Διπλωματική Εργασία :

TAP:
A Distributed Directory Service over Grid using a Peer-to-Peer as back end with Distributed Hash Tables

Νταντής Αδριανός

Επιβλέπων Καθηγητής: Σαμολαδάς Βασίλης
Μέλος Επιτροπής: Πετράκης Ευριπίδης
Μέλος Επιτροπής: Κουμπαράκης Μανόλης
Τμήμα Ηλεκτρονικών Μηχανικών και Μηχανικών Υπολογιστών
Πολυτεχνείο Κρήτης
Χανιά 2005

Επιβλέπων Καθηγητής: Σαμολαδάς Βασίλης

Μέλος Επιτροπής: Πετράκης Ευριπίδης

Μέλος Επιτροπής: Κουμπαράκης Μανόλης

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Περίληψη

Ο σκοπός αυτής της εργασίας είναι η χρησιμοποίηση των ιδιοτήτων των Grid συστημάτων και των κλιμακωτών ιδιοτήτων των untrusted peer-to-peer distributed hash tables, παρέχοντας μια υλοποίηση για Distributed Directory Services.

Η συνεισφορά αυτής της διατριβής είναι τρισκελής. Στο πρώτο σκέλος παρέχεται μια γενική περιγραφή των Grid συστημάτων και μια λεπτομερέστερη ανάλυση του Globus Toolkit και των βασικών του στοιχείων. Στο δεύτερο σκέλος παρέχεται μια γενική περιγραφή των peer-to-peer δικτύων και μια ειδικότερη για το JXTA platform. Επιπλέον, γίνεται μια αναφορά στην ιδέα των distributed indexes και τον τρόπο που χρησιμοποιούνται σε peer-to-peer (P2P) εφαρμογές και πιο συγκεκριμένα στο GISP, μια υλοποιήση ενός Distributed Hash Tables (DHT) στο JXTA. Τέλος, το τρίτο σκέλος περιλαμβάνει τη δημιουργία ενός Grid service που προσφέρει distributed directory services, υποστηριγμένο απο το JXTA και χρησιμοποιώντας Distributed Hash Tables μέσω του GISP.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέπων καθηγητή Βασίλη Σαμολαδά για την πολύτιμη βοήθεια και καθοδήγηση του κατά την διάρκεια της εκπόνησης αυτής της διπλωματικής εργασίας. Ακόμη θα ήθελα να ευχαριστήσω τους γονείς μου που προσέφεραν την δυνατότητα να βρίσκομαι σε αυτή την ευχάριστη θέση της απόκτησης του πτυχίου μου με την ολοκλήρωση αυτής της διατριβής.

Αυτή η διατριβή, καθώς και το όνομα της (ΤΑΠ), είναι αφιερωμένη σε έναν πολύ καλό μου φίλο που ήταν άρρωστος και τώρα ευτυχώς είναι πολύ καλύτερα.

Κεφάλαιο 1

Εισαγωγή

Πριν δεκαπέντε χρόνια ελάχιστοι ήταν αυτοί που πραγματικά πίστευαν στην δημιουργία της παγκόσμιας κοινότητας. Ακόμα λιγότεροι ήταν αυτοί που πίστευαν ότι το κυρίαρχο μέσο για την επίτευξη της θα ήταν η ραγδαία εξάπλωση του «παγκόσμιου ιστού» (μετά από χρόνια ονομάστηκε παγκόσμιος). Πολλές οι συνέπειες αυτής της δημιουργίας, πολιτισμικές, πολιτικές, κοινωνικές. Ευμενείς ή δυσμενείς δεν γνωρίζουμε, αυτό που είναι εντελώς προφανές είναι ο τεράστιος όγκος πληροφορίας και δυνατοτήτων που «ζεί» και «μεταβάλεται» μέσα στον παγκόσμιο ιστό. Η επιστήμη δεν έμεινε άπρακτη, μια που αυτή τον γέννησε, αναπτύχθηκε, προσαρμόστηκε και ανακάλυψε/δημιούργησε καινούργιους τομείς έρευνας και εφαρμογών.

Η ανάπτυξη της επιστήμης των υπολογιστών εξελίχθηκε και διακλαδώθηκε, ιδιαίτερα, στο πεδίο της διαχείρισης πληροφοριακών συστημάτων. Οι τεχνολογίες του internet (και Intranet) έχουν εξαπλωθεί σε όλους τους τομείς και οδηγούν τη ροή του εμπορίου, της πληροφορίας και των επικοινωνιών παγκοσμίως. Δύο από τους τεχνολογικούς τομείς που αναγκαστικά αναπτύχθηκαν ραγδαία είναι τα δίκτυα υπολογιστών (computer networks) και η κατανεμημένη υπολογιστική ισχύς (distributed computing). Δύο διαφορετικές τεχνολογίες, που βασίζονται σε αυτούς τους τομείς, έχουν αναπτυχθεί με σκοπό την καλύτερη διαχείριση υπολογιστικών κοινοτήτων μεγάλης κλίμακας (large-scale computational societies) : peer-to-peer (P2P) [11, 12, 13] και Grid computing [2]. Και οι δύο τεχνολογίες φαίνεται να έχουν τον ίδιο στόχο, αλλά βασίζονται σε διαφορετικές κοινότητες, οι οποίες επικεντρώνονται σε διαφορετικές ανάγκες και προϋποθέσεις

Ένα πρόβλημα που και οι δύο τεχνολογίες καλούνται να λύσουν είναι η ανάγκη για μία υπηρεσία που δεδομένου ενός ονόματος κάποιας οντότητας θα βρίσκει αυτή την οντότητα, έτσι θα είναι δυνατή η προσπέλαση των ιδιοτήτων της ή ακόμα και της ίδιας. Λόγω της κατανεμημένης φύσης των δύο τεχνολογιών, η υπηρεσία πρέπει να είναι και αυτή κατανεμημένη

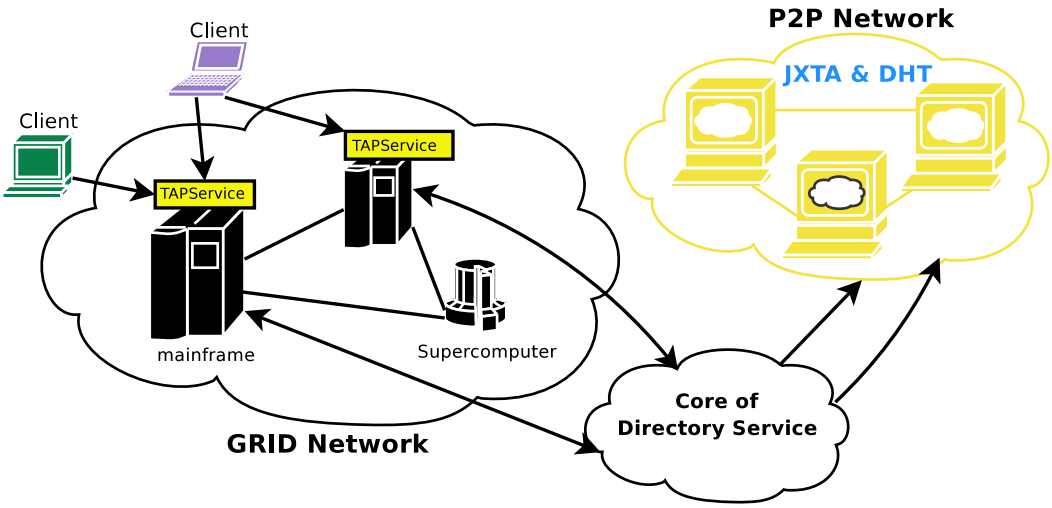
τόσο στην υλοποίηση της όσο και στις υπηρεσίες που προσφέρει. Αυτό το είδος της υπηρεσία ονομάζεται Distributed Directory Service και η υλοποίηση μιας τέτοιας υπηρεσίας είναι το αντικείμενο αυτής της εργασίας. Στις επόμενες δύο ενότητες θα γίνει μια εισαγωγική αναφορά στις τεχνολογίες Grid και Peer-to-Peer, ενώ στην τρίτη ενότητα θα κάνουμε μια εισαγωγή στην αρχιτεκτονική της υπηρεσίας μας.

