικά πολλούς δεύτερους γύρους επικοινωνίας σε σχέση με άλλες διεργασίες που εκτελούσαν πολύ λίγους.

Τα ακριβή ποσοστά δεύτερων γύρων επικοινωνίας παρουσιάζονται τόσο στους πίνακες Α.1, Α.3, Α.4 και Α.7 του Παραρτήματος Α, όσο και στα Γραφήματα 4.4 και 4.5 του συγκεκριμένου κεφαλαίου.

Σενάριο με τυχαία χρονικά διαστήματα (*R*=10)				
S=20, t=5, $rInt$=2.3				
A/A	**Συνολικός χρόνος εκτέλεσης (100 λειτουργίες)**		**Μέσος χρόνος εκτέλεσης (100 λειτουργίες)**	
	(1 γύρος επικοινωνίας)	**(2 γύροι επικοινωνίας)**	**(1 γύρος επικοινωνίας)**	**(2 γύροι επικοινωνίας)**
1	14,33	2,19	0,15	0,36
2	12,93	4,11	0,15	0,27
3	17,75	1,22	0,18	0,30
4	17,71	3,80	0,20	0,29
5	18,83	2,95	0,21	0,33
6	10,74	4,09	0,13	0,26
7	13,08	0,57	0,13	0,28
8	14,37	3	0,16	0,29
9	15,10	2,31	0,16	0,29
10	13,19	0	0,13	0
		Συνολικός μέσος χρόνος εκτέλεσης	0,16	0,30
	Συνολικός χρόνος Εκτέλεσης (80 λειτουργίες)	**Μέσος χρόνος Εκτέλεσης (80 λειτουργίες)**		
Writer	13,53	0,17		

Πίνακας 4.6 Χρόνοι εκτέλεσης για σενάριο τυχαίων χρονικών διαστημάτων

Σενάριο με τυχαία χρονικά διαστήματα (*R*=10)				
S=20, t=2, $rInt$=6.3				
A/A	**Συνολικός χρόνος εκτέλεσης (100 λειτουργίες)**		**Μέσος χρόνος εκτέλεσης (100 λειτουργίες)**	
	(1 γύρος επικοινωνίας)	**(2 γύροι επικοινωνίας)**	**(1 γύρος επικοινωνίας)**	**(2 γύροι επικοινωνίας)**
1	17.61	0	0.18	0
2	17.23	0	0.17	0
3	23.12	0	0.23	0
4	24.90	0	0.25	0
5	12.45	9.30	0.17	0.33
6	14.70	0.60	0.15	0.30
7	15.74	1.04	0.16	0.27
8	18.47	1.39	0.19	0.35
9	18,27	1.55	0.19	0.31
10	14.11	0.26	0.14	0.26
		Συνολικός μέσος χρόνος εκτέλεσης	0,18	0,30
	Συνολικός χρόνος Εκτέλεσης (160 λειτουργίες)	**Μέσος χρόνος Εκτέλεσης (160 λειτουργίες)**		
Writer	27,63	0,17		

Πίνακας 4.7 Χρόνοι εκτέλεσης για σενάριο τυχαίων χρονικών διαστημάτων

Στους Πίνακες 4.6 και 4.7 που παρουσιάζονται πιο πάνω, παρουσιάζονται τα αποτελέσματα των χρόνων εκτέλεσης των σεναρίων. Τα αποτελέσματα που πήραμε είναι αναμενόμενα, καθώς παρατηρούμε ότι και για τα δύο σενάρια που τρέξαμε, ο μέσος χρόνος εκτέλεσης των λειτουργιών ανάγνωσης που εκτέλεσαν δεύτερο γύρο επικοινωνίας είναι περίπου διπλάσιος για κάθε διεργασία Reader. Ακόμα, παρατηρούμε ότι οι λειτουργίες γραφής είναι σαφώς γρήγορες, καθώς εκτελούνται στον ίδιο περίπου χρόνο με τις λειτουργίες ανάγνωσης που έκαναν ένα γύρο επικοινωνίας. Επίσης, όπου παρατηρείται μηδενική τιμή στους Πίνακες 4.6 και 4.7 εννοείται ότι η συγκεκριμένη διεργασία ανάγνωσης δεν έχει εκτελέσει δεύτερο γύρο επικοινωνίας, έτσι δεν μπορεί να συμπεριληφθεί στην εύρεση του Συνολικού Μέσου Χρόνου Εκτέλεσης των λειτουργιών που εκτέλεσαν δεύτερο γύρο επικοινωνίας.

Το συνολικό ποσοστό των λειτουργιών ανάγνωσης που εκτελούν δεύτερο γύρο επικοινωνίας για όλα τα σενάρια, όπως υπολογίζεται από τους Πίνακες Α.1 και Α.2 του Παραρτήματος Α, φτάνει περίπου το 15% (14.97%). Συγκριτικά με τα αποτελέσματα της προσομοίωσης που έγινε στο άρθρο [2] στο προσομοιωτή δικτύων *ns-2*, όπου στη χειρότερη περίπτωση το αντίστοιχο ποσοστό των λειτουργιών ανάγνωσης που εκτελούν δεύτερο γύρο επικοινωνίας είναι γύρω στο 7.5%, τα αποτελέσματα της υλοποίησης μας δείχνουν ότι ο αλγόριθμος ανταποκρίνεται ικανοποιητικά σε ρεαλιστικό περιβάλλον. Το γεγονός ότι ο χρόνος εκτέλεσης των λειτουργιών γραφής είναι περίπου ο ίδιος με τις λειτουργίες ανάγνωσης που εκτελούν ένα γύρο επικοινωνίας επαληθεύει ότι οι λειτουργίες ανάγνωσης είναι γρήγορες, και εφόσον οι λειτουργίες ανάγνωσης εκτελούν το πολύ δύο γύρους επικοινωνίας η υλοποίηση μας είναι ημιγρήγορη.

Κεφάλαιο 5

Συμπεράσματα

5.1 Επίλογος

Μέσα από αυτή τη διπλωματική εργασία μπορέσαμε να υλοποιήσουμε και να αξιολογήσουμε έναν αλγόριθμο Ατομικών Αντικειμένων Γραφής-Ανάγνωσης σε ένα παγκόσμιο κατανεμημένο σύστημα υπολογιστών, το *Planetlab*. Έτσι, μπορέσαμε να εξάγουμε χρήσιμα συμπεράσματα για τον αλγόριθμο ως προς την αξιοπιστία, την ευρωστία και την ταχύτητα εκτέλεσης του.

Όπως έχουμε παρατηρήσει στα αποτελέσματα του προηγούμενου κεφαλαίου, η υλοποίηση που κάναμε έχει μειωμένη απόδοση σε σχέση με τα αποτελέσματα της προσομοίωσης που έγιναν στο προσομοιωτή δικτύου *ns-2* και παρουσιάζονται στο άρθρο [2]. Αυτή η μείωση στην απόδοση οφείλεται στις διαφορές που έχει ένα ρεαλιστικό δίκτυο όπως το *Planetlab* με ένα προσομοιωμένο δίκτυο όπως είναι το *ns-2*. Η συμφόρηση που μπορεί να παρατηρηθεί στο *Planetlab* λόγω του ότι είναι κτισμένο πάνω από το διαδίκτυο, η απώλεια μηνυμάτων, η ετεροχρονισμένη παράδοση των μηνυμάτων επηρέασαν τα αποτελέσματα μας, τόσο στο ποσοστό λειτουργιών που εκτελούν δεύτερο γύρο επικοινωνίας όσο και σε χρόνο εκτέλεσης. Παρόλα ταύτα, το ποσοστό των λειτουργιών που εκτελούν δεύτερο γύρο επικοινωνίας σε ελάχιστες περιπτώσεις ξεπερνά το 35% (περίπου 1 στις 10 εκτελέσεις), πράγμα που σημαίνει ότι στις πλείστες εκτελέσεις της υλοποίησης μας τα αποτελέσματα μας είναι ικανοποιητικά.

5.2 Προβλήματα που αντιμετωπίστηκαν

Μια από τις δυσκολίες που αντιμετωπίστηκαν, ήταν η εκμάθηση προγραμματισμού υποδοχών ροής πράγμα που έκανα για πρώτη φορά κατά τη διάρκεια του πτυχίου. Αυτή η μεθοδολογία προγραμματισμού απαιτεί έναν διαφορετικό τρόπο σκέψης και αντίληψης ως προς την εφαρμογή και τη διασύνδεση των διεργασιών μεταξύ τους. Αυτό είχε ως αποτέλεσμα να χρειαστώ περισσότερο χρόνο να ολοκληρώσω την υλοποίηση της εφαρμογής.

Συν τοις άλλοις, οι μη επαρκείς πληροφορίες που παίρνει κάποιος που χρησιμοποιεί τις μηχανές του *Planetlab* συνέβαλαν στην καθυστέρηση της υλοποίησης και των πειραματικών αποτελεσμάτων. Πιο συγκεκριμένα, υπάρχουν διάφοροι λόγοι για τους οποίους δεν μπορεί να ενωθεί κάποιος σε μια μηχανή του *Planetlab*, όπως η μη μετάδοση του δημόσιου κλειδιού του στη μηχανή που προσπαθεί να ενωθεί, η μηχανή να μην είναι διαθέσιμη εκείνη την χρονική περίοδο ή ακόμα και να μην του επιτρέπεται να ενωθεί στην μηχανή για λόγους ασφαλείας από τον *PI* της μηχανής. Ένας έμπειρος χρήστης του κατανεμημένου συστήματος πιθανόν να γνωρίζει αυτά τα προβλήματα μέσα από την εμπειρία και την αλληλεπίδραση του με το σύστημα, αλλά ένας άπειρος χρήστης ή κάποιος που χρησιμοποιεί το *Planetlab* για πρώτη φορά δεν παίρνει σαφή πληροφόρηση από τις μηχανές ή από το ίδιο το σύστημα για αυτά τα προβλήματα με αποτέλεσμα να δυσκολευτεί μέχρι να εξοικειωθεί το τρόπο λειτουργίας της πλατφόρμας.

Ένα πολύ σημαντικό και μη εξαιρετέο πρόβλημα που έπρεπε να αντιμετωπιστεί ήταν οι περιορισμοί των μηχανών του *Planetlab*, καθώς κάθε "μέρισμα" έχει περιορισμένο μέγεθος μνήμης που μπορεί να χρησιμοποιήσει σε κάθε μηχανή. Λόγω της φύσης της εφαρμογής μας, είχαμε να εκτελέσουμε σε κάθε μηχανή του *Planetlab* τουλάχιστον τέσσερις διεργασίες καθώς δεν είχαμε στη διάθεση μας όσες μηχανές απαιτούν τα πειράματα που θέλαμε να τρέξουμε, είτε επειδή οι πόροι των μηχανών ήταν δεσμευμένοι για άλλες εφαρμογές, είτε επειδή απλά δεν είχαμε πρόσβαση. Σαν αποτέλεσμα αυτού, δεν μπορέσαμε να τρέξουμε πειράματα για πάνω από τριάντα περίπου *Readers*, διότι οι πόροι που απαιτούσαν ήταν πολύ περισσότεροι από τους διαθέσιμους και ειδικότερα όταν οι διεργασίες ήταν διεργασίες *Server*. Οι διεργασίες

Server κάθε φορά που δέχονταν νέα αίτηση (*connection request*), δημιουργούσαν ένα νέο νήμα, έτσι στην περίπτωση που είχαμε είκοσι *Readers*, στις μηχανές που έτρεχαν οι διεργασίες *Server* έτρεχαν ουσιαστικά ογδόντα εικονικές διαφορετικές διεργασίες *Server*. Οι πόροι που δεσμεύονταν από αυτές τις διεργασίες δημιούργησαν αρκετά προβλήματα στο δίκτυο του *Planetlab*, πρόβλημα το οποίο ανάγκασε σε μερικές περιπτώσεις τους διαχειριστές του να τερματίσουν τις διεργασίες μας. Οι διαχειριστές του *Planetlab* διατηρούν κάθε δικαίωμα να επέμβουν στην περίπτωση που κάποια εφαρμογή προκαλεί κάποιο πρόβλημα στο δίκτυο ή στους υπόλοιπους χρήστες του. Το γεγονός αυτό δυσκόλεψε περισσότερο τη διεξαγωγή των πειραμάτων και τη συγκομιδή αποτελεσμάτων. Η επέμβαση αυτή γίνεται όταν κριθεί αναγκαία, για να διατηρηθεί η ευρωστία του δικτύου και να εξασφαλιστεί η ισόνομη κατανομή των πόρων του *Planetlab*. Σαν πρώτο βήμα, οι διαχειριστές ενημερώνουν τον χρήστη ότι έχει χρησιμοποιήσει υπερβολικούς πόρους από τις μηχανές που έχει στη διάθεση του, στέλνοντας του ένα ηλεκτρονικό μήνυμα (*email*).

```
pl_mom reset slice cyprus_SF on planetlab1.sics.se

From: PlanetLab Support (support@planet-lab.org)

Sent: Monday, May 12, 2008 12:53:33 AM
Reply-to: PlanetLab Support (support@planet-lab.org)

To: pl-mom@planet-lab.org; cyprus_SF@slices.planet-lab.org

Sometime before Sun May 11 20:44:56 2008 GMT, swap space was nearly exhausted
on planetlab1.sics.se.

Slice cyprus_SF was reset since it was the largest consumer of physical memory at
524.9 MB (51.9%) (325.9 MB writable).

Please reply to this message explaining the nature of your experiment, and what you are
doing to address the problem.

http://summer.cs.princeton.edu/status/tabulator.cgi?table=slices/table_cyprus_SF

cyprus_SF processes prior to reset:

  PID     VIRT      SZ      RES %CPU %MEM COMMAND

 5801   7.5 MB    1.9 MB    2.4 MB  0.0  0.2 sshd: cyprus_SF@pts/4
 18809   6.8 MB    1.7 MB    2.0 MB  0.0  0.1 sshd: cyprus_SF [priv]
 10306   4.5 MB    1.1 MB    1.9 MB  0.0  0.1 ./server 1600 30
  5011   6.8 MB    1.7 MB    1.8 MB  0.0  0.1 sshd: cyprus_SF [priv]
 10076   4.5 MB    1.1 MB    1.8 MB  0.0  0.1 ./server 1600 30
 11453   4.6 MB    1.1 MB    1.8 MB  0.0  0.1 ./server 1700 30
 10299   4.5 MB    1.1 MB    1.8 MB  0.0  0.1 ./server 1500 30
 ..........................................................................
 7430   0 bytes   0 bytes   12.0 KB  0.0  0.0 [server] <defunct>
 7487   0 bytes   0 bytes   12.0 KB  0.0  0.0 [server] <defunct>

 Sun May 11 20:44:56 2008 GMT planetlab1.sics.se reset cyprus_SF
```

Σχήμα 5.1 Προειδοποιητικό ηλεκτρονικό μήνυμα *Planetlab*

Η μορφή του μηνύματος βρίσκεται στο Σχήμα 5.1. Τέλος, με την παραλαβή ενός πανομοιότυπου ηλεκτρονικού μηνύματος όπως ακριβώς αυτού στο Σχήμα 5.1, οι διεργασίες του χρήστη τερματίζονται. Το πρόβλημα αυτό δεν αναφέρεται με σκοπό να θίξει την πράξη αυτή, αλλά για να καταγραφούν επακριβώς οι δυσκολίες που αντιμετωπίστηκαν στην εκτέλεση των σεναρίων και να ενημερώσει τυχόν νέους χρήστες του *Planetlab* που θα έχουν σαν έναυσμα την παρούσα Διπλωματική Εργασία, για τις υπάρχουσες δικλίδες ασφαλείας.

5.3 Μελλοντική εργασία

Σε αυτή την Ατομική Διπλωματική Εργασία έχουμε μελετήσει, υλοποιήσει και αξιολογήσει τον Αλγόριθμο Ατομικών Αντικειμένων Γραφής-Ανάγνωσης. Στο άρθρο που βασίστηκε το [2] έχει αποδειχθεί ότι δεν μπορούν να υπάρξουν γρήγορες υλοποιήσεις *SWMR* (*Single Writer Multiple Readers*), όταν ο αριθμός των νοητών αναγνωριστικών V των διεργασιών *Reader* είναι $\frac{S}{t}-2$ ή μεγαλύτερος. Έτσι, έχουμε υλοποιήσει μια ημιγρήγορη εφαρμογή όπου οι λειτουργίες γραφής είναι γρήγορες - εκτελούνται δηλαδή σε ένα γύρο επικοινωνίας- και ένα ποσοστό των λειτουργιών ανάγνωσης χρειάζονται έναν δεύτερο γύρο επικοινωνίας.

Υπάρχουν πολλά περιθώρια βελτίωσης και επέκτασης της υλοποίησης μας, καθώς η διαπροσωπία της εφαρμογής μας δεν είναι πολύ φιλική ως προς τη αλληλεπίδραση με το χρήστη και ο κώδικας μπορεί να δεχθεί σημαντική βελτίωση ως προς τον τρόπο συγγραφής του. Επίσης, μπορούν να βρεθούν και άλλοι παράμετροι που μπορούν να αλλάξουν για να αξιολογηθεί ο αλγόριθμος εκτός από αυτές που αναφέραμε, όπως π.χ. ο αριθμός των *Servers* που χρησιμοποιούνται σε κάθε σενάριο, έχοντας έτσι διαφορετικά αποτελέσματα και άλλου είδους συμπεράσματα.

Τέλος, θα μπορούσαν να υλοποιηθούν βοηθητικά προγράμματα (*scripts*) τα οποία θα είναι υπεύθυνα να ενώνονται πάνω στις μηχανές του *Planetlab* ή στις μηχανές του Πανεπιστημίου Κύπρου και να ξεκινούν τις διάφορες διεργασίες που αποτελούν την υλοποίηση του αλγόριθμου, καθώς είναι μια χρονοβόρα διαδικασία η οποία μπορεί να αποτρέψει κάποιον από την χρήση της εφαρμογής μας. Όσον αφορά την χρήση του

Planetlab, δεν περιορίζει τη χρήση του μόνο σε εφαρμογές που χρησιμοποιούν το μοντέλο εξυπηρετητή-πελάτη όπως είναι αυτή που υλοποιήθηκε. Υπάρχει μια πληθώρα εφαρμογών που μπορούν να αναπτυχθούν σε αυτό και το συγκεκριμένο έγγραφο δίνει αρκετές πληροφορίες και βοήθειες για κάποιον που επιθυμεί να δημιουργήσει την δική του εφαρμογή στο *Planetlab*.

Βιβλιογραφία

[1] H. Attiya, A. Bar-Noy, and D. Dolev, "*Sharing memory robustly in message passing systems*", J. of the ACM, 42(1):124–142, 1996.

[2] Chryssis Georgiou, Nicolas C. Nicolaou, and Alex A. Shvartsman, "*Fault-Tolerant SemiFast Implementations of Atomic Read/Write Registers*", in Proc. of the 18th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2006), pp 281-290, Cambridge, MA, 2006.

[3] P. Dutta, R. Guerraoui, R. R. Levy, and A. Chakraborty, "*How fast can a distributed atomic read be?*", In Proceedings of the 23rd ACM symposium on Principles of distributed computing, pages 236–245, 2004.

[4] Jeri K. Hanly and Elliot B. Koffman, "*Problem solving and program design in C*", Addison Wesley, 2004.

[5] Joseph Jaja, "*An introduction to parallel algorithms*", Addison Wesley Publishing Company, Maryland, 1992.

[6] James F. Kurose and Keith W. Ross, "*Computer Networking, A top-down approach featuring the Internet*", Third Edition, Addison Wesley, 2005.

[7] Jon Lasser, "*Think Unix*", Que Publishings, Maryland, 2000.

[8] Andrew S. Tanenbaum and Maarten van Steen, "*Distributed Systems: Principles and Paradigms*", Second Edition, Addison Wesley, 2006.

[9] Σημειώσεις μαθήματος "*ΕΠΛ 431 – Σύνθεση Παράλληλων Αλγορίθμων*", Εαρινό Εξάμηνο, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου, 2007.

[10] Fedora Operating System, http://www.fedora.org.

[11] Planetlab, http://www.planet-lab.org.

[12] Planetlab user manual, http://www.planet-lab.org/doc/guides.

Παράρτημα Α - Πίνακες Αποτελεσμάτων

Στο παράρτημα αυτό παρουσιάζονται τα αποτελέσματα των σεναρίων που εκτέλεσε η εφαρμογή μας. Πιο συγκεκριμένα, παρουσιάζονται πίνακες με τα συγκεντρωτικά ποσοστιαία αποτελέσματα για κάθε σενάριο αλλά και ξεχωριστά για κάθε *Reader*.

Συγκεντρωτικός Πίνακας Ποσοστιαίων Αποτελεσμάτων Σεναρίου με τυχαία χρονικά διαστήματα															
S=20															
	rInt = 2.3					**rInt = 4.3**					**rInt = 6.3**				
	t=1	**t=2**	**t=3**	**t=4**	**t=5**	**t=1**	**t=2**	**t=3**	**t=4**	**t=5**	**t=1**	**t=2**	**t=3**	**t=4**	**t=5**
R=10	33.5	44.7	19	14.7	4	4.1	19.1	36.8	19.5	5.3	5.5	12.9	32.8	52.3	2.43
R=20	16.3	10.4	22.4	54.2	8.39	4.9	30.05	17.65	35.1	7.9	10.05	11.25	39.3	43.1	2.85
R=30	-	-	-	-	0.93	-	-	-	-	11	-	-	-	-	7.27

Πίνακας Α.1 Συγκεντρωτικά Ποσοστιαία Αποτελέσματα σεναρίου τυχαίων χρονικών διαστημάτων

Συγκεντρωτικός Πίνακας Ποσοστιαίων Αποτελεσμάτων Σεναρίου με σταθερά χρονικά διαστήματα															
S=20															
	rInt = 2.3					**rInt = 4.3**					**rInt = 6.3**				
	t=1	**t=2**	**t=3**	**t=4**	**t=5**	**t=1**	**t=2**	**t=3**	**t=4**	**t=5**	**t=1**	**t=2**	**t=3**	**t=4**	**t=5**
R=10	23.5	25.9	7.8	20.1	6.1	0.5	0.8	22.5	32.4	1.4	0.3	2	19	14	2
R=20	3.85	24.4	18.1	20.25	3.15	5.11	17.89	33.5	19.55	1.15	0.45	2.35	3.25	7.05	0.55
R=30	-	-	-	-	7	-	-	-	-	1.03	-	-	-	-	1.7

Πίνακας Α.2 Συγκεντρωτικά Ποσοστιαία Αποτελέσματα σεναρίου σταθερών χρονικών διαστημάτων

Πίνακας Αποτελεσμάτων ξεχωριστά για κάθε *Reader* (Σενάριο με τυχαία χρονικά διαστήματα, R=10)															
S=20															
	rInt = 2.3					**rInt = 4.3**					**rInt = 6.3**				
A/A	**t=1**	**t=2**	**t=3**	**t=4**	**t=5**	**t=1**	**t=2**	**t=3**	**t=4**	**t=5**	**t=1**	**t=2**	**t=3**	**t=4**	**t=5**
1	21	34	11	35	3	2	1	65	36	4	1	17	25	51	1
2	18	2	33	0	4	0	50	52	48	3	0	3	59	73	2
3	20	43	1	57	5	2	55	81	0	4	0	1	54	71	D
4	30	86	19	81	4	1	58	65	3	8	7	0	68	0	D
5	0	82	1	0	4	0	2	38	32	7	2	16	17	51	3
6	40	28	11	63	3	19	3	51	41	5	1	16	47	0	5
7	72	1	11	37	3	2	1	9	35	5	0	0	0	82	0
8	65	83	56	0	4	0	2	2	0	9	0	20	2	80	3
9	50	88	42	50	6	5	12	0	0	12	0	55	56	80	3
10	19	0	5	84	4	0	7	5	0	1	44	1	0	27	D
%	**33.5**	**44.7**	**19**	**14.7**	**4**	**4.1**	**19.1**	**36.8**	**19.5**	**5.3**	**5.5**	**12.9**	**32.8**	**52.3**	**2.43**

Πίνακας Α.3 Αποτελέσματα σεναρίου τυχαίων χρονικών διαστημάτων ξεχωριστά για κάθε *Reader*

Πίνακας Αποτελεσμάτων ξεχωριστά για κάθε *Reader* (Σενάριο με τυχαία χρονικά διαστήματα, R=20)															
S=20															
	rInt = 2.3					**rInt = 4.3**					**rInt = 6.3**				
A/A	**t=1**	**t=2**	**t=3**	**t=4**	**t=5**	**t=1**	**t=2**	**t=3**	**t=4**	**t=5**	**t=1**	**t=2**	**t=3**	**t=4**	**t=5**
1	36	0	0	0	8	2	5	0	51	4	32	4	63	37	7
2	0	25	58	69	5	1	5	43	72	6	2	9	19	0	0
3	2	84	4	62	5	1	69	46	32	8	0	7	12	84	0
4	47	0	0	90	9	1	61	0	10	13	3	5	24	73	0
5	1	3	64	0	8	0	81	6	29	3	1	50	54	0	0
6	1	1	1	76	12	0	51	0	0	2	2	2	63	0	1
7	3	76	69	73	13	0	7	0	10	17	15	1	74	77	1
8	58	0	4	44	9	0	12	4	81	0	6	2	8	82	0
9	0	2	0	83	5	0	81	0	72	9	4	4	76	13	1
10	0	0	25	45	15	0	53	19	82	17	14	1	81	82	0
11	3	4	1	3	D	2	3	12	77	3	2	17	7	60	10
12	0	0	11	12	8	19	4	43	0	10	0	39	24	36	0
13	46	8	66	61	13	0	70	0	0	6	30	1	55	0	12
14	76	0	0	95	6	0	1	7	80	5	15	4	44	0	4
15	0	0	77	78	11	63	59	53	20	14	28	2	7	88	3
16	48	0	32	90	3	3	1	51	0	9	37	6	51	82	5
17	0	1	1	54	6	0	12	2	0	8	3	3	58	23	5
18	5	0	1	74	6	4	25	57	86	7	0	5	47	24	5
19	0	2	0	75	5	2	1	6	0	1	3	61	2	57	0
20	0	2	30	0	4	0	0	4	0	16	1	2	17	44	3
%	**16.3**	**10.4**	**22.4**	**54.2**	**8.39**	**4.9**	**30.05**	**17.65**	**35.1**	**7.9**	**10.05**	**11.25**	**39.3**	**43.1**	**2.85**

Πίνακας Α.4 Αποτελέσματα σεναρίου τυχαίων χρονικών διαστημάτων ξεχωριστά για κάθε *Reader*

Πίνακας Αποτελεσμάτων ξεχωριστά για κάθε *Reader* (Σενάριο με σταθερά χρονικά διαστήματα, R=10)															
S=20															
	rInt = 2.3					**rInt = 4.3**					**rInt = 6.3**				
A/A	**t=1**	**t=2**	**t=3**	**t=4**	**t=5**	**t=1**	**t=2**	**t=3**	**t=4**	**t=5**	**t=1**	**t=2**	**t=3**	**t=4**	**t=5**
1	9	6	1	14	2	1	0	0	66	0	1	19	0	0	1
2	72	1	0	44	6	1	0	33	0	0	0	0	1	0	1
3	57	4	0	0	9	1	0	48	55	3	0	D	47	0	2
4	0	0	1	0	4	0	1	49	49	2	0	0	52	0	2
5	29	D	14	0	1	1	0	0	0	1	0	1	40	2	1
6	56	80	1	19	14	0	0	39	43	0	0	0	47	0	1
7	7	7	3	0	6	0	2	46	0	2	0	0	1	55	2
8	4	81	57	0	5	0	1	7	50	2	1	0	0	0	2
9	1	71	0	69	13	0	0	3	61	0	0	0	2	48	2
10	0	9	1	55	1	1	4	0	0	4	1	0	0	35	6
%	**23.5**	**25.9**	**7.8**	**20.1**	**6.1**	**0.5**	**0.8**	**22.5**	**32.4**	**1.4**	**0.3**	**2**	**19**	**14**	**2**

Πίνακας A.5 Αποτελέσματα σεναρίου σταθερών χρονικών διαστημάτων ξεχωριστά για κάθε *Reader*

Πίνακας Αποτελεσμάτων ξεχωριστά για κάθε *Reader* (Σενάριο με σταθερά χρονικά διαστήματα, R=20)															
S=20															
	rInt = 2.3					**rInt = 4.3**					**rInt = 6.3**				
A/A	**t=1**	**t=2**	**t=3**	**t=4**	**t=5**	**t=1**	**t=2**	**t=3**	**t=4**	**t=5**	**t=1**	**t=2**	**t=3**	**t=4**	**t=5**
1	0	17	7	0	0	1	D	29	72	0	0	0	1	0	4
2	1	1	1	67	8	2	3	40	19	3	1	2	0	0	0
3	1	45	4	0	0	9	3	36	0	0	1	0	0	7	0
4	0	54	5	0	7	2	28	37	19	0	0	0	1	1	0
5	1	3	43	79	0	0	24	25	0	0	1	1	0	0	0
6	1	54	1	0	0	0	24	25	0	0	1	0	0	34	2
7	1	4	59	9	6	D	57	51	0	4	1	35	49	7	1
8	1	1	2	5	0	3	52	25	56	0	1	0	1	18	1
9	0	1	9	0	3	3	3	43	0	2	0	0	0	0	0
10	0	25	0	0	1	1	22	26	0	1	0	1	0	39	1
11	1	56	1	84	3	1	0	32	0	2	1	0	0	0	0
12	0	1	9	0	0	11	62	27	0	3	0	2	0	0	0
13	0	61	69	0	9	15	8	52	0	3	0	1	0	0	0
14	0	44	1	79	0	11	29	31	0	0	0	0	0	0	0
15	1	1	0	1	14	15	3	0	70	0	1	2	13	0	0
16	1	1	74	21	2	5	2	29	0	1	1	0	0	2	1
17	0	1	17	0	8	2	17	53	0	1	0	3	0	25	0
18	0	56	41	69	2	1	4	49	50	2	0	0	0	1	0
19	0	1	12	0	0	1	1	29	66	1	0	0	0	7	0
20	0	61	17	1	0	9	0	31	D	0	0	0	0	0	1
%	**3.85**	**24.4**	**18.1**	**20.25**	**3.15**	**5.11**	**17.89**	**33.5**	**19.55**	**1.15**	**0.45**	**2.35**	**3.25**	**7.05**	**0.55**

Πίνακας Α.6 Αποτελέσματα σεναρίου σταθερών χρονικών διαστημάτων ξεχωριστά για κάθε *Reader*

Πίνακας Αποτελεσμάτων ξεχωριστά για κάθε *Reader* (Σενάριο με τυχαία χρονικά διαστήματα, R=30)							
S=20, t=5							
A/A	**rInt = 2.3**	**rInt = 4.3**	**rInt = 6.3**	**A/A**	**rInt = 2.3**	**rInt = 4.3**	**rInt = 6.3**
1	0	6	7	**16**	0	23	4
2	0	13	15	**17**	0	23	8
3	0	0	4	**18**	0	18	3
4	14	11	12	**19**	0	2	16
5	0	12	12	**20**	0	10	9
6	0	16	19	**21**	0	19	5
7	14	0	8	**22**	0	0	1
8	0	5	16	**23**	0	18	0
9	0	9	12	**24**	0	22	0
10	0	0	12	**25**	0	3	13
11	0	23	0	**26**	0	7	0
12	0	7	0	**27**	0	22	5
13	0	7	7	**28**	0	11	6
14	0	21	7	**29**	0	1	6
15	0	4	0	**30**	0	17	11

Πίνακας Α.7 Αποτελέσματα σεναρίου τυχαίων χρονικών διαστημάτων ξεχωριστά για κάθε *Reader*

Πίνακας Αποτελεσμάτων ξεχωριστά για κάθε *Reader* (Σενάριο με σταθερά χρονικά διαστήματα, R=30)							
S=20, t=5							
A/A	**rInt = 2.3**	**rInt = 4.3**	**rInt = 6.3**	**A/A**	**rInt = 2.3**	**rInt = 4.3**	**rInt = 6.3**
1	10	0	0	**16**	11	0	4
2	0	2	0	**17**	7	2	0
3	0	1	0	**18**	0	2	0
4	14	0	1	**19**	12	1	5
5	17	1	1	**20**	15	2	1
6	0	0	4	**21**	1	0	0
7	4	2	4	**22**	11	0	4
8	4	2	0	**23**	10	0	4
9	1	2	0	**24**	0	0	4
10	9	2	0	**25**	1	2	0
11	11	2	4	**26**	7	1	4
12	1	1	1	**27**	10	1	0
13	D	2	5	**28**	7	1	1
14	9	2	0	**29**	1	1	0
15	8	0	0	**30**	15	0	4

Πίνακας Α.8 Αποτελέσματα σεναρίου σταθερών χρονικών διαστημάτων ξεχωριστά για κάθε *Reader*

Σημείωση: Οι διεργασίες στις οποίες αναγράφεται το αποτέλεσμα τους ως "*D*", είναι οι διεργασίες οι οποίες κατέρρευσαν κατά την εκτέλεση της εφαρμογής και ως εκ τούτου δεν έχουμε αποτελέσματα για αυτές.