ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
Τ.Ε.Ι. ΔΥΤΙΚΗΣ ΕΛΛΑΔΑΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΚΑΤΑΝΕΜΗΜΕΝΑ ΠΡΩΤΟΚΟΛΛΑ ΔΙΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ

ΑΡΝΑΟΥΤΟΓΛΟΥ ΧΡΙΣΤΙΝΑ Α.Μ. 1160
ΚΟΛΟΒΕΛΩΝΗ ΑΝΑΣΤΑΣΙΑ Α.Μ. 1209

Επιβλέπων καθηγητής : Βασίλειος Ταμπακάς

Αθήνα 2016

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΚΑΤΑΝΕΜΗΜΕΝΑ ΠΡΩΤΟΚΟΛΛΑ ΔΙΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ

ΑΡΝΑΟΥΤΟΓΛΟΥ ΧΡΙΣΤΙΝΑ Α.Μ. 1160
ΚΟΛΟΒΕΛΩΝΗ ΑΝΑΣΤΑΣΙΑ Α.Μ. 1209

Επιβλέπων καθηγητής : Βασίλειος Ταμπακάς

ΠΕΡΙΛΗΨΗ

Σήμερα, σχεδόν όλα τα μεγάλα συστήματα που βασίζονται σε υπολογιστές είναι πλέον κατανεμημένα. Η επεξεργασία των πληροφοριών κατανέμεται σε πολλούς υπολογιστές και δεν περιορίζεται σε μία μόνο μηχανή. Επομένως η τεχνολογία των κατανεμημένων συστημάτων έχει μεγάλη σημασία για τα υπολογιστικά συστήματα των εταιρειών. Το γεγονός αυτό, σε συνδυαμό με την 'περιέργειά' μας να γνωρίσουμε καλύτερα ένα άγνωστο για μας νέο πεδίο μας ώθησε στην επιλογή να επιλέξουμε την πτυχιακή μας εργασία πάνω στα κατανεμημένα συστήματα με θέμα τα "Κατανεμημένα πρωτόκολλα διάδοσης πληροφορίας".

Αντικείμενο της εργασίας είναι η διάδοση πληροφορίας στα κατανεμημένα συστήματα. Η εργασία περιλαμβάνει διερεύνηση και προγραμματισμό συστήματος.

Στόχος της εργασίας είναι η καταγραφή και εξοικείωση με τους γνωστούς αλγορίθμους διάδοσης πληροφορίας (wave algorithms) και την εφαρμογή τους για την επίλυση γνωστών προβλημάτων στα κατανεμημενα συστήματα (δρομολόγηση, αρχικοποίηση, κινητά συστήματα κλπ.). Επιλεγμένα πρωτόκολλα θα προγραμματιστούν σε πραγματικό περιβάλλον και θα χρησιμοποιηθούν σε προσομοιωμένο περιβάλλον.

ΕΥΧΑΡΙΣΤΙΕΣ

Θα θέλαμε να ευχαριστήσουμε τον επιβλέποντα καθηγητή μας κ. Βασίλειο Ταμπακά για την καθοδήγηση και τη βοήθειά του και τις οικογένειές μας που μας στήριξαν στις σπουδές μας.

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΡΟΛΟΓΟΣ

Ο Leslie Lamport δηλώνει για ένα κατανεμημένο σύστημα ότι *"Γνωρίζεις ότι έχεις ένα όταν, ένα σφάλμα ενός ηλεκτρονικού υπολογιστή που ούτε ήξερες ότι υπήρχει, μπορεί να αχρηστέψει τον υπολογιστή σου."*

Ο Andrew S. Tanenbaum σημειώνει ότι *"Τα κατανεμημένα συστήματα απαιτούν ριζικά διαφορετικό λογισμικό από αυτό που χρησιμοποιούν τα κεντρικοποιημένα συστήματα."*

Ο Γεώργιος Κουλούρης δίνει τον ορισμό *"Τα κατανεμημένα συστήματα είναι μια συλλογή από αυτόνομους υπολογιστές που συνδέονται μεταξύ τους μέσω ενός δικτύου, και χρησιμοποιούν ειδικά σχεδιασμένο λογισμικό για την παροχή ενοποιημένων υπολογιστικών υπηρεσιών."* και σημειώνει ότι *"Σε ένα τέτοιο σύστημα, οι διεργασίες που εκτελούνται από τους δικτυωμένους υπολογιστές επικοινωνούν μεταξύ τους και συντονίζουν τις κινήσεις τους μόνο μέσω της ανταλλαγής μηνυμάτων."*

Οι Burns και Willings σημειώνουν ότι πρόκειται περί *"...ενός πληροφοριακού συστήματος πολλαπλών αυτόνομων υπολογιστικών στοιχείων, τα οποία συνεργάζονται για την επίτευξη ενός κοινού στόχου..."*

Μια σύντομη περιγραφή:

Το βιβλίο αποτελείται από 4 κεφάλαια

- Στο πρώτο (1) κάνουμε μία εισαγωγή στα κατανεμημένα συστήματα. Αναλύουμε τις σχεδιαστικές προκλήσεις και τα αρχιτεκτονικά μοντέλα και αναφερόμαστε τις συνηθέστερες τοπολογίες.

- Στο δεύτερο (2) αναλύουμε τα προβλήματα των κατανεμημένων συστημάτων, όπως τη μετάδοση πληροφορίας και διάσχιση δικτύου, το πρόβλημα συγχρονισμού ρολογιών, το πρόβλημα εκλογής αρχηγού και τη δρομολόγηση. Επίσης γίνεται αναφορά στα πρωτόκολλα που χρησιμοποιούνται για την επίλυση των σχετικών προβλημάτων.

- Στο τρίτο (3) υλοποιούμε σε java κάποιους από τους αλγορίθμους που αναλύονται στο δεύτερο κεφάλαιο, τον αλγόριθμο εκλογής αρχηγού σε τοπολογία δακτυλίου, τον αλγόριθμο του LeLann και τον ECHO.

- Στο τέταρτο (4) κάνουμε μία αναφορά στην οργάνωση και ανάπτυξη κώδικα, στα ολοκληρωμένα περιβάλλοντα ανάπτυξης λογισμικού όπως το NetBeans IDE και σε άλλα θέματα που σχετίζονται με την ανάπτυξη κώδικα.

ΚΕΦΑΛΑΙΟ 1

Εισαγωγή στα Κατανεμημένα Συστήματα

1.1 Γενικά

Τα περισσότερα υπολογιστικά συστήματα σήμερα είναι κατανεμημένα (τραπεζικά δίκτυα, internet, mobile and wireless systems).

Ένα κατανεμημένο σύστημα αποτελείται από γεωγραφικά ανεξάρτητες, αυτόνομες υπολογιστικές συσκευές, που επικοινωνούν μεταξύ τους και λειτουργούν συντονισμένα για την επίτευξη ενός κοινού στόχου.

Πιο αναλυτικά τα κατανεμημένα συστήματα απαρτίζονται από πολλούς απομακρυσμένους υπολογιστές (κόμβους) μεταξύ των οποίων υπάρχει επικοινωνία με μηνύματα μέσω ενός δικτύου. Στην επικοινωνία μπορεί να προκύψει κάποιο σφάλμα κατά την αποστολή των μηνυμάτων ενώ η σειρά παράδοσης μπορεί να διαφέρει απο τη σειρά αποστολής.

1.1.1 Λειτουργικά συστήματα

Το λειτουργικό σύστημα είναι ένα ολοκληρωμένο σύνολο προγραμμάτων που χρησιμοποιείται για την διαχείριση των πόρων (ΚΜΕ, μνήμη, δίσκοι, συσκευές εισόδου-εξόδου) του υπολογιστη καθώς επίσης και για τη δημιουργία ενός περιβάλλοντος επικοινωνίας του χρήστη με τον υπολογιστή.

Τα νεότερα λειτουργικά συστήματα είναι τα εξής:

1. Τα λειτουργικά συστήματα πολυεπεξεργαστών τα οποία χρησιμοποιούνται στα πολυεπεξεργαστικά συστήματα. Οι χρήστες βλέπουν ένα μόνο λειτουργικό σύστημα στο οποίο υπάρχει ένας μόνο εικονικός χώρος διευθύνσεων και ένα κοινό ρολόι (κοινή ώρα) και διατηρείται ουρά έτοιμων διεργασιών. Συνήθως αφιερώνεται ένας επεξεργαστής για την εκτέλεση του πυρήνα του ΛΣΠ. Επίσης γίνεται χρήση κρυφών μνημών από κάθε επεξεργαστή με συνεκτικές και μη συνεκτικές κρυφές μνήμες. Μια διεργασία δρομολογείται σε έναν επεξεργαστή με τον πιο αποδοτικό τρόπο (π.χ. σε ποια κρυφή μνήμη με επεξεργαστή είναι αποθηκευμένα πρόσφατα δεδομένα της).

2. Τα λειτουργικά συστήματα δικτύων χρησιμοποιούνται στα πολύ-υπολογιστικά συστήματα (ομογενή ή ετερογενή) και επιτρέπουν τη χρήση πόρων των υπόλοιπων υπολογιστών χωρίς να κρύβουν πως δεν είναι τοπικοί πόροι (μη διαφανής λειτουργία). Είναι το ίδιο με τα λειτουργικά συστήματα του κάθε υπολογιστή με μικρές προσθήκες που υλοποιούν τις δικτυακές υπηρεσίες. Ωστόσο δεν προσφέρεται η εικόνα ενός μόνο ολοκληρωμένου συστήματος και η συχνή υπηρεσία των λειτουργικών συστημάτων δικτύων είναι το καταμεριζόμενο σύστημα αρχείων δικτύου.

3. Τα κατανεμημένα λειτουργικά συστήματα χρησιμοποιούνται σε ομογενή συστήματα πολύ υπολογιστών και προσφέρουν την εικόνα ενός μόνο συστήματος με κοινή μνήμη και κοινό ρολόι. Η λειτουργία τους δε είναι διαφανής, ο χρήστης δεν γνωρίζει ποιον επεξεργαστή χρησιμοποιεί ή που είναι αποθηκευμένα τα αρχεία του. Είναι σύνθετα συστήματα που επιλύουν με διαφανή τρόπο ένα μεγάλο αριθμό προβλημάτων (π.χ. δημιουργία κοινού χρόνου /συγχρονισμό ρολογιών, κατανεμημένο σύστημα αρχείων,

κατανεμημένη δρομολόγηση διεργασιών και μηνυμάτων, αμοιβαίος αποκλεισμός, θέματα ασφάλειας και συνέπειας, συνεπή ονομασία κ.λ.π.). Απαιτούν τη χρήση πανομοιότυπων πυρήνων για κάθε επεξεργαστή και στενό συγχρονισμό τους. Ο κάθε πυρήνας ελέγχει το δικό του περιβάλλον.

1.1.2 Λειτουργικά συστήματα δικτύων - κατανεμημένα λειτουργικα συστήματα

Μέχρι τις αρχές της προηγούμενης δεκαετίας τα λειτουργικα συστηματα δικτύων παρουσίαζαν πολύ μεγαλύτερη χρήση/ανάπτυξη από τα κατανεμημένα λειτουργικά συστήματα . Αυτό οφειλόταν σε μια σειρά πρακτικών προβλημάτων των λειτουργικων συστηματων δικτύων όπως η μεγάλη πολυπλοκότητά τους, η ανάγκη ομοιογένειας και των όμοιων τοπικών λειτουργικων συστημάτων που θέτει φραγμό στην χρήση των και η ανάγκη της στενής συνεργασίας στα κατανεμημένα λειτουργικά συστήματα που μεταφράζεται σε μερική απώλεια του ελέγχου στα τοπικά συστήματα και θέτει θέματα ασφάλειας αλλά και τοπικής απόδοσης.

Η απάντηση στα προηγούμενα είναι η δημιουργία κατανεμημένων συστημάτων με περιορισμένο βαθμό κατανομής τα οποία μπορούν να υλοποιηθούν με ένα απλούστερο ενδιάμεσο λογισμικό (middleware). Τα κατανεμημένα συστήματα εκτελούνται πάνω από τα τοπικά λειτουργικα συστηματα δικτύων υποστηρίζοντας συγκεκριμένες κατανεμημένες εφαρμογές. Δεν απαιτούν απώλεια της τοπικής αυτοδυναμίας του κάθε υπολογιστή ούτε απόλυτη ομοιογένεια στα υποσυστήματα και στο λογισμικό τους.

Τα συστήματα περιορισμένης κατανομής δημιουργούν ένα περιορισμένο μοντέλο κατανεμημένης επεξεργασίας κατάλληλο για τις κατανεμημένες εφαρμογές (π.χ. κατανεμημένο σύστημα αρχείων) που θέλουν να υποστηρίξουν. Για παράδειγμα το μοντέλο αυτό μπορεί να υποστηρίζει υπηρεσίες όπως είναι οι κλήσεις απομακρυσμένων διαδικασιών (remote procedure call), οι κλήσεις απομακρυσμένων μεθόδων, οι υπηρεσίες κατανεμημένης ονομασίας τέλος οι υπηρεσίες ασφάλειας.

1.1.3 Σύγκριση με παράλληλα συστήματα

Ενα παράλληλο σύστημα είναι ένα ειδικά σχεδιασμένο σύστημα που περιέχει πολλαπλους εσωτερικούς επεξεργαστές ή περιέχει πολλαπλούς υπολογιστές που είναι στενά διασυνδεμένοι μεταξύ τους για την λύση ενός μεγάλου προβλήματος. Η προσέγιση αυτή αυξάνει την απόδοση απο τον ακολουθιακό υπολογιστή και η ιδέα είναι οτι η ταυτόχρονη λειτουργία των n υπολογιστών έχει σαν αποτέλεσμα να είναι n φορές ταχύτερος. Αυτό σπάνια συμβαίνει στην πράξη για διάφορους λόγους.

Σήμερα με την ευρεία ανάπτυξη του Διαδικτύου (Internet) και του Παγκόσμιου Ιστού (World Wide Web), η διαθεσιμότητα των συστημάτων υπολογιστών είναι κρίσιμη για ορισμένες εφαρμογές όπως το ηλεκτρονικό εμπόριο (e-commerce). Έτσι για παράδειγμα οι διακομιστές ιστού (Web Servers) που χρησιμοποιούνται για να φιλοξενήσουν πολλές υπηρεσίες διαδικτύου όχι μόνο πρέπει να είναι σε θέση να αντιμετωπίζουν τις εκατονταδες αιτήσεις χρηστών αποτελεσματικά αλλά και πρέπει να λειτουργούν αδιάλειπτα σε 24ωρη βάση.

Από την άλλη μεριά υπάρχουν εφαρμογές που δεν απαιτούν μόνο την επιτάχυνση των υπολογισμών καιαπρόσκοπη λειτουργία των συστημάτων υπολογιστών ή διακομιστών αλλά

πρόσβαση σε απομακρυσμένα συστήματα υπολογιστών με σκοπό την ανταλλαγή δεδομένων και υπηρεσιών. Τέτοιες εφαρμογές είναι η πρόσβαση ιστοσελίδων σε διακομιστές Ιστού που υπάρχουν στο διαδίκτυο, η παροχή υπηρεσιών ηλεκτρονικού εμπορίου στους πελάτες για διάφορα προιόντα ή πρόσβαση σε συστήματα βάσεων δεδομένων για τραπεζικές συναλλαγές και κρατήσεις αεροπορικών εισιτηρίων. Οι εφαρμογές αυτές δεν απαιτούν απαιτητικούς υπολογισμούς αλλά επικεντρώνονται στη διευκόλυνση της επικοινωνίας μεταξύ απομακρυσμένων υπολογιστών για την διάφανη ανταλλαγή και ακεραιότητα των δεδομένων.

Ένας τρόπος για να επιτύχουμε την επικοινωνία μεταξύ των απομακρυσμένων υπολογιστών είναι η αύξηση της ταχύτητας του δικτύου επικοινωνίας που συνδέονται οι υπολογιστες. Γι' αυτό μια καλή λύση είναι να συνδέσουμε τους απομακρυσμένους και ανεξάρτητους υπολογιστές με ένα υπερ-ταχύ δίκτυο επικοινωνίας. Αυτό έχει σαν αποτέλεσμα τη δημιουργία συστημάτων που είναι γνωστά ως κατανεμημένα συστήματα. Συνεπώς ένα κατανεμημένο σύστημα είναι ένα συστημα που αποτελείται απο ανεξάρτητους πόρους που είναι χαλαρά διασυνδεμένοι μεταξύ τους μέσω διαδικτύου για την ταχεία ανταλλαγή υπηρεσιών στις διάφορες εφαρμογές.

1.2 Σχεδιαστικές προκλήσεις

Οι σχεδιαστές των κατανεμημένων συστημάτων, ανεξαρτήτως του βαθμού κατανομής ενός τέτοιου συστήματος έρχονται αντιμέτωποι με μία σειρά γενικών και ειδικών απαιτήσεων που σχετίζονται με την κατανεμημένη φύση των απαιτήσεων αυτών. Παρακάτω θα αναλύσουμε τις ειδικές απαιτήσεις ενός συστήματος.

Γενικές απαιτήσεις:

- Ευελιξία (flexibility) – Η δυνατότητα επέκτασης του συστήματος.
- Ανοιχτή υλοποίηση (openness) - Συνεργασία με διαφορετικούς κατασκευαστές.
- Επίδοση (performance) - Χρόνος απόκρισης, ρυθμός απόδοσης, βαθμός χρήσης πόρων.

Ειδικές απαιτήσεις:

- Διαφάνεια (transparency)
- Κλιμάκωση (scalability)
- Αξιοπιστία (reliability)
- Ανεκτικότητα σφαλμάτων (fault tolerance)
- Ασφάλεια (security)

1.2.1 Διαφάνεια

Ένας σημαντικός στόχος των κατανεμημένων συστημάτων είναι η διαφάνεια. Ένα κατανεμημένο σύστημα, πρέπει να εμφανίζεται στον χρήστη και τις εφαρμογές ως ένα ενιαίο σύνολο χωρίς να διακρίνεται η διασπορά των πόρων σε διαφορετικούς ηλεκτρονικούς υπολογιστές και η ανάγκη της μεταξύ τους επικοινωνίας. Η διαφάνεια μπορεί να κατηγοριοποιηθεί όπως φαίνεται παρακάτω:

- Διαφάνεια προσπέλασης (access transparency)
  Ενοποιημένη προσπέλαση σε όλους τους πόρους του συστήματος, ανεξάρτητα από το αν είναι τοπικοί ή απομακρυσμένοι. Ένα κατανεμημένο σύστημα μπορεί, για παράδειγμα, να διαθέτει υπολογιστικά συστήματα, τα οποία να χρησιμοποιούν διαφορετικά λειτουργικά συστήματα, όπου το κάθε ένα από αυτα, να έχει τις δικές του συμβάσεις ονομασίας αρχείων. Οι διαφορές στην ονομασία και στον χειρισμό των αρχείων θα πρέπει να παραμένουν κρυφές από τους χρήστες και τις εφαρμογές.

- Διαφάνεια τοποθεσίας (location transparency)
  Ενοποιημένη προσπέλαση σε όλους τους πόρους του συστήματος, ανεξάρτητα από την πραγματική γεωγραφική τους θέση. Ο χρήστης δεν μπορεί να προσδιορίσει τη φυσική θέση ενός πόρου μέσα στο σύστημα.

- Διαφάνεια μετανάστευσης (migration transparency)
  Μπορεί να υπάρξει μετακίνηση πόρων, χωρίς να επηρεάζεται ο τρόπος προσπέλασής τους.

- Διαφάνεια μετάθεσης (location transparency)
  Ακόμα ισχυρότερη είναι η διαφάνεια στις περιπτώσεις όπου οι πόροι μπορούν να μετακινούνται ακόμα και κατά τη διάρκεια της προσπέλασής τους χωρίς αυτό να γίνεται αντιληπτό από τον χτήστη. Για παράδειγμα, όταν ένας χρήστης χρησιμοποιεί εν κινήσει τον ασύρματο φορητό του υπολογιστή και παραμένοντας συνδεδεμένος σε όλη τη διάρκεια της μετακίνησής του, τότε υπάρχει διαφάνεια μετάθεσης.

- Διαφάνεια αναπαραγωγής (replication transparency)
  Επιτρέπει τη χρήση πολλαπλών αντιγράφων των πόρων, με σκοπό τη βελτίωση της αξιοπιστίας και της απόδοσης του συστήματος χωρίς όμως οι χρήστες να αντιλαμβάνονται την ύπαρξη αυτών των αντιγράφων. Προκειμένου να επιτευχθεί η απόκρυψη των αντιγράφων από τους χρήστες, πρέπει όλα τα αντίγραφα να έχουν το ίδιο όνομα. Επίσης, τα αντίγραφα θα πρέπει να έχουν και διαφάνεια τοποθεσίας αφού θα ήταν πρακτικά αδύνατο να γίνεται αναφορά σε αντίγραφα που βρίσκονται σε διαφορετικές θέσεις.

- Διαφάνεια ταυτοχρονισμού (concurrency transparency)
  Δυνατότητα ταυτόχρονης προσπέλασης ενός πόρου από διαφορετικούς χρήστες. Για παράδειγμα, δύο ανεξάρτητοι χρήστες μπορούν να κάνουν ταυτόχρονη προσπέλαση των ίδιων πινάκων σε μία κοινόχρηστη βάση δεδομένων χωρίς να επηρεάζουν ο ένας τον άλλο κατά τη διάρκεια των ενεργειών τους.

- Διαφάνεια αποτυχίας (failure transparency)
  Επιτρέπει την απόκρυψη των αστοχιών, δίνοντας στους χρήστες και τις εφαρμογές τη δυνατότητα να ολοκληρώσουν την εργασία τους, παρά την παρουσία κάποιου σφάλματος σε κάποιο επιμέρους σημείο του συστήματος. Σε αυτό το σημείο ταιριάζει απόλυτα ο ορισμός του Leslie Lamport για τα κατανεμημένα συστήματα: *"Γνωρίζεις ότι έχεις ένα όταν, ένα σφάλμα ενός ηλεκτρονικού υπολογιστή που ούτε ήξερες ότι υπήρχει, μπορεί να αχρηστέψει τον υπολογιστή σου."* Η κάλυψη των αστοχιών αποτελεί ένα από τα δυσκολότερα ζητήματα στα κατανεμημένα συστήματα.

- Διαφάνεια διατήρησης (persistence transparency)
  Απόκρυψη του αν ένας πόρος (λογισμικού) βρίσκεται στη μνήμη ή στο δίσκο. Για παράδειγμα, πολλές αντικειμενοστρεφείς βάσεις δεδομένων διαθέτουν μηχανισμούς για την άμεση κλήση μεθόδων που σχετίζονται με αποθηκευμένα αντικείμενα. Ο διακομιστής βάσεων δεδομένων αντιγράφει πρώτα την κατάσταση του αντικειμένου από το δίσκο στην κύρια μνήμη, εκτελεί τη λειτουργία, και ενδεχομένως ξαναγράφει την κατάσταση πίσω στο δευτερεύοντα αποθηκευτικό χώρο. Ο χρήστης όμως δεν αντιλαμβάνεται ότι ο διακομιστής μεταφέρει την κατάσταση μεταξύ πρωτεύουσας και δευτερεύουσας μνήμης.

1.2.2 Κλιμάκωση

Καθώς τα κατανεμημένα συστήματα αναπτύσσονται, οι ρυθμοί εκτέλεσης των διεργασιών πρέπει να διατηρούνται στα ίδια επίπεδα. Η ανάπτυξη αυτή, μπορεί να αφορά τον αριθμό των χρηστών, τον αριθμό των υπολογιστών που συνδέονται ή ακόμα και την απόδοση του συστήματος. Η κλιμάκωση διευκολύνεται από τη χρήση αλγορίθμων που αποφεύγουν καταστάσεις συμφόρησης κατά την πρόσβαση κοινών δεδομένων. Τα δεδομένα θα πρέπει να οργανώνονται ιεραρχικά, χρησιμοποιώντας τις κατάλληλες δομές δεδομένων στην υλοποίηση, ώστε να επιτυγχάνονται οι καλύτεροι δυνατοί χρόνοι προσπέλασης. Μπορεί επίσης να γίνει χρήση αντιγράφων για τα δεδομένα τα οποία προσπελαύνονται συχνά προκειμένου να επιτευχθούν όσο το δυνατόν καλύτερες επιδόσεις του συστήματος.

- Έλεγχος του κόστους των φυσικών πόρων
  Όσο αυξάνεται η απαίτηση για ένα πόρο, θα πρέπει να είναι δυνατή η επέκταση του συστήματος, με λογικό κόστος, έτσι ώστε αυτή να ικανοποιήται. Για παράδειγμα, η συχνότητα πρόσβασης σε ένα φάκελο στο τοπικό δίκτυο είναι πιθανό να αυξηθεί όσο αυξάνεται ο αριθμός των χρηστών. Επομένως, πρέπει να είναι δυνατή η εγκατάσταση πρόσθετων διαχειριστών αποθηκευτικών μονάδων για να αποφευχθεί κώλυμα στην απόδοση, το οποίο θα προέκυπτε αν ένας και μόνο διακομιστής αποθηκευτικών μονάδων είχε να διαχειριστεί όλα τα αιτήματα πρόσβασης σε ένα φάκελο.

- Έλεγχος της απώλειας απόδοσης
  Έστω η διαχείριση ενός συνόλου πληροφοριών του οποίου το μέγεθος είναι ανάλογο με τον αριθμό των χρηστών ή των πόρων του συστήματος. Οι αλγόριθμοι που χρησιμοποιούν ιεραρχικές δομές έχουν μεγαλύτερη δυνατότητα κλιμάκωσης από αυτούς που χρησιμοποιούν γραμμικές δομές. Αλλά ακόμα και με τις ιεραρχικές δομές, μια αύξηση στο πλήθος εγγραφών θα έχει σαν αποτέλεσμα κάποια απώλεια στην απόδοση. Για να διαθέτει ένα σύστημα δυνατότητα κλιμάκωσης, η μέγιστη απώλεια απόδοσης δεν πρέπει να είναι χειρότερη από αυτό.

- Παρεμπόδιση εξάντλησης πόρων λογισμικού
  Ένα παράδειγμα πιθανής εξάντλησης των πόρων του συστήματος, είναι η ονοματολογία που χρησιμοποιείται για τις διευθύνσεις των υπολογιστών στο διαδίκτυο. Η χρήση 32 bits για το σκοπό αυτό θέτει ένα φυσικό όριο στις δυνατότητες του συστήματος. Η κατάλληλη λύση στο πρόβλημα αυτό είναι δύσκολη λόγω της αδυναμίας πρόβλεψης των απαιτήσεων των χρηστών με την πάροδο του χρόνου.

- Αποφυγή κωλυμάτων απόδοσης
  Οι αλγόριθμοι πρέπει να αποκεντρώνονται για να αποφευχθούν οι απώλειες απόδοσης.

1.2.3 Αξιοπιστία

Η αξιοπιστία είναι μια έννοια ορθά συνδεδεμένη με την ανοχή στα σφάλματα που μελετάται παρακάτω. Τα κατανεμημένα συστήματα έχουν τη δυνατότητα να είναι πιο αξιόπιστα από ότι τα συστήματα ανεξάρτητων υπολογιστών, επειδή διαθέτουν ανεκτικότητα στα σφάλματα. Αυτό σημαίνει ότι ενώ ορισμένες μονάδες του συστήματος μπορεί να αντιμετωπίσουν κάποιο σφάλμα, οι υπόλοιπες που συνεχίζουν να λειτουργούν κανονικά, μπορούν να αναλάβουν το έργο των μονάδων που βγήκαν εκτός λειτουργίας. Για το λόγο αυτό οι κατανεμημένες αρχιτεκτονικές είναι ένας παραδοσιακός τρόπος σχεδιασμού υψηλά αξιόπιστων υπολογιστικών συστημάτων. Η διασφάλιση της σωστής λειτουργίας ενός κατανεμημένου συστήματος κατά την παρουσία σφαλμάτων απαιτεί την χρήση σύνθετων αλγοριθμικών λύσεων.

1.2.4 Ανεκτικότητα σφαλμάτων

Ορισμένες φορές μπορεί να προκύψουν σφάλματα στο υλικό ή το λογισμικό μιας μονάδας, μπορεί να σταματήσουν να λειτουργούν ή να λειτουργούν μερικώς, ενώ την ίδια στιγμή άλλες μονάδες συνεχίζουν κανονικά την λειτουργία τους. Έτσι όμως οι διεργασίες μπορεί να παράγουν λανθασμένα αποτελέσματα ή να σταματήσουν πριν ολοκληρώσουν τον υπολογισμό. Για το λόγο αυτό, κάθε μέρος του κατανεμημένου συστήματος πρέπει να έχει ανοχή στην πιθανή ύπαρξη σφαλμάτων.

- Ανίχνευση σφαλμάτων
  Κάποια σφάλματα μπορούν να ανιχνευθούν ενώ άλλα όχι. Για παράδειγμα η ανίχνευση ενός κατεστραμμένου διακομιστή στο διαδίκτυο είναι σχεδόν αδύνατη. Το ζητούμενο είναι να υπάρχει διαχείριση των αστοχιών οι οποίες δεν μπορούν να ανιχνευθούν αλλά μπορούν να υποτεθούν.

- Αντιμετώπιση σφαλμάτων
  Μερικά σφάλματα τα οποία ανιχνεύονται μπορούν να αντιμετωπιστούν πλήρως ή να γίνουν λιγότερο σοβαρά. Για παράδειγμα, μηνύματα που δεν φτάνουν στο προορισμό τους μπορούν να σταλούν εκ νέου.

- Ανοχή σφαλμάτων
  Τα τερματικά μπορούν να σχεδιαστούν έτσι ώστε να ανέχονται τις αστοχίες, το οποίο συνεπάγεται ότι και οι χειριστές του συστήματος θα πρέπει επίσης να τις ανέχονται. Για παράδειγμα, όταν ένας web browser δεν μπορεί να επικοινωνίσει με ένα web server, ο χρήστης ενημερώνεται προκειμένου να μην περιμένει.

- Ανάκτηση από σφάλματα
  Ο σχεδιασμός λογισμικού θα πρέπει να επιτρέπει την επαναφορά των δεδομένων στην περίπτωση που καταστραφεί μία υπολογιστική μονάδα. Οι διεργασίες που πραγματοποιούνται από ορισμένα προγράμματα θα μείνουν ελλιπείς όταν προκύψει ένα σφάλμα, και τα μόνημα δεδομένα που ενημερώνουν μπορεί να μην είναι συνεπή.

1.2.5 Ασφάλεια

Πολλοί από τους πόρους πληροφοριών που είναι που είναι διαθέσιμοι και διατηρούνται στα κατανεμημένα συστήματαέχουν υψηλή πραγματική αξία για τους χρήστες τους. Είναι λοιπόν ιδιαίτερα σημαντικό να προφυλάσονται. Η ασφάλεια των πόρων πληροφοριών έχει τρία συστατικά: **εμπιστοσύνη** (προστασία από έκθεση σε μη εξουσιοδοτημένα άτομα), **ακεραιότητα** (προστασία από αλλαγές ή διαφθορά) και **διαθεσιμότητα** (προστασία από παρεμβολές με σκοπό να υπαάρξει πρόσβαση στους πόρους).

Σε ένα κατανεμημένο σύστημα, διεργασίες που εκτελούνται από διαφορετικό υπολογιστή μπορούν να επικοινωνίσουν μεταξύ τους. Κατά την επικοινωνία όμως αυτή, εάν υπάρχει ελεύθερη πρόσβαση σε όλους τους πόρους του τοπικού δικτύου, η ασφάλεια κινδυνεύει. Για την προφύλαξη του δικτύου μπορεί να γίνει χρήση ενός firewall περιορίζοντας έτσι την εισερχόμενη και εξερχόμενη κίνηση. Παρόλα αυτά, η προκειμένη λύση δεν διασφαλίζει την κατάλληλη χρήση των πόρων από τις διεργασίες που εκτελούνται μέσα στο τοπικό δίκτυο, αλλά ούτε και την κατάλληλη χρήση των πόρων του δικτύου που δεν προστατεύονται από firewalls.

Το μεγάλο ζητούμενο εδώ είναι να επιτυγχάνεται η αποστολή ευαίσθητων πληροφοριών μέσω μηνυμάτων με ασφαλή τρόπο, αλλά ταυτόχρονα να είναι γνωστή και να μπορεί να πιστωποιηθεί η ακριβής ταυτότητα ενός χρήστη εξ' αποστάσεως. Και οι δύο αυτές προκλήσεις, μπορούν να αντιμετωπιστούν με τη χρήση τεχνικών κρυπτογράφησης που έχουν αναπτυχθεί για το σκοπό αυτό. Παρόλα αυτά το πρόβλημα δεν μπορεί να αντιμετωπιστεί πλήρως.

1.3 Κατανεμημένες υπηρεσίες

- Υπηρεσίες επικοινωνίας
  Η επικοινωνία στα κατανεμημένα συστήματα βασίζεται στη μεταβίβαση μηνυμάτων σε χαμηλό επίπεδο, όπως αυτή υποστηρίζεται από το υποκείμενο δίκτυο. Τα σημερινά συστήματα μπορεί να αποτελούνται από εκατομύρια διεργασίες, οι οποίες είναι διάσπαρτες σε ένα πιθανότατα αναξιόπιστο δίκτυο όπως το internet.

- Υπηρεσίες εκτέλεσης

- Υπηρεσίες ονομασίας
  Τα ονόματα παίζουν σημαντικό ρόλο σε όλα τα συστήματα υπολογιστών. Σε κάθε πόρο του συστήματος αποδίδεται ένα μοναδικό όνομα. Κάθε όνομα μπορεί να αναχθεί στην οντότητα στην οποία αναφέρεται και έτσι, μία διεργασία μπορεί να προσπελάσει την ονοματισμένη οντότητα. Προκειμένου να επιτευχθεί η αναγωγή ονομπάτων χρειάζεται η εφαρμογή ενός συστήματος ονομασίας, του οποίου η υλοποίηση συχνά κατανέμεται σε πολλά μηχανήματα. Χρησιμοποιήται επίσης και υπηρεσία ευρετηρίου.

- Υπηρεσίες εντοπισμού
  Υπηρεσίες εντοπισμού παρέχονται στα κινητά κατανεμημένα συστήματα.

- Υπηρεσίες αρχείων
  Είναι ένα παγκόσμιο σύστημα αρχείων που παρέχει ένα διαφανή τρόπο για την προσπέλαση οποιουδείποτε αρχείου του συστήματος με τον ίδιο τρόπο. Χρησιμοποιήται και εδώ υπηρεσία ευρετηρίου.

- Υπηρεσίες συναλλαγών

- Υπηρεσίες αντικειμένων
  Τα αντικείμενα, στα κατανεμημένα συστήματα, παίζουν σημαντικό ρόλο στην επίτευξη της διαφάνειας κατανομής. Σχεδόν τα πάντα αντιμετωπίζονται σαν αντικείμενα και οι πελάτες έχουν στη διάθεσή τους και μπορούν να καλούν υπηρεσίες και πόρους με τη μορφή αντικειμένων.

- Υπηρεσίες αναπαραγωγής
  Στα κατανεμημένα συστήματα, τα δεδομένα αναπαράγωντε με στόχο την αξιοπιστία και τη βελτίωση της απόδοσης. Βέβαια, όταν ενημερώνεται ένα αντίγραφο, θα πρέπει να είναι εξασφαλισμένο ότι θα ενημερώνονται και τα άλλα αντίγραφα, αλλιώς δεν θα είναι πλέον ίδια.

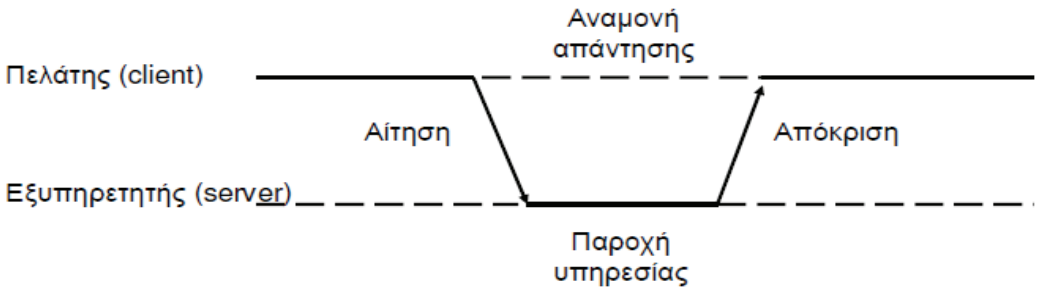
- Υπηρεσίες ασφαλείας
  Η ασφάλεια είναι πολύ σημαντική στα κατανεμημένα συστήματα καθός ένα σφάλμα, μπορεί να αχρηστέψει ένα σύστημα. Ασχολείται με την προστασία των πόρων κάθε είδους, ώστε η δυνατότητα πρόσβασης να περιορίζεται μόνο στα εξουσιοδοτημένα άτομα. Κάποιοι από τους μηχανισμούς ασφαλείας είναι η κρυπτογράφηση, η πιστοποίηση ταυτότητας, η εξουσιοδότηση και η παρακολούθηση.

1.4 Αρχιτεκτονικά Μοντέλα

Για τη διαχείριση της ανάπτυξης πολύπλοκων συστημάτων, είναι απαραίτητος ο διαχωρισμός τους σε επιμέρους τμήματα. Τα αρχιτεκτονικά μοντέλα αποτελούν τους τρόπους οργάνωσης των τμημάτων αυτών καθώς επίσης και τους τρόπους δόμησης των συσχετίσεων μεταξύ τους. Η αρχιτεκτονική προσέγγιση βοηθάει στην κατανόηση του συστήματος, στην οργάνωση της ανάπτυξής του, στην επαναχρησιμοποίηση των συστατικών του και τέλος στη συντήριση και εξέληξη του συστήματος. Για τα κατανεμημένα συστήματα τα δύο βασικά αρχιτεκτονικά μοντέλα είναι το μοντέλο πελάτη-εξυπηρετητη και το μοντέλο ομότιμων. Εδώ θα σχοληθούμε περισσότερο με την ανάλυση του μοντέλου πελάτη-εξυπηρετητη, ενώ θα γίνει και μια απλή αναφορά στο μοντέλο ομότιμων.

1.4.1 Μοντέλο πελάτη-εξυπηρετητή

Η έννοια των πελατών (clients) οι οποίοι ζητούν παροχή υπηρεσιών από εξυπηρετητές (servers), βοηθάει στην κατανόηση και τη διαχείριση της πολυπλοκότητας των κατανεμημένων συστημάτων. Στο βασικό μοντέλο πελάτη-εξυπηρετητή, οι διεργασίες ενός κατανεμημένου συστήματος υποδιαιρούνται σε δύο ομάδες. Πελάτης (client) είναι μια διεργασία που στέλνει αίτηση στον εξυπηρετητή για μια υπηρεσία και περιμένει την απάντησή του. Εξυπηρετητής (server) είναι η διεργασία που είναι υπεύθυνη για την υλοποίηση της υπηρεσία που αιτήται ο πελάτης.



Εικόνα 1

Όταν σε ένα δίκτυο υπάρχει αξιοπιστία, η επικοινωνία πελάτη-εξυπηρετητή γίνεται με τη βοήθεια ενός απλού ασυνδεσμικού πρωτοκόλλου. Ένας πελάτης προκειμένου να ζητήσει μία υπηρεσία, δημιουργεί ένα μήνυμα προσδιορίζοντας την ταυτότητα του εξυπηρετητή που θέλει μαζί με τα απαιτούμενα δεδομένα εισόδου. Στη συνέχεια το μήνυμα στέλνεται στον εξυπηρετητή και αυτός με τη σειρά του το επεξεργάζεται και μορφοποιεί τα αποτελέσματα σε ένα απαντητικό μήνυμα το οποίο στέλνει πίσω στον πελάτη. Το πρωτόκολλο αίτησης-απάντησης δουλεύει άψογα αν το μήνυμα δεν χαθεί ή δεν αλλοιωθεί. Βέβαια υπάρχει πάντα και η πιθανότητα αστοχίας. Όμως όπως έχει αναφερθεί και σε προηγούμενη ενότητα, εάν εντοπιστεί κάποιο σφάλμα, τότε ο πελάτης μπορεί να ξαναστείλει την αίτηση.

Ως εναλλακτική λύση, πολλά συστήματα χρησιμοποιούν συνδεσμικά πρωτόκολλα, αν και δεν είναι ιδιαίτερα καλή λύση λόγω της χαμηλής της απόδοσης και μεγάλου κόστους εγκατάστασης και κατάργησης. Με αυτό τον τρόπο λειτουργούν τα πρωτόκολλα εφαρμογών του internet τα οποία βασίζονται σε αξιόπιστες συνδέσεις IP/TCP. Για παράδειγμα, όταν ένας πελάτης ζητά μία υπηρεσία, εγκαθιστά πρώτα μία σύνδεση με τον εξυπηρετητή και στη συνέχεια στέλνει την αίτηση. Ο εξυπηρετητής με τη σειρά του χρησιμοποιεί την ίδια σύνδεση για να στείλει το απαντητικό μήνυμα, μετά το οποίο η σύνδεση καταργείται.

1.4.1.1 Παραλλαγές πελάτη-εξυπηρετητή

Πολλαπλοί εξυπηρετητές
Στόχος των πολλαπλών εξυπηρετητών είναι η αναβάθμιση της απόδοσης και της ανθεκτικότητας του συστήματος. Υπηρεσίες οι οποίες έχουν μεγάλη ζήτηση, όπως για παράδειγμα οι μηχανές αναζήτησης, μπορεί να διαθέτουν πολλούς εξυπηρετητές αφιερωμένους σε μία συγκεκριμένη εργασία. Στην περίπτωση της πολλαπλότητας υπάρχουν δύο προσεγγίσεις, είτε τα αντικείμενα που υπολποιούν την υπηρεσία μοιράζονται μεταξύ πολλαπλών εξυπηρετητών, είτε αντίγραφα των αντικειμένων διατηρούνται σε κάθε εξυπηρετητή.

Πληρεξούσιοι εξυπηρετητές (proxy servers) και προσορινή αποθήκη (caching)
Βασικοί στόχοι εδώ είναι η αναβάθμιση της διαθεσιμότητας και της απόδοσης μίας υπηρεσίας (π.χ. εξυπηρετητής ιστού) με την μείωση του φορτίου στο δίκτυο ευρείας κλίμακας (π.χ. διαδίκτυο) και πάνω στην ίδια την υπηρεσία. Η προσορινή αποθήκη αποθηκεύει πρόσφατα δεδομένα που έχουν παραδοθεί από τον εξυπηρετητή στον πελάτη και βρίσκονται πιο κοντά του από ότι τα πραγματικά δεδομένα, ενώ ένας πληρεξούσιος εξυπηρετητής είναι μία διεργασία που αναλαμβάνει να παίξει το ρολο του ενδιάμεσου ανάμεσα στον πραγματικό εξυπηρετητή και στον πελάτη που ζητά για παράδειγμα κάποια ιστοσελίδα ή το κατέβασμα κάποιου αρχείου του. Μόλις ο πληρεξούσιος εξυπηρετητής λάβει ένα αίτημα από έναν πελάτη, ελέγχει αν διαθέτει την πληροφορία στη δική του προσωρινή μνήμη, και αν τη διαθέτει τη στέλνει κατευθείαν χωρίς να χρειάζεται επικοινωνία με τον πραγματικό εξυπηρετητή. Αν δεν διαθέτει τη συγκεκριμένη πληροφορία τότε αναλαμβάνει να τη ζητήσει, να την προωθήσει στον πελάτη αλλά και να την αποθηκεύσει προσωρινά ώστε να την προωθήσει και πάλι σε περίπτωση που ζητηθεί ξανά. Εκτός από την αποθήκευση αντικειμένων πιο κοντά στους πελάτες, οι πληρεξούσιοι εξυπηρετητές μπορεί να αναλάβουν και άλλους ρόλους, όπως για παράδειγμα την μετατροπή μίας υπηρεσίας από μία μορφή σε μία άλλη.

Κινητός κώδικας (mobile code)
Κώδικας μπορεί να μετακινηθεί από τον εξυπηρετητή και να εκτελεστεί στον υπολογιστή που φιλοξενεί τη διεργασία του πελάτη. Ένα παράδειγμα τέτοιου κώδικα είναι τα Java applets που εκτελούνται τοπικά από το πρόγραμμα περιήγησης ιστού του πελάτη. Το θετικό εδώ είναι ότι δεν υπάρχουν καθυστερήσεις στο δίκτυο επικοινωνίας και έτσι η αλληλεπίδραση με τον χρήστη είναι άμεση. Σε άλλες περιπτώσεις, οι εξυπηρετητές παρέχουν λειτουργικότητα που δεν είναι τυποποιημένη και διαθέσιμη στα προγράμματα περιήγησης και χρειάζεται να σταλεί κώδικας από τον εξυπηρετητή στον πελάτη που όταν εκτελεστεί θα επικοινωνεί με τον εξυπηρετητή. Ένας τέτοιος κώδικας μπορεί να υλοποιεί το λεγόμενο μοντέλο ώθησης (push model) κατά το οποίο ο εξυπηρετητής ξεκινάει μία αλληλεπίδραση, και όχι ο πελάτης. Για παράδειγμα μπορεί ο εξυπηρετητής μόλις αποκτήσει κάποια δεδομένα να τα στέλνει από μόνος του στον πελάτη, χωρίς να περιμένει να του τα ζητήσει.

Κινητοί πράκτορες (mobile agents)
Κινητός πράκτορας είναι ένα εκτελέσιμο πρόγραμμα το οποίο περιλαμβάνει κώδικα και δεδομένα και ταξιδεύει μεταξύ υπολογιστών μέσω δικτύου και διεκπεραιώνει συγκεκριμένες εργασίες, όπως για παράδειγμα η συλλογή πληροφοριών, για λογαριασμό κάποιας οντότητας, στην οποία επιστρέφει κάποια αποτελέσματα. Εάν μπαίναμε στην διαδικασία να συγκρίνουμε αυτή την αρχιτεκτονική με εκείνη του στατικού πελάτη που εκτελεί απομακρισμένες κλήσεις σε αντικείμενα, πιθανώς συνοδευόμενες από μεταφορά μεγάλου όγκου δεδομένων, θα διαπιστώναμε μία μείωση του κόστους και του χρόνου επικοινωνίας λόγω της

αντικατάστασης των απομακρυσμένων κλήσεων με τοπικές. Από την άλλη πλευρά όμως, οι κινητοί πράκτορες όπως και ο κινητός κώδικας, συνιστούν πιθανή απειλή στην ασφάλεια του υπολογιστή που επισκέπτονται, καθώς μπορεί να περιέχουν κακόβουλο λογισμικό, όπως ιούς. Για τον λόγο αυτό, η εκτέλεση κινητών πρακτόρων θα πρέπει να επιτρέπεται μόνο μετά από ασφαλή αναγνώριση της ταυτότητας της οντότητας για λογαριασμό της οποίας αυτοί λειτουργούν.

Υπολογιστές δικτύου (network computers)
Οι υπολογιστές δικτύου είναι υπολογιστές χαμηλού κόστους οι αποίοι δεν διαθέτουν κάποιο μόνιμο μέσο αποθήκευσης (π.χ. σκληρό δίσκο), παρά μόνο μνήμη RAM. Ακόμη και τα προγράμματα τα οποία τρέχουν σε αυτούς τους υπολογιστές, καθώς και το λειτουργικό τους σύστημα, βρίσκονται αποθηκευμένα σε απομακρυσμένους εξυπηρετητές αρχείων. Τα περισσότερα προγράμματα, είναι γραμμένα σε κάποια από τις εκδόσεις της Java, έτσι ώστε να μπορούν να είναι ανεξάρτητα της πλατφόρμας στην οποία θα τρέξουν. Έτσι, το μόνο που χρειαζεται να διαθέτει ένας υπολογιστής δικτύου είναι μία γρήγορη σύνδεση στο internet.

Ελαφριοί πελάτες (thin clients)
Η παραλλαγή του ελαφρύ πελάτη αναφέρεται σε έναν υπολογιστή ή ένα στρώμα λογισμικού που προσφέρει κυρίως την διεπαφή χρήσης μίας εφαρμογής, αφού όλη η επιχειρισιακή λογική εκτελείται σε έναν άλλο απομακρισμένο υπολογιστή. Σε αυτή την παραλλαγή, όπως και σε εκείνη των υπολογιστών δικτύου, παρατηρούμε χαμηλό κόστος λειτουργίας και διαχείρησης, αλλά αντί τα προγράματα των εφαρμογών να μεταφορτώνονται στον υπολογιστή του πελάτη, αυτά εκτελούνται σε εξυπηρετητές και χρησιμοποιήται το δίκτυο για την μεταφορά των δεδομένων. Ένας ελαφρύς πελάτης μπορεί να τρέχει σε έναν υπολογιστή με περιορισμένες δυνατότητες για εφαρμογές όπως διαχείρηση ηλεκτρονικού ταχυδρομείου και περιήγηση ιστού, ενώ μπορεί να συνδέεται σε ένα μεγαλύτερο δίκτυο, για παράδειγμα, μίας εταιρείας ή ενός σχολείου. Η δομή του ελαφρύ πελάτη διευκολύνει τον εντοπισμό προβλημάτων λειτουργίας και μειώνει την πολυπλοκότητα για τον τελικό χρήστη, καθώς αυτός περιορίζεται μόνο στην εκμάθηση μίας εφαρμογής αντί ενός υπολογιστικού περιβάλλοντος γενικού σκοπού. Τέλος, διευκολύνει την παροχή μηχανισμών ασφάλησης, καθώς αυτοί μπορούν να υλοποιηθούν κεντρικά στον εξυπηρετητή υπολογισμών.

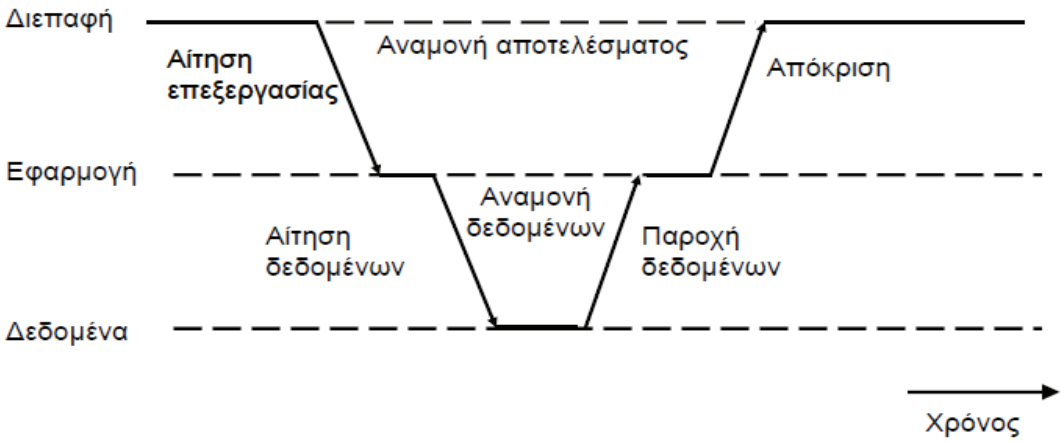
Βαριοί πελάτες (fat clients)
Σε αυτή την παραλλαγή, ο εξυπηρετητής είναι υπεύθυνος μόνο για τη διαχείρηση των δεδομένων. Το λογισμικό του πελάτη υλοποιεί τη λογική της εφαρμογής και τις αλληλεπιδράσεις με το χρήστη του συστήματος. Στο μοντέλο του βαρύ πελάτη, οι λειτουργίες επεξεργασίας της εφαρμογής εκτελούνται τοπικά και έτσι μεταβιβάζεται μεγαλύτερος φόρτος επεξεργασίας στον πελάτη, σε αντίθεση με το μοντέλο του ελαφρύ πελάτη. Ταιριάζει περισσότερο σε νέα συστήματα πελάτη-εξυπηρετητή, στα οποία οι δυνατότητες του συστήματος του πελάτη είναι εκ των προτέρων γνωστές. Είναι πιο περίπλοκο από το μοντέλο του ελαφρύ πελάτη, ειδικά στον τομέα της διαχείρησης. Οι νέες εκδόσεις της εφαρμογής πρέπει να εγκαθίστανται σε κάθε πελάτη.

1.4.1.2 Αρχιτεκτονική τριών επιπέδων

Η αρχιτεκτονική client-server συχνά θεωρήται ως αρχιτεκτονική δύο επιπέδων αφού ένας εξυπηρετητής μπορεί να λειτουργεί και ως πελάτης σε αρκετές περιπτώσεις. Για παράδειγμα, μια μηχανή αναζήτης στο internet είναι ταυτόχρονα και εξυπηρετητής και πελάτης, ως εξυπηρετητής ανταποκρίνεται στα ερωτήματα των προγραμμάτων περιήγησης και ως πελάτης ενεργοποιεί προγράμματα ανίχνευσης ιστού.

Πολλές αφαρμογές πελάτη-εξυπηρετητή έχουν ως βασικό ρόλο την υποστήριξη της πρόσβασης των χρηστών σε βάσεις δεδομένων. Για το λόγο αυτό μπορούμε να διαχωρίσουμε την αρχιτεκτονική σε τρεία επίπεδα:

- επίπεδο παρουσίασης (διεπαφής χρήστη)
- επίπεδο εφαρμογής (επεξεργασίας)
- επίπεδο δεδομένων



Εικόνα 2

Επίπεδο παρουσίασης
Το επίπεδο παρουσίασης ή αλλιώς επίπεδο διεπαφής χρήστη υλοποιήται από τους πελάτες. Το επίπεδο αυτό αποτελείται από τα προγράμματα που επιτρέπουν στους τελικούς χρήστες να αλληλεπιδρούν με τις εφαρμογές. Σε αυτό το επίπεδο μπορεί να συναντίσουμε διάφορους βαθμούς πολυπλοκότητας. Για παράδειγμα, ένα πρόγραμμα διεπαφής χρήστη μπορεί να είναι μια απλή οθόνη χαρακτήρων. Σήμερα, συναντάμε πολύ μεγάλες λειτουργικές δυνατότητες, οι οποίες επιτρέπουν στις εφαρμογές να μοιράζονται το ίδιο παράθυρο γραφικών και να το χρησιμοποιούν για να ανταλλάσουν δεδομένα μέσω των ενεργειών του χρήστη. Ένα παράδειγμα είναι η διαγραφή αρχείων, όπου για να διαγράψουμε το αρχείο που θέλουμε, επιλέγουμε το εικονίδιο που το αντιπροσωπεύει και το μετακινούμε πάνω στο εικονίδιο που αντιπροσωπεύει τον κάδο ανακύκλωσης.

Επίπεδο εφαρμογής
Το επίπεδο εφαρμογής ή αλλιώς επίπεδο επεξεργασίας είναι αυτό που βρίσκεται ανάμεσα στα επίπεδα παρουσίασης και δεδομένων και θα μπορούσαμε ίσως να πούμε ότι είναι και το πιό σημαντικό αφού σε αυτό γίνονται όλες οι διεργασίες. Ένα παράδειγμα που μπορούμε να εξετάσουμε σε αυτό το επίπεδο είναι μία μηχανή αναζήτησης στο internet. Οι χρήστες κάνουν

αναζήτηση σε μία βάση δεδομένων πληκτρολογώντας μια σειρά από λέξεις-κλειδιά, και μια λίστα με σχετικές ιστοσελίδες εμφανίζεται. Πίσω από τη μηχανή αναζήτησης υπάρχει ένα πρόγραμμα, το οποίο μετατρέπει τις λέξεις-κλειδιά που δίνει ο χρήστης σε ένα ή περισσότερα ερωτήματα της βάσης δεδομένων και τοποθετεί τα αποτελέσματα σε μία λίστα την οποία στη συνέχεια βλέπει ο χρήστης.

Επίπεδο δεδομένων
Το επίπεδο δεδομένων περιέχει τα προγράμματα τα οποία χειρίζεται το επίπεδο εφαρμογής. Το πιό σημαντικό σε αυτό το επίπεδο είναι η ιδιότητα διατήρησης των δεδομένων ακόμα και όταν δεν εκτελείται καμία εφαρμογή. Το επίπεδο δεδομένων αποτελείται από ένα σύστημα αρχείων, αλλά είναι πιό συνηθισμένο να χρησιμοποιήται μια βάση δεδομένων. Στο μοντέλο πελάτη-εξυπηρετητή, το επίπεδο δεδομένων υλοποιήται συνήθως στην πλευρά του εξυπηρετητή. Ένα ακόμα σημαντικό χαρακτηριστικό είναι η διατήρηση της συνέπειας. Όταν χρησιμοποιούνται βάσεις δεδομένων, διατήρηση της συνέπειας σημαίνει ότι σε αυτό το επίπεδο αποθηκεύονται επίσης μεταδεδομένα, όπως περιγραφές πινακων, περιορισμοί για τις καταχωρίσεις, και μεταδεδομένα για τις συγκεκριμένες εφαρμογές. Ακόμα ένα χαρακτηριστικό που αξίζει να αναφερθεί, είναι η ανεξαρτησία των δεδομένων. Τα δεδομένα είναι ανεξάρτητα από τις εφαρμογές, με τέτοιο τρόπο ώστε οι αλλαγές σε αυτή την οργάνωση να μην επηρεάζουν τις εφαρμογές, αλλά ούτε οι εφαρμογές να επηρεάζουν την οργάνωση των δεδομένων.

1.4.2 Μοντέλο ομότιμων

Τα ομότιμα συστήματα είναι αποκεντρωμένα συστήματα στα οποία κάθε δικτυακός κόμβος μπορεί να πραγματοποιεί υπολογιστικές λειτουργίες. Το συνολικό σύστημα είναι σχεδιασμένο για να εκμεταλεύεται την υπολογιστική ισχύ και τα αποθηκευτικά μέσα ενός μεγάλου αριθμού υπολογιστών συνδεδεμένων μέσω δικτύου. Τα περισσότερα ομότιμα συστήματα είναι προσωπικά, αλλά παρατηρείται αύξηση της χρήσης αυτής της τεχνολογίας και από επιχειρήσεις. Στο μοντέλο ομότιμων κόμβων (peer-to-peer) κάθε διεργασία μπορεί να παίξει όλους τους ρόλους (πελατης, εξυπηρετητής). Οι κόμβοι μπορούν να εισέρχονται και να εξέρχονται από το σύστημα με μεγάλη συχνότητα, χωρίς αυτό να επηρεάζει τη διαθεσιμότητα των υπηρεσιών του συστήματος λόγω πλεονασμού πόρων. Σε κάθε κόμβο υπάρχουν πόροι, όπως για παράδειγμα καταχωρημένα δεδομένα, οι οποίοι είναι διαθέσιμοι τμηματικά στο δίκτυο, από τη στιγμή της σύνδεσης έως τη στιγμή της αποσύνδεσης του κόμβου. Η πληροφορία του κάθε πόρου έχει μια δομή, την οποία ο κάθε κόμβος πρέπει να διαφημίσει στους γειτονικούς κόμβους ώστε και εκείνοι με τη σειρά τους να μπορούν να διατηπώσουν ερωτήματα προκειμένου να πάρουν την συγκεκριμένη πληροφορία. Τα αιτήματα για έναν πόρο διασκορπίζονται μέσα στο σύστημα και έτσι όσοι κόμβοι έχουν σχετική πληροφορία τη στέλνουν, σαν μία συνολική απάντηση από το δίκτυο, σε αυτόν που την ζήτησε. Έτσι, μπορούν να υλοποιηθούν ακόμα και εφαρμογές μεγάλης κλίμακας, αφού η αποθήκευση, η επεξεργασία και το φορτίο επικοινωνίας για την πρόσβαση στα δεδομένα κατανέμεται σε πολλά υπολογιστικά συστήματα και συνδέσμους δικτύων.

1.5 Συγχρονα και Ασυγχρονα κατανεμημενα συστηματα

Σε ένα κατανεμημένο σύστημα είναι δύσκολο να μπουν χρονικοί περιορισμοί στο χρόνο που χρειάζεται για την εκτέλεση διεργασιών,την παράδοση μηνυμάτων, ή την παρέκκλιση των ρολογιών.

Υπάρχουν δύο προσεγγίσεις:

- Η πρώτη κάνει ισχυρές υποθέσεις για το χρόνο (σύγχρονα κατανεμημένα συστήματα –synchronousdistributed systems).

- Η δεύτερη δεν κάνει καμία υπόθεση για το χρόνο (ασύγχρονα κατανεμημένα συστήματα –asynchronous distributed systems).

1.5.1 Σύγχρονα κατανεμημένα συστήματα

Σύγχρονα κατανεμημένα συστήματα είναι εκείνα τα οποία με διάφορες τεχνικές καταφέρνουν να έχουν συγχρονισμένα τοπικά ρολόγια, παρόλη την ύπαρξη του φαινομένου της ταχύτητας ολίσθησης.

Ένα σύγχρονο κατανεμημένο σύστημα είναι αυτό στο οποίο ορίζονται τα ακόλουθα όρια :

- ο χρόνος να εκτελεστεί κάθε βήμα μιας διεργασίας έχει γνωστό άνω και κάτω όριο

- κάθε μήνυμα που μεταδίδεται πάνω από ένα κανάλι λαμβάνεται μέσα σε ένα γνωστό πεπερασμένο χρόνο

- κάθε διεργασία έχει ένα τοπικό ρολόι του οποίου ο ρυθμός παρέκκλισης από τον πραγματικό χρόνο έχει ένα γνωστό όριο

Στα κατανεμημένα συστήματα είναι δύσκολο να βρεις ρεαλιστικές τιμές ορίων και σχεδόν αδύνατο να παρέχεις εγγυήσεις για τις τιμές αυτές. Παρόλα αυτά η μοντελοποίηση ενός αλγορίθμου ως ένα σύγχρονο κατανεμημένο σύστημα μπορεί να είναι χρήσιμη , διότι μας δίνει μια ιδέα για το πώς θα συμπεριφερόταν σε ένα πραγματικό κατανεμημένο σύστημα. Σύγχρονα συστήματα μπορούν να δημιουργηθούν αρκεί να υπάρχουν εκ των προτέρων γνωστές οι απαιτήσεις των διεργασιών και να τους δοθεί εγγυημένα χρόνος στην ΚΜΕ και χωρητικότητα στο δίκτυο , τα οποία να καλύπτουν αυτές τις απαιτήσεις.

1.5.2 Ασύγχρονα κατανεμημένα συστήματα

Ορισμένα κατανεμημένα συστήματα, όπως το Internet, ανήκουν σε ένα διαφορετικό μοντέλο με τα ακόλουθα χαρακτηριστικά :

- Δεν υπάρχουν όρια στην ταχύτητα εκτέλεσης των διεργασιών . Κάθε βήμα μπορεί να χρειαστεί ένα αυθαίρετα μεγάλο χρονικό διάστημα για να εκτελεστεί.

- Δεν υπάρχουν όρια στις καθυστερήσεις μετάδοσης μηνυμάτων . Ένα μήνυμα μπορεί να ληφθεί μετά από αυθαίρετα μεγάλο χρονικό διάστημα.

- Ο ρυθμός παρέκκλισης ενός ρολογιού είναι αυθαίρετος.

- Αυτό το μοντέλο ταιριάζει ακριβώς στο Internet στο οποίο δεν υπάρχουν εσωτερικά όρια στους φόρτους των εξυπηρετητών και του δικτύου.

- Ορισμένα προβλήματα μπορούν να λυθούν ακόμα και με αυτές τις υποθέσεις (π.χ . οι browsers που επιτρέπουν στους χρήστες να κάνουν και άλλα πράγματα όσο περιμένουν).

- Κάθε λύση που είναι έγκυρη για ένα ασύγχρονο κατανεμημένο σύστημα είναι έγκυρη και για σύγχρονο κατανεμημένο σύστημα.

- Τα πραγματικά κατανεμημένα συστήματα είναι συνήθως ασύγχρονα λόγω του ότι οι διεργασίες μοιράζονται τους επεξεργαστές και τα κανάλια επικοινωνίας.

- Υπάρχουν και ορισμένα προβλήματα που δεν μπορούν να λυθούν για ασύγχρονα κατανεμημένα συστήματα και πρέπει να γίνει χρήση ορισμένων χρονικών περιορισμών.

- Για παράδειγμα η ανάγκη για την παράδοση στοιχείων μιας συνεχούς ροής δεδομένων πολυμέσων σε μια συγκεκριμένη διορία. Για τέτοιες περιπτώσεις χρειάζονται (στοιχεία από) σύγχρονα κατανεμημένα συστήματα.

1.5.3 Σύγκριση

Στην πράξη τα κατανεμημένα συστήματα τείνουν να έχουν ασύγχρονη συμπεριφορά ακόμη και αν είναι κατασκευασμένα από το ίδιο υλικό, για τους εξής λόγους:

1. Της προσθήκης στρωμάτων λογισμικού για τη διαχείριση των διεργασιών και τη δημιουργία /διαχείριση καναλιών επικοινωνίας.

2. Του διαφορετικού φόρτου εργασίας του κάθε Η/Υ.

3. Του φαινομένου του clock drift.

Για την επίτευξη του συγχρονισμού απαιτείται η χρήση ειδικού ενδιάμεσου λογισμικού που η εφαρμογή του θεωρείται ιδιαίτερα ακριβή.

Το ασύγχρονο είναι ευρύτερο μοντέλο. Οι τεχνικές /αλγόριθμοι που αναπτύσσονται για ασύγχρονα συστήματα μπορούν να χρησιμοποιηθούν άμεσα και σε σύγχρονα. Το αντίστροφο μπορεί να ισχύσει μόνο με τη χρήση ειδικών λογισμικών που είναι γνωστά ως συγχρονιστές (synchronizers).

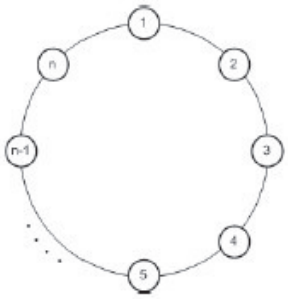
Τα σύγχρονα συστήματα δείχνουν καλύτερη απόδοση από τα ασύγχρονα. Στην τελική σύγκριση πρέπει να ληφθεί υπόψη και το κόστος υλοποίησης του συγχρονισμού. Οι σύγχρονοι αλγόριθμοι γενικά είναι πιο απλοί από τους ασύγχρονους. Γενικά το σύγχρονο περιβάλλον επιτρέπει πιο απλό και εύκολο σχεδιασμό και στους σύγχρονους αλγορίθμους οι απαιτήσεις αξιοπιστίας ικανοποιούνται πιο εύκολα.

1.6 Συνηθέστερες τοπολογίες δικτύων κατανεμημένων συστημάτων

Το δίκτυο ενός κατανεμημένου συστήματος αποτελείται από διεργασίες και το δίκτυο επικοινωνίας. Το δίκτυο επικοινωνίας μπορεί να αναπαρασταθεί ως ένας γράφος G=(V,E),όπου V είναι το σύνολο των κόμβων (κάθε κόμβος αντιστοιχεί σε μία διεργασία) και Ε είναι το σύνολο των πλευρών του γράφου. Ο γράφος που αντιστοιχεί σε ένα κατανεμημένο σύστημα ονομάζεται τοπολογία δικτύου του κατανεμημένου συστήματος και δεχόμαστε ότι κάθε τοπολογία είναι διασυνδεδεμένη (connected), δηλαδή ότι υπάρχει πάντα ένα μονοπάτι που ενώνει δύο οποιουσδήποτε κόμβους. Παρακάτω αναφέρονται οι συνηθέστερες τοπολογίες δικτύων με μεγαλύτερη έμφαση στις τοπολογίες δακτυλίου και δένδρου με τις οποίες θα ασχοληθούμε περοσσότερο στα επόμενα κεφάλαια.

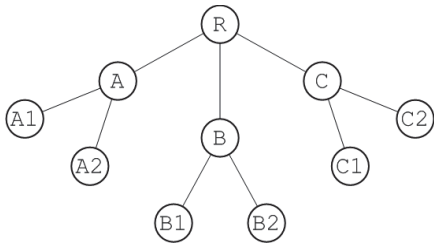
1.6.1 Δακτύλιος (ring)

Ένα σύγχρονο κατανεμημένο σύστημα αποτελείται από κάποιες διεργασίες οι οποίες είναι τοποθετημένες σε ένα δίκτυο δακτυλίου όπως φαίνεται στο σχήμα. Η κάθε διεργασία έχει μία μοναδική ταυτότητα, δεν μπορεί να όμως γνωρίζει τις ταυτότητες των άλλων διεργασιών, αλλά μπορεί να ξεχωρίσει τον δεξιόστροφο γείτονά της από τον αριστερόστροφο. Θεωρούμε ότι οι διεργασίες είναι αριθμιμένες από το 1 έως το n με δεξιόστροφη κατεύθυνση, όπως φαίνεται στο σχήμα, χωρίς όμως οι ίδιες οι διεργασίες ναα γνωρίζουν αυτό τον τρόπο αρίθμισης. Κάποια κατανεμημένα συστήματα χρησιμοποιούν τους δακτυλίους ως φυσική συνδεσμολογία των επεξαργαστών αλλά στις περισσότερες περιπτώσεις δημιουργήται ένας υπερκέιμενος δακτύλιος ως τοπολογία ελέγχου του κατανεμημένου συστήματος. Σε κάποιες περιπτώσεις χρειάζεται να οριστεί ένας συνδετικός δακτύλιος σε ένα γράφο όπου δύο διαδοχικοί κόμβοι είναι υποχρεωτικά γείτονες στο υποκείμενο φυσικό δίκτυο. Άλλες φορές χρειάζεται να οριστεί ένας εικονικός δακτύλιος όπου δύο διαδοχικοί κόμβοι δεν είναι υποχρεωτικά γείτονες.



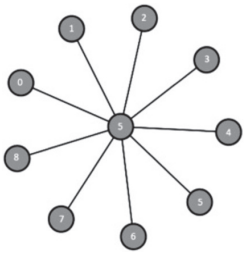
1.6.2 Δένδρα (trees)

Ένα δένδρο είναι ένας μη κατευθυνόμενος γράφος στον οποίο οποιεσδήποτε δύο κορυφές συνδέοντε με ένα και μόνο απλό μονοπάτι. Ορισμένα κατανεμημένα συστήματα χρησιμοποιούν τα δένδρα ως φυσική συνδεσμολογία των επεξαργαστών αλλά στις περισσότερες περιπτώσεις δημιουργήται ένα υπερκείμενο δένδρο ως τοπολογία ελέγχου του κατανεμημένου συστήματος, όπως συμβαίνει και στους δακτυλίους. Κάθε διασυνδεδεμένος γράφος G=(V,E) περιέχει πάντα ένα συνδετικό δένδρο G'=(V,E') το οποίο μπορεί να οριστεί μα διάφορα κριτήρια όπως ως συνδετικό δένδρο ελάχιστης διαμέτρου όπου ο συνολικός χρόνος ενός κατανεμημένου υπολογισμού ελαχιστοποιήται, ως συνδετικό δένδρο ελάχιστου βάρους που χρησιμοποιήται για την ελαχιστοποίηση του συνολικού κόστους επικοινωνίας, ως συνδετικό δένδρο κατά βάθος όπου χρησιμοποιήται σε συγκεκριμένα προβλήματα των κατανεμημένων συστημάτων όπως στη δρομολόγηση και τέλος ως συνδετικό δένδρο περιορισμένου βαθμού το οποίο μειώνει το τοπικό κόστος υπολογισμού σε κάθε κόμβο και κάθε κόμβος έχει τον ελάχιστο αριθμό γειτόνων.



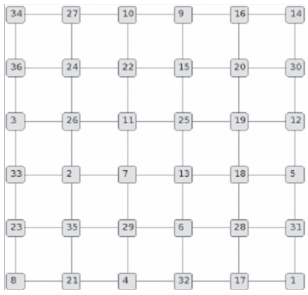
1.6.3 Άλλες τοπολογίες

Αστέρες (stars)
Οι αστέρες στην πραγματικότητα είναι μία ειδική περίπτωση δένδρου με βάθος 1. Η ρίζα του δένδρου ονομάζεται κέντρο του αστέρα.

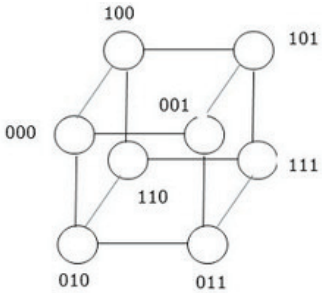


Κλίκες (cliques)
Στις κλίκες ή αλλιώς στους πλήρεις γράφους κάθε κόμβος συνδέεται απευθείας με τον καθένα από τους υπόλοιπους με μία πλευρά, δηλαδή κάθε κόμβος έχει βαθμό N-1.

Πλέγματα (grids) ή κυλινδρικά πλέγματα (torus)
Στο πλέγμα οι κόμβοι γνωρίζουν ποιοί γείτονες είναι πάνω, κάτω, δεξιά και αριστερά τους.

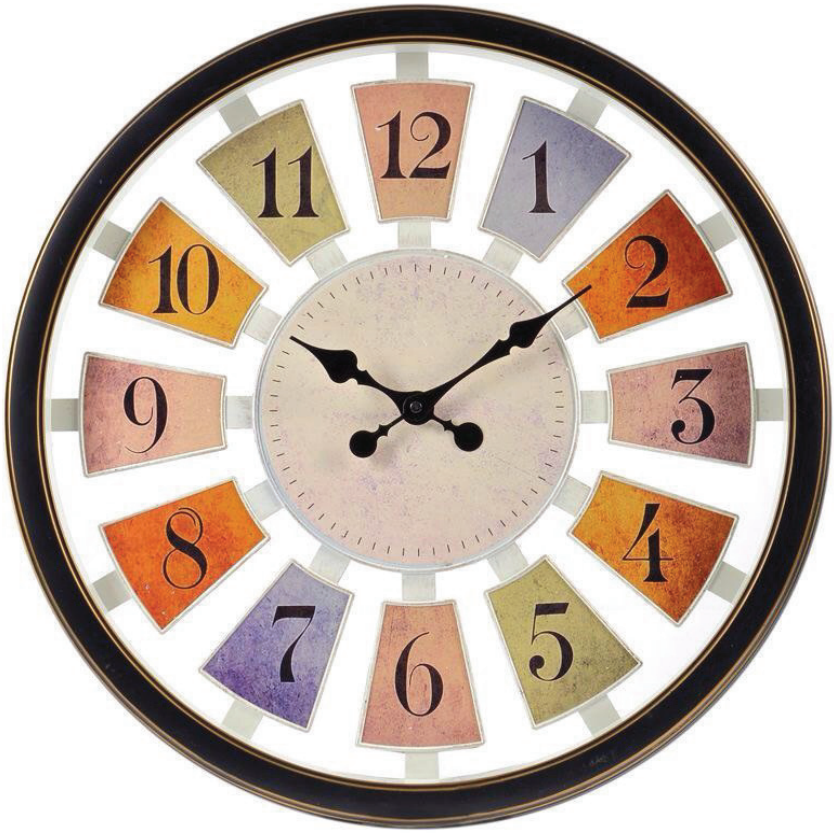


Υπερκύβοι (hypercubes)
Στους υπερκύβους ο αριθμός των κόμβων είναι πάντα δύναμη του 2 και η δύναμη αυτή ονομάζεται διάσταση του υπερκύβου. Αν η δύναμη είναι ίση με 3, τότε ο υπερκύβος θα έχει οκτώ επεξεργαστές που είναι αριθμημένοι ως εξής: 000, 001, 010, 011, 100, 101, 110, 111. Στους υπερκύβους οι παράλληλες πλευρές πρέπει να έχουν την ίδια ονομασία.



ΚΕΦΑΛΑΙΟ 2

ΠΡΟΒΛΗΜΑΤΑ ΣΤΑ ΚΑΤΑΝΕΜΗΜΕΝΑ ΣΥΣΤΗΜΑΤΑ



2.1 Μετάδοση πληροφορίας - Διάσχιση δικτύου

Κατά τον σχεδιασμό των κατανεμημένων αλγορίθμων στις διάφορες εφαρμογές προκύπτουν διάφορα προβλήματα. Τα προβλήματα αυτά σχετίζονται με την αναμετάδοση (broadcasting) της πληροφορίας όπως για παράδειγμα την διάδοση ενός start ή ενός termination μηνύματος. Στόχος είναι η επίτευξη ενός σφαιρικού συγχρονισμού στο δίκτυο μεταξύ των διεργασιών και ο έλεγχος της εκτέλεσης συγκεκριμένων γεγονότων σε κάθε διεργασία που κρατά μέρος της εισόδου της συνάρτησης. Η υλοποίηση των εργασίων που προαναφέρθηκαν γίνεται πάντα με το πέρασμα μηνυμάτων σύμφωνα με κάποιο προδιαγραφόμενο και τοπολογικά εξαρτώμενο σχήμα που εξασφαλίζει τη συμμετοχή όλων των διεργασιών. Τα προβλήματα που μόλις αναφέραμε είναι τόσο σημαντικά που η επίλυση περισσότερο πολύπλοκων προβλημάτων όπως για παράδειγμα το πρόβλημα εκλογής αρχηγού που θα δούμε παρακάτω, μπορούν να βασιστούν σε αυτά.

Οι αλγόριθμοι διάδοσης πληροφορίς με πέρασμα μηνυμάτων ονομάζονται κυματικοί αλγόριθμοι (wave algorithms). Οι κυματικοί αλγόριθμοι οδηγούνται πάντα σε τερματισμό, όπως επίσης και σε μία απόφαση σε ένα τουλάχιστον κόμβο. Η απόφαση αυτή έπεται μετά από ένα γεγονός (που έχει σχέση με το κυματικό πρωτόκολλο) σε κάθε κόμβο.

Στο μοντέλο αυτό του κατανεμημένου συστήματος, η επικοινωνία γίνεται μέσω της ανταλλαγής μηνυμάτων και υπάρχει πάντα ένα μονοπάτι που συνδέει κάθε ζευγάρι επεξεργαστών (διασυνδεδεμένο δίκτυο). Ο χρονισμός μπορεί να είναι σύγχρονος ή ασύγχρονος και το δίκτυο στατικό, δηλαδή δεν υπάρχουν αλλαγές στην τοπολογία του. Επίσης, τα κανάλια επικοινωνίας δεν έχουν κατευθύνσεις, θεωρούμε ότι είναι διπλής κατεύθυνσης και τέλος πρέπει να αναφέρουμε ότι κάθε αλγόριθμος είναι σχεδιασμένος για μία συγκεκριμένη τοπολογία.

Σε κάθε κυματικό αλγόριθμο, γίνεται διάκριση των διεργασιών σε αρχικοποιητές (initiators ή starters) και μη αρχικοποιητές (non-initiators ή followers). Μία διεργασία είναι αρχικοποιητής αν αρχίζει την εκτέλεση του τοπικού της αλγορίθμου τυχαία. Ένας μη-αρχικοποιητής εμπλέκεται στον αλγόριθμο μόνο όταν ένα μήνυμα του αλγορίθμου φτάνει και πυροδοτεί την εκτέλεση των τοπικών αλγορίθμων. Το πρώτο γεγονός του αρχικοποιητή είναι ένα εσωτερικό γεγονός ή ένα γεγονός αποστολής μηνύματος, ενώ το πρώτο γεγονός ενός μη-αρχικοποιητή είναι ένα γεγονός λήψης μηνύματος. Εάν υπάρχει ακριβώς ένας αρχικοποιητής σε κάθε υπολογισμό, τότε ο αλγόριθμος καλείται συγκεντρωτικός, ενώ αν ο αλγόριθμος αρχίσει τυχαία από ένα αυθαίρετο υποσύνολο των διεργασιών, τότε καλέιται αποκεντρωτικός. Συνήθως οι συγκεντρωτικοί αλγόριθμοι παρουσιάζουν καλύτερη πολυπλοκότητα μηνυμάτων. Επίσης, ένας αλγόριθμος μπορεί να σχεδιαστεί για μία συγκεκριμένη τοπολογία, όπως για παράδειγμα για έναν δακτύλιο ή ένα δέντρο. Τέλος ένας αλγόριθμος μπορεί να υποθέσει τη διαθεσιμότητα διαφορετικών τύπων αρχικής γνώσης στους επεξεργαστές, όπως την ταυτότητα διεργασίας, των αριθμό των αποφάσεων και την πολυπλοκότητα.

2.1.1 Πρωτόκολλο ECHO

Ο αλγόριθμος echo είναι απο τους σημαντικότερους κυματικούς αλγόριθμους και αναφέρεται σε δίκτυα γενικής τοπολογίας. Είναι κεντρικοποιημένο πρωτόκολλο γιατί αρχικοποιείται από έναν κόμβο και χρησιμοποιείται συχνά ως δομικό μέρος άλλων προβλημάτων με σκοπό τη μετάδοση πληροφορίας και το συγχρονισμό.Ο χρονισμός είναι ασύγχρονος. Ο αρχικοποιητής μεταδίδει την πληροφορία σε όλους τους άλλους κόμβους και στη συνέχεια όλοι οι μη – αρχικοποιητές θα μεταδώσουν – στείλουν πίσω πληροφορία στον αρχικοποιητή .

Η περιγραφή του πρωτοκόλλου ECHO είναι η εξής :
Ο αρχικοποιητής στέλνει σε όλους τους γείτονες το μήνυμα token και περιμένει να λάβει από όλους τους γείτονες το μήνυμα token και τέλος αποφασίζει. Οι μη-αρχικοποιητές όταν λάβουν το πρώτο token «σημειώνουν» τον αποστολέα ως πατέρα τους έπειτα αναμεταδίδουν το token σε όλους τους γείτονες, εκτός του πατέρα τους και όταν λάβουν από όλους τους γείτονες ένα token το στέλνουν στον πατέρα τους. Κάθε κόμβος στέλνει το πολύ ένα μήνυμα σε κάθε πλευρά. Επομένως κάποια στιγμή το πρωτόκολλο θα ολοκληρωθεί. Ο γράφος Τ που σχηματίζεται είναι ένα δένδρο γιατί ο αριθμός των πλευρών του είναι κατά ένα μικρότερος του αριθμού των κόμβων του. Το δένδρο Τ έχει ρίζα τον αρχικοποιητή. Πέρα απο τη διάδοση της πληροφορίας το πρωτοκολλο echo έχει τη πλεονέκτημα πως σχηματίζει το γεννητικό δένδρο του γράφου.

2.1.2 Πρωτόκολλο polling

Ο αλγόριθμος αναφέρεται σε πλήρεις γράφους ή αστέρες όπου υπάρχει ένα κανάλι επικοινωνίας για κάθε ζευγάρι κόμβων. Είναι ένας μόνο αρχικοποιητής και στην περίπτωση του αστέρα ο αρχικοποιητής είναι στο κέντρο. Η πληροφορία μεταφέρεται από τον αρχικοποιητή στους υπόλοιπους κόμβους.

Επεξήγηση του πρωτοκόλλου:
Ο αρχικοποιητής αρχικά στέλνει το token σε όλους τους γείτονες και στη συνέχεια περιμένει να λάβει το token από όλους τους γείτονες . Όταν συμβεί αυτό τότε εκτελεί ένα τοπικό κώδικα (decide) που εξαρτάται από τον γενικότερο αλγόριθμο. Οι μη-αρχικοποιητές περιμένουν να λάβουν το token και στη συνέχεια να στείλουν το token στον αρχικοποιητή.

2.1.3 Πρωτόκολλο του TARRY (1985)- Διάσχιση γενικού γράφου

Ο αλγόριθμος διάσχισης ενός γενικού γράφου δόθηκε απο τον Tarry το 1985. Ο αρχικοποιητής επιλέγει ένα κανάλι και στέλνει το token στον αντίστοιχο γείτονα. Ένας κόμβος που λαμβάνει για πρώτη φορά το token σημειώνει τον γείτονα απο όπου τον έλαβε ως πατέρα του και στέλνει το token σε έναν άλλο γείτονα του. Ένας κόμβος ποτέ δεν στέλνει δεύτερη φορά το token απο το ίδιο κανάλι. Όταν ένας κόμβος λάβει το token και ήδη το έχει στείλει σε όλους τους γείτονες του τότε το στέλνει στον πατέρα του. Ένας κόμβος εκτελεί την εντολή decide όταν έχει στείλει το token σε όλους τους γειτονές του. Από όλους τους κόμβους αποφασίζει μόνο ο αρχικοποιητής. Το πρωτόκολλο του TARRY σχηματίζει ένα γεννητικό (συνδετικό) δένδρο (spanning tree). Το γεννητικό αυτό δένδρο δεν είναι κατ' ανάγκη κατά βάθος(Depth First Search –DFS).

2.2 Πρόβλημα συγχρονισμού ρολογιών

Ρολόι σε έναν ηλεκτρονικό υπολογιστή ονομάζουμε το ηλεκτρονικό κύκλωμα που παρακολουθεί το χρόνο. Με το κύκλωμα αυτό, δημιουργήται από το λειτουργικό σύστημα το ρολόι λογισμικού που μπορεί να μετρήσει κατά προσέγγιση τον πραγματικό χρόνο. Σε ένα κατανεμημένο σύστημα όσοι επεξεργαστές υπαρχουν, τόσα θα είναι και τα ανεξάρτητα φυσικά ρολόγια. Οι κρύσταλοι χαλαζία όμως, από τους οποίους αποτελείται το ρολόι δεν είναι δυνατό πάντα να ταλαντώνονται με τον ίδιο τρόπο, λόγω διαφόρων αιτιών, όπως για παράδειγμα λόγω διαφορετικής θερμοκρασίας. Έτσι οι τιμές των ρολογιών καταλήγουν να αποκλίνουν μεταξύ τους όλο και περισσότερο. Η ταχύτητα ολίσθισης του ρολογιού (clock drift) είναι η μεταβολή της διαφοράς μεταξύ των τιμών του ρολογιού και ενός "τέλειου" ρολογιού αναφοράς ανά μονάδα χρόνου, όπως αυτή μετριέται από ένα τέλειο ρολόι. Για τα συνηθισμένα ρολόγια η ταχύτητα ολίσθισης μπορεί να είναι 10^{-5} ή 10^{-6} secs/sec. Αυτό έχει ως αποτέλεσμα ένα συνηθισμένο ρολόι να αποκλίνει από το τέλειο ρολόι περίπου 1sec ανά 11,6 ημέρες.

Ο συγχρονισμός των ρολογιών στα κατανεμημένα συστήματα επιτυγχάνεται δύσκολα και μόνο κάτω από ορισμένες συνθήκες. Στον αστρονομικό χρόνο, ο χρόνος ορίζεται σε σχέση με την περιστροφή της γης γύρω από τον άξονά της και γύρω από τον ήλιο. Ηλιακή μέρα είναι το διάστημα μεταξύ δύο διαδοχικών αφίξεων του ήλιου στο υψηλότερο δυνατό σημείο του ουρανού, και ηλιακό δευτερόλεπτο είναι το 1/86400 της ηλιακής μέρας. Η διάρκεια όμως της ημέρας δεν είναι σταθερή, αφού υπάρχει μία μακροπρόθεσμη τάση ελάττωσης της ταχύτητας της γης(αύξηση της διάρκειας της ημέρας). Προκειμένου να αντιμετωτιστεί το πρόβλημα αυτό, οι αστρονόμοι υπολόγιζαν το δευτερόλεπτο με βάση πολλές ηλιακές μέρες (ως μέσο όρο) και το ονόμασαν μέσο ηλιακό δευτερόλεπτο.

Ο μέσος χρόνος Γκρίνουιτς (Greenwich Mean Time, GMT), ή ώρα Γκρίνουιτς, είναι ο τοπικός μέσος ηλιακός χρόνος του πρώτου μεσημβρινού της Γης ο οποίος διέρχεται από το Γκρίνουιτς της Αγγλίας. Από τον μεσημβρινό αυτό αρχίζει η μέτρηση του γεωγραφικού μήκους στη Γη, καθώς επίσης και του χρόνου κατά την μεσημβρινή διάβαση του δίσκου του ήλιου από αυτόν. Την GMT χρησιμοποιούν όλα τα αστεροσκοπεία της Γης, οι επιστημονικές αποστολές, όλα τα αεροσκάφη, τα ποντοπόρα πλοία, οι σταθμοί παράκτιας επικοινωνίας, τα διεθνή αεροδρόμια, τα διεθνή ειδησεογραφικά πρακτορεία, τα κέντρα έρευνας και διάσωσης,τα μεγάλα στρατιωτικά κέντρα κ.τ.λ.

Από το 1948, για τη μέτρηση του χρόνου δεν χρησιμοποιούνται πλέον οι αστρονομικές παρατηρήσεις αλλά οι φυσικές ιδιότητες του ατόμου του καισίου 133 που δημιουργεί την "ατομική ώρα". Ο χρόνος που χρειάζεται το άτομο του καισίου 133 για να εκτελέσει 9.192.631.770 μεταπτώσεις ονομάζεται ατομικό δευτερόλεπτο. Αρκετές χώρες υπολογίζουν την ώρα, μέσω εθνικών εργαστηρίων, με βάση ατομικά ρολόγια καισίου 133. Περιοδικά κάθε εργαστήριο αναφέρει την ώρα του στο "Διεθνές Γραφείο Ώρας του Παρισιού" (BIH) το οποίο υπολογίζει τη μέση τιμή και δημιουργεί τη Διεθνή Ατομική Ώρα (International Atomic Time, TAI). TAI είναι το μέσο πλήθος χτύπων των ρολογιών από τα μεσάνυχτα της 1ης Ιανουαρίου 1958 διά του αριθμού 9.192.631.770. Η Ελλάδα συμμετέχει στην ατομική ώρα με τρία ρολόγια καισίου 133 που διαθέτει στο Εργαστήριο Χρόνου και Συχνότητας στο Ελληνικό Ινστιτούτο Μετεορολογίας (ΕΙΜ). Το πρόβλημα της ατομικής ώρας είναι ότι 86.400 δευτερόλεπτα του TAI αντιστοιχούν σε ένα χρονικό διάστημα μικρότερο κατά τρία χιλιοστά του δευτερολέπτου από τη μέση ηλιακή μέρα, αυτό συμβαίνει επιδή η ηλιακή μέρα συνεχώς μεγαλώνει. Παρόλα αυτά, το παραπάνω πρόβλημα έχει ως αποτέλεσμα το μεσημέρι να πέφτει όλο και νωρίτερα.

Το πρόβλημα της ατομικής ώρας έρχεται να λύσει το Διεθνές Γραφείο Ώρας προσθέτοντας δευτερόλεπτα στο TAI όταν η διαφορά με την ηλιακή ώρα φτάνει τα 800 χιλιοστά του δευτερολέπτου (ως τώρα έχουν εισαχθεί 30 επιπλέον δευτερόλεπτα). Η ώρα που προκύπτει από αυτό ονομάζεται Παγκόσμια Συντονισμένη Ώρα (Universal Coordinated Time – UTC) και έχει αντικαταστήσει τη μέση ώρα Γκρίνουιτς. Η διόρθωση στην ώρα UTC λαμβάνεται σοβαρά υπόψη σε διάφορα συστήματα που βασίζονται στον πραγματικό χρόνο όπως στο σύστημα μεταφοράς της ηλεκτρικής ενέργειας και στους ηλεκτρονικούς υπολογιστές.

Οι αλγόριθμοι συγχρονισμού ρολογιών στα κατανεμημένα συστήματα χωρίζονται σε δύο κατηγορίες, στους αλγορίθμους που εφαρμόζονται όταν είναι διαθέσιμος ένας εξυπηρετητής ώρας, και στους αλγορίθμους που εφαρμόζονται όταν δεν είναι διαθέσιμος ένας εξυπηρετητής ώρας. Η πρώτη κατηγορία έχει σκοπό να συγχρονίσει όλους τους κόμβους με την ώρα του εξυπηρετητή, ενώ στην δεύτερη περίπτωση, οι αλγόριθμοι προσπαθούν να προσφέρουν ένα σχετικό χρονισμό μεταξύ των κόμβων.

2.2.1 Αλγοριθμος του Cristian

Ο αλγόριθμος του Cristian ανήκει στην πρώτη κατηγορία από αυτές που αναφέραμε παραπάνω, όπου είναι διαθέσιμος ένας εξυπηρετητής ώρας. Ο συγκεκριμένος αλγόριθμος βασίζεται στην εργασία του Cristian (1989) και σε προηγούμενες εργασίες. Κάθε κόμβος στέλνει περιοδικά ένα μήνυμα στον εξυπηρετητή ώρας ρωτώντας την τρέχουσα ώρα. Στη συνέχεια, ο κόμβος αυτός, με βάση την απάντηση που λαμβάνει, διορθώνει το ρολόι του. Στη διόρθωση πρέπει να ληφθούν υπόψη και οι καθυστερήσεις του δικτύου. Ποτέ όμως ένα ρολόι δεν πρέπει να γυρίσει προς τα πίσω, γιατί αυτό θα μπορούσε να προκαλέσει σοβαρά προβλήματα όπως το ότι η ώρα ενός αντικειμενικού αρχείου που μεταγλωττίστηκε αμέσως μετά την αλλαγή θα είναι προγενέστερη από την ώρα τροποποίησης του πηγαίου αρχείου, αν η τροποοίηση αυτή έγινε ακριβώς πριν την αλλαγή. Μία τέτοια διόρθωση θα πρέπει να γίνει βαθμιαία. Ένα ακόμη πρόβλημα εδώ είναι ότι για να φθάσει πίσω στον αποστολέα η απάντηση από τον εξυπηρετητή απαιτήται κάποιος χρόνος ο οποίος εκτός από το ότι είναι μη μηδενικός είναι και απροσδιόριστος αφού ποικίλλει ανάλογα με το φόρτο του δικτύου. Ο Cristian κατάφερε να χρονομετρήσει αυτή τη διαφορά. Ο αποστολέας μπορεί να καταγράψει με ακρίβεια το διάστημα που μεσολαβεί μεταξύ της αποστολής της αίτησης και της λήψης της απάντησης. Τα δύο γεγονότα μετριούνται από το ίδιο ρολόι, οπότε το αποτέλεσμα θα είναι σχετικά εκριβές. Έχοντας ως στόχο την βελτίωση της ακρίβειας, ο Cristian πρότεινε να υπάρχουν παραπάνω από ένας διακομιστές ώρας από τους οποίους για παράδειγμα να βγαίνει ένας μέσος όρος. Αλλιώς μπορεί να προτιμάται το μήνυμα που φτάνει πρώτο αφού πιθανότατα έχει συναντήσει τη λιγότερη κυκλοφορία στο δρόμο του και για αυτό μπορεί να θεωρηθεί το πιό αντιπροσωπευτικό του καθαρού χρόνου διάδοσης.

2.2.2 Αλγόριθμος Berkeley

Ο αλγόριθμος του Berkeley σε αντίθεση με τον αλγόριθμο του Cristian δεν χρησιμοποιεί εξυπηρετητή ώρας. Στον αλγόριθμο του Cristian ο διακομιστής ώρας είναι παθητικός, λαμβάνει αιτήσεις για την ώρα και το μόνο που κάνει είναι να απαντά στις αιτήσεις αυτές. Στο σύστημα Berkeley UNIX, χρησιμοποιήται η ακριβώς αντίθετη προσέγγιση (Gusella και Zatti, 1989). Εδώ ο εξυπηρετητής ώρας έχει ενεργό ρόλο αφού κάνει περιοδικούς ελέγχους σε κάθε μηχάνημα έχοντας σκοπό να πάρει πληροφορία για την τρέχουσα ώρα του. Σύμφωνα με τις απαντήσεις υπολογίζει μία μέση ώρα και δίνει εντολή σε όλα τα άλλα μηχανήματα να

συντονίσουν τα ρολόγια τους σε αυτή την ώρα, ή να μειώσουν την ταχύτητα των ρολογιών τους προκειμένου να επιτευχθεί η μείωση που χρειάζεται. Η ώρα του εξυπηρετητή πρέπει να ρυθμίζεται περιοδικά, με μη αυτόματο τρόπο από τον χειριστή. Ας δούμε ένα απλό παράδειγμα για την καλύτερη κατανόηση του αλγορίθμου. Υποθέτουμε ότι έχουμε έναν εξυπηρετητή με ώρα 3:00. Ο διακομιστής ρωτάει τα μηχανήματα για την ώρα τους. Έστω ότι λαμβάνει τις απαντήσεις 2:50 και 3:25. Τότε ο εξυπηρετητής θα πρέπει να προχωρίσει το ρολόι του στις 3:05 που είναι ο μέσος όρος (θεωρώντας τις καθυστερήσεις δικτύου αμελητέες) και να ζητήσει από τα μηχανήματα να συντονίσουν τα ρολόγια τους σε αυτή την ώρα. Το ρολόι που έστειλε την απάντηση 2:50 θα πρέπει να ρυθμιστεί ώστε να δείχνει 3:05 ενώ το ρολόι που έδειχνε 3:25 θα πρέπει να μειώσει την ταχύτητά του μέχρι να δείχνει την ίδια ώρα με αυτή του εξυπηρετητή.

2.2.3 Πρωτόκολλο NTP

Ένας από τους πιο διαδεδομένους αλγόριθμους στο internet είναι το Πρωτόκολλο Δικτυακής Ώρας (Network Time Protocol, NTP). Το NTP χρησιμοποιήται για το συγχρονισμό ρολογιών στο internet και καταφέρνει να συγχρονίσει τα ρολόγια σε παγκόσμιο επίπεδο με ακρίβεια της κλίμακας των 1-50 msec. Αυτή η ακρίβεια επιτυγχάνεται μέσω της χρήσης προηγμένων αλγορίθμων συγχρονισμού ρολογιών. Πιο συγκεκριμένα, όλοι οι κόμβοι του δικτύου οργανώνονται σε μία λογική ιεραρχία κατά στρώματα. Στο πρώτο στρώμα (ρίζα) βρίσκονται οι πρωτεύοντες εξυπηρετητές, ενώ στο δεύτερο βρίσκονται οι δευτερεύοντες εξυπηρετητές που επικοινωνούν κατευθείαν με τους πρωτεύοντες του πρώτου επιπέδου. Αυτή η ιεραρχία συνεχίζει με τον ίδιο τρόπο, κ έτσι κάθε επίπεδο επικοινωνεί με το αμέσως επόμενο. Ο συγχρονισμός γίνεται με ανταλλαγή μηνυμάτων μεταξύ δύο διαδοχικών/γειτονικών στρωμάτων. Πάντα λαμβάνονται υπόψη οι καθυστερήσεις μετάδοσης. Τα στρώματα που βρίσκονται πιο κοντά στη ρίζα απολαμβάνουν τη μεγαλύτερη ακρίβεια ως προς το συγχρονισμό των ρολογιών, αφού εισάγεται νέο σφάλμα σε κάθε νέο στρώμα συγχρονισμού. Η λογική δομή του δικτύου είναι δυνατόν να αλλάξει αν π.χ. ένας πρωτεύων εξυπηρετητής δεν έχει πλέον κατευθείαν προσπέλαση σε ώρα UTC. Οι εξυπηρετητές που βρίσκονται στο ίδιο επίπεδο ανταλλάσουν συνεχώς μηνύματα για τον καλύτερο συγχρονισμό τους αλλά και για τον εντοπισμό μη αξιόπιστων τιμών ώρας.

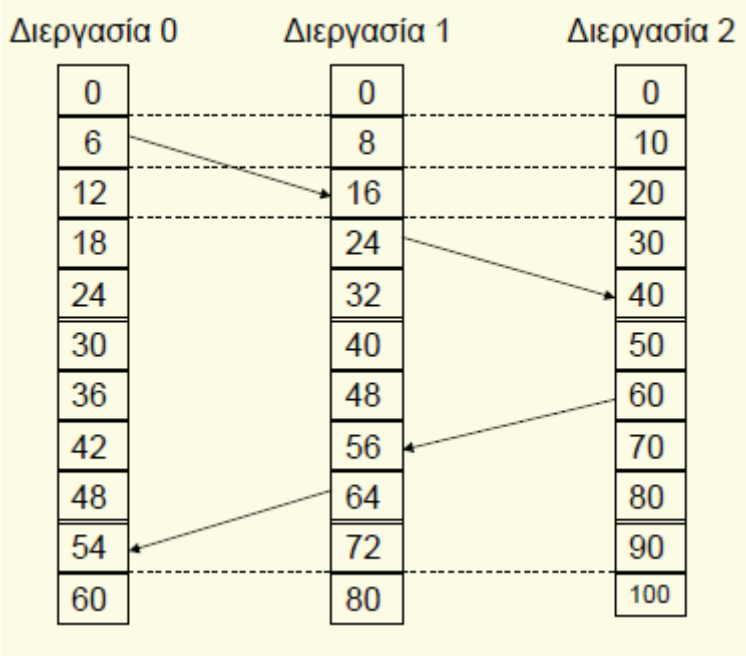
2.2.4 Λογικά ρολόγια του Lamport

Ο συγχρονισμός των ρολογιών με την πραγματική ώρα είναι σχεδόν αδύνατο να επιτευχθεί. Το πιο σημαντικό όμως στις περισσότερες περιπτώσεις είναι απλά να συμφωνούν όλα τα μηχανήματα στην ίδια ώρα, ώστε να μπορεί να καθοριστεί η σωστή σειρά που συμβαίνουν τα γεγονότα σε ένα κατανεμημένο σύστημα. Ο Lamport (1978) πρότινε ένα μηχανισμό που εξυπηρετούσε αυτό το σκοπό τον οποίο ονόμασε “λογικά ρολόγια”.

Για το συγχρονισμό των ρολογιών ο Lamport όρισε τη σχέση “συμβαίνει-πριν” . Για παράδειγμα, αν έχουμε τη σχέση a-->b, σημαίνει ότι όλες οι διεργασίες συμφωνούν ότι το a συνέβη πριν από το b. Αν τα a και b ανήκουν στην ίδια διεργασία (κόμβο) και το a έγινε πριν το b, τότε η σχέση a-->b είναι αληθής. Αν το a είναι το γεγονός αποστολής ενός μηνύματος και το b είναι το γεγονός λήψης του μηνύματος αυτού, τότε η σχέση a-->b και πάλι είναι αληθής.ένα μήνυμα δεν μπορεί να ληφθεί πριν σταλεί, αλλά ούτε και την ίδια ώρα που στέλνεται, αφού χρειάζεται μία ορισμένη και μη μηδενική ποσότητα χρόνου για να φθάσει. Η σχέση “συμβαίνει-πριν” είναι μεταβατική, δηλαδή αν a-->b και b-->c τότε a-->c. Αν τα a και

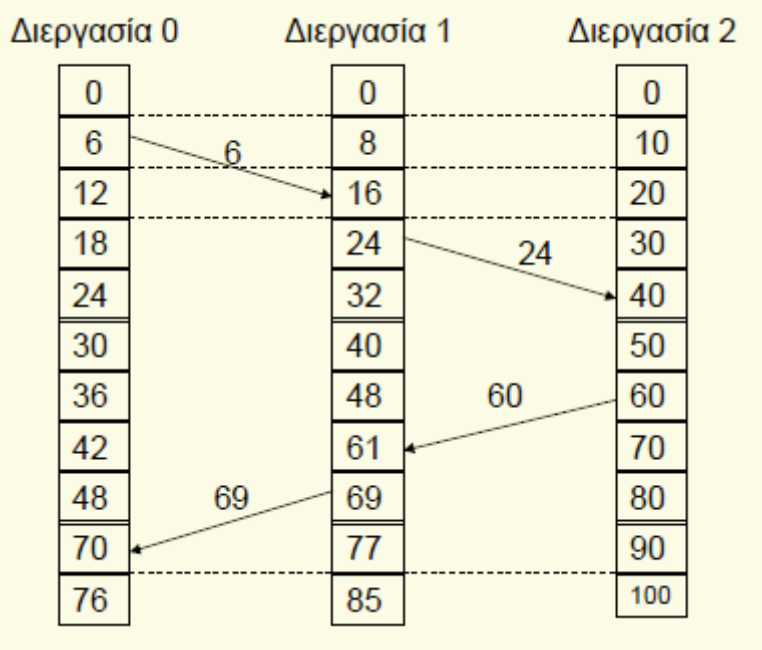
b λάβουν χώρα σε διαφορετικές διεργασίες που δεν ανταλλάσουν μηνύματα ούτε καν έμμεσα, (δηλαδή μέσω τρίτων), τότε δεν είναι αληθές ούτε το a-->b ούτε το b-->a. Σε αυτή την περίπτωση λέγεται ότι τα γεγονότα γίνονται ταυτόχρονα και δεν μπορούμε να κάνουμε καμία υπόθεση σχετικά με το ποιό γεγονός συνέβει πρώτο.
Χρειαζόμαστε ένα τρόπο μέτρησης του χρόνου τέτοιο ώστε για κάθε συμβάν a να μπορούμε να δώσουμε την τιμή ώρας C(a), με την οποία να συμφωνούν όλες οι διεργασίες. Για αυτές τις διεργασίες πρέπει να ισχύει ότι, αν a-->b, τότε C(a)-->C(b). Αναδιατυπώνοντας τις παραπάνω συνθήκες, προκύπτει ότι αν a και b είναι δύο συμβάντα της ίδιας διεργασίας και το a συμβεί πριν το b, τότε ισχύει ότι C(a)<C(b). Επίσης, αν a είναι το γεγονός αποστολής ενός μηνύματος και b το γεγονός λήψης του μηνύματος από άλλη διεργασία, τότε κάθε διεργασία πρέπει να συμφωνεί ότι C(a)<C(b). Όπως έχει αναφερθεί αρκετές φορές ως τώρα, η ώρα δεν πρέπει ποτέ να μειώνεται, παραμόνο να γίνονται διορθώσεις με την πρόσθεση θετικών τιμών.

Στον αλγόριθμο των λογικών ρολογιών, αν συμβεί ένα εσωτερικό γεγονός ή ένα γεγονός send στη διεργασία p_i τότε $C_i=C_i+1$. Μαζί με κάθε μήνυμα m που στέλνει η διεργασία p_i μεταφέρεται και η τιμή (χρονοσφραγίδα) του γεγονότος send(m). Όταν συμβεί ένα γεγονός receive(m) σε μία διεργασία p_i τότε επιλέγεται η μεγαλύτερη τιμή τοπικού ρολογιού, μεταξύ της τρέχουσας τιμής του τοπικού ρολογιού και της χρονοσφραγίδας που μεταφέρει το μήνυμα m. Στη συνέχεια προκειμένου να είμαστε σίγουροι ότι πάντα η ώρα της λήψης θα είναι μεγαλύτερη από την ώρα της αποστολής, η τιμή αυτή αυξάνεται κατά 1.



εικόνα 3

Στην εικόνα 3 βλέπουμε τρεις διεργασίες στις οποίες γίνεται ανταλλαγή τεσσάρων μηνυμάτων. Η κάθε διεργασία έχει το δικό της ρολόι που λειτουργεί με το δικό του ρυθμό. Ο αλγόριθμος του Lamport καλείται να διορθώσει τα ρολόγια.



εικόνα 4

Στην εικόνα4 έχει εφαρμοστεί ο αλγόριθμος του Lamport. Παρατηρούμε ότι το μήνυμα που φεύγει από τη διεργασία 0 στο χτύπο 6 φτάνει στη διεργασία 1 στο χτύπο 16, πράγμα πολύ πιθανό οπότε δεν γίνεται καμία τροποποίηση. Στη συνέχεια στο χτύπο 24 της διεργασίας 1 φεύγει μήνυμα το οποίο καταλήγει στο χτύπο 40 της διεργασίας 2, πολύ πιθανό επίσης, οπότε και πάλι δε χρειάζεται να γίνει καμία τροποποίηση. Αντιθέτως όταν από τη διεργασία 2 φεύγει μήνυμα στο χτύπο 60 δεν είναι δυνατό να φτάνει στην επόμενη διεργασία στο χτύπο 56, οπότε η διεργασία 1 ρυθμίζει το ρολόι της ένα χτύπο πιο μπροστά. Το ίδιο ισχύει και στο επόμενο μήνυμα, η διεργασία 0 έχει ρυθμίσει το ρολόι της στο 70 (69+1).

2.2.5 Διανυσματικά ρολόγια

Τα διανυσματικά ρολόγια ή αλλιώς διανυσματικές χρονοσφραγίδες καλύπτουν την αδυναμία των λογικών ρολογιών του Lamport. Στα λογικά ρολόγια Lamport ισχύει η σχέση ότι αν a-->b, τότε C(a)<C(b), δεν ισχύει πάντα όμως το αντίστροφο. Δηλαδή, αν γνωρίζουμε ότι C(a)<C(b), δεν μπορούμε να είμαστε σίγουροι ότι το γεγονός a προηγήθηκε του b. Το πρόβλημα είναι ότι οι χρονοσφραγίδες του Lamport δεν αποτυπώνουν την αιτιότητα. Στο παραπάνω παράδειγμα μπορεί η σειρά να τειρήται αυστηρά αλλά αυτό δεν έχει καμία σημασία όταν δύο άρθρα ή αποκρίσεις δεν σχετίζονται. Η αιτιότητα μπορεί να αποτυπωθεί με τις διανυσματικές χρονοσφραγίδες. Στις διανυσματικές χρονοσφραγίδες, ο παραλήπτης είναι ενήμερος για τον αριθμό των συμβάντων τα οποία έχουν λάβει χώρα σε μία διεργασία, αλλά και για τον αριθμό των συμβάντων που έχουν λάβει χώρα σε άλλες διεργασίες πριν την αποστολή ενός μηνύματος από μία διεργασία.

2.2.6 Καθολικές καταστάσεις

Συχνά είναι χρήσιμο να γνωρίζουμε την καθολική κατάσταση στην οποία βρίσκεται μία δεδομένη χρονικά στιγμή ένα σύστημα. Η καθολική κατάσταση (global state) ενός κατανεμημένου συστήματος αποτελείται από την τοπική κατάσταση κάθε διεργασίας και τα μηνύματα τα οποία έχουν σταλεί αλλά δεν έχουν παραδοθεί ακόμη. Η τοπική κατάσταση

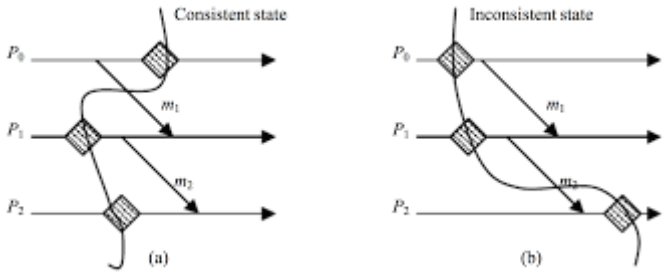
ορίζεται με βάση τις ανάγκες της εφαρμογής που εκτελείται από το κατανεμημένο σύστημα. Στην περίπτωση για παράδειγμα ενός κατανεμημένου συστήματος βάσεων δεδομένων, η τοπική κατάσταση μπορεί να αποτελείται από εκείνες τις εγγραφές οι οποίες αποτελούν μέρος της βάσης αυτής και να μην περιλαμβάνει προσωρινές εγγραφές που χρησιμοποιούνται για υπολογισμούς.

Η συλλογή της καθολικής κατάστασης ενός κατανεμημένου συστήματος δεν είναι καθόλου εύκολη διαδικασία, αυτό συμβαίνει λόγω των πολλών κόμβων, των καθυστερήσεων επικοινωνίας των καναλιών και των διαφορετικών ρολογιών των επεξεργαστών. Από την άλλη μερία όμως, η γνώση της καθολικής κατάστασης ενός κατανεμημένου συστήματος μπορεί να είναι αρκετά χρήσιμη για διάφορους λόγους. Για παράδειγμα, όταν γνωρίζουμε ότι οι τοπικοί υπολογισμοί έχουν σταματήσει και ότι δεν υπάρχουν άλλα μηνύματα τα οποία δεν έχουν παραδοθεί ακόμη, τότε πιθανότατα το σύστημα να βρίσκεται πλέον σε μία κατάσταση όπου δεν μπορεί να υπάρξει πρόοδος. Σε αυτή την περίπτωση μπορούμε να καταλήξουμε σε δύο συμπεράσματα, είτε ότι στο σύστημα υπάρχει αδιέξοδο, είτε ότι ένας κατανεμημένος υπολογισμός φυσιολογικά έχει τερματιστεί.

Κατανεμημένο στιγμιότυπο (distributed snapshot)

Οι Chandy και Lamport (1985) εισήγαγαν την έννοια του κατανεμημένου στιγμιότυπου προκειμένου να γίνεται καταγραφή της καθολικης κατάστασης ενός κατανεμημένου συστήματος με έναν απλό και σαφή τρόπο. Ένα κατανεμημένο στιγμιότυπο αντανακλά μία κατάσταση στην οποία μποσεί να βρεθεί το κατανεμημένο σύστημα. Ένα στιγμιότυπο πρέπει να αντανακλά μια συνεπή καθολική κατάσταση του συστήματος. Πιο συγκεκριμένα αυτό σημαίνει ότι, αν έχουμε καταγράψει πως μία διεργασία έχει λάβει ένα μήνυμα από μία άλλη, τότε πρέπει να έχουμε καταγράψει επίσης ότι η τελευταία έστειλε όντως εκείνο το μήνυμα. Σε αντίθετη περίπτωση, ένα στιγμιότυπο θα περιέχει την καταγραφή μηνυμάτων, τα οποία έχουν ληφθεί αλλά δεν έχουν σταλεί ποτέ, γεγονός που δεν θέλουμε να συμβεί. Όμως από την άλλη, επιτρέπεται μια διεργασία να έχει στείλει ένα μήνυμα το οποίο να μην έχει παραληφθεί ακόμα.

Η έννοια μιας καθολικής κατάστασης μπορεί να αναπαρασταθεί γραφικά με αυτό που ονομάζουμε τομή και φαίνεται στην παρακάτω εικόνα. Στην εικόνα 5(a) παρουσιάζεται μία συνεπής τομή με τη γραμμή που διασχίζει τον άξονα του χρόνου των διργασιών. Η τομή αντιπροσωπεύει το τελευταίο συμβάν που έχει καταγραφεί για κάθε διεργασία. Σε αυτή την περίπτωση, μπορεί να επαληθευτεί εύκολα ότι όλες οι καταγεγραμμένες λήψεις μηνυμάτων αντιστοιχούν σε κάποιο καταγεγραμμένο συμβάν αποστολής. Αντιθέτως στην εικόνα 5(b) παρατηρούμε μία ασυνεμή τομή. Η λήψη του μηνύματος m_2 από τη διεργασία P_2 έχει καταγραφεί, αλλά το στιγμιότυπο δεν περιέχει αντίστοιχο συμβάν αποστολής. Έτσι, ο παραλήπτης του m_2 δεν μπορεί να ανγνωριστεί με αυτή την τομή.



Εικόνα 5

Για να απλοποιήσουμε την εξήγηση του αλγορίθμου για τη λήψη ενός κατανεμημένου στιγμιοτύπου, προϋποθέτουμε ότι το κατανεμημένο σύστημα μπορεί να αναπαρασταθεί ως ένα σύνολο διεργασιών οι οποίες συνδέονται μεταξύ τους μέσω μονόδρομων καναλιών επικοινωνίας από σημείο σε σημείο.

2.3 Πρόβλημα εκλογής αρχηγού

Στους κατανεμημένους αλγορίθμους πολλές φορές είναι απαραίτητο μία διεργασία να ορίζεται ως αρχηγός ώστε να ενεργεί ως συντονιστής. Ο αρχηγός είναι απαραίτητος σε αρκετές περιπτώσει όπως για παράδειγμα στη μετάδοση πληροφορίας (wave protocols), όταν χρειάζεται να υπάρχει ένας αχικοποιητής (initiator), στον αμοιβαίο αποκλεισμό σε κατανεμημένα συστήματα, στην εξισορρόπηση φορτίου (load balancing) και στην ανάκαμψη του συστήματος μετά από ένα πρόβλημα. Σε ένα πρωτόκολλο εκλογής ιχύει ότι ο κάθε κόμβος εκτελεί τον ίδιο τοπικό αλγόριθμο, ο οποίος είναι αποκεντρωμένος, δηλαδή μπορεί να αρχικοποιηθεί από οποιοδήποτε αριθμό κόμβων. Επίσης, κάθε εκτέλεση του αλγορίθμου έχει ως αποτέλεσμα μία τερματική κατάσταση στην οποία μία μόνο διεργασία είναι σε κατάσταση elected ενώ όλες οι υπόλοιπες είναι σε κατάσταση defeated. Ανάλογα με το πρωτόκολλο που χρησιμοποιήται, οι ηττημένοι κόμβοι είναι δυνατόν να μη γνωρίζουν πως έχουν τελικά ηττηθεί και να βρίσκονται σε άλλες καταστάσεις όπως awake, sleep, candidate.

2.3.1 Εκλογή σε δίκτυα τοπολογίας δένδρου

Αν η τοπολογία του δικτύου είναι δένδρο ή υπάρχει διαθέσιμο γεννητικό δένδρο, μπορεί να γίνει εκλογή αρχηγου με χρήση του αλγορίθμου διάδοσης κύματος του δένδρου (wave tree algorithm). Υποθέτουμε ότι κάθε κόμβος έχει μία μοναδική ταυτότητα. Σε κάποιες περιπτώσεις μπορεί να μην είναι όλα τα φύλλα του δένδρου αφυπνησμένα, οπότε θα πρέπει πριν ξεκινήσει η διαδικασία να αφυπνιστούν. Οι κόμβοι οι οποίοι θέλουν να ξεκινήσουν το πρωτόκολλο εκλογής (αρχικοποιητές) υποθέτουμε ότι μπαίνουν αυτόματα σε κατάσταση awake, ενώ οι υπόλοιποι βρίσκονται σε αδράνεια, δηλαδή σε κατάσταση sleep. Οι awake κόμβοι στέλνουν στους γείτονές τους ένα μήνυμα αφύπνησης wake_up. Ένας sleep κόμβος που λαμβάνει το μήνυμα wake_up αφυπνίζεται και η κατάστασή του γίνεται αυτομάτως awake. Ένας awake κόμβος καταλαβαίνει ότι όλοι οι κόμβοι είναι awake όταν λάβει από όλους τους γείτονές του wake_up μήνυμα, οπότε και η διαδικασία εκλογής μπορεί να ξεκινήσει.

Η διαδικασία ξεκινά από τα φύλλα του δένδρου τα οποία στέλνουν μήνυμα <tok,id> στον πατέρα τους, όπου το id είναι η ταυτότητά τους. Κάθε κόμβος, λαμβάνοντας υπόψιν και τη δική του ταυτότητα, συγκρίνει τις ταυτότητες και κρατάει στην τοπική μεταβλητή v_p τη μικρότερη ταυτότητα. Σε περίπτωση που ένας κόμβος δεν λάβει μήνυμα από ένα από τα παιδιά του, τον ονομάζουμε υπόλοιπο κόμβο (αυτό θεωρούμε πως είναι αληθές αρχικά για τα φύλλα), τότε στέλνει το μήνυμα <tok,v_p> στον υπόλοιπο κόμβο. Οι κόμβοι που θα λάβουν ένα μήνυμα από κάθε γείτονά τους, αποφασίζουν για την μικρότερη ταυτότητα σε όλο το δίκτυο και επομένως βρίσκουν τον αρχηγό. Οι κόμβοι που αποφασίζουν πρώτοι στέλνουν ένα μήνυμα σε όλους τους γείτονές τους (εκτός από τον υπόλοιπο κόμβο), ενημερώνοντας τους υπόλοιπους για τον αρχηγό.

Δύο είναι οι κόμβοι που τελικά αποφασίζουν και βρίσκουν τον αρχηγό. Αυτοί ενημερώνουν τους υπόλοιπους κόμβους. Αν τα κανάλια δεν είναι FIFO τότε είναι δυνατόν ένας κόμβος να λάβει ένα μήνυμα τύπου <tok,r> και μετά να λάβει ένα μήνυμα τύπου wake_up. Στην περίπτωη αυτή μπορεί να αποθηκεύσει το <tok,r> για να το επεξεργαστεί αφου ολοκληρωθεί η πρώτη φάση.

2.3.2 Πρωτόκολλο LeLann

Ο LeLann ήταν εκείνος που παρουσίασε για πρώτη φορά (1977) έναν αλγόριθμο για την εκλογή αρχηγού στα κατανεμημένα συστήματα και η πρότασή του αφορούσε κυκλικά τοποθετημένους επεξεργαστές (τοπολογία δακτυλίου). Σύμφωνα με τον αλγόριθμο του LeLann, κάθε επεξεργαστής-αρχικοποιητής εκπέμπει ένα μήνυμα με το οποίο γνωστοποιεί την ταυτότητά του στους υπόλοιπους. Το μήνυμα αυτό αφού έχει περάσει από όλους τους κόμβους κάνοντας μια πλήρη περιστροφή επιστρέφει στον αρχικοποιητή, έτσι κάθε αρχικοποιητής γνωρίζει τις ταυτότητες όλων των επεξεργαστών. Ο αρχικοποιητής τώρα μπορεί να συγκρίνει όλες τις ταυτότητες και να εκλέξει αρχηγό, για παράδειγμα αυτόν με την μεγαλύτερη ταυτότητα. Οι κόμβοι που δεν είναι αρχικοποιητές, έχουν ως μοναδικό ρόλο να προωθούν στον επόμενο κόμβο τα μηνύματα που λαμβάνουν και δεν έχουν συμμετοχή στην εκλογή αρχηγού. Θεωρήται ότι τα κανάλια είναι FIFO και κάθε αρχικοποιητής πρώτα παράγει το δικό του μήνυμα και μετά μπορεί να λάβει κάποιο άλλο. Ο αλγόριθμος του LeLann είναι πολύ απλός αλλά μπορεί να έχει μεγάλο αριθμό διακινούμενων μηνυμάτων αν για παράδειγμα κάθε κόμβος είναι και αρχικοποιητής.

2.3.3 Πρωτόκολλο Chang & Roberts

Οι Chang & Roberts βελτιώνουν τον αλγόριθμο του LeLann, απαιτώντας κατά μέσο όρο, λιγότερα μηνύματα σε γραμμικό χρόνο, παρόλο που διατηρεί τον αριθμό μηνυμάτων για τη χειρότερη περίπτωση του αλγορίθμου του LeLann. Έχει τα προτερήματα της εύκολης υλοποίησης, της επεκτασιμότητας σε άλλες τοπολογίες και της πολύ καλής απόδοσης κάτω από ορισμένες συνθήκες χρονισμού του δικτύου. Χαρακτηριστικά του αλγορίθμου είναι η άγνοια του συνολικού αριθμού των επεξεργαστών που παίρνουν μέρος στην εκλογή, η κίνηση μηνυμάτων προς τη μια κατεύθυνση μόνο και η δυνατότητα μη ταυτόχρονης αρχής του αλγορίθμου απ' όλους τους επεξεργαστές, όπου δίνει κλύτερα αποτελέσματα από εκείνα του σύγχρονου ξεκινήματος.

Κάθε επεξεργαστής γνωρίζει την ταυτότητά του. Με την παρακάτω διαδικασία, θα εκλεγεί αρχηγός ο επεξεργαστής με τη μεγαλύτερη ταυτότητα. Μόλις ξεκινήσει η διαδικασία εκλογής, ο επεξεργαστής παράγει ένα μήνυμα με την ταυτότητά του και το προωθεί στον αριστερό γείτονά του. Ο επεξεργαστής που λαμβάνει με τη σειρά του το μήνυμα αυτό, συγκρίνει την ταυτότητα με την δική του και αν η δική του είναι μικρότερη, τότε είναι πιθανό, το μήνυμα που έλαβε, να είναι αυτό του νικητή, οπότε το προωθεί προς τα αριστερά. Αν σε αντίθετη περίπτωση, το μήνυμα που έλαβε έχει μικρότερη ταυτότητα από τη δική του, τότε αποκλείεται να μεταφέρει τον αριθμό του νικητή και το μήνυμα αγνοείται. Τέλος, αν η ταυτότητα του μηνύματος είναι ίδια με τη δική του, τότε αυτός εκλέγεται αρχηγός και η διαδικασία τερματίζεται. Ο αλγόριθμος βρίσκει τη μοναδική μέγιστη ταυτότητα επεξεργαστή στο σύστημα αφού η μοναδική φορά κίνησης των μηνυμάτων και η τοπολογία του δακτυλίου αναγκάζουν ένα μήνυμα να περάσει από όλους τους επεξεργαστές μέχρι να φτάσει ξανά πίσω στον εκπομπό του. Μόνο το μήνυμα με την μεγαλύτερη ταυτότητα καταφέρνει να συμπληρώσει ένα πλήρη κύκλο χωρίς να κοπεί από κάποιον ενδιάμεσο επεξεργαστή. Άρα, μόνο ο επεξεργαστής με τη μεγαλύτερη ταυτότητα δέχεται πίσω το μήνυμά του και έτσι εκλέγεται αρχηγός.

2.3.4 Πρωτόκολλο Peterson/Dolev-Klawe- Rodeh

Ο αλγόριθμος των Peterson/Dolev-Klawe- Rodeh βελτιώνει τον αλγόριθμο των Chang & Roberts επιτυγχάνοντας τη μειωμένη πολυπλοκότητα μηνυμάτων για την εκλογή αρχηγού σε δακτυλίους μονής κατεύθυνσης. Κατά την έναρξη αυτού του αλγορίθμου, όλες οι ταυτότητες είναι ενεργές, και σε κάθε γύρο κάθε ενεργή ταυτότητα συγκρίνει τον εαυτό της με τις ενεργές γειτονικές ταυτότητες. Ας πούμε εδώ ότι θέλουμε να εκλέξουμε αρχηγό τη μικρότερη ταυτότητα. Αν μία ενεργή ταυτότητα συγκριθεί με τις γειτονικές και είναι η μικρότερη, τότε παραμένει ενεργή και προχωρά στον επόμενο γύρο, αλλιώς γίνεται παθητική. Με αυτό τον τρόπο, έπιτα από κάποιους γύρους θα έχει μείνει μόνο μία ταυτότητα ενεργή, η οποία θα είναι και η νικήτρια. Οι ταυτότητες οι οποίες γίνονται παθητικές, συνεχίζουν να λαμβάνουν μηνύματα αλλά δεν κάνουν συγκρίσεις, παραμόνο τα προωθούν. Ο αλγόριθμος μπορεί να εφαρμοστεί επίσης και σε δακτυλίους διπλής κατεύθυνσης.

2.3.5 Πρωτόκολλο Itai & Rodeh

Ο αλγόριθμος Itai & Rodeh είναι επίσης βασισμένος στον αλγόριθμο Chang & Roberts, χρησιμοποιηείται για την επίλυση προβλημάτων σε ασύγχρονους δακτυλίους και δίνει λύση στην περίπτωση που το πρόβλημα εκλογής αρχηγού είναι κάπως διαφοροποιημένο. Η διαφοροποίηση είναι ότι δεν υπάρχουν διακεκριμένες ταυτότητες στο δίκτυο, άρα οι ταυτότητες των επεξεργαστών δεν μπορούν να χρησιμοποιηθούν αφού μπορεί να υπάρχουν ίδιες ταυτότητες στο δίκτυο. Παρατηρούμε μία συμμετρία στο δίκτυο την οποία θα πρέπει να διασπάσουμε. Και σε αυτό τον αλγόριθμο αναζητάμε τον επεξεργαστή με το μεγαλύτερο (ή μικρότερο) αριθμό και τα μηνύματα μπορούν να κινηθούν προς μία και μόνο κατεύθυνση.

Οι επεξεργαστές διαλέγουν κάποιους τυχαίους φυσικούς αριθμούς για ταυτότητές τους. Έπειτα γίνεται ταυτόχρονα η εκλογή του μεγαλύτερου και ο έλεγχος για τη μοναδικότητα των αριθμών. Αν υπάρξει κόλλημα στον αλγόριθμο, τότε αυτός επαναλαμβάνεται. Το μέγεθος του δακτυλίου πρέπει να είναι γνωστό σε όλους τους επεργαστές. Οι επαναλήψεις του αλγορίθμου σημειώνονται από από τον αριθμό της φάσης ο οποίος μεταφέρεται και από τα μηνύματα του αλγορίθμου.

Λόγω του ασυγχρονισμού του δικτύου και της ομοιομορφίας των επεξεργαστών απέναντι στον αλγόριθμο, δημιουργούνται δύο βασικά προβλήματα τα οποία πρέπει να αντιμετωπιστούν ώστε να μπορέσει να λειτουργήσει σωστά ο αλγόριθμος.

Το πρώτο πρόβλημα είναι ότι ένας επεξεργαστής θέλει πρώτα να σιγουρευτεί ότι υπάρχει τουλάχιστον ένας ακόμα που προσπαθεί να εκλεγεί ώστε να αποφασίσει να σταματήσει την προσπάθεια για την εκλογή του. Σε αντίθετη περίπτωση ο αλγόριθμος θα έπεφτε σε αδιέξοδο, αν δηλαδή δεν διεκδικούσε την εκλογή κανένας επεξεργαστής. Εδώ το πρόβλημα αντιμετοπίζεται αφού οι επεξεργαστές που αντιλαμβάνονται ότι υπάρχουν ισχυρότεροι υποψήφιοι επεξεργαστές και δεν πρόκειται να εκλεγούν, αποσύρρονται από την διαδικασία της εκλογής. Αν ως εδώ δεν υπάρξει μοναδικός νικητής τότε συνεχίζουν μόνο οι επεξεργαστές που παρέμειναν ενεργοί στο τέλος της προηγούμενης φάσης. Ο αλγόριθμος είναι αδύνατο να φτάσει σε αδιέξοδο γιατί αποκλείεται το ενδεχόμενο να παραιτηθούν όλοι οι επεξεργαστές την ίδια στιγμή. Κάθε φορά που τρέχει ο αλγόριθμος θα υπάρχει κάποιος (ή κάποιοι) που θα είναι μεγαλύτερος (ή μεγαλύτεροι).

Το δεύτερο πρόβλημα είναι ότι ένας επεξεργαστής δυσκολεύεται να διακρίνει τα δικά του μηνύματα έχοντας ώς μόνο δεδομένο τις ταυτότητες, αφού η αρίθμιση των επεξεργαστών δεν είναι εγγυημένα μοναδική. Το συγκεκριμένο πρόβλημα μπορεί να λυθεί μόνο αν όλοι οι επεξεργαστές γνωρίζουν ποιό είναι το πλήθος των επεξεργαστών που υπάρχουν στον δακτύλιο. Έτσι όλες οι ταυτότητες θα ξεκινούν τη διαδρομή τους με τη συνοδεία ενός μετρητή με την τιμή 1 ο οποίος θα αυξάνεται κατά 1 κάθε φορά που θα περνάει από έναν επεξεργαστή, άρα κάθε φορά που θα φτάνει σε έναν επεξεργαστή η τιμή που θα είναι ίση με το πλήθος, τότε θα ξέρει πως έχει ολοκληρωθεί ένας κύκλος και έχει ξανά το δικό του μήνυμα.

Τώρα μπορεί να γίνει και η περιγραφή του αλγορίθμου. Μόλις λοιπόν γίνει αντιληπτό από έναν κόμβο ότι έχει ξεκινήσει η διαδικασία εκλογής αρχηγού στο δακτύλιο, τότε επιλέγει τυχαία έναν φυσικό αριθμό που θα τον χρησιμοποιήσει ως ταυτότητα. Στη συνέχεια στέλνει ένα μήνυμα προς την κατεύθυνση κίνησης των μηνυμάτων το οποίο περιέχει τον αριθμό της φάσης στην οποία βρίσκεται ο αλγόριθμος, την ταυτότητα του επεξεργαστή που διάλεξε, τον μετρητή που αναφέραμε παραπάνω και μία μεταβλητή η οποία σηματοδοτεί την ύπαρξη ή όχι κάποιου μοναδικού νικητή. Σκοπός κάθε μηνύματος είναι η επιστροφή του στον κόμβο από τον οποίο ξεκίνησε και η εκλογή αυτού του κόμβου. Είναι φανερό ότι όταν συμβεί αυτό θα σημαίνει ότι η ταυτότητα του κόμβου φυσικά θα είναι η μέγιστη, αλλά και μοναδική. Ένα μήνυμα που κινήται μέσα στον δακτύλιο μπορεί είτε να προωθηθεί, είτε να απορριφθεί. Ένα μήνυμα μπορεί να απορριφθεί αν έχει μικρότερη φάση από αυτή στην οποία βρίσκεται ο αλγόριθμος τη δεδομένη στιγμή, ή αν έχει μικρότερη ταυτότητα από ότι ο κόμβος στον οποίο βρίσκεται. Στε αντίθετη περίπτωση ο κόμβος στον οποίο βρίσκεται το μήνυμα γίνεται ανενεργός και το μήνυμά μας προωθήται. Στην περίπτωση που δύο ταυτότητες είναι ίσες, οι κόμβοι επιλέγουν νέες ταυτότητες, ξεκινάνει νέα φάση και η διαδικασία επαναλαμβάνεται.

2.4 Δρομολόγηση

Δρομολόγηση είναι η διαδικασία κατά την οποία ένας κόμβος επιλέγει ένα ή περισσότερους γειτονικούς κόμβους για να προωθήσει ένα πακέτο πληροφορίας, προκειμένου αυτό να φτάσει στον τελικό προορισμό του. Αυτό επιτυγχάνεται μέσω των αλγορίθμων δρομολόγισης. Για τη δρομολόγηση είναι απαραίτητο να υπάρχουν πληροφορίες για την τοπολογία του δικτύου σε κάθε κόμβο. Υπάρχουν λοιπόν οι πίνακες δρομολόγησης, οι οποίοι συλλέγουν και οργανώνουν αυτές τις πληροφορίες. Έτσι η διαδικασία της δρομολόγησης χωρίζεται σε δύο μέρη, τον υπολογισμό και ενημέρωση των πινάκων, που πραγματοποιήται κατά την αρχηκοποίηση και όταν αλλάξει η τοπολογία, και την προώθηση των πακέτων πληροφορίας με βάση τους πίνακες δρομολόγησης.

Ας δούμε σε αυτό το σημείο ποιά είναι τα κριτήρια της δρομολόγησης.

- Ορθότητα (correctness): Ο αλγόριθμος πρέπει να μεταφέρει σωστά τα πακέτα ως τον τελικό τους προορισμό.
- Πολυπλοκότητα (comlexity): Ο αλγόριθμος για τον υπολογισμό των τιμών των πινάκων θα πρέπει τα καταναλώνει όσο το δυνατό λιγότερους πόρους, δηλαδή χρόνο, χώρο και αριθμό μηνυμάτων.
- Αποδοτικότητα (efficiency): Η δρομολόγηση πρέπει να γίνεται μέσω του καλύτερου μονοπατιού ώστε να αποφεύγονται οι καθυστερήσεις λόγω συσσώρευσης μηνυμάτων σε μέρη του δικτύου. Ένας αλγόριθμος λέγεται βέλτιστος όταν χρησιμοποιεί τα συντομότερα μονοπάτια.

- Βιωσιμότητα (robustness): Οι πίνακες δρομολόγησης πρέπει να ενημερώνονται όταν υπάρχουν αλλαγές στην τοπολογία.
- Προσαρμοστικότητα (adaptiveness): Ο αλγόριθμος πρέπει να διατηρεί την ισορροπία σχετικά με τη ροή των πακέτων στο δίκτυο και να προτιμά κόμβους με μικρότερο φόρτο.
- Δικαιοσύνη (fairness): Ο αλγόριθμος θα πρέπει να εξυπηρετεί τον κάθε κόμβο στον ίδιο βαθμό.

Το δίκτυο όπως γνωρίζουμε αναπαρίσταται με έναν γράφο, οι κόμβοι του γράφου αναπαριστούν τις διαδικασίες (υπολογιστές) του δικτύου, και οι ακμές τις συνδέσεις του. Ένας αλγόριθμος μπορούμε να πούμε ότι είναι βέλτιστος αν πληρεί μία από τις παρακάτω προυποθέσεις.

1. Ελάχιστες μεταβάσεις (minimum hop): Ο αριθμός των μεταβάσεων από κόμβο σε κόμβο καθορίζουν το κόστος του μονοπατιού. Σκοπός είναι να βρεθεί το μονοπάτι με τον ελάχιστο αριθμό μεταβάσεων.
2. Μικρότερες αποστάσεις (shortest path): Σε κάθε ακμή του γράφου αντιστοιχεί ένας πάντα θετικός αριθμός που είναι το βάρος. Για να υπολογίσουμε το κόστος του μονοπατιού αθροίζουμε τα βάρη των ακμών του. Σκοπός είναι να βρεθεί το μονοπάτι με το ελάχιστο κόστος.
3. Ελάχιστη καθυστέρηση (minimum delay): Σε κάθε ακμή αντιστοιχούμε δυναμικά ένα βάρος το οποίο εξαρτάται από την κυκλοφορία των πακέτων στην ακμή αυτή. Σκοπός μας εδώ είναι να γίνει επιλογή του μονοπατιού εκείνου που έχει την μικρότερη καθυστέρηση λόγω κίνησης.

2.4.1 Chandy-Misra

Οι Chandy και Misra πρότειναν έναν αλγόριθμο ο οποίος υπολογίζει όλα τα ελάχιστα μονοπάτια προς έναν προορισμό, χρησιμοποιώντας έναν κατανεμημένο υπολογισμό, ο οποίος αρχικοποιήται από έναν απλό κόμβο και στον οποίο συμμετέχουν και άλλοι κόμβοι μόνο αφού λάβουν κάποιο μήνυμα. Κάθε κόμβος γνωρίζει του γείτονές του και το κόστος των μονοπατιών που οδηγούν σε αυτούς.

2.4.2 Floyd-Warshal(μη κατανεμημένος)

Ο μη κατανεμημένος αλγόριθμος των Floyd και Warshal λύνει το πρόβλημα υπολογισμού του συντομότερου μονοπατιού μεταξύ όλων των ζευγαριών των κόμβων ενός δικτύου. Ο αλγόριθμος αυτός μπορεί να υπολογίσει τη μικρότερη απόσταση μεταξύ δύο οποιονδείποτε κόμβων.

2.4.3 Toueg

Ο Toueg (1980) βασίστηκε στον αλγόριθμο των Floyd και Warshal προκειμένου να υπολογίσει συγχρόνως τους πίνακες δρομολόγησης σε όλους τοους κόμβους ενός δικτύου με κατανεμημένο τρόπο. Θα μπορούσαμε να πούμε πιο απλά ότι ο Toueg πήρε τον αλγόριθμο των Floyd και Warshal και τον μετέτρεψε σε κατανεμημένο χρησιμοποιόντας τις κατάλληλες τοπικές μεταβλητές σε κάθε κόμβο και την κατάλληλη ανταλλαγή μηνυμάτων. Το πρωτόκολλο του Toueg υπολογίζει για κάθε ζευγάρι κόμβων το συντομότερο μονοπάτι μεταξύ τους και αποθηκεύει το πρώτο κανάλι του μονοπατιού στον ένα κόμβο.

Στο σύστημά μας ισχύει ότι:

- Κάθε κύκλος στο δίκτυο έχει ένα θετικό βάρος.
- Κάθε κόμβος στο δίκτυο αρχικά γνωρίζει τα χαρακτηριστικά των άλλων κόμβων.
- Κάθε κόμβος γνωρίζει ποιοί από τους κόμβους είναι γείτονές του, αφού αποθηκεύονται σε πίνακα, και γνωρίζει επίσης τα βάρη των ακμών που εξέρχονται από αυτόν.

Ο αλγόριθμος του Toueg έχει αρκετά καλή πολυπλοκότητα και χρησιμοποιεί τα βέλτιστα μονοπάτια σε ένα δίκτυο. Δεν έχει όμως ανοχή σε αλλαγές, αφού όταν η τοπολογία του δικτύου αλλάξει πρέπει να γίνουν οι υπολογισμοί πάλι από την αρχή. Όλοι οι κόμβοι πρέπει να επιλέγουν από κοινού τον ίδιο κόμβο για το επόμενό τους βήμα. Για να επιτευχθεί αυτό ο κάθε κόμβος πρέπει να γνωρίζει από την αρχή τα ονόματα των υπόλοιπων κόμβων. Για να γίνει αυτό πρέπει πριν την εκτέλεση του αλγορίθμου του Toueg να γίνει χρήση ενός κυματικού αλγορίθμου.

2.4.4 Merlin-Segall

Οι Merlin και Segall πρότειναν έναν αλγόριθμο ο οποίος υπολογίζει τους πίνακες δρομολόγησης για κάθε προορισμό ξεχωριστά. Ο αλγόριθμος δεν βελτιώνει αυτόν του Toueg αφού οι υπολογισμοί για διαφορετικούς κόμβους είναι ανεξάρτητοι. Για κάθε προορισμό v ο αλγόριθμος ξεκινά με ένα δένδρο T_v που "δείχνει" προς τον v και επαναληπτικά ενημερώνει το δένδρο αυτό ώστε να γίνει ένα βέλτιστο sink tree για τον κόμβο v.

Ένα γεννητικό δένδρο του οποίου όλοι οι κόμβοι δείχνουν προς τον v λέμε ότι είναι ένα sink tree προς τον v. για να πούμε ότι είναι βέλτιστο sink tree προς τον v θα πρέπει για κάθε v που ανήκει στο V να υπάρχει ένα δένδρο τέτοιο ώστε για κάθε κόμβο που ανήκει στο V, το μονοπάτι από το u στο v να μέσα από το T_v να είναι ένα βέλτιστο μονοπάτι.

ΚΕΦΑΛΑΙΟ 3

ΥΛΟΠΟΙΗΣΗ ΕΠΙΛΕΓΜΕΝΩΝ ΑΛΓΟΡΙΘΜΩΝ ΣΕ JAVA

3.1 Αλγόριθμος Echo

3.1.1 EchoClient.java

```
import java.io.*;
import java.net.*;

public class EchoClient {

   public static void main(String[] args) throws IOException {

      // Ip tou server
      String serverHostname = new String ("127.0.0.1");


         System.out.println ("Attemping to connect to host "+serverHostname + " on port 10008.");

      Socket echoSocket = null;
      PrintWriter out = null;
      BufferedReader in = null;

      try {
         // Dimiourgia neou socket me thn ip kai port to 10008
         echoSocket = new Socket(serverHostname, 10008);

         // Anoigei ton PrintWriter gia na grafei sto socket
         out = new PrintWriter(echoSocket.getOutputStream(), true);

         // Anoigei neo BufferReader gia na diavazei apo to socket
         in = new BufferedReader(new InputStreamReader( echoSocket.getInputStream()) );

      } catch (UnknownHostException e) {
         System.err.println("Don't know about host: " + serverHostname);
         System.exit(1);
      } catch (IOException e) {
         System.err.println("Couldn't get I/O for the connection to: " + serverHostname);
         System.exit(1);
      }

      // Anoigei neo BufferReader gia na diavazei apo to pliktrologio
         BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
         String userInput;

      System.out.println ("Type Message (\"Bye\" to quit)");


      // Oso o xristis pliktrologei lexeis
         while ((userInput = stdIn.readLine()) != null)
      {
```

```
            // To grafei sto socket to mhnyma tou xrhsth
               out.println(userInput);

            // An grapsei Bye tote termatizei to while
            if (userInput.equals("Bye"))
               break;

               System.out.println("echo: " + in.readLine());
            }

         // Kleinei tous porous pou eixe anoixei
            out.close();
            in.close();
            stdIn.close();
            echoSocket.close();
      }
}
```

3.1.2 EchoServer.java

```
import java.net.*;
import java.io.*;

public class EchoServer extends Thread
{
   protected Socket clientSocket;

   public static void main(String[] args) throws IOException
   {
      ServerSocket serverSocket = null;

      try {
         // Dimiourgei neo socket gia ton server sto port 10008
         serverSocket = new ServerSocket(10008);
         System.out.println ("Connection Socket Created");
         try {

            // O server trexei gia panta
            while (true) {

               System.out.println ("Waiting for Connection");

               // Perimenei mexri na syndetei neos client
               new EchoServer (serverSocket.accept());
            }
          }
          catch (IOException e){

             System.err.println("Accept failed.");
```

```
                System.exit(1);
            }
        }
        catch (IOException e){
            System.err.println("Could not listen on port: 10008.");
            System.exit(1);
        }
        finally
        {
            try {
                // Kleinei to socket tou server
                serverSocket.close();
            }
            catch (IOException e){
                System.err.println("Could not close port: 10008.");
                System.exit(1);
            }
        }
    }

    // Constructor gia ton EchoServer pou dexetai to Socket
    private EchoServer (Socket clientSoc)
    {
        clientSocket = clientSoc;

        // me thn run trexei h start pou einai apo kato
        start();
    }

    public void run()
    {
        System.out.println ("New Communication Thread Started");

        try {
            // Anoigei neo PrintWriter gia to socket tou server pou grafei
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

            // Anoigei neo BufferReader gia to socket tou server pou diavazei
                                    BufferedReader in = new BufferedReader( new
InputStreamReader( clientSocket.getInputStream()) );

            String inputLine;

            // Oso o client grafei
            while ((inputLine = in.readLine()) != null) {

                System.out.println ("Server: " + inputLine);

                // O server apantaei me thn idia lexi
                out.println(inputLine);
```

```
            if (inputLine.equals("Bye."))
                break;
        }

        // Kleinei tous porous
         out.close();
         in.close();
         clientSocket.close();
      }
      catch (IOException e)
      {
         System.err.println("Problem with Communication Server");
         System.exit(1);
      }
   }
}
```

3.2 Αλγόριθμος εκλογής αρχηγού σε τοπολογία δακτυλίου

3.2.1 RingImplement.java

```
import java.io.*;
import java.net.*;
import java.time.Clock;
import java.util.*;
import java.util.logging.*;


public class RingImplement extends Thread {
   int pid, portNo, delay, ID;
   ServerSocket listenSock;
   Socket socket;
   Timer clock;
   GuiShell guiBox;
   boolean active = true, restart,  init = true;


   RingImplement(int ID, int portNo, GuiShell guiElement, boolean restart) {
      this.pid = ID;
      this.portNo = portNo;
      this.guiBox = guiElement;
      this.ID = ID;
      this.restart = restart;

      delay = ID*4* 1000;
   }

   public void timer() throws IOException{

         listenSock = new ServerSocket(portNo);
      listenSock.setSoTimeout(delay);
   }

   public String polls() {

      System.out.println("Process: " + pid + " Starting election");
      String msg = "Election - " + pid;
      guiBox.outputStatus("Process " + pid + " : " + msg);
      return msg;
   }
   public void run() {

      try {
         timer();
         sendToSock();
      } catch (IOException ex) {
         Logger.getLogger(RingImplement.class.getName()).log(Level.SEVERE, null, ex);
```

```
    }
  }

  public boolean startElection(boolean bool){

      String message = polls();
    pushMsg(message);
    bool = false;
    return bool;
  }

  public void sendToSock() throws IOException {

    while (active) {
      try {
        if (restart == true) {
          restart =  startElection(restart);
        }
        if (pid == 1 && init == true) {
          init = startElection(init);
        }
        socket = listenSock.accept();
                                BufferedReader intoProcess = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        String receiveMsg = intoProcess.readLine();
        System.out.println("Process: " + pid + " " + receiveMsg);
        decodeMessage(receiveMsg);

      } catch (SocketTimeoutException x) {
        String message = polls();
        pushMsg(message);

      } finally {
        socket.close();
      }
    }
  }

  public int newPort(int ID) {

    int portNxt = ID + 50000;
    if (ID == GuiShell.N) {
      portNxt = 50000;
    }
    return portNxt + 1;
  }

  public int lastPro(int ID) {

    int lastPort = ID;
```

```
    if (lastPort == 1) {
        lastPort = GuiShell.N+1;
    }
    return lastPort - 1;
}

@SuppressWarnings({ "unused", "deprecation" })
public void decodeMessage(String text) {

    String message = text, header, proStart = "any", proStart2;

    int cordPro = 0;
    String elec = "Election";
    String crash = "Crash";
    String restart = "Restart";
    String cord = "Co-ordinator";
    String alive = "Alive";

    StringTokenizer decodeMsg = new StringTokenizer(text);
    header = decodeMsg.nextToken();


    switch (header) {

        case "Election":
            if (text.length() > 12) {
                proStart = text.substring(11, 12);
            }
            if (!proStart.equals(String.valueOf(pid))) {
                pushMsg(message + "," + pid);
                guiBox.outputStatus("Process " + pid + " : " + message + "," + pid);
            } else {
                for (int k = 11; k < text.length(); k++) {
                    if (cordPro < Integer.parseInt(text.substring(k, k + 1))) {
                        cordPro = Integer.parseInt(text.substring(k, k + 1));
                    }
                    k++;
                }
                message = "Co-ordinator - " + cordPro;
                pushMsg(message);
                guiBox.outputStatus("Process " + pid + " : " + message);
            }
            break;
        case "Co-ordinator":
            proStart2 = text.substring(text.length() - 1);
            if (!proStart2.equals(String.valueOf(pid))) {
                pushMsg(message);
                guiBox.outputStatus("Process " + pid + " : " + message);
            } else {
                if (clock == null) {
```

```
                clock = new Timer();
                clock.schedule(new TimerTask() {
                    @Override
                    public void run() {
                        String mssg = "Alive - " + pid;
                        pushMsg(mssg);
                    }
                }, 0, 4 * 1000);
            }
        }
        break;
    case "Alive":

        String creatorProcess = text.substring(text.length() - 1);
        if (clock != null && pid < Integer.parseInt(creatorProcess)) {
            clock.cancel();
        }
        if (lastPro(Integer.parseInt(creatorProcess)) != pid) {
            pushMsg(text);
        } else {
            pushMsg("Process Alive");
        }
                break;
    case "Crash":
        try {
            listenSock.close();
            if (clock != null) {
                clock.cancel();
            }
            active = false;
            guiBox.outputStatus("Process " + pid + ": Crashed");
            this.stop();
        } catch (IOException ex) {
             Logger.getLogger(RingImplement.class.getName()).log(Level.SEVERE, null,
ex);
        }
        break;
    case "Restart":
        ID = pid;
        break;

    }
}


@SuppressWarnings("static-access")
public void pushMsg(String message) {

try {
    int port = newPort(ID);
```

```
            socket = new Socket("127.0.0.1", port);
            PrintWriter toProcess = new PrintWriter(socket.getOutputStream());
            this.sleep(150);
            toProcess.println(message);
            toProcess.close();
            socket.close();
        } catch (ConnectException ex) {
            if (ID == GuiShell.N) {
                ID = 0;
            }
            ID++;
            pushMsg(message);
        } catch (UnknownHostException ex) {
            Logger.getLogger(RingImplement.class.getName()).log(Level.SEVERE, null, ex);
        } catch (IOException ex) {
            Logger.getLogger(RingImplement.class.getName()).log(Level.SEVERE, null, ex);
        } catch (InterruptedException ex) {
            Logger.getLogger(RingImplement.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

3.2.2 GuiShell.java

```
import java.awt.Color;
import java.io.IOException;
import java.io.PrintWriter;
import java.net.ConnectException;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;


// --------------------Created by Abhinav Bansal -----------------------------

public class GuiShell {

    // PLithos komvwn
    public static int Plithos = 4;
    public static final int N = Plithos+1;
```

```
RingImplement process;

private static JFrame MainWindow = new JFrame();

private static JButton Start = new JButton("Start Process");
private static JButton B_crash = new JButton("crash");
private static JButton B_restart = new JButton("restart");
private static JLabel L_Message = new JLabel("Select process ");
private static JLabel L_Conv = new JLabel();
public static JTextArea TA_status = new JTextArea();
public static JScrollPane SP_Conversation = new JScrollPane();
public static JComboBox jComboBox1 = new javax.swing.JComboBox();

public static void main(String args[]){
   BuildMainWindow();
   Initialise();
}


private static void Initialise() {

   Start.setEnabled(true);
}


private static void BuildMainWindow() {

   MainWindow.setTitle("main");
   MainWindow.setSize(100, 500);
   MainWindow.setVisible(true);
   ConfigurationMainWindow();
   algoRithm();
}


public void startAction(java.awt.event.ActionEvent evt) {

   for (int i = 1; i < N; i++) {
      process = new RingImplement(i, 50000 + i, this, false);
      process.start();
      jComboBox1.addItem("Process " + i);
   }
}


public void restartAction(java.awt.event.ActionEvent evt) {

   for (int i = 1; i < N; i++) {
      try {
```

```
            Socket socket = new Socket("127.0.0.1", 50000 + i);
            PrintWriter pw = new PrintWriter(socket.getOutputStream());
            pw.println("Restart");
            pw.close();
            socket.close();
        } catch (ConnectException ex) {
            System.out.println("Process " + i + " is crashed");
        } catch (UnknownHostException ex) {
            Logger.getLogger(GuiShell.class.getName()).log(Level.SEVERE, null, ex);
        } catch (IOException ex) {
            Logger.getLogger(GuiShell.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    String restart = jComboBox1.getSelectedItem().toString();
    int ID = Integer.parseInt(restart.substring(restart.length() - 1));
    process = new RingImplement(ID, 50000 + ID, this, true);
    process.start();
    outputStatus("Process " + ID + "has been restarted");

}

  private static void algoRithm() {

    Start.addActionListener(new java.awt.event.ActionListener()
    {
        @Override
        public void actionPerformed(java.awt.event.ActionEvent evt)
        {
              GuiShell gui = new GuiShell();
             gui.startAction(evt);
        }
    });

    //_____________________CRASH BUTTON_____________________________
    B_crash.addActionListener( new java.awt.event.ActionListener()
    {
        @Override
        public void actionPerformed(java.awt.event.ActionEvent evt)
        {
             crashActionPerform(evt);
        }

        private void crashActionPerform(java.awt.event.ActionEvent evt) {

            String crash = jComboBox1.getSelectedItem().toString();
            try {
                                        Socket socket = new Socket("127.0.0.1", 50000 +
Integer.parseInt(crash.substring(crash.length() - 1)));
                PrintWriter pw = new PrintWriter(socket.getOutputStream());
```

```
                pw.println("Crash");
                pw.close();
                socket.close();
            } catch (UnknownHostException ex) {
                Logger.getLogger(GuiShell.class.getName()).log(Level.SEVERE, null, ex);
            } catch (IOException ex) {
                Logger.getLogger(GuiShell.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    });


    B_restart.addActionListener(new java.awt.event.ActionListener()
    {
        @Override
        public void actionPerformed(java.awt.event.ActionEvent evt)
        {
                GuiShell gui = new GuiShell();
                gui.restartAction(evt);
        }
    });
}

public void outputStatus(String text) {
    TA_status.append(text);
    TA_status.append("\n");
    TA_status.append("\n");
}


private static void ConfigurationMainWindow() {

    MainWindow.setSize(375,350);
    MainWindow.getContentPane().setLayout(null);
    B_restart.setBounds(250, 40, 81, 25);
    B_crash.setBounds(250, 10, 81, 25);
    Start.setBounds(13, 270, 323, 25);

    MainWindow.getContentPane().add(B_restart);
    MainWindow.getContentPane().add(B_crash);
    MainWindow.getContentPane().add(Start);
    MainWindow.getContentPane().add(L_Message);
    MainWindow.getContentPane().add(L_Conv);
    MainWindow.getContentPane().add(jComboBox1);
    jComboBox1.setBounds(120, 10, 90,20);
    L_Message.setBounds(10, 10, 160, 20);
    L_Conv.setBounds(100,70,140,16);

    TA_status.setColumns(20);
    TA_status.setRows(5);
```

```
        TA_status.setEditable(false);

        SP_Conversation.setViewportView(TA_status);
        MainWindow.getContentPane().add(SP_Conversation);
        SP_Conversation.setBounds(10, 90, 330, 180);
        MainWindow.getContentPane().add(SP_Conversation);
    }

}
```

3.3 Αλγόριθμος LeLann

3.3.1 LelannMutualExclusion.java

```
// Java imports
import java.util.Queue;
import java.util.LinkedList;
import java.util.Random;

// Visidia imports
import visidia.simulation.process.algorithm.Algorithm;
import visidia.simulation.process.messages.Door;

public class LelannMutualExclusion extends Algorithm {

    // All nodes data
    int procId;
    int next = 0;
    // Higher speed means lower simulation speed
    int speed = 1;

    // Token
    boolean token = false;

    // To display the state
    boolean waitForCritical = false;
    boolean inCritical = false;

    // Critical section thread
    ReceptionRules rr = null;
    // State display frame
    DisplayFrame df;

    public String getDescription() {

        return ("Lelann Algorithm for Mutual Exclusion");
    }

    @Override
    public Object clone() {
        return new LelannMutualExclusion();
    }

    //
    // Nodes' code
    //
    @Override
    public void init() {

        procId = getId();
```

```
        Random rand = new Random( procId+1 );

        rr = new ReceptionRules( this );
        rr.start();

        // Display initial state + give time to place frames
        df = new DisplayFrame( procId );
        displayState();
        try { Thread.sleep( 5000 ); } catch( InterruptedException ie ) {}

        // Start token round
        if ( procId == 0 ) {
           token = false;
           TokenMessage tm = new TokenMessage(MsgType.TOKEN);
           boolean sent = sendTo( next, tm );
        }

        while( true ) {

           // Wait for some time
           int time = ( 3 + rand.nextInt(10)) * speed * 1000;
           System.out.println("Process " + procId + " wait for " + time);
           try {
                 Thread.sleep( time );
           } catch( InterruptedException ie ) {}

           // Try to access critical section
           waitForCritical = true;
           askForCritical();

           // Access critical
           waitForCritical = false;
           inCritical = true;

           displayState();

           // Simulate critical resource use
           time = (1 + rand.nextInt(3)) * 1000;
           System.out.println("Process " + procId + " enter SC " + time);
           try {
                 Thread.sleep( time );
           } catch( InterruptedException ie ) {}
           System.out.println("Process " + procId + " exit SC ");

           // Release critical use
           inCritical = false;
           endCriticalUse();
        }
}
```

```
//--------------------
// Rules
//-------------------

// Rule 1 : ask for critical section
synchronized void askForCritical() {

      while( !token ) {

         displayState();
         try { this.wait(); } catch( InterruptedException ie) {}
      }
}

// Rule 2 : receive TOKEN
synchronized void receiveTOKEN(int d){

      System.out.println("Process " + procId + " reveiced TOKEN from " + d );
      next = ( d == 0 ? 1 : 0 );

      if ( waitForCritical == true ) {

         token = true;
         displayState();
         notify();

      } else {
         // Forward token to successor
         TokenMessage tm = new TokenMessage(MsgType.TOKEN);
      System.out.println( tm+" "+next );
         boolean sent = sendTo( next, tm );
      }
}

// Rule 3 :
void endCriticalUse() {

      token = false;
      TokenMessage tm = new TokenMessage(MsgType.TOKEN);
      boolean sent = sendTo( next, tm );

      displayState();
}

// Access to receive function
public TokenMessage recoit ( Door d ) {

      TokenMessage sm = (TokenMessage)receive( d );
      return sm;
}
```

```
    // Display state
    void displayState() {

        String state = new String("\n");
        state = state + "--------------------------------------\n";
        if ( inCritical )
           state = state + "** ACCESS CRITICAL **";
        else if ( waitForCritical )
           state = state + "* WAIT FOR *";
        else
           state = state + "-- SLEEPING --";

        df.display( state );
    }
}
```

ΚΕΦΑΚΑΙΟ 4

ΕΡΓΑΛΕΙΟ ΠΑΡΟΥΣΙΑΣΗΣ ΤΩΝ ΑΛΓΟΡΙΘΜΩΝ

4.1 Οργάνωση και ανάπτυξη κώδικα

Με τους όρους ανάπτυξη ή προγραμματισμός κώδικα στην επιστήμη των υπολογιστών συχνά αναφερόμαστε στη διαδικασία ανάπτυξης, ελέγχου και συντήρησης πηγαίου κώδικα προγραμμάτων για ηλεκτρονικούς υπολογιστές. Η διαδικασία αυτή μπορεί να αποδειχθεί πολύ δύσκολη, απαιτώντας από τον προγραμματιστή πολλές διαφορετικές δεξιότητες και γνώσεις. Εκτός αυτού, πλέον η πολυπλοκότητα και το μέγεθος του μεγαλύτερου μέρους των σύγχρονων προγραμμάτων έχει μεγαλώσει σε τέτοιο βαθμό που κάνει την ανάπτυξή τους ακόμα πιο δύσκολη. Για το λόγο αυτό έχουν αναπτυχθεί πολλά εργαλεία τα οποία διευκολύνουν αρκετά την παραγωγή ποιοτικού κώδικα. Στο κεφάλαιο αυτό θα δοθεί μεγαλύτερη έμφαση στο εργαλείο NetBeans.

4.2 Ποιότητα κώδικα

Η ερμηνεία του όρου ποιότητα σε ότι αφορά τον κώδικα μιας εφαρμογής μπορεί να ποικίλει ανάλογα με τη φύση αυτής. Για παράδειγμα σε μία εφαρμογή που χειρίζεται δεδομένα σε πραγματικό χρόνο ίσως δοθεί μεγαλύτερη έμφαση στην ταχύτητα εκτέλεσης και όχι στις υπόλοιπες συνιστώσες ποιότητας. Αντίθετα,σε μία εφαρμογή που πραγματοποιεί διατραπεζικές συναλλαγές είναι απαραίτητη μεγαλύτερη ασφάλεια. Παρ΄ όλα αυτά, είναι γενικά αποδεκτό ότι, οποιαδήποτε και αν είναι η προσέγγισή της ανάπτυξης λογισμικού, το πρόγραμμα τελικά θα πρέπει να πληρεί κάποια βασικά κριτήρια. Τα πιο σημαντικά από αυτά τα χαρακτηριστικά περιγράφονται παρακάτω:

Αποδοτικότητα (Efficiency)

Η αποδοτικότητα αφορά την όσο το δυνατόν πιο αποδοτική χρήση των πόρων του συστήματος. Με τον όρο πόροι ενός συστήματος συνήθως αναφερόμαστε στη μνήμη, τόσο την μόνιμη (σκληρός δίσκος) όσο και την προσωρινή (RAM - cache), τον επεξεργαστή, τις περιφερειακές συσκευές, το δίκτυο, κ.ά.

Αξιοπιστία (Reliability)

Ο παραγόμενος κώδικας πρέπει να είναι αξιόπιστος. Δηλαδή, θα πρέπει να υλοποιεί με ακρίβεια αυτά που έχουν οριστεί από τις προδιαγραφές προβλέποντας τα διάφορα προβλήματα που μπορεί να προκύψουν κατά τη διάρκεια της εκτέλεσης (όπως η υπερχείλιση ή η έλλειψη μνήμης).

Ευρωστία (Robustness)

Ο προγραμματιστής θα πρέπει να προβλέψει όσο το δυνατόν περισσότερες περιπτώσεις κατά τις οποίες, λόγω της ασυμβατότητας των διαφόρων δεδομένων ή στοιχείων προερχόμενων από το χρήστη, υπάρχει πιθανότητα να προκύψουν σφάλματα κατά την ώρα εκτέλεσης. Τέτοιου είδους σφάλματα θα πρέπει να συμβαίνουν όσο γίνεται πιο σπάνια και στην περίπτωση που συμβούν να περιγράφονται κατάλληλα με μηνύματα σφάλματος.

Μεταφερσιμότητα (Portability)

Η τεχνολογία πλέον εξελίσσεται ταχύτατα τόσο σε επίπεδο υλικού όσο και σε επίπεδο λογισμικού. Έτσι είναι πολύ χρήσιμο ο παραγόμενος κώδικας να μπορεί να μεταφέρεται σε οποιοδήποτε περιβάλλον και να μπορεί να εκτελεστεί χωρίς κάποια επιπλέον προσπάθεια επαναπρογραμματισμού.

Αναγνωσιμότητα (Readability)

Η ανάπτυξη και η συντήρηση του κώδικα δεν είναι πλέον θέμα ενός μόνο ατόμου. Η ανάπτυξη γίνεται στα πλαίσια ομάδων όπου κάθε άτομο αναλαμβάνει ένα μικρό κομμάτι της όλης εφαρμογής. Έτσι είναι πολύ χρήσιμο ο κώδικας που παράγουν τα διάφορα μέλη των ομάδων να είναι ευανάγνωστος και κατανοητός έτσι, ώστε να μπορεί οποιοδήποτε μέλος της ομάδας να διαβάσει οποιοδήποτε μέρος του κώδικα εφαρμογής.

4.3 Έλεγχος ποιότητας κώδικα

Ο έλεγχος του κατά πόσο ο κώδικας μίας εφαρμογής είναι ποιοτικός ή όχι δεν είναι κάτι το τετριμμένο. Αντίθετα, αποτελεί μία πολύπλοκη διαδικασία η οποία απαιτεί αρκετό χρόνο και προσπάθεια. Όσο αργότερα στην διαδικασία ανάπτυξης αρχίσει να γίνεται ο έλεγχος τόσο πιο δύσκολο είναι να επιτευχθεί το επιθυμητό αποτέλεσμα. Επιπρόσθετα, δεν είναι εύκολο για τον προγραμματιστή να ελέγχει την ποιότητα του κώδικα που παράγει, αφού, αν μπορούσε εύκολα να εντοπίσει τα σημεία στα οποία υστερούσε, θα τα είχε διορθώσει από την αρχή.

Για την, κατά κάποιο τρόπο, πιστοποίηση της ποιότητας του παραγόμενου κώδικα έχουν αναπτυχθεί πολλά εργαλεία. Τα πιο διαδεδομένα κάνουν στατική ανάλυση κώδικα (static code analysis), δηλαδή διαβάζουν και αναλύουν τον κώδικα ελέγχοντας για τυχόν λάθη, παραλήψεις ή μη αποδοτικές τεχνικές προγραμματισμού. Ο PMD, είναι ένα παράδειγμα δυναμικού αναλυτή για κώδικα Java. Ο αναλυτής αυτός διαβάζει τον κώδικα και παράγει μία αναφορά στην κατάλληλη μορφή (πχ html, xml, κτλ) ανάλογα με το τι του έχει ορίσει ο χρήστης.Η αναφορά αυτή περιέχει παρατηρήσεις για διάφορα θέματα (ανάλογα με τους κανόνες που έχουν οριστεί) χωρισμένες σε πέντε επίπεδα σοβαρότητας. Ο προγραμματιστής μπορεί, διαβάζοντας την αναφορά, να τροποποιήσει τον κώδικά του κάνοντάς τον πιο ποιοτικό.

Βέβαια η ποιότητα του κώδικα δεν μπορεί να διασφαλιστεί μόνο κάνοντας χρήση τέτοιου είδους εργαλείων. Υπάρχουν πολλές περιπτώσεις κατά τις οποίες γίνεται σπατάλη πόρων ή υπάρχουν λογικά λάθη οι οποίες δεν μπορούν να εντοπιστούν. Εντούτοις, η χρήση τους παρέχει ενός είδους μέτρο για την ποιότητα του παραγόμενου κώδικα.

4.4 Έλεγχος ορθότητας κώδικα

Ο έλεγχος της ορθότητας του κώδικα μιας εφαρμογής, δηλαδή η πιστοποίηση ότι ο κώδικας κάνει ακριβώς αυτό για το οποίο σχεδιάστηκε, αποτελεί ένα πολύ δύσκολο και χρονοβόρο κομμάτι της ανάπτυξης λογισμικού. Το μεγαλύτερο και πιο δύσκολα αντιμετωπίσιμο πρόβλημα είναι ο εντοπισμός του κομματιού της όλης εφαρμογής το οποίο δε συμπεριφέρεται όπως θα έπρεπε ύστερα από κάποια αλλαγή σε ένα άλλο κομμάτι αυτής. Στις περισσότερες των περιπτώσεων, αν δε γίνουν κάποιες ενέργειες πρόληψης, η αντιμετώπιση τέτοιου είδους προβλημάτων παίρνει τόσο πολύ χρόνο που βγάζει όλο το χρονοδιάγραμμα ανάπτυξης εκτός ορίων.

Μία λύση για την πρόωρη αντιμετώπιση τέτοιου είδους προβλημάτων είναι η χρήση ανεξάρτητων κομματιών κώδικα ελέγχου, γνωστά ως “test units”. Τα κομμάτια κώδικα αυτά, αναπτύσσονται για κάθε λειτουργία της εφαρμογής σε όλα τα επίπεδα της υλοποίησης και ελέγχουν κάθε φορά, αν το αποτέλεσμα είναι το αναμενόμενο. Το γεγονός αυτό επιτρέπει κάθε φορά που γίνεται μία αλλαγή σε κάποιο σημείο της εφαρμογής να ελέγχεται, αν όλα τα υπόλοιπα κομμάτια εξακολουθούν να εκτελούνται όπως και πριν. Στην περίπτωση που κάτι πάει λάθος ο προγραμματιστής μπορεί να βρει πολύ εύκολα ποια κομμάτια δεν ανταποκρίνονται όπως θα έπρεπε.

Υπάρχουν πολλά περιβάλλοντα εργασίας που υλοποιούν τέτοιου είδους test units και είναι διαθέσιμα για μια πληθώρα γλωσσών προγραμματισμού. Κατά τη διάρκεια ανάπτυξης και ελέγχου της εφαρμογής που αναπτύχθηκε στα πλαίσια της παρούσας εργασίας χρησιμοποιήθηκε το NetBeans IDE, ένα πολύ σημαντικό τέτοιου είδους λογισμικό για Java εφαρμογές που χρησιμοποιείται ευρύτατα. Αυτό παρέχει όλες τις απαιτούμενες υπηρεσίες για τον έλεγχο της ορθότητας του παραγομένου κώδικα.

4.5 Συστήματα διαχείρισης εκδόσεων (VCS)

Όπως προαναφέρθηκε, πλέον το μέγεθος των εφαρμογών δεν επιτρέπει την ανάπτυξή τους από ένα μόνο άνθρωπο, αλλά κάνει αναγκαία τη συνεργασία πολλών. Αυτό δημιουργεί ένα πρόβλημα που δεν υπήρχε πριν: την έγκαιρη και χωρίς προβλήματα παρακολούθηση των αλλαγών που γίνονται από κάθε μέλος της ομάδας. Έτσι υπάρχει ανάγκη για την ύπαρξη κάποιου είδους μηχανισμού ο οποίος θα παρακολουθεί τις αλλαγές, θα εμποδίζει τις συγκρούσεις μεταξύ διαφορετικών εκδόσεων ίδιων αρχείων και θα δίνει τη δυνατότητα να έχουν όλοι πρόσβαση σε οτιδήποτε νέο προστεθεί. Τέτοιου είδους μηχανισμοί ονομάζονται "Συστήματα Διαχείρισης Εκδόσεων" (Version Control Systems – VCS). Τα συστήματα διαχείρισης εκδόσεων είναι συνήθως αυτόνομα προγράμματα τα οποία διαχειρίζονται πολλαπλές εκδόσεις ίδιων μονάδων πληροφορίας. Οι διαφορετικές εκδόσεις διαχωρίζονται μέσω κάποιου είδους αύξοντα αριθμού και συνοδεύονται από σχόλια που προσθέτει ο χρήστης και συνδέονται με το χρήστη που πραγματοποίησε τις αλλαγές.

Τα συστήματα διαχείρισης εκδόσεων χρησιμοποιούνται για την αποθήκευση πηγαίου κώδικα, εγγράφων τεκμηρίωσης της εφαρμογής, ακόμα και configuration files. Καθώς το λογισμικό σχεδιάζεται, αναπτύσσεται και εγκαθίσταται, είναι πολύ συνηθισμένο να υπάρχουν διαφορετικές εκδόσεις του λογισμικού σε διαφορετικά τοπικά αντίγραφα, και οι προγραμματιστές να αναπτύσσουν ταυτόχρονα αναβαθμίσεις τους. Σφάλματα και άλλα προβλήματα παρουσιάζονται σε συγκεκριμένες εκδόσεις (γιατί στην προσπάθεια αντιμετώπισης των προβλημάτων, δημιουργούνται νέα). Για την αντιμετώπιση αυτών των προβλημάτων, είναι σημαντικό να υπάρχει η δυνατότητα να ανακτούνται διαφορετικές εκδόσεις του συστήματος οι οποίες δεν αντιμετωπίζουν τα προβλήματα που έχει η έκδοση με την οποία ασχολείται κάποιος προγραμματιστής. Επιπλέον, μερικές φορές είναι απαραίτητο να δημιουργούνται ταυτόχρονα δύο ή παραπάνω εκδόσεις του ίδιου συστήματος (όπως όταν προσπαθούν να λυθούν τα προβλήματα μιας έκδοσης αλλά δεν έχει υλοποιούνται νέες λειτουργίες, ενώ παράλληλα δημιουργούνται εκδόσεις του συστήματος με νέες λειτουργίες).

Οι παραπάνω λόγοι καθιστούν τη χρήση των συστημάτων διαχείρισης εκδόσεων απαραίτητη. Τα συστήματα VCS είναι συνήθως αυτόνομα προγράμματα τα οποία διαχειρίζονται πολλαπλές εκδόσεις ίδιων μονάδων πληροφορίας. Οι διαφορετικές εκδόσεις διαχωρίζονται μέσω κάποιου είδους αύξοντα αριθμού, συνοδεύονται από σχόλια που προσθέτει ο χρήστης και συνδέονται με τον χρήστη που πραγματοποίησε τις αλλαγές. Παρακάτω περιγράφονται τα δύο πιο γνωστά συστήματα διαχείρισης εκδόσεων, το CVS (Concurrent Versioning System) και το SVN (Subversion).

4.5.1 CVS (Concurrent Versioning System)

Πρόκειται για ένα λογισμικό ανοιχτού κώδικα που δημιουργήθηκε από τον Dick Grune τη δεκαετία του 1980. Το CVS χρησιμοποιεί την αρχιτεκτονική πελάτη - εξυπηρετητή: στον εξυπηρετητή αποθηκεύονται οι εκδόσεις του έργου και οι πελάτες συνδέονται για να λάβουν (check out) ένα πλήρες αντίγραφο του έργου, να εργαστούν πάνω σε αυτό το αντίγραφο και στο τέλος να στείλουν (chek in) τις αλλαγές τους. Συνήθως, οι πελάτες συνδέονται στον εξυπηρετητή μέσω ενός τοπικού δικτύου ή μέσω του διαδικτύου. Όμως, ο πελάτης και ο εξυπηρετητής μπορεί να βρίσκονται στο ίδιο μηχάνημα στο οποίο το CVS αποθηκεύει τις εκδόσεις ενός έργου αν οι προγραμματιστές είναι εργάζονται στο ίδιο σύστημα. Το πρόγραμμα του εξυπηρετητή "τρέχει" σε λειτουργικό Unix (αν και ο εξυπηρετητής CVSNT υποστηρίζει διάφορες εκδόσεις των Windows, της Microsoft, και Linux), ενώ οι CVS πελάτες μπορεί να τρέχουν σε οποιοδήποτε από τα πιο διαδεδομένα λειτουργικά συστήματα.

Πολλοί προγραμματιστές μπορεί να εργάζονται στο ίδιο έργο ταυτόχρονα, όπου ο καθένας επεξεργάζεται τα αρχεία του δικού του τοπικού αντίγραφου, και στέλνει τις αλλαγές στον εξυπηρετητή. Για να αποφευχθούν οι συγκρούσεις σε μέρη του κώδικα που δύο ή παραπάνω προγραμματιστές έχουν κάνει αλλαγές, ο εξυπηρετητής δέχεται τις αλλαγές μόνο από την πιο πρόσφατη έκδοση των αρχείων. Γι αυτό το λόγο οι προγραμματιστές πρέπει να κρατούν ενημερωμένο το τοπικό τους αντίγραφο ενσωματώνοντας συχνά τις αλλαγές των υπόλοιπων χρηστών. Αυτό γίνεται αυτόματα από το CVS και ζητείται η μεσολάβηση του χρήστη μόνο όταν δημιουργούνται συγκρούσεις με τον κώδικα του εξυπηρετητή που έχει αλλαχθεί από κάποιον άλλο χρήστη και το τοπικό αντίγραφο του προγραμματιστή που προσπαθεί να στείλει τις δικές του αλλαγές. Αν η αποστολή είναι επιτυχής, τότε ο αριθμός της έκδοσης όλων των αρχείων αυξάνεται , και ο CVS εξυπηρετητής αποθηκεύει σε ένα αρχείο καταγραφής την περιγραφή που έχει εισάγει ο χρήστης, την ημερομηνία αποστολής, και το όνομα του αποστολέα.

Οι προγραμματιστές μπορούν να συγκρίνουν εκδόσεις του συστήματος, να ζητήσουν ένα πλήρες ιστορικό των αλλαγών ή ένα συγκεκριμένο στιγμιότυπο του έργου βάση μιας ημερομηνίας ή ενός αριθμού έκδοσης. Επιπλέον, οι χρήστες μπορούν μέσω της εντολής *update* να ενημερώσουν το τοπικό τους αντίγραφο λαμβάνοντας την τελευταία έκδοση του έργου που υπάρχει στον εξυπηρετητή.

Το CVS χρησιμοποιεί την συμπίεση *delta (delta compression)* για την αποδοτική αποθήκευση διαφορετικών εκδόσεων του ίδιου αρχείου. Η υλοποίηση αυτή ευνοεί τα αρχεία με πολλές γραμμές (συνήθως αρχεία κειμένου) και σε ακραίες καταστάσεις μπορεί το σύστημα να αποθηκεύσει ξεχωριστά αντίγραφα για κάθε έκδοση ενός αρχείου.

4.5.2 SVN (Subversion)

Το SVN δημιουργήθηκε από την εταιρία CollabNet το 2000. Χρησιμοποιείται για την αποθήκευση εκδόσεων αρχείων πηγαίου κώδικα, ιστοσελίδων και εγγράφων τεκμηρίωσης. Ο στόχος της δημιουργίας του είναι να γίνει ο πιο συμβατός διάδοχος του ευρέως χρησιμοποιημένου CVS, έχοντας διορθώσει τα προβλήματα του CVS και έχοντας προσθέσει λειτουργίες που έλλειπαν από το CVS . Το SVN είναι πολύ γνωστό στην κοινότητα λογισμικού ανοιχτού κώδικα και χρησιμοποιείται σε πολλά έργα όπως: *Apache Software Foundation, KDE, Free Pascal, FreeBSD, GCC, Python, Django, Ruby, Mono, SourceForge.net, ExtJS* και *Tigris.org*. Το σύστημα SVN χρησιμοποιεί ως web server τον

Apache HTTP Server και ως πρωτόκολλο το WebDAV/DeltaV. Επιπλέον, υπάρχει μία ανεξάρτητη διεργασία εξυπηρετητή που χρησιμοποιεί ένα προσαρμοσμένο στο TCP/IP πρωτόκολλο.

Οι αποστολές στο SVN είναι καθαρά ατομικές εργασίες. Όταν μια διαδικασία αποστολής διακοπεί, δεν προκαλεί την αστάθεια στην αποθήκη λογισμικού. Επιπλέον, τα αρχεία που αντιγράφηκαν, μετακινήθηκαν ή και διαγράφηκαν διατηρούν πλήρες ιστορικό των εκδόσεων τους. Επίσης, διατηρούνται οι εκδόσεις φακέλων, μετονομασιών και αρχείων μεταδεδομένων (χωρίς όμως να διατηρούνται ημερομηνίες για τις εκδόσεις τους). Ολόκληρες δενδρικές δομές φακέλων μπορούν να μετακινηθούν να αντιγραφούν πολύ εύκολα και γρήγορα και να διατηρηθεί το πλήρες ιστορικό των εκδόσεών τους. Άλλη μια δυνατότητα του SVN είναι η διατήρηση εκδόσεων για symbolic links.

Το κόστος της αποστολής έκδοσης ή της ενημέρωσης αυξάνεται ανάλογα με το μέγεθος των αλλαγών και όχι με το μέγεθος των δεδομένων. Αυτό οφείλεται στον τρόπο με τον οποίο αποθηκεύει τις εκδόσεις των αρχείων. Το SVN προσφέρει δύο τύπους αποθήκης δεδομένων: το σύστημα FSFS (Fast Secure File System) και το σύστημα Berkeley DB. Το FSFS αποδίδει γρηγορότερα σε φακέλους με μεγάλο αριθμό αρχείων και καταλαμβάνει λιγότερο χώρο στο δίσκο λόγο της μικρότερης καταγραφής των αλλαγών. Το SVN αντιμετωπίζει κάποιους περιορισμούς με την χρήση του Berkeley DB οδηγώντας την αποθήκη δεδομένων σε καταστάσεις απώλειας δεδομένων όταν οι προσπελάσεις στην βάση διακοπούν βίαια.

Όμως το σύστημα SVN έχει κάποιους περιορισμούς και προβλήματα. Ένα από αυτά είναι η υλοποίηση της διαδικασίας μετονομασίας φακέλων και αρχείων. Αυτή τη στιγμή υλοποιεί αυτή τη διαδικασία κάνοντας αντιγραφή στο νέο όνομα και διαγραφή του παλιού. Μόνο τα ονόματα αλλάζουν και το SVN εξακολουθεί να χρησιμοποιεί το παλιό όνομα σε παλαιότερες εκδόσεις. Ωστόσο, μπορεί να μπερδευτεί όταν αρχεία τροποποιούνται και μετακινούνται στην ίδια αποστολή. Επιπλεόν, το Subversion δεν έχει λειτουργίες της αποθήκης δεδομένων. Για παράδειγμα, μερικές φορές χρειάζεται να σβήσουμε οριστικά ιστορικές καταχωρήσεις για συγκεκριμένα δεδομένα, όμως δεν υπάρχει στο SVN τέτοια ενσωματωμένη λειτουργία ώστε να πραγματοποιείται εύκολα.

4.6 Ολοκληρωμένα Περιβάλλοντα Ανάπτυξης Λογισμικού (IDEs)

Για τη δημιουργία κάποιου προγράμματος σήμερα αυτό που είναι απαραίτητο είναι η ύπαρξη κάποιου είδους κειμενογράφου και ενός μεταγλωττιστή (compiler) ή διερμηνέα(interpreter) μέσω του οποίου θα είναι δυνατή η μετατροπή του κώδικα από υψηλού επιπέδου γλώσσα σε γλώσσα μηχανής. Έτσι, για παράδειγμα, για την ανάπτυξη μίας Java εφαρμογής σε ένα windows σύστημα αρκεί η ύπαρξη ενός σημειωματάριου (notepad) και η εγκατάσταση της επιθυμητής έκδοσης της Java Virtual Machine. Σε αυτήν την περίπτωση ο προγραμματιστής κατά την διάρκεια της ανάπτυξης της εφαρμογής του θα πρέπει να μεταγλωττίζει και να τρέχει την εφαρμογή μέσω της γραμμής εντολών ξανά και ξανά, έχοντας ως μόνο του βοηθό στην περίπτωση ύπαρξης κάποιου σφάλματος τα μηνύματα σφάλματος του μεταγλωττιστή. Αυτό σημαίνει την κατανάλωση πολύ χρόνου και κόπου ακόμη και για τις πιο απλές εφαρμογές. Όσο το μέγεθος και η πολυπλοκότητα των εφαρμογών (άρα και του κώδικα) αυξάνεται, τα μεγέθη αυτά αυξάνονται σχεδόν εκθετικά, πράγμα που καθιστά αυτή τη μέθοδο ανάπτυξης μη πρακτική.

Τη λύση σε αυτό το πρόβλημα δίνουν τα ολοκληρωμένα περιβάλλοντα ανάπτυξης λογισμικού (Integrated Development Environment) — ευρέως γνωστά ως IDEs. Τα IDEs είναι ολοκληρωμένες εφαρμογές μέσα στις οποίες μπορεί να πραγματοποιηθεί κάθε απαραίτητη διαδικασία για την ανάπτυξης μίας εφαρμογής. Επίσης είναι σχεδιασμένα να μεγιστοποιούν την παραγωγικότητα του προγραμματιστή παρέχοντας ισχυρά συνδεδεμένα συνιστούντα μέρη με παρόμοια διεπαφή χρήστη. Αυτό σημαίνει πως ο προγραμματιστής, στην περίπτωση που χρησιμοποιούνται διαφορετικά προγράμματα ανάπτυξης, καταβάλλει λιγότερη προσπάθεια να μεταβεί από το ένα πρόγραμμα στο άλλο. Ωστόσο, επειδή ένα IDE είναι από τη φύση του πολύπλοκο, η παραγωγικότητα του προγραμματιστή αυξάνεται όσο αποκτά μεγαλύτερη γνώση και οικειότητα από την χρήση αυτού του εργαλείου.

Οι συνηθέστερες λειτουργίες που παρέχονται είναι η συγγραφή του κώδικα, η τροποποίησή του, η μεταγλώττισή του, η εκτέλεσή του καθώς και κάποια διαδικασία εντοπισμού σφαλμάτων. Ο βασικός σκοπός τους είναι να περιορίσουν και κατά κάποιο τρόπο να κρύψουν όλες τις ρυθμίσεις που χρειάζονται για το συντονισμό των διαφορετικών κομματιών, ο συνδυασμός των οποίων είναι απαραίτητος για την επιτυχημένη εκτέλεση μίας εφαρμογής. Αυτό έχει σαν συνέπεια την ευκολότερη εκμάθηση μιας γλώσσας προγραμματισμού καθώς και την αύξηση της παραγωγικότητας.

Γνωστά IDEs

Τα τελευταία χρόνια έχουν αναπτυχθεί πολλά ολοκληρωμένα περιβάλλοντα ανάπτυξης λογισμικού. Μερικά από τα διασημότερα και ευρύτερα χρησιμοποιούμενα είναι τα παρακάτω:

- Visual Studio της Microsoft
- Eclipse
- NetBeans της Sun
- IntelliJ IDEA της JetBrains.

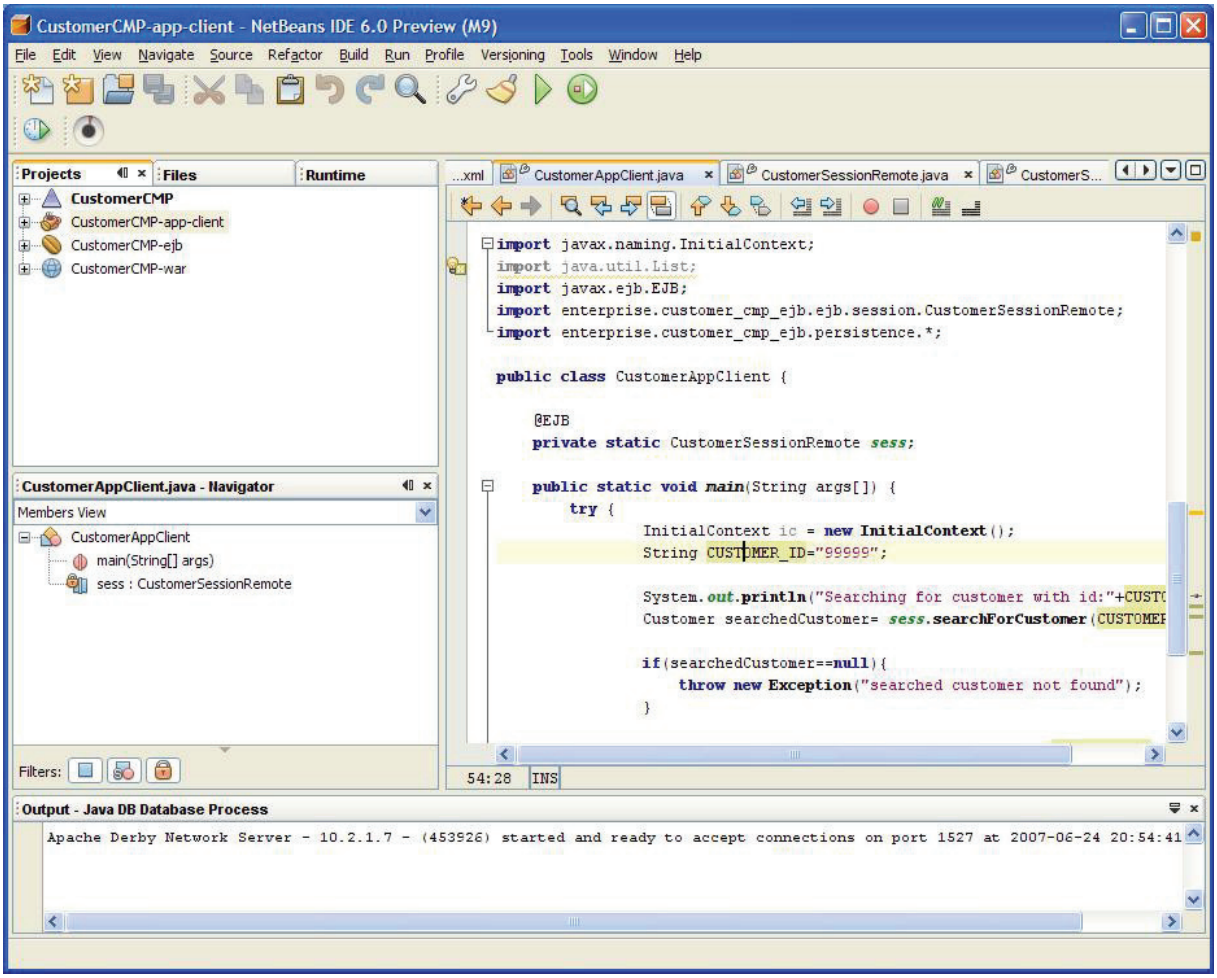
Η υλοποίηση των αλγορίθμων που αναπτύχθηκαν στα πλαίσια της παρούσας εργασίας έγινε στο περιβάλλον ανάπτυξης NetBeans IDE.

4.6.1 NetBeans IDE

Το NetBeans IDE είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης εφαρμογών που βασίζεται στην πλατφόρμα NetBeans. Η πλατφόρμα NetBeans IDE επιτρέπει την ανάπτυξη της εφαρμογής NetBeans IDE από ένα σύνολο συστατικών μονάδων λογισμικού που ονομάζονται modules. Ένα module είναι ένα συμπιεσμένο αρχείο Java το οποίο περιέχει κλάσεις, που είναι προορισμένες να αλληλεπιδρούν με NetBeans Open APIs, και ένα αρχείο που το χαρακτηρίζει ως module. Εφαρμογές που έχουν αναπτυχθεί πάνω σε modules μπορούν να επεκταθούν προσθέτωντας νέα modules.

Το NetBeans IDE διανέμεται δωρεάν και είναι λογισμικό ανοιχτού κώδικα. Υποστηρίζει την ανάπτυξη όλων των είδών εφαρμογών Java (lava SE, web, EJB και mobile εφαρμογές). Επιπλέον, υποστηρίζει την ανάπτυξη εφαρμογών με την χρήση του Ant τον έλεγχο εκδόσεων, με την χρήση των CVS, Subversion, Mercurial και Clearcase, και την επεξεργασία κώδικα (refactoring).

Όλες οι λειτουργίες του IDE παρέχονται από τα modules. Κάθε module παρέχει μία απαιτούμενη λειτουργία όπως υποστήριξη ανάπτυξης εφαρμογών Java, επεξεργασία κώδικα, υποστήριξη συστημάτων διαχείρισης εκδόσεων CVS και SVN. Το NetBeans IDE παρέχει όλα τα απαιτούμενα modules για την ανάπτυξη εφαρμογών Java στο αρχικό πακέτο που κατεβάζει κανείς, το οποίο επιτρέπει στον χρήστη να αρχίσει άμεσα την ανάπτυξη μιας εφαρμογής. Τα modules επιτρέπουν την επέκταση του IDE. Νέες λειτουργίες, όπως η υποστήριξη επιπλέον γλωσσών προγραμματισμού μπορούν να προστεθούν μετά την αρχική εγκατάσταση.



NetBeans IDE

4.7 Συστήματα παρακολούθησης προβλημάτων (issue tracking systems)

Τα συστήματα παρακολούθησης προβλημάτων είναι πακέτα λογισμικού που διαχειρίζονται προβλήματα και τα διατηρούν σε μία λίστα, όπως απαιτείται από κάποιον οργανισμό. Τα συστήματα παρακολούθησης προβλημάτων χρησιμοποιούνται από οργανισμούς για να καταχωρούν, να ενημερώνουν και να επιλύουν προβλήματα που αναφέρονται από πελάτες ή ακόμα και προβλήματα που αναφέρονται από το προσωπικό της εταιρίας. Επιπλέον, ενα σύστημα παρακολούθησης προβλημάτων διατηρεί μία γνωσιακή βάση με πληροφορίες για τον κάθε πελάτη, λύσεις σε συχνά εμφανιζόμενα προβλήματα και άλλα παρόμοια δεδομένα.

Στην τεχνολογία λογισμικού, τα συστήματα παρακολούθησης προβλημάτων είναι σχεδιασμένα να συνεισφέρουν στην παροχή εγγύησης ποιότητας και για να βοηθήσουν του προγραμματιστές να αναφέρουν σφάλματα που μπορεί να υπάρχουν στον κώδικα στον οποίο εργάζονται. Το κύριο όφελος ενός συστήματος παρακολούθησης σφαλμάτων είναι να παρέχουν μια γενική εποπτεία των αιτήσεων που δημιουργούνται κατά την ανάπτυξη κώδικα (οι οποίες μπορεί να αφορούν λάθη ή βελτιώσεις του κώδικα) και την κατάστασή τους. Η λίστα με τις αιτήσεις ταξινομημένες κατά σειρά προτεραιότητας προσφέρει πολύ σημαντικά δεδομένα για την σχεδίαση των επόμενων βημάτων της εφαρμογής ή ακόμα και τις απαιτήσεις της επόμενης έκδοσης της.

Σε ένα επιχειρηματικό περιβάλλον, ένα σύστημα παρακολούθησης σφαλμάτων μπορεί να χρησιμοποιηθεί για την παραγωγή αναφορών για την παραγωγικότητα των προγραμματιστών όσο αναφορά τη διόρθωση σφαλμάτων. Ωστόσο, οι αναφορές αυτές μπορεί να οδηγήσουν σε εσφαλμένα συμπεράσματα γιατί διαφορετικά σφάλματα έχουν διαφορετικά επίπεδα σοβαρότητας και πολυπλοκότητας. Η σοβαρότητα ενός σφάλματος μπορεί να μην έχει άμεση σχέση με την πολυπλοκότητα του. Για αυτό μπορεί να υπάρχουν διαφορετικές απόψεις μεταξύ των managers και των σχεδιαστών.

Παρακάτω περιγράφονται δύο γνωστά συστήματα παρακολούθησης σφαλμάτων, το *Bugzilla* και το *JIRA*.

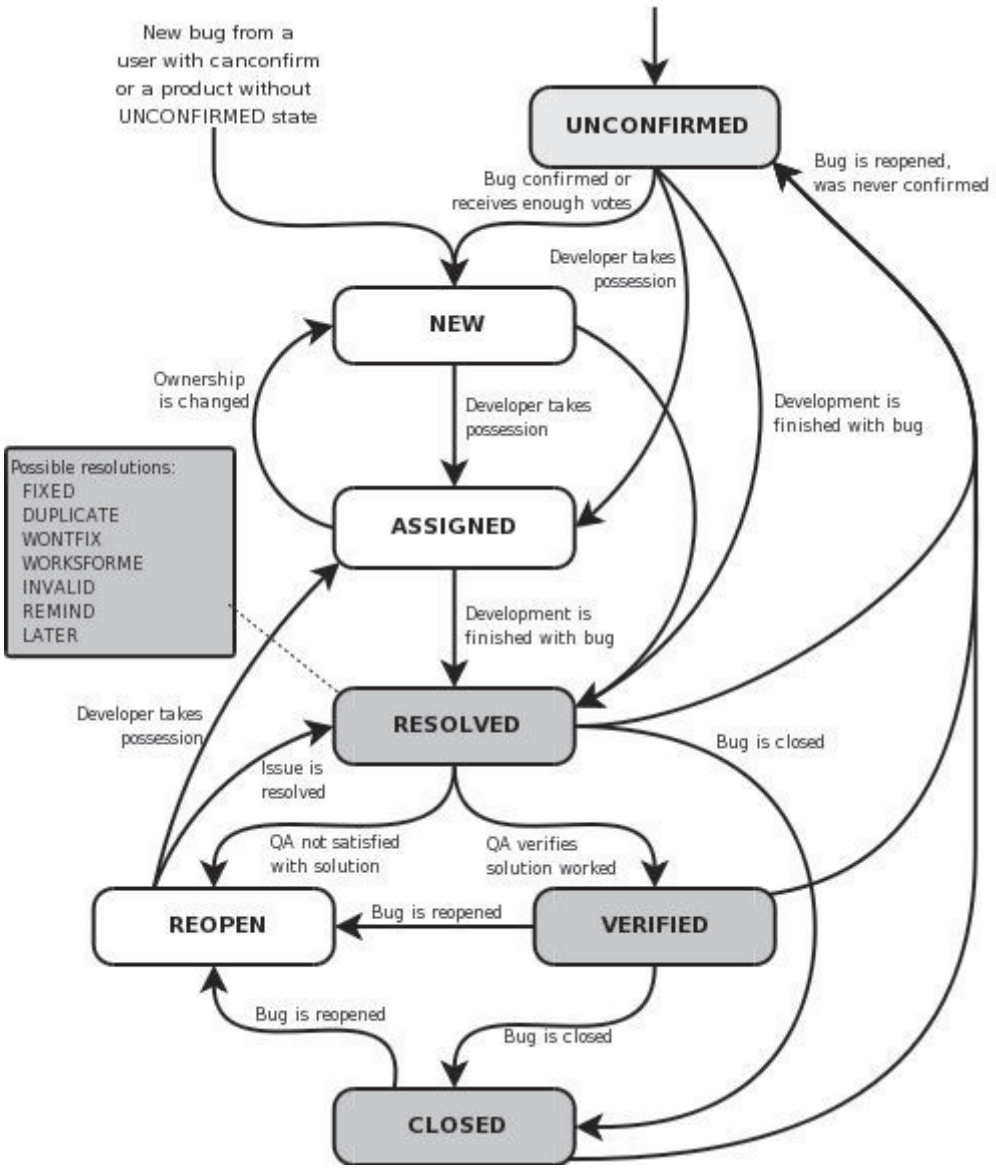
4.7.1 Bugzilla

Πρόκειται για ένα γενικού σκοπού, διαδικτυακό σύστημα παρακολούθησης σφαλμάτων και ένα εργαλείο ελέγχου το οποίο αρχικά αναπτύχθηκε και χρησιμοποιήθηκε από το έργο Mozilla, και κυκλοφόρησε υπό την άδεια Mozilla Public. Κυκλοφόρησε ως λογισμικό ανοιχτού κώδικα από την Netscape Communications το 1998, και υιοθετήθηκε από πολλούς οργανισμούς.

Το Bugzilla έχει κάποιες απαιτήσεις συστήματος για την εγκατάσταση του και αναφέρονται παρακάτω:

- Ένα συμβατό σύστημα διαχείρισης βάσης δεδομένων.
- Την κατάλληλη έκδοση του Perl 5.
- Μια συλλογή από Perl modules.
- Ένας συμβατός web server.
- Ένας συμβατός mail transfer agent, ή οποιοδήποτε SMTP server.

Επί του παρόντος, τα συστήματα βάσεων δεδομένων που είναι συμβατά είναι τα: MySQL, PostgreSQL, και Oracle. Το σύστημα Bugzilla συνήθως εγκαθίσταται σε λειτουργικό Linux και λειτουργεί χρησιμοποιώντας τον Apache HTTP Server αλλά ο Microsoft Internet Information Services ή οποιοσδήποτε άλλος web server που υποστηρίζει CGI μπορεί να χρησιμοποιηθεί.



Ο κύκλος ζωής μιας αίτησης σφάλματος

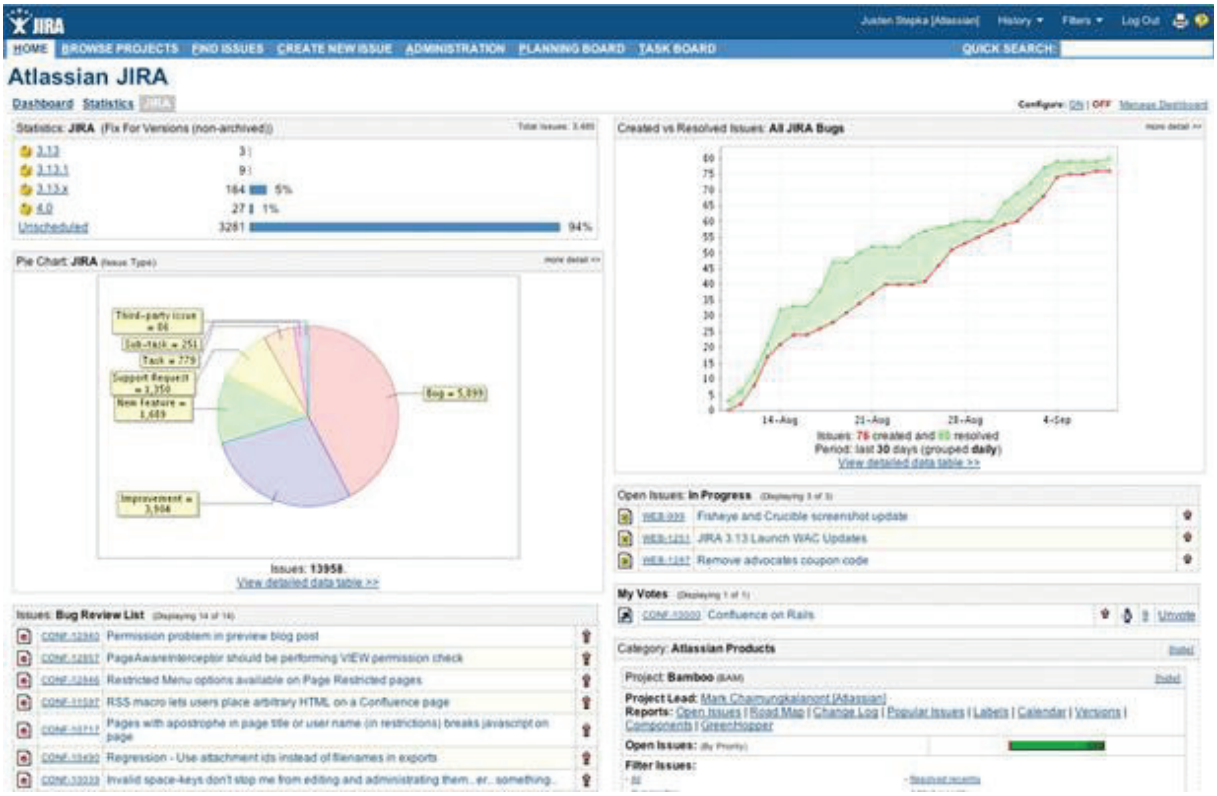
Παρόλο που ο κώδικας του Bugzilla έχει την προοπτική να το μετατρέψει σε ένα εργαλείο διαχείρισης έργων, ή σε εργαλείο διαχείρισης εργασιών, οι προγραμματιστές του Bugzilla επέλεξαν να επικεντρωθούν στον σχεδιασμό ενός συστήματος παρακολύθησης λαθών. Υπάρχουν κάποιες απαραίτητες σχεδιαστικές απαιτήσεις που περιλαμβάνουν τα εξής:

- τη δυνατότητα να τρέχει με εργαλεία δωρεάν διάθεσης και ανοιχτού κώδικα. Ενώ η ανάπτυξη του Bugzilla περιλαμβάνει την προσπάθεια υποστήριξης εμπορικών βάσεων δεδομένων, εργαλείων και λειτουργικών συστημάτων, δεν υπάρχει η πρόθεση αυτό να γίνει εις βάρος των εργαλείων ανοιχτού κώδικα.
- η διατήρηση της ταχύτητας και της αποτελεσματικότητας με οποιοδήποτε κόστος. Ένα από τα χαρακτηριστικά του Bugzilla που έλκει τους προγραμματιστές είναι η ταχύτητα του και η μη επιβαρυμένη υλοποίηση του, ελαχιστοποιώντας όσο είναι δυνατό τις κλήσεις προς τη βάση δεδομένων. Οι προσπελάσεις στην βάση προσπαθούν να είναι όσο το δυνατό λιγότερο απαιτητικές και αποφεύγεται η δημιουργία βαριάς HTML.

- η χρήση εισιτήριων για τα σφάλματα. Τα σφάλματα στον κώδικα μπορούν να σταλούν από οποιονδήποτε και να ανατεθούν σε έναν συγκεκριμένο προγραμματιστή. Διάφορες ενημερώσεις της κατάστασης του σφάλματος επιτρέπονται, μαζί με σημειώσεις από τον χρήστη που εντόπισε το σφάλμα καθώς και παραδείγματα στα οποία εμφανίζεται. Στην πράξη, τα περισσότερα έργα Bugzilla που επιτρέπουν την δημόσια καταχώρηση σφαλμάτων, αναθέτουν όλα τα σφάλματα σε κάποιον, ο οποίος έχει καθήκον την ανάθεση των σφαλμάτων σε προγραμματιστές και τον ορισμό του επιπέδου προτεραιότητας.

4.7.2 JIRA

Το JIRA είναι ένα εμπορικό πρόγραμμα για επιχειρήσεις, αναπτύχθηκε από την εταιρία Atlassian και συνήθως χρησιμοποιείται ως σύστημα παρακολούθησης σφαλμάτων κώδικα και ως σύστημα διαχείρισης έργων. Χρησιμοποιείται σε πολλά έργα λογισμικού ανοιχτού κώδικα με πάνω από 12.000 χρήστες σε περισσότερες από 100 χώρες. Το JIRA χρησιμοποιείται για την παρακολούθηση λαθών και σε πολλά μεγάλης κλίμακας ανοιχτού κώδικα, δημόσια έργα όπως το *Secondlife* της Linden Lab's, ένα παρόμοιο έργο που ονομάζεται *Opensimulator* και σε πολλά άλλα έργα ευρέως γνωστά στην κοινότητα των προγραμματιστών. Το σύστημα αυτό είναι πολύ χρήσιμο για μεγάλα ή/και δημόσια έργα.



Το περιβάλλον του JIRA

Το JIRA επιτρέπει την απεικόνιση του κύκλου ζωής ενός προβλήματος στις επιχειρηματικές διεργασίες. Επιπλέον, προσφέρει αρκετούς τρόπους για την παροχή σε πραγματικό χρόνο πληροφοριών σχετικές με προβλήματα ή ροές εργασίας, στην κατάλληλη μορφή.

Το JIRA είναι γραμμένο σε Java και για απομακρυσμένη κλήση μεθόδων (Remote Procedure Calls) υποστηρίζει τα: SOAP, XML RPC και ένα Java API. Επιπλέον συνεργάζεται με αρκετά συστήματα διαχείρισης εκδόσεων όπως: Subversion, CVS, Clearcase, Visual SourceSafe, Mercurial, και Perforce.

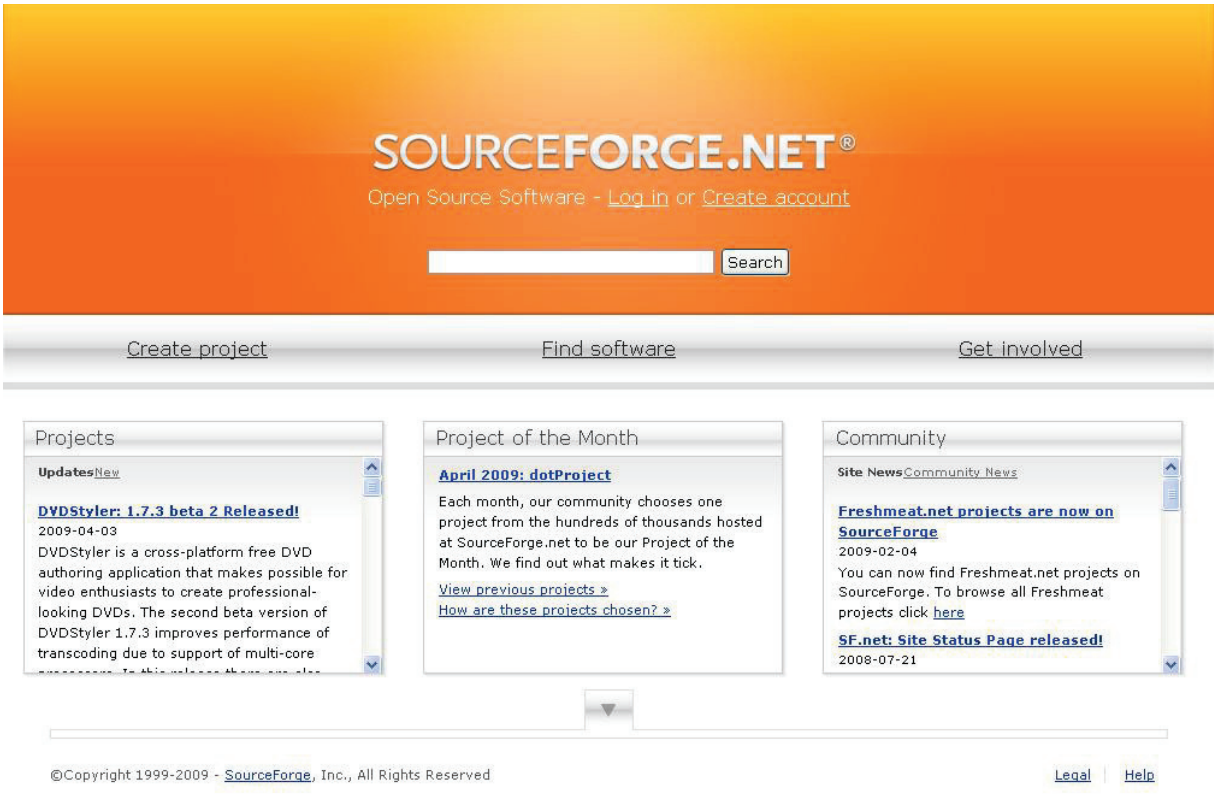
Το JIRA έχει ένα σύστημα plugin το οποίο μέσω του JIRA API, δίνει τη δυνατότητα στους προγραμματιστές να ενσωματώνουν έναν μεγάλο αριθμό λειτουργιών που δημιουργούνται είτε από την κοινότητα του JIRA είτε από τρίτους. Επιπλέον συνεργάζεται και με γνωστά IDEs όπως το Eclipse και το IntelliJ IDEA χρησιμοποιώντας το Atlassian IDE Connector ένα έργο ανοιχτού κώδικα από την Atlassian.

Πολλές ομάδες προγραμματιστών έχουν υιοθετήσει το JIRA στα έργα τους. Μερικές από αυτές είναι οι: JBoss, Spring Framework, OpenSymphony, Fedora Commons και Codehaus Xfire.

4.8 Συστήματα διαχείρισης έργων λογισμικού (project management systems)

4.8.1 SourceForge

Το SourceForce είναι μία web-based αποθήκη κώδικα. Είναι ένα κεντρικοποιημένο σύστημα στο οποίο οι προγραμματιστές μπορούν να ελέγχουν και να διαχειρίζονται την ανάπτυξη λογισμικού ανοιχτού κώδικα. Η εταιρία SourceForge Inc. χειρίζεται την ιστοσελίδα στην οποία τρέχει μία έκδοση του SourceForge Enterprise Edition. Το SourceForce φιλοξενεί πάνω από 180.000 έργα και έχει πάνω από 1,9 εκατομμύρια εγγεγραμμένους χρήστες, παρόλο που δεν είναι όλοι ενεργοί. Σύμφωνα με έρευνα της Compete.com το SourceForce.net δέχεται πάνω από 28 εκατομμύρια ετησίως.



Η ιστοσελίδα του SourceForge

Οι προγραμματιστές έχουν πρόσβαση σε ένα κεντρικοποιημένο χώρο αποθήκευσης και σε εργαλεία διαχείρισης έργων. Όμως το SourceForce είναι ευρύτερα γνωστό για την παροχή συστημάτων διαχείρισης εκδόσεων όπως CVS, SVN, Git και Mercurial. Επιπλέον, προσφέρει wikis, ανάλυση και μετρικές κώδικα, πρόσβαση σε μία βάση δεδομένων MySQL και μοναδικές διευθύνσεις sub-domain όπως: http://project-name.sourceforge.net.

4.8.2 GForge

Το GForge είναι μία σύγχρονη και επεκτάσιμη πλατφόρμα η οποία διαθέτει μία εύχρηστη διεπαφή χρήστη και μπορεί να συνδέσει ένα μεγάλο σύνολο εργαλείων, τα οποία μπορεί να είναι εργαλεία για την διαχείριση πηγαίου κώδικα μέχρι και εξαιρετικά προσαρμόσιμοι trackers, εργαλεία διαχείρισης εργασιών, εργαλεία διαχείρισης εγγράφων, forums, mailing lists. Όλα αυτά τα εργαλεία ελέγχονται από ένα κεντρικοποιημένο σύστημα και συντηρούνται αυτόματα από το σύστημα αυτό.

Λειτουργίες του GForge

Το GForge αποτελεί παρακλάδι του συστήματος SourceForge. Κυκλοφορεί υπό την άδεια GNU General Public License. Εκτός από τα εργαλεία που μπορούν να ενσωματωθούν στο GForge προσφέρει hosting σε έργα, έλεγχο εκδόσεων μέσω CVS και Subversion, σύστημα παρακολούθησης σφαλμάτων στον κώδικα, και μετάδοση μηνυμάτων.

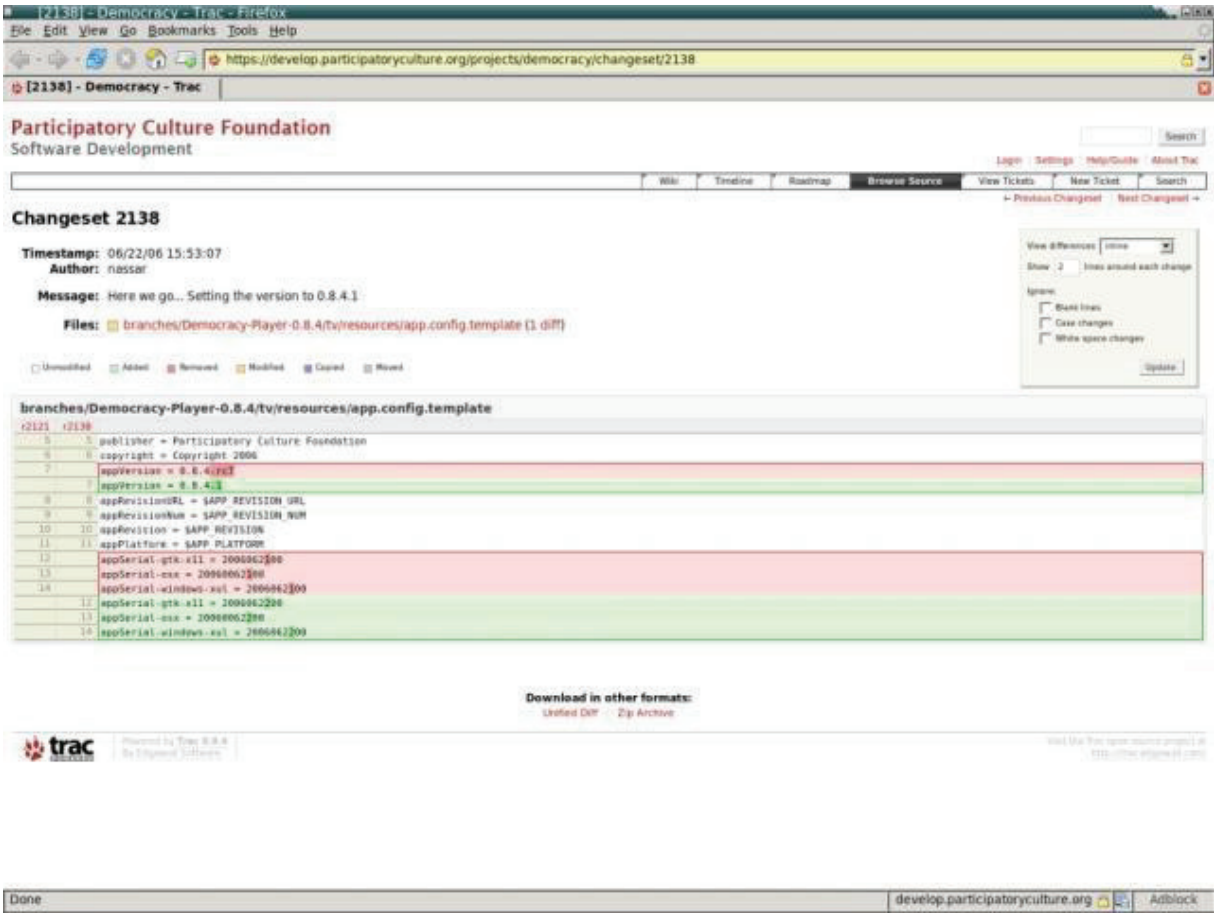
Το μόνο που χρειάζεται να κάνει κάποιος για να χρησιμοποιήσει το GForge είναι: αρχικά, να εγγραφεί στο σύστημα με τη χρήση συνθηματικού και κωδικού πρόσβασης, και μόλις επιβεβαιωθούν τα στοιχεία του να καταχωρίσει το έργο του. Όταν το έργο εγκριθεί για καταχώρηση, ο διαχειριστής το μόνο που κάνει είναι να επισκέπτεται την ιστοσελίδα και στην σελίδα διαχειριστή (admin) του παρέχεται ένας μεγάλος αριθμός υπηρεσιών για την διαχείριση του έργου του. Αυτές είναι οι παρακάτω:

- Δημιουργία νέων tracker ή επεξεργασία παλιών.
- Αλλαγή ρυθμίσεων των επιλογών roles και Observer που παρέχουν άδεια σε μέρη του έργου και ρυθμίζουν το επίπεδο πρόσβασης σε αυτά.
- Προσθήκη μελών στο έργο.
- Δημιουργία νέων εργασιών για το έργο χρησιμοποιώντας τον διαχειριστή εργασιών του χρήστη και υπολογισμός του εκτιμώμενου χρόνου ολοκλήρωσής του με τη βοήθεια του Microsoft Project.
- Σύνδεση του κώδικα που στέλνει ο προγραμματιστής με κάποια συγκεκριμένη εργασία.
- Καταχώρηση ατελειών μετά από την κυκλοφορία κάποιας έκδοσης του συστήματος.
- Αποστολή αλλαγών στον κώδικα και σύνδεση τους με τις ατέλειες που έχουν καταχωρηθεί.

Το GForge μπορεί να ενσωματωθεί στο Eclipse, το Visual Studio και το Microsoft Project με την χρήση ενός plugin διατίθεται μαζί με το GForge.

4.8.3 Trac

Το TRAC είναι ένα web-based, ανοιχτού κώδικα σύστημα, με ενσωματωμένο wiki, το οποίο χρησιμοποιείται για την διαχείριση της ανάπτυξης ενός έργου, καθώς και την παρακολούθηση προβλημάτων. Ακολουθεί μία μινιμαλιστική προσέγγιση για web-based διαχείριση ανάπτυξης έργων.



Το περιβάλλον του Trac

Η δημιουργία αυτού του συστήματος είναι εμπνευσμένη από το CVSTrac, και στην αρχή ονομάστηκε *svntrac* λόγο της ικανότητας του να αλληλεπιδρά με το Subversion. Το TRAC έχει γραφτεί στη γλώσσα προγραμματισμού Python. Μέχρι τα μέσα του 2005 κυκλοφορούσε υπό την άδεια GNU General Public License, αλλά από την έκδοση 0.9 κυκλοφορεί υπό την άδεια modified BSD license. Και οι δύο άδειες είναι ελεύθερης διάθεσης.

Το Trac επιτρέπει την διασύνδεση μεταξύ μιας βάσης δεδομένων σφαλμάτων, ενός συστήματος διαχείρισης εκδόσεων και του περιεχομένου ενός wiki. Επίσης, χρησιμοποιείται και ως διαδικτυακή διεπαφή συστημάτων διαχείρισης εκδόσεων όπως: Subversion, Git, Mercurial, Bazaar και Darcs. Περιλαμβάνει και άλλες λειτουργίες, μερικές από αυτές περιγράφονται παρακάτω:

- Υποστήριξη πολλαπλών έργων.
- Σύστημα ticket (για την παρακολούθηση σφαλμάτων, τη διαχείριση λειτουργιών).
- Παραμετροποιημένες αναφορές.
- Χρονοδιάγραμμα όλων των πρόσφατων δραστηριοτήτων.
- RSS Feed
- Επεκτασιμότητα (μέσω των plugins της Python).

Είναι καταγεγραμμένοι πάνω από 450 οργανισμοί που χρησιμοποιούν το Trac. Ανάμεσα σε αυτούς είναι το εργαστήριο Jet Propulsion της NASA, που το χρησιμοποιεί για την διαχείριση διάφορων deep space/near space έργων, και το WebKit, ένα εργαλείο που χρησιμοποιείται στον φυλλομετρητή Safari της Apple.

4.9 Μετρικές κώδικα (code metrics)

Οι μετρικές κώδικα χρησιμοποιούνται για την αξιολόγηση του κώδικα μιας εφαρμογής οι οποίες συνήθως περιλαμβάνουν μετρήσεις για τα παρακάτω:

- Αριθμός γραμμών κώδικα, που χρησιμοποιείται για την μέτρηση του μεγέθους του λογισμικού που αναπτύσσεται.
- Κυκλωματική πολυπλοκότητα (Cyclomatic Complexity), που χρησιμοποιείται για την μέτρηση της πολυπλοκότητας ενός προγράμματος. Αυτό γίνεται υπολογίζοντας τον αριθμό των γραμμικά ανεξάρτητων μονοπατιών του πηγαίου κώδικα του προγράμματος.
- Σφάλματα ανά γραμμή κώδικα.
- Κάλυψη κώδικα, που χρησιμοποιείται για να υπολογιστεί το μέγεθος του κώδικα που έχει δοκιμαστεί.
- Ταίριασμα κλάσεων, όπου μετρούνται οι εξαρτήσεις μεταξύ κλάσεων και αυτό γίνεται μέσω των παραμέτρων μιας κλάσης, των κλήσεων των μεθόδων της και των υλοποιήσεων interface.

Έχουν αναπτυχθεί αρκετά εργαλεία τα οποία χρησιμοποιούνται για την στατική ανάλυση κώδικα (static code analysis), τα οποία διαβάζουν και αναλύουν τον κώδικα ελέγχοντας για τυχόν λάθη, παραλήψεις ή μη αποδοτικές τεχνικές προγραμματισμού.

PMD

Το PMD είναι ένα στατικός αναλυτής πηγαίου κώδικα γλώσσας προγραμματισμού Java. Εκτός από τον υπολογισμό κάποιων μετρικών κώδικα, βοηθάει στη συγγραφή κώδικα χρησιμοποιώντας κάποια πρότυπα. Αυτό βοηθάει στην διατήρηση της ομοιογένειας του κώδικα, κυρίως σε περιπτώσεις όπου μια εφαρμογή αναπτύσσεται από διαφορετικές ομάδες ατόμων.

Χρησιμοποιείται για την αναγνώριση προβλημάτων που μπορεί να υπάρχουν στον κώδικα όπως:

- Πιθανά σφάλματα - Κενά *try/catch/finally/switch* blocks.

- Νεκρός κώδικας - Μη χρησιμοποιούμενες τοπικές μεταβλητές, παράμετροι ή κρυφές μέθοδοι.
- Κενές if/while δηλώσεις.
- Πολύπλοκες εκφράσεις.
- Κλάσεις με υψηλές μετρήσεις κυκλωματικής πολυπλοκότητας.
- Πολλαπλά αντίγραφα του ίδιου κώδικα - Αντιγραμμένος κώδικα μπορεί να συνεπάγεται με αντιγραμμένα σφάλματα, καθώς επίσης και την μείωση της ευκολίας συντήρησης.

Συνήθως τα λάθη που ανιχνεύει το PMD δεν είναι πραγματικά λάθη, αλλά μη αποδοτικός κώδικας, και πολλές φορές η εφαρμογή εξακολουθεί να τρέχει ακόμα και αν δεν διορθωθούν.

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Σημειώσεις για το μάθημα “Εισαγωγή στα κατανεμημένα συστήματα” Βασίλειου Ταμπακά
2. “Παράλληλα κατανεμημένα συστήματα και αλγόριθμοι” Αγγελικής Πραγιάτη, Ναύπακτος 2006
3. “Κατανεμημένα Συστήματα με Java” Ι.Κ. Κάβουρας, Ι.Ζ. Μήλης, Γ.Β. Ξυλωμένος, Α.Α. Ρουκουνάκη, Εκδόσεις Κλειδάριθμος, 2005
4. "Θεμελιώδη Ζητήματα Κατανεμημένων Συστημάτων", τόμος Ι, Πανεπιστημιακές σημειώσεις, Π. Σπυράκης, Β. Ταμπακάς, Πάτρα 1999
5. "Θεμελιώδη Ζητήματα Κατανεμημένων Συστημάτων", τόμος ΙΙ, Πανεπιστημιακές σημειώσεις, Π. Σπυράκης, Β. Ταμπακάς, Πάτρα 1999
6. “Κατανεμημένα συστήματα, αρχές και υποδείγματα”,Andrew S. Tanenbaum and Maarten van Steen, Εκδόσεις Κλειδάριθμος, 2005
7. “Προγραμματισμός και Αρχιτεκτονική Συστημάτων Επεξεργασίας”, Στυλιανός Παπαδάκης, Κωνσταντίνος Διαμαντάρας, Εκδόσεις Κλειδάριθμος, 2012
8. “Κατανεμημένα Συστήματα”, Καραϊσκος Ζαφείριος, Λάρισα 2010
9. https://github.com/ksashikumar/Ring-Election-Algorithm/blob/master/RingElection.java
10. https://github.com/ipefixe/Distributed-white-board/blob/master/Algorithms/LeLann/src/LelannMutualExclusion.java
11. http://old.ceid.upatras.gr/courses/katanemhmena/ds2/index.php/2009-09-17-08-27-11/63
12. http://stackoverflow.com/questions/7946862/java-echo-server-tcp-and-udp-implementation