1.1 Εισαγωγή στο **Grid**

Η ιδέα του Grid ξεκίνησε ως μια προσπάθεια ένωσης των υπερυπολογιστών, με κύριο άξονα επιστημονικές εφαρμογές, τώρα όμως έχει εξελιχθεί σε κάτι πολύ γενικότερο. Ως Grid ορίζεται ένα διαμοιραζόμενο περιβάλλον υλοποιημένο μέσω της ανάπτυξης μιας μόνιμης, τυποποιημένης υπηρεσιακής δομής, η οποία υποστηρίζει τη δημιουργία και διαχείριση διαμεριζόμενων πόρων εντός κατανεμημένων κοινοτήτων. Το Grid έχει τη δυνατότητα να συνδέει δυναμικά πόρους και να υποστηρίζει την εκτέλεση μιας μεγάλης κλίμακας κατανεμημένης εφαρμογής. Οι πόροι μπορεί να είναι υπολογιστές, αποθηκευτικός χώρος, αισθητήρες, προγράμματα υπολογιστών και δεδομένα, όλα όμως πρέπει να επικοινωνούν μέσω κάποιου δικτύου (Internet or Intranet). Στο Grid υπάρχει η έννοια εικονικός οργανισμός που στην ουσία είναι μια ομάδα από ατόμων και/ή ιδρυμάτων που οριοθετούνται και λειτουργούν βάση κάποιων κανόνων. Μέσω της αυστηρής πολιτικής αυξάνεται η ικανότητα των πόρων να παρέχουν ποιοτικές υπηρεσίες (Quality of Service [4]), αλλά διευκολύνουν και άλλες σημαντικές διαδικασίες, όπως την αναβάθμιση λογισμικού.

1.2 Εισαγωγή στα δίκτυα **Peer-to-Peer**

Ο όρος Peer-to-peer επικεντρώνεται στην απομάκρυνση από το συγκεντρωτικό μοντέλο του πελάτη/εξυπηρετητή σε ένα εντελώς κατανεμημένο υπολογιστικό μοντέλο. Ως Peer-to-peer (P2P) ορίζεται μια κλάση από εφαρμογές που εκμεταλλεύονται κάποιους πόρους (αποθηκευτικό χώρο, κύκλους επεξεργαστή, περιεχόμενο, την παρουσία ανθρώπου) που είναι διαθέσιμοι στο διαδίκτυο [12]. Οι πόροι είναι κατανεμημένοι και προσπελαύονται μέσα από περιβάλλον με ασταθή και αβέβαια επικοινωνία και με απρόβλεπτες διευθύνσεις IP. Το P2P είναι σχεδιασμένο έτσι ώστε να είναι ανεξάρτητο του DNS και εντελώς ή κατά κύριο λόγο αυτόνομο και χωρίς κεντρικούς εξυπηρετητές δικτύου (central servers). Οι πιο συνηθισμένες υλοποιήσεις κατασκευάζουν ένα δίκτυο με δομή ανεξάρτητη της δομής του υποκείμενου δικτύου (που συ-



Σχήμα 1.1: Τα τρία βασικά μέρη του TAP.

νήθως είναι το διαδίκτυο). Οι κόμβοι που συμμετέχουν σε αυτό το δίκτυο ονομάζονται peers και κανείς τους δεν έχει σφαιρική γνώση για ολόκληρο το δίκτυο.

1.3 Εισαγωγή στην υπηρεσία TAP

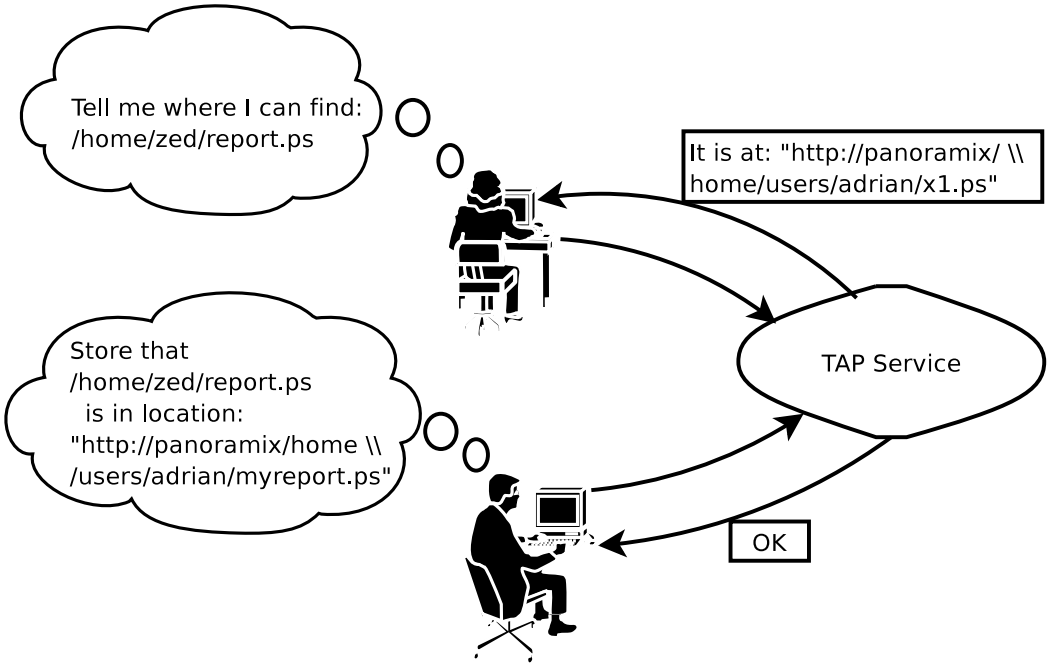
Το TAP είναι ένα σύστημα που προσφέρει Distributed Directory Services. Είναι βασισμένο σε δύο διαφορετικές τεχνολογίες, στο Grid και στα P2P δίκτυα, και αποτελείται από τρία βασικά μέρη. Το πρώτο μέρος είναι το Grid δίκτυο, και είναι υλοποιημένο με το Globus Toolkit 3.2. Το δεύτερο είναι το P2P δίκτυο και αποτελείται από 2 κομμάτια, το JXTA και το GISP. Το JXTA είναι μια πλατφόρμα ανάπτυξης P2P εφαρμογών, όπου «πάνω» του είναι υλοποιημένο το GISP, που είναι ένα DHT. Τέλος, είναι το ενδιάμεσο, Core of Directory Service (CDS), μέρος που διαμορφώνει το Directory Service και φέρνει σε επικοινωνία τα προηγούμενα δύο. Στο Σχ. 1.1 φαίνεται μια σχηματική αναπαράσταση του TAP.

Ο χρήστης έρχεται σε επικοινωνία με έναν Grid server και δηλώνει την κλήση που επιθυμεί να εκτελέσει. Αν η επικοινωνία είναι επιτυχής, ο Grid server μεταβιβάζει στο CDS την κλήση που του απεύθυνε ο χρήστης. Το CDS επεξεργάζεται την κλήση και αφού την μετατρέψει κατάλληλα, επικοινωνεί με το GISP για να φέρει σε πέρας αυτή την κλήση. Το GISP είναι ένα Distributed Hash Table και οι μόνες κλήσεις που προσφέρει είναι διάβασμα και γράψιμο ζευγαριών τύπου κλειδί/τιμή. Για να εκτελεστεί μία κλήση του χρήστη, μπορεί να χρειαστεί το CDS να εκτελέσει περισσότερες από μία κλήσεις προς το GISP.

Ο χρήστης επικοινωνεί απευθείας με τον Grid server, και μόνο με αυτόν. Για να στεφθεί με επιτυχία αυτή η επικοινωνία, πρέπει να τηρηθούν τα συγκεκριμένα πρωτόκολλα που διέπουν το Grid αλλά και η αντίστοιχη πολιτική που εφαρμόζεται από τον εικονικό οργανισμό που υπόκειται η υπηρεσία. Έτσι, το TAP χρησιμοποιεί τους αυστηρούς μηχανισμούς ασφάλειας του Grid, authorization και authedication, για να διασφαλίσει καλύτερη ασφάλεια στους χρήστες του, περισσότερα για την ασφάλεια στο Globus-Toolkit στην ενότητα 2.1.3. Εκτός από αυτού του είδους την ασφάλεια, επικίνδυνων (μοχθηρών) χρηστών, παρέχεται και ασφάλεια από το JXTA, που προστατεύει το P2P από την παραποίηση πληροφορίας από κάποιον «ψεύτικο» κόμβο. Περισσότερα για την ασφάλεια στο JXTA στην ενότητα 2.2.2.

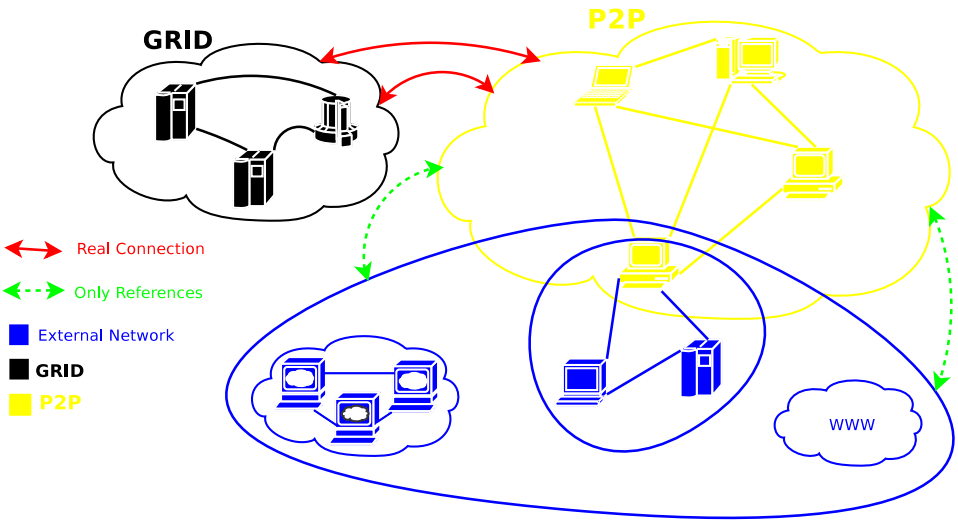
Η λειτουργικότητα που προσφέρει το TAP στους χρήστες είναι να μπορούν να δημιουργήσουν και να διαχειριστούν ένα directory structure, όπου κάθε στοιχείο του θα είναι ένα URL (RFC 2396) με κύριο μηχανισμό προσπέλασης ένα ή περισσότερα δίκτυα και με κάποιο DNS. Αυτή η δομή θα είναι γενική (global) για όλους και ο καθένας θα διαχειρίζεται ένα ή περισσότερα μέρη της ή και όλη αν έχει την κατάλληλη εξουσιοδότηση. Ο χρήστης θα μπορεί να δημιουργήσει, να διαγράψει και να ανακτήσει κάποια καταχώρηση από την δομή. Για παράδειγμα, θα μπορεί να εισάγει ότι το όνομα "/project/tap/documents/thesis" αντιστοιχεί στο αρχείο που βρίσκεται στη θέση που δηλώνει το URL="http://panoramix.tuc.gr/adrian/thesis.ps". Η δομή υποστηρίζει permissions στην μορφή του UNIX, δηλαδή read:write (rw). Υποστηρίζεται και η δυνατότητα δημιουργίας ομάδων (groups) και ομαδοποίησης των χρηστών όπως και στο UNIX. Οπότε, κάθε καταχώρηση της δομής έχει permission της μορφής "rw:rw:rw", όπου η πρώτη τριάδα αντιστοιχεί στον ιδιοκτήτη (owner) της καταχώρησης, η δεύτερη στην ομάδα (group) και η τρίτη έχει γενικό χαρακτήρα και απευθύνεται σε όλους (all) τους χρήστες. Μια δυνατή χρήση φαίνεται και στο Σχ. 1.2.

Κάθε στοιχείο της δομής (αρχείο ή κατάλογος) θα αναφέρεται (δείχνει) και σε ένα URL, στο ίδιο URL μπορούν να δείχνουν και περισσότερα από ένα στοιχεία. Κάθε URL θα δείχνει και σε μια διεύθυνση ενός αρχείου ή καταλόγου, κάπου σε έναν υπολογιστή ενός δικτύου. Το δίκτυο αυτό μπορεί να είναι κάποιο (ή καποια) τοπικο δίκτυο ή και όλο το διαδίκτυο. Συμπερασματικά, το TAP κάνει αναφορά σε τρία δίκτυα, το Grid δίκτυο, το P2P δίκτυο και το εξωτερικό δίκτυο που δείχνουν τα URLs. Τα τρία αυτά δίκτυα μπορούν να είναι εντελώς ανεξάρτητα μεταξύ τους, από πλευράς μελών, και αρκεί μονο να υπάρχει επικοινωνία μεταξύ του πρώτου και του δεύτερου. Επίσης, οποιοδήποτε δίκτυο από τα τρία, μπορεί να είναι υπερσύνολο ή και υποσύνολο των άλλων δύο. Για παράδειγμα, μπορεί το «εξωτερικό» δίκτυο να αναφέρεται σε υπολογιστές που είναι μέλη του P2P αλλά και σε άλλους που είναι μέλη του



Σχήμα 1.2: Παράδειγμα χρήσης του TAP με δύο χρήστες.

Grid. Στην Σχ. 1.3 φαίνεται μια πιθανή λειτουργία του TAP, με άξονα τα επιμέρους δίκτυα.



Σχήμα 1.3: Μια πιθανή περίπτωση ανάμιξης των τριών δικτύων.

Στο TAP χρησιμοποιήσαμε τη μέθοδο-δομή Distributed Hash Tables (DHT) για καλύτερη απόδοση, κυρίως όταν ψάχνουμε (lookup) για ένα αντικείμενο. Για να την χρησιμοποιήσουμε έπρεπε πρώτα να κατασκευαστεί μια δομή δεδομένων μέσα στο DHT. Αυτή την υλοποιήσαμε, στηριζόμενοι στο ημερολόγιο (log) των «πράξεων» που έχουν γίνει. Κάθε χρήστης ή εφαρμογή που χρησιμοποιεί το TAP δεν αποθηκεύει τη δομή δεδομένων (αρχεία-καταλόγους) του κατευθείαν μέσα στο TAP. Αντίθετα, η δομή δεδομένων συνεπάγεται από την ιστορία των πράξεων στα logs και το TAP, συγκεκριμένα το CDS, αποθηκεύει τις εγγραφές των logs στο GISP. Οι χρήστες ανανεώνουν τη δομή δεδομένων προσθέτοντας εγγραφές από το ημερολόγιο. Στηριζόμενο στο ημερολόγιο (logging), το TAP είναι πιο ανεκτικό στα λάθη-αποτυχία του πελάτη (client failure). Το αποτέλεσμα που θα παραχθεί, αν έχουμε ταυτόχρονες ανανεώσεις στο ίδιο τμήμα της δομής δεδομένων, είναι αποδεκτό και δεν οδηγεί σε προβληματική (διεφθαρμένη) δομή. Αυτό επιτυγχάνεται συνδυάζοντας το logging, που αποτρέπει την κατάρρευση της δομής αφού το μόνο που θα γίνει θα είναι μια εισαγωγή μιας εγγραφής ημερολογίου, και την χρησιμοποίηση των ρολογιών Lamport (περισσότερα στην ενότητα3.2), για το συγχρονισμό των εγγραφών.

Η δομή αυτής της αναφοράς έχει ως εξής: Στο κεφάλαιο 2 παρουσιάζουμε τη σχετική δουλειά που υπάρχει, συγκεκριμένα αναφερόμαστε στην πρώτη ενότητα στα Grid, Globus Toolkit, Grid services, στη δεύτερη στα P2P, JXTA, GISP και στις υπόλοιπες σε κάποια άλλα πιο ειδικά θέματα. Στο τρίτο κεφάλαιο αναλύουμε εκτενώς την υπηρεσία TAP και κάνουμε

μερικά σχόλια για συμπεράσματα και μελλοντική δουλειά.

Κεφάλαιο 2

Σχετική δουλειά

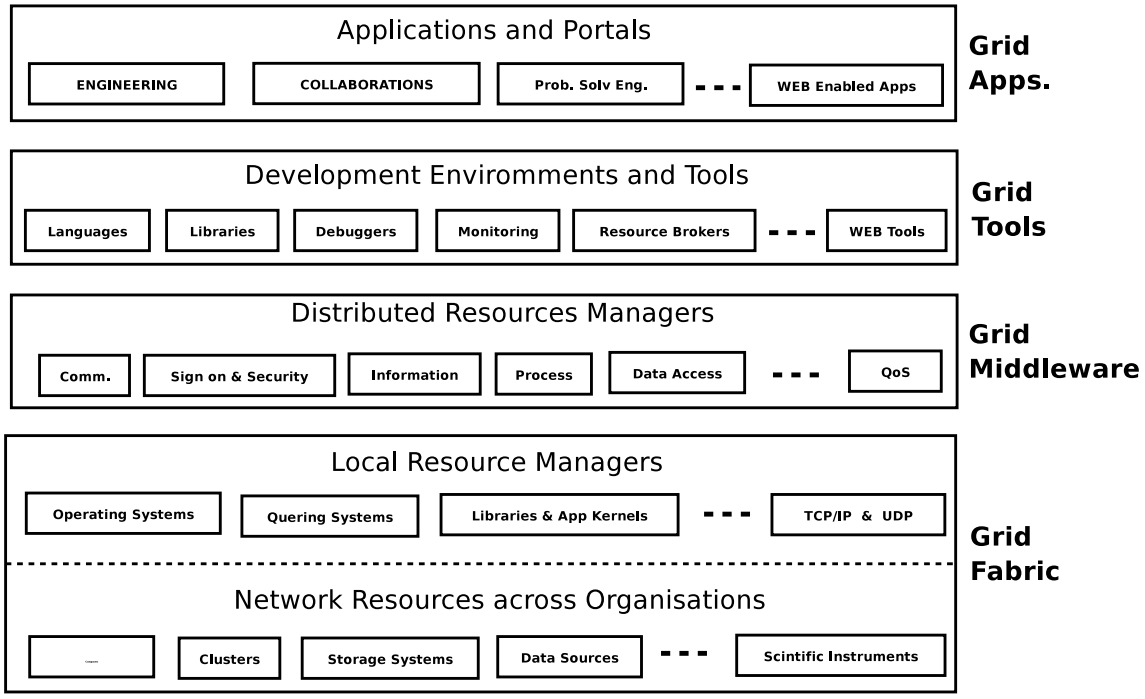
2.1 **Grid**

2.1.1 Η αρχιτεκτονική του **Grid**

Το Grid διαμορφώνει μια εικονική πλατφόρμα, έχοντας ενσωματωμένους τους πόρους, τη διαχείριση και την επικοινωνία τους. Τα μεγάλης κλίμακας Grid δίκτυα είναι πραγματικά κατανεμημένα, ετερογενή και δυναμικά συστήματα που υπόσχονται άπειρους υπολογιστικούς κύκλους και αποθήκευση, καθώς επίσης και πρόσβαση σε όργανα, συσκευές απεικόνισης, κ.ο.κ., αδιαφορώντας για τη γεωγραφική τους θέση. Για να επιτευχθεί αυτή η λειτουργία, ένα Grid πρέπει να είναι έτσι σχεδιασμένο που να παρέχει ένα ενιαίο υπολογιστικό περιβάλλον σε όλους τους χρήστες και παράλληλα να υπάρχει αποτελεσματική συνένωση των πόρων.

Κάθε Grid πρέπει να πληρεί και τις τρεις παρακάτω προδιαγραφές:

- Ετερογενές: Η πληθωρικότητα του είδους των πόρων, που μπορεί να αποτελείται από διάφορα είδη πόρων, τα οποία μπορεί να υπόκεινται σε διαφορετικές πολιτικές διαχείρισης.
- Κλιμακωτό: Ο αριθμός των πόρων, που μπορεί να είναι πάρα πολλοί και απομακρυσμένοι. Οπότε, οι εφαρμογές που χρησιμοποιούν πόρους σε μεγάλη γεωγραφική έκταση πρέπει να είναι ανεκτικές στις καθυστερήσεις.
- Δυναμικό ή Προσαρμοστικό: Στο Grid η αποτυχία κάποιου πόρου ή υπηρεσίας δεν θεωρείται η εξαίρεση αλλά ο κανόνας. Ο διαχειριστής των πόρων ή η εφαρμογή πρέπει να συμπεριφέρεται δυναμικά, ώστε να λαμβάνει την μέγιστη απόδοση από κάθε πόρο και υπηρεσία.



Σχήμα 2.1: Τα τμήματα ενός *Grid*.

Κάθε Grid αποτελείται από τέσσερα τμήματα, διαχωρισμένα σύμφωνα με το ρόλο που διαδραματίζουν μέσα στο ίδιο το Grid. Μια σχηματική απεικόνισή τους φαίνεται στην Σχ. 2.1 και εξηγούνται συνοπτικά παρακάτω.

Το Grid Fabric αποτελείται από όλους τους πόρους, που συνήθως είναι γεωγραφικά κατανεμημένοι και προσβάσιμοι μέσω του διαδικτύου. Οι πόροι μπορεί να είναι υπολογιστές (κάθε είδους) που τρέχουν κάποιο λειτουργικό όπως το UNIX ή τα Windows. Μπορεί να είναι clusters και να τρέχουν λειτουργικό σύστημα για cluster ή άλλου είδους όπως το Condor. Επίσης, μπορεί να είναι βάσεις δεδομένων, αποθηκευτικοί χώροι ή κάποια επιστημονικά όργανα.

Το Grid Middleware προσφέρει βασικές υπηρεσίες όπως απομακρυσμένη διαχείρισή διεργασιών (remote process management), διαπίστευση (Authedication), δέσμευση πόρων, πρόσβαση σε αποθηκευμένη πληροφορία, κ.α. .

Το τμήμα των Grid Tools διαθέτει υψηλού επιπέδου υπηρεσίες που επιτρέπουν στους προγραμματιστές να αναπτύξουν εφαρμογές. Επιπλέον, διαθέτει «μεσίτες» (brokers) που λειτουργούν ως αντιπρόσωποι του χρήστη-προγραμματιστή και μπορούν να διαχειριστούν ή να δρομολογήσουν υπολογισμούς μέσω των γενικών πόρων.

Στο Grid Applications περιλαμβάνονται εφαρμογές διαφόρων ειδών. Εδώ μπορεί ο καθένας

να χρησιμοποιήσει κάποια εφαρμογή που ήδη υπάρχει ή και να χτίσει την δικιά του πάνω στις ήδη υπάρχουσες. Οι εφαρμογές μπορούν να αναπτυχθούν σε γλώσσες που είναι κατάλληλες για Grid όπως η Java, η C++, το σύστημα MPI. Τα Grid Portals προσφέρουν εφαρμογές, με τη μορφή υπηρεσιών στο διαδίκτυο. Οι χρήστες μπορούν να υποβάλλουν το πρόβλημα τους ή να συλλέξουν τα αποτελέσματα ενός προβλήματος μέσω ενός web interface.

2.1.2 Grid Applications

Αν και αρχικά η τεχνολογία Grid αναπτύχθηκε κυρίως για επιστημονικούς σκοπούς, το εμπορικό ενδιαφέρον είναι μεγάλο. Σήμερα, πολλές είναι οι εφαρμογές που αναπτύσσουν Grid συστήματα, είτε αυτές έχουν ερευνητικό είτε εμπορικό σκοπό. Παρακάτω, αναφέρουμε μερικές από τις πιο σημαντικές. Μια μεγαλύτερη λίστα μπορεί να βρεθεί στα [5, 6].

- Globus: www.globus.org (USA)
- NetSolve: www.cs.utk.edu/netsolve (USA)
- Legion: legion.virginia.edu (USA)
- NASA IPG: www.ipg.nasa.gov (USA)
- UNICORE: juelich.de/unicore (Europe)
- Globe: www.cs.vu.nl/ steen/globe (Europe)
- CERN Data Grid: grid.web.cern.ch/grid (Europe)

2.1.3 Globus Toolkit

Το Globus Project [7] είναι μια συνδυασμένη προσπάθεια από ερευνητές και προγραμματιστές από όλο τον κόσμο εστιασμένη στα υπολογιστικά Grid και είναι οργανωμένο σε τέσσερις δραστηριότητες: στην έρευνα, στην ανάπτυξη λογισμικού, σε πραγματικές δοκιμές και σε εφαρμογές.

Με στενή συνεργασία με πραγματικά προγράμματα Grid στην επιστημονική έρευνα και τη βιομηχανία, αναπτύσσει και προωθεί τα τυποποιημένα πρωτόκολλα Grid (από το Global Grid Forum (GGF) [8]) και είναι βασισμένο στο Open Grid Service Architecture (OGSA), μια υλοποίηση του Open Grid Services Infrastructure (OGSI). Για να επιτρέψει τη δια-λειτουργικότητα και την κοινή υποδομή, αναπτύσσει και προωθεί τυποποιημένο λογισμικό (APIs) και Grid

SDKs για να επιτρέψει τη διανομή και τη φορητότητα του κώδικα. Επιπλέον προσφέρεται ένα αρθρωτό «σύνολο τεχνολογιών» και επιτρέπει την επαυξητική ανάπτυξη εργαλείων και εφαρμογών πάνω σε Grid. Με όλα τα παραπάνω αποτελεί μια ανοικτή πηγή - βάση λογισμικού ως αναφορά για την οικοδόμηση της υποδομής και των εφαρμογών του Grid.

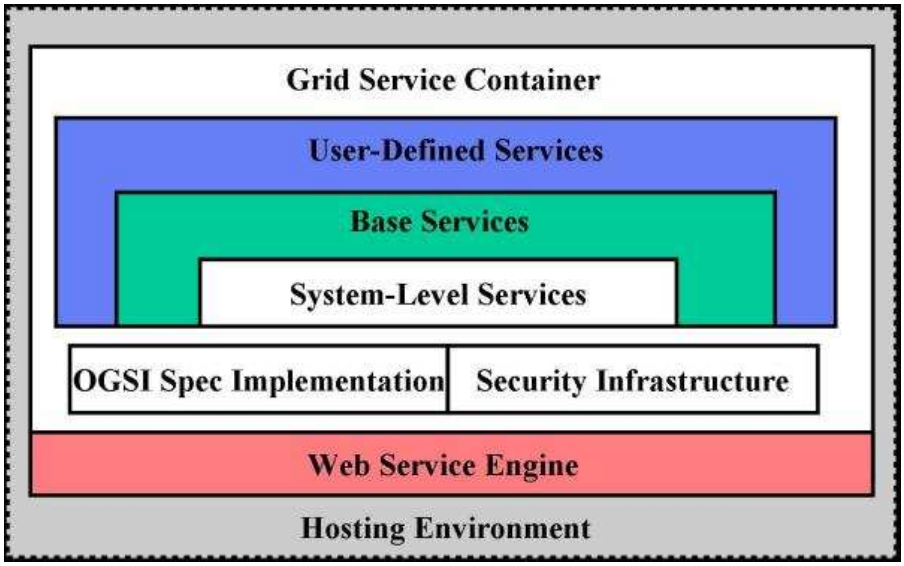
Στα πλαίσια αυτής της διπλωματικής θα αναφερθούμε στο περιβάλλον του Globus Toolkit 3.2 και στα βασικά κομμάτια από τα οποία αυτό αποτελείται. Ουσιαστικά αποτελείται από το βασικό πυρήνα (Core) του Globus Toolkit, την υποδομή της ασφάλειας και 3 βασικές υπηρεσίες:

- Ο πυρήνας του Globus (Core of Globus toolkit), που αποτελείται από τις βασικές και απαραίτητες υπηρεσίες για κάθε υπόσταση του.
- Υποδομή Ασφάλειας (Grid Security Infrastructure), παρέχει συναρτήσεις ασφάλειας όπως ατομική ή αμοιβαία διαπίστευση, εμπιστευτική επικοινωνία, και εξουσιοδότηση.
- Υπηρεσίες Διαχείρισης Δεδομένων (Data Management Services), προσφέρουν υποστήριξη για μεταφορά αρχείων ανάμεσα σε μηχανές του Grid και διαχείριση των μεταφορών αυτών.
- Υπηρεσίες Διαχείρισης Πόρων (Resource Management Services), υποστηρίζουν κατανομή πόρων, υποβολή εργασίας σε απομακρυσμένο μηχάνημα και λήψη αποτελεσμάτων και διαχείριση της διεργασίας αυτής.
- Υπηρεσίες Πληροφορίας (Information Services), προσφέρουν υπηρεσίες για τη συλλογή πληροφοριών πάνω στο Grid και ερωτήσεις πάνω στις πληροφορίες αυτές.

Ο πυρήνας του **Globus**

Τα λευκά τμήματα του Σχ. 2.2 αποτελούν τα μέρη του πυρήνα του Globus, και όλα μαζί αποτελούν τα βασικά (απαραίτητα) τμήματα για τα Grid services.

- Το *OGSI Reference Implementation* παρέχει υλοποιημένα όλα τα προκαθορισμένα από το OGSI interfaces, με τη μορφή APIs καθώς και διαφόρων εργαλείων.
- Το *Security Infrastructure* παρέχει τα εξής: το SOAP ως transport level message protection, end-to-end mutual authetication, single sign-on service authorization.
- Τα *System-Level Services* είναι run-time services που είναι πολύ γενικά και μπορούν να χρησιμοποιηθούν από όλα τα άλλα Grid services.



Σχήμα 2.2: Τα τμήματα του πυρήνα του Globus.

- Τα τρία προηγούμενα μαζί με τα User-Defined Services, Base Services, αλληλεπιδρούν με το Grid Service Container. Το τεύταιο είναι ένα τυπικό run-time περιβάλλον ανάπτυξης υπηρεσιών.

Υπηρεσίες Ασφάλειας

Το Globus Toolkit χρησιμοποιεί την υποδομή ασφάλειας (Grid Security Infrastructure GSI) για να κάνει δυνατή την ασφαλή διαπίστευση και επικοινωνία πάνω από ένα ανοικτό δίκτυο. Παρέχει επίσης διάφορες χρήσιμες υπηρεσίες, συμπεριλαμβανομένης της αμοιβαίας διαπίστευσης και ενιαίο single sign-on. Το GSI βασίζεται στη κρυπτογράφηση με δημόσιο κλειδί, X.509 πιστοποιητικά, και το ασφαλές πρωτόκολλο επικοινωνίας SSL. Επίσης, επεκτάσεις σε αυτά τα πρότυπα έχουν προστεθεί για την υποστήριξη επιπλέον λειτουργικότητας.

Τα αρχικά κίνητρα πίσω από το GSI είναι:

- Η ανάγκη για την ασφαλή επικοινωνία μεταξύ των στοιχείων ενός υπολογιστικού Grid.
- Η ανάγκη να υποστηριχθεί η ασφάλεια μέχρι τα όρια της εφαρμογής, αποφεύγοντας ένα κεντρικά-ρυθμισμένο σύστημα ασφάλειας.
- Η ανάγκη να υποστηριχθεί single sign-on για τους χρήστες του Grid, συμπεριλαμβα-

νομένου delegation of credentials για τους υπολογισμούς που περιλαμβάνουν πολλούς πόρους.

Το βασικό κομμάτι του GSI είναι το CAS, το οποίο επιτρέπει στους προμηθευτές των πόρων να προσδιορίσουν τις πολιτικές ελέγχου πρόσβασης σε αυτούς γενικά (π.χ. σε ομάδες χρηστών), αλλά και λεπτομερή εξουσιοδότηση - πολιτική διαχείρισης ελέγχου πρόσβασης σε συγκεκριμένους χρήστες. Τέλος, αν και οι προμηθευτές των πόρων διατηρούν την απόλυτη κυριότητα πάνω σε αυτούς, τίθενται οι ίδιοι κάτω από τις καθημερινές πολιτικές διαχείρισης του Grid (π.χ. προσθήκη και διαγραφή των χρηστών, τροποποίηση των προνομίων τους κ.λ.π.).

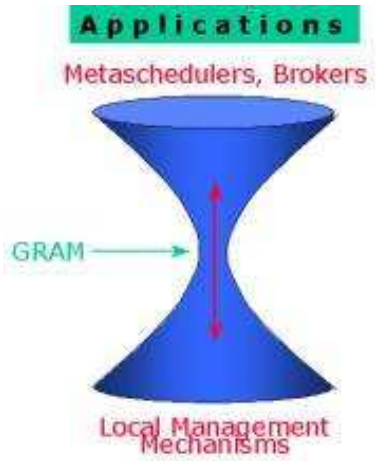
Υπηρεσίες Διαχείρισης Δεδομένων

Κατά την οικοδόμηση ενός Grid, το σημαντικότερο στοιχείο μέσα σε αυτό είναι τα δεδομένα, για τα οποία είναι αναγκαίο να καθορισθούν οι απαιτήσεις τους, το πώς θα κινούνται αυτά γύρω από την υποδομή του Grid και τέλος να ορισθεί η πρόσβαση στα απαραίτητα δεδομένα κατά τρόπο ασφαλή και αποδοτικό. Αυτές τις προδιαγραφές έρχεται να καλύψει η υπηρεσία διαχείρισης δεδομένων, η οποία αποτελείται από τα παρακάτω κομμάτια :

- Το GridFTP, ένα υψηλής απόδοσης, ασφαλές και αξιόπιστο πρωτόκολλο μεταφοράς δεδομένων, βελτιστοποιημένο για high-bandwidth wide-area networks το οποίο είναι βασισμένο στο γνωστό FTP. Έχει επιλεχθεί ένα σύνολο χαρακτηριστικών πρωτοκόλλου και επεκτάσεις οι οποίες καθορίστηκαν σε IETF RFCs, καθώς επίσης προστέθηκαν χαρακτηριστικά για να καλύψουν τις απαιτήσεις από τα τρέχοντα προγράμματα Grid.

- Αξιόπιστη υπηρεσία μεταφοράς αρχείων (RFT) είναι μια υπηρεσία βασισμένη σε OGSA που παρέχει τη διεπαφή για την επίβλεψη και τον έλεγχο των μεταφορών αρχείων τρίτων χρησιμοποιώντας τους κεντρικούς εξυπηρετητές GridFTP.

- Υπηρεσία θέσης αντιγράφου (RLS), η οποία διατηρεί και παρέχει την πρόσβαση στις πληροφορίες χαρτογράφησης - αντιστοίχησης από τα λογικά ονόματα για τα αντικείμενα δεδομένων σε target ονόματα. Τα τελευταία μπορούν να αντιπροσωπεύσουν τις φυσικές θέσεις των αντικειμένων, ή μια είσοδο στο RLS που χαρτογραφεί ένα άλλο επίπεδο λογικής ονομασίας.



Σχήμα 2.3: Η υπηρεσία διαχείρησης δεδομένων (GRAM).

- Την υπηρεσία XIO που είναι μια επέκταση της βιβλιοθήκης εισόδου/εξόδου του Globus Toolkit, η οποία προσφέρει ένα απλό και διαισθητικό API (άνοιγμα, κλείσιμο, διάβασμα, γράψιμο) για swappable IO υλοποιήσεις.

Υπηρεσίες Διαχείρισης Πόρων

Η υπηρεσία αυτή, γνωστή και ως GRAM (Globus Resource Allocation Manager), παρέχει μια ασφαλή και ελεγχόμενη απομακρυσμένη πρόσβαση σε ετερογενείς πόρους, καθώς και τη διαχείριση απομακρυσμένων υπολογισμών. Το GRAM απλοποιεί τη χρήση των απομακρυσμένων συστημάτων, με την παροχή μιας ενιαίας, ομοιόμορφης και ευέλικτης διεπαφής για την αίτηση και τη χρησιμοποίηση απομακρυσμένων πόρων για την εκτέλεση εργασιών. Η πιο κοινή χρήση του GRAM (και η καλύτερα υποστηριζόμενη) είναι απομακρυσμένη υποβολή και έλεγχος εργασιών και χρησιμοποιείται για να υποστηρίξει τις διανεμημένες εφαρμογές υπολογισμού.

Το GRAM χρησιμοποιεί την υποδομή ασφάλειας Grid (GSI) ώστε να παρέχει την αμοιβαία διαπίστευσης χρηστών και απομακρυσμένων πόρων. Στόχος του είναι να μειώσει τον αριθμό των ελέγχων που απαιτούνται για τη χρησιμοποίηση απομακρυσμένων πόρων (όπως χρονοπρογραμματισμός, συστήματα αναμονής, συστήματα reservation, και διεπαφές ελέγχου). Αυτή η ικανότητα παρομοιάζεται (όπως και πολλά άλλα κομμάτια του Globus) με το λαιμό της κλεψύδρας, με τις εφαρμογές και τις υψηλότερου επιπέδου υπηρεσίες (όπως brokers, metaschedulers) επάνω και τους τοπικούς μηχανισμούς ελέγχου και πρόσβασης από κάτω, όπως

φαίνεται στην Σχ. 2.3. Και οι δύο πλευρές εργάζονται μόνο με το GRAM, έτσι ο αριθμός των αλληλεπιδράσεων, τα APIs και τα πρωτόκολλα που πρέπει να χρησιμοποιηθούν μειώνονται πολύ.

Το Globus Toolkit 3 παρέχει δύο υλοποιήσεις του GRAM. Η πρώτη είναι αυτή που συμφωνεί με τις ιδιότητες και τα πρωτόκολλα του OGSI (WS GRAM), ενώ η δεύτερη (Pre-WS GRAM) είναι αυτή που υπήρχε και στο Globus Toolkit 2.x και προσφέρεται κυρίως για λόγους συμβατότητας.

Υπηρεσίες Πληροφορίας

Οι υπηρεσίες πληροφορίας είναι ένα ζωτικής σημασίας κομμάτι της υποδομής του Grid. Διατηρεί τη γνώση για τη διαθεσιμότητα, την ικανότητα καθώς και την τρέχουσα χρησιμοποίηση των πόρων. Μέσα σε οποιοδήποτε Grid, τόσο οι πόροι δεδομένων όσο και οι πόροι CPU θα κυμανθούν, ανάλογα με τη διαθεσιμότητά τους για να επεξεργαστούν και να μοιραστούν δεδομένα, καθώς αυτοί οι πόροι δεσμεύονται και απελευθερώνονται, ανανεώνουν τη διαθεσιμότητά τους στις υπηρεσίες πληροφορίας έτσι ώστε ο πελάτης, ο μεσίτης (broker) ή ο διαχειριστής των πόρων του Grid να χρησιμοποιήσει αυτή την πληροφορία για να λάβει ενημερωμένες αποφάσεις σχετικά με τις αναθέσεις των πόρων. Η υπηρεσία πληροφορίας έχει ως σκοπό να παρέχει:

- Αποδοτική ανάκληση της πληροφορίας κατάστασης για κάθε συγκεκριμένο πόρο.
- Κοινούς μηχανισμούς έρευνας και ανακάλυψης πόρων σε όλες τις οντότητες Grid.

Οι παροχείς υπηρεσιών πληροφορίας είναι προγράμματα που παρέχουν τις πληροφορίες στον κατάλογο για την κατάσταση των πόρων. Παραδείγματα της πληροφορίας που συγκεντρώνονται :

- Στατική πληροφορία του Host (όπως για το λειτουργικό σύστημα όνομα και έκδοση, για τον επεξεργαστή κατασκευαστή, μοντέλο, ταχύτητα, cache, αριθμός επεξεργαστών, για τη μνήμη τη συνολική φυσική και εικονική, για συσκευές, για υπηρεσίες όπως τύπο, πρωτόκολλο, port κ.ο.κ.).
- Δυναμική πληροφορία για το φόρτο εργασίας του κόμβου, τις καταχωρήσεις στη σειρά αναμονής, κ.ο.κ. .
- Πληροφορία για σύστημα αποθήκευσης, όπως συνολικός αποθηκευτικός χώρος, ελεύθερος χώρος, κ.ο.κ. .

- Πληροφορία δικτύου, όπως bandwidth , καθυστέρηση (latency), μετρημένη και προβλεφθείσα κ.ο.κ. .

- Ιδιαίτερα δυναμική πληροφορία για την διαθέσιμη ελεύθερη φυσική και εικονική μνήμη, ελεύθερος αριθμός επεξεργαστών, κ.ο.κ. .

Η υλοποίηση αυτής της υπηρεσίας πληροφοριών έχει δύο εκδοχές. Η πρώτη είναι ως μια υπηρεσία παρακολούθησης και ανακάληψης (Monitoring and Discovery Service MDS), όπου χρησιμοποιεί το ελαφρύ πρωτόκολλο πρόσβασης καταλόγου, το OpenLDAP (Lightweight Directory Access Protocol), ως διεπαφή για την πληροφορία των πόρων, και υπήρχε στο Globus Toolkit 2.x. Το LDAP (RFC 2251) είναι ένα πρωτόκολλο για πρόσβαση και «εργασία» σε καταλόγους μέσω TCP/IP, και το OpenLDAP είναι μια ανοιχτή υλοποίηση του LDAP. Στους LDAP καταλόγους αποθηκεύεται πληροφορία με μοναδικό όνομα και τα σχετικά χαρακτηριστικά της. Κάθε χαρακτηριστικό έχει έναν τύπο και μια τιμή.

Με την έκδοση του GT3 παρουσιάστηκε μια άλλη πρόταση για την υλοποίηση αυτής της υπηρεσίας, που ονομάστηκε Index Service. Κάθε Grid service instance έχει μια συγκεκριμένη παρτίδα από δεδομένα υπηρεσίας σχετικά με αυτό. Η ιδέα του Index Service είναι να προσφέρει ένα interface για τις διαδικασίες που δημιουργούν, συναθροίζουν και ερωτούν για αυτά τα δεδομένα υπηρεσίας. Το Index Service χρησιμοποιεί μια εκτεταμένη δομή για να διαχειρίζεται στατικά και δυναμικά δεδομένα για Grids που είναι χτισμένα με το GT3. Το Index Service δεν γεννάει ειδικούς τύπους δεδομένων, αλλά αυτοί εξαρτώνται από την κάθε υπηρεσία και το πως είναι σχηματισμένη. Το Index Service περισυλλέγει, συναθροίζει, και ερωτά για Service Data, παρακολουθεί data feed και δημιουργεί δεδομένα υπηρεσίας δυναμικά και κατά απαίτηση. Μπορεί να χρησιμοποιηθεί για δεικτοδότηση (indexing) της πληροφορίας κατάστασης των δεδομένων υπηρεσίας από πολυάριθμα grid service instances, για χρησιμοποίηση σε αναζήτηση, επιλογή και βελτιστοποίηση των πόρων.

Το Index Service προσφέρει τις εξής βασικές ικανότητες:

- Ένα interface για να συνδέονται εξωτερικές Service Data Provider εφαρμογές στα service instances.

- Μια γενική υποδομή για να υπάρχει η δυνατότητα να συναθροίζονται (aggregate) τα δεδομένα υπηρεσίας.

- Μια Registry των Grid υπηρεσιών.

- Ένα δυναμικό data-generating and indexing node, κατάλληλο για χρήση σε ιεραρχικές υπηρεσίες. Αυτό λειτουργεί στο ίδιο σκεπτικό με το SDK2 και παρέχει παρόμοιες λειτουργίες.

Η ανεύρεση δεδομένων (πόρων) συχνά απαιτεί συγκεκριμένη, και κάποιες φορές δυναμική πληροφορία για τα instances. Έτσι, κάθε υπηρεσία πρέπει να παρέχει τουλάχιστον κάποια κοινά δεδομένα υπηρεσίας, αλλά και περισσότερα αν το επιδιωκει. Αυτά τα κοινά δεδομένα θα πρέπει να έχουν τη δυνατότητα να απαντούν στις παρακάτω συναρτήσεις (λειτουργίες):

- *FindServiceData*: Παρέχει την λειτουργία να συναθροίζει δεδομένα υπηρεσίας από άλλες υπηρεσίες.

- *NotificationSource*: Χρησιμοποιείται για συνδρομή (subscription) πελατών στο να εκδηλώσουν ενδιαφέρον για μια υπηρεσία και μπορεί να στέλνει μηνύματα γνωστοποίησης (notification).

- *Pull/Push (synchronous/asynchronous)*: Η συμμετοχή (subscription) και FindServiceData είναι φόρμες ερώτησης για δεδομένα υπηρεσίας. Το FindServiceData είναι σύγχρονη (pull) ερώτηση, ενώ subscription notification είναι ασύγχρονη απάντηση (push).

- *Registry*: Πρέπει να υποστηρίζει την ανεύρεση, επιστρέφοντας τα GSHs των ανάλογων υπηρεσιών. Το GSH είναι ένα URL και κάθε υπηρεσία έχει το ίδιο για πάντα. Δεν δίνει καμία περιγραφή αλλά έχει τη μορφή ταυτότητας και δείχνει στο GSR, το οποίο δίνει περιγραφή της υπηρεσίας. Το Registry επιτρέπει έγγραφή soft-state για τις υπηρεσίες, ώστε να μπορούν να εγγραφούν οι ίδιες τα δικά τους GSH, και έτσι να επιτρέπουν την ανεύρεσή τους.

2.1.4 Grid Service

Σε αυτή την ενότητα θα εξηγήσουμε τι είναι ένα Grid service, αναδεικνύοντας τα πλεονεκτήματα που έχει έναντι ενός Web service, αλλά και αναφέροντάς τα κύρια χαρακτηριστικά του. Θα αναφερθούμε στον όρο Grid Service όπως προκύπτει από την OGSA (Open Grid Service Architecture). Τρία χαρακτηριστικά που χρησιμοποιούνται από τα Web Services και υιοθετήθηκαν από τα Grid Services είναι το SOAP (Simple Object Acess Protocol), η WSDL (Web Service Description Language [9]) και το WS-Inspection [10]. Το WS-Inspection αποτελεί μια υιοθέτηση μερικών απλών XML υποθέσεων για την εύρεση περιγραφών υπηρεσιών, που δημοσιεύθηκαν από τους ίδιους τους προμηθευτές της υπηρεσίας (service provider).

Τα Web Services είναι ευρέως διαδεδομένα, αλλά τους λείπουν κάποια επιθυμητά χαρακτηριστικά, τα οποία διαθέτουν τα Grid Services. Αυτά είναι τα εξής (και αναλύονται παρακάτω):

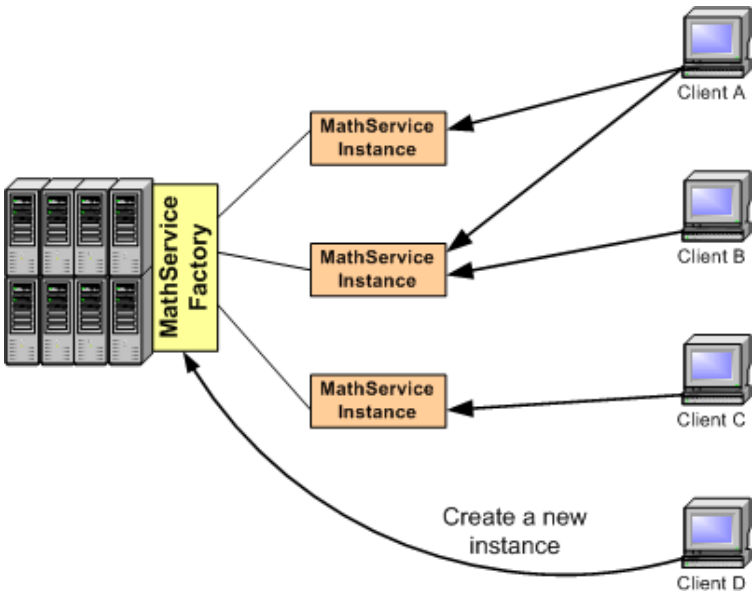
- Stateful and potentially transient services
- Service Data
- Notifications
- Service Groups
- portType extension
- Lifecycle management
- GSH & GSR

Stateful and potentially transient services

Το πρώτο χαρακτηριστικό είναι και από τα πιο σημαντικά. Τα Grid Services είναι stateful, δηλαδή έχουν τη δυνατότητα να «θυμούνται» τι είχε γίνει στις προηγούμενες κλήσεις του κάθε πελάτη. Αν ένας πελάτης, προκειμένου να εκτελέσει μια πάρα πολύ μεγάλη διαδικασία, χρειαστεί να εκτελέσει μια αλυσίδα από μικρότερες, το Grid Service παρέχει αυτή τη δυνατότητα χωρίς να χρειαστεί ο πελάτης να περνάει ως παράμετρο τα αποτελέσματα της προηγούμενης στην επόμενη. Κάτι που υποστηρίζεται από ελάχιστα Web Services, και όχι κατά το μέτρο και τις ιδιότητες που προσφέρονται από τα Grid Services.

Επιπλέον, τα Grid Services υποστηρίζουν και τη δυνατότητα κάθε υπηρεσίας να είναι transient (πρόσκαιρη). Δηλαδή, κάθε instance της υπηρεσίας να είναι αυτόνομο για κάθε πελάτη, και να έχει πρόσβαση μόνο αυτός στα αποτελέσματα, κατά τη διάρκεια αλλά και μετά το τερματισμό του instance. Ως Grid service instance αναφερόμαστε στην συγκεκριμένη αρχικοποίηση ενός Grid Service.

Τα Grid Services το πετυχαίνουν αυτό επιτρέποντάς στους προγραμματιστές να χρησιμοποιούν την προσέγγιση του factory/instance για κάθε υπηρεσία. Αντί να υπάρχει μια μεγάλη (και stateless) υπηρεσία που να είναι κοινή για όλους τους χρήστες, μπορούμε να έχουμε ένα κεντρικό factory για την υπηρεσία, που να είναι υπεύθυνο να δημιουργεί instances της υπηρεσίας. Όταν ένας πελάτης θέλει να εκτελέσει μια κλήση της υπηρεσίας (π.χ. MathService) θα συνεργάζεται με το instance και όχι με το factory. Ένα παράδειγμα με πολλές περιπτώσεις συνεργασίας φαίνεται στο Σχ. 2.4.



Σχήμα 2.4: Χρησιμοποίηση του factory.

Τα Grid Services μπορούν να είναι transient αλλά μπορεί και όχι. Δεν είναι απαραίτητο κάθε Grid Service να είναι transient, αλλά εξαρτάται από την εφαρμογή και η κάθε εφαρμογή χρησιμοποιεί όποιον από τους δυο τρόπους ταιριάζει στις ανάγκες της.

Lifecycle management

Τα Grid Services μπορούν να δημιουργούνται και να καταστρέφονται δυναμικά. Οι υπηρεσίες μπορούν να καταστρέφονται ρητά (κατ΄ εντολή), ή πιθανώς να καταστράφηκαν, ή να είναι απροσπέλαστες επειδή έχουμε αποτυχία του συστήματος, π.χ. αποτυχία του λειτουργικού συστήματος ή του δικτύου. Όλα αυτά είναι υπολογισμένα και υλοποιημένα μέσα από το interface και της ιδιότητες που διέπουν το Lifecycle management των υπηρεσιών του.

Service Data

Το χαρακτηριστικό Service Data (Δεδομένα Υπηρεσίας) επιτρέπει την εισαγωγή ενός σετ από δομημένα δεδομένα σε κάθε υπηρεσία, τα οποία μπορούν να είναι προσβάσιμα κατευθείαν μέσω του interface της. Τα Web Services επιτρέπουν μονάχα operations να περιλαμβάνονται στα WSDL interfaces. Αντίθετα, με το Service Data μπορεί πολύ εύκολα να συμπεριληφθεί οποιαδήποτε δομημένη μορφή δεδομένων, όπως classes, arrays, και πολλά άλλα.

Γενικώς, τα Δεδομένα Υπηρεσίας που μπορεί να προσφέρει κάθε υπηρεσία μπορούν να διαχωριστούν σε δυο κατηγορίες:

- Πληροφορία κατάστασης: Περιέχει πληροφορία για την παρούσα κατάσταση της υπηρεσίας, όπως αποτελέσματα «πράξεων», ενδιάμεσα αποτελέσματα, πληροφορία runtime, κ.τ.λ..
- Service metadata: Πληροφορία για την ίδια την υπηρεσία, όπως δεδομένα του συστήματος, υποστηριζόμενα interfaces, κόστος χρησιμοποίησης της υπηρεσίας, κ.τ.λ. .

Notifications

Ένα Grid Service μπορεί να διαμορφωθεί ώστε να λειτουργεί ως notification source, και ορισμένοι πελάτες μπορούν να λειτουργούν ως notification sinks (subscribers). Αυτό σημαίνει ότι αν κάτι αλλάξει στο Grid Service, αυτή η αλλαγή θα γνωστοποιηθεί στους κατάλληλους πελάτες (δεν γνωστοποιούνται όλες οι αλλαγές, μόνο οι ζητούμενες).

Service Groups

Κάθε υπηρεσία μπορεί να διαμορφωθεί ώστε να λειτουργεί ως service group, όπου θα συναθροίζει (aggregate) άλλες υπηρεσίες. Πολύ εύκολα μπορούν να πραγματοποιηθούν διαδικασίες όπως «πρόσθεσε καινούργια υπηρεσία στο group», «απέβαλε αυτή την υπηρεσία από το group», αλλά και πιο σημαντικές όπως «βρες μια υπηρεσία σε αυτό το group που να πληρεί αυτή την συνθήκη». Αν και η λειτουργία του Service Group που προκύπτει από το OGSI είναι αρκετά απλή, το GT3 επιτρέπει αρκετές επεκτάσεις και υπηρεσίες πάνω σε αυτό το θέμα.

portType extension

Κάθε Web Service εκθέτει το interface του μέσω ενός WSDL document. Αυτό το interface συχνά ονομάζεται portType και κάθε κανονικό Web Service έχει μόνο ένα portType. Από την άλλη, τα Grid Services υποστηρίζουν μια επέκταση (extension) του portType και έτσι μπορούν να ορίσουν ένα πιο ευέλικτο portType. Ένα παράδειγμα φαίνεται στο Σχ. 2.5.

Για παράδειγμα, οι προδιαγραφές του OGSI υποχρεώνουν όλα τα Grid Services να επεκτείνουν (extend) ένα προτυποποιημένο portType. Χάρις όμως την επέκταση του portType, μπορεί ο καθένας να ορίσει το δικό του portType ως επέκταση του Grid Service και απλά να κληρονομεί (εξτενδ) τις κλήσεις του πρωτότυπου portType. Με τα Web Services θα έπρεπε

Σχήμα 2.5: Επέκταση του πρωτότυπου portType.

να συμπεριλάβουμε στον ορισμό μας όλες τις κλήσεις (συμπεριλαμβανομένου και του Grid Service) σε ένα μόνο portType.

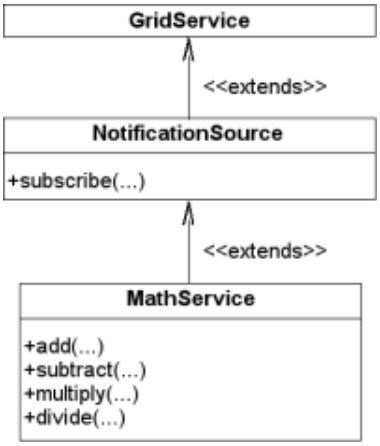
Εκτός από το πρωτότυπο portType, το OGSI ορίζει και πολλά αλλά πρότυπα portTypes από τα οποία μπορούμε να επεκτείνουμε ένα portType και προσθέσουμε τη λειτουργικότητα που επιθυμούμε στην υπηρεσία μας. Ένα παράδειγμα φαίνεται στο Σχ. 2.6.

Γενικά, τα Grid Services μπορούν να έχουν τρία είδη portTypes, όπως φαίνεται και από το Σχ. 2.7.

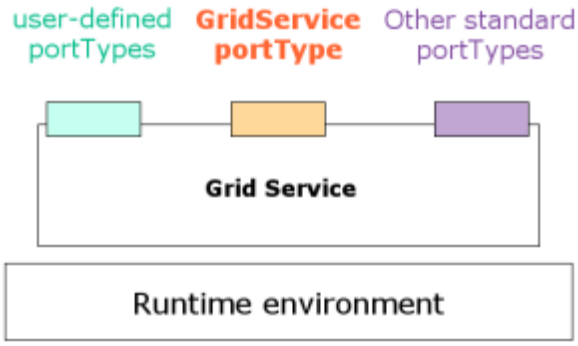
GSH & GSR

Αφού τα Grid Services είναι και αυτά Web Services, διευθυνσιοδοτούνται και αυτά με URIs. Ωστόσο, το OGSI παρουσιάζει έναν πολύ πιο ισχυρό σχήμα διευθυνσιοδότησης.

Ως "Grid Service URI" ορίζεται το Grid Service Handle (GSH). Κάθε GSH είναι μοναδικό. Δεν μπορούν να υπάρχουν δυο Grid Services με το ίδιο GSH. Αυτό που παρέχει το GSH είναι το πού βρίσκεται η υπηρεσία, αλλά δεν αναφέρει τίποτα για πως θα επικοινωνήσει κάποιος με το Grid Service (τι μεθόδους έχει, τι είδους μηνύματα δέχεται/στέλνει, κ.α.). Για να μάθει κάποιος αυτή την πληροφορία χρειάζεται το Grid Service Reference (GSR). Το GSR μπορεί να έχει διάφορες μορφές, αλλά επειδή η επικοινωνία γίνεται συνήθως με το SOAP, αυτό είναι ένα αρχείο WSDL. Κάθε πελάτης γνωρίζοντας μόνο το GSH μπορεί, εν τέλει, να επικοινωνήσει με το Grid Service, αφού του παρέχονται οι ανάλογες υπηρεσίες που παρέχοντάς τες το GSH,



Σχήμα 2.6: Ένα παράδειγμα χρήσης της επέκτασης του portType.



Σχήμα 2.7: Τα τρία είδη portType στα Grid Services.

βρίσκουν το GSR και επιτρέπουν την επικοινωνία και όλα τα επακόλουθα.

2.2 Peer to Peer networks

2.2.1 Τα δίκτυα P2P

Η τεχνολογία P2P γνώρισε απότομη ανάπτυξη, κυρίως μέσω εφαρμογών που έδιναν στο χρήστη τη δυνατότητα να επιτρέπει σε άλλους χρήστες να έχουν read-access σε κάποια αρχεία τους, αλλά να έχει και ο ίδιος στα αρχεία των άλλων. Η πρώτη και η πιο διάσημη τέτοιου είδους εφαρμογή (file-sharing) ήταν το Napster. Από τότε μέχρι σήμερα πολλές και διαφόρων ειδών είναι οι προσπάθειες που έχουν γίνει, μπορούν όμως να διαχωριστούν σε τρεις κατηγορίες (γενιές).

Στα P2P πρώτης γενιάς, οι κόμβοι είναι όλοι «ίσοι», αλλά η επικοινωνία τους εξαρτάται από έναν κεντρικό εξυπηρετητή, τέτοιο ήταν και το Napster. Δεν λαμβάνουν υπόψη τους καθόλου την τοπολογία του δικτύου ενώ η μονιμότητα (persistence) και η διαθεσιμότητα ούτε εγγυάται ούτε θεωρείται απαραίτητη. Δηλαδή δεν εγγυώνται ότι αν γίνει μια ερώτηση για ένα αντικείμενο και το αντικείμενο υπάρχει μέσα στο δίκτυο, τότε σίγουρα θα βρεθεί. Είναι φανερή η ευαισθησία του συστήματος σε περίπτωση αποτυχίας του κεντρικού εξυπηρετητή, όπως και το ότι είναι πολύ ευάλωτη σε θέμα ασφάλειας, ενώ η απόδοση του μειώνεται αρκετά όταν υπάρχουν πάρα πολλοί κόμβοι.

Τα P2P της δεύτερης γενιάς προσπαθούν να ελαχιστοποιήσουν την εξάρτηση από κάποιον κεντρικό εξυπηρετητή. Πρωτοπόρες εφαρμογές ήταν το Gnutella και το Freenet. Κάθε νέος κόμβος πρέπει να ξέρει κάποιον που είναι ήδη μέλος. Από αυτόν και από τους γνωστούς αυτού, εφαρμόζοντας έναν αλγόριθμο πλημμύρας (flooding algorithm), θα αντλήσει κάποια πληροφορία για την τοπολογία του δικτύου. Ο αλγόριθμος σταματάει όταν ξεπεραστεί το Time-To-Live (που υπάρχουν διάφοροι ορισμοί του). Αντίθετα με την τοπολογία, η μονιμότητα (persistence) και η διαθεσιμότητα, ούτε εγγυάται ούτε θεωρείται απολύτως απαραίτητη.

Η τρίτη γενιά P2P συστημάτων ξεκίνησε από εφαρμογές όπως τα Chord [14], Pastry [15], Tapestry, CAN [16] και παρείχαν μια δομή γνωστή ως Distributed Hash Table (DHT). Σε αυτά τα συστήματα, κάθε κόμβος έχει ένα μοναδικό κλειδί, κρυπτογραφημένο από μια hash function βασισμένη σε κάποιο μοναδικό χαρακτηριστικό όπως το IP. Μοναδικό κλειδί έχει και κάθε αντικείμενο που εισάγεται στο P2P. Σε κάθε hash table αποθηκεύονται ζευγάρια της μορφής: το κλειδί και το αντικείμενο για το οποίο είναι υπεύθυνο. Τα κλειδιά των κόμβων και τα ζευγάρια κλειδί-αντικείμενο περνούν και αυτά από μια hash function και ανατίθενται σε κάποιο κλειδί (κόμβο). Οι κόμβοι είναι συνδεδεμένοι μεταξύ τους με συγκεκριμένη συνδεσμολογία, ανάλογα με την εφαρμογή. Το Chord χρησιμοποιεί ένα κυκλικό δαχτυλίδι, το CAN έναν N-

διαστάσεων καρτεσιανό χώρο και το Tapestry καμία σχηματική δομή. Τα ζευγάρια κλειδί-τιμή ανατίθενται και σώζονται στους κόμβους που καθορίζει ο εκάστοτε αλγόριθμος, παράλληλα υπάρχει και η δυνατότητα να αποθηκεύονται και σε παραπάνω από έναν κόμβους (cloning). Χάρις στη δομημένη τοπολογία, η απόδοση των ερωτήσεων είναι πολύ καλύτερη από ότι στις δύο προηγούμενες γενιές. Το πιο σημαντικό χαρακτηριστικό αυτής της τεχνολογίας είναι ότι το DHT αυτό-οργανώνεται σε κάθε αλλαγή.

Η δουλειά μας είναι επικεντρωμένη στην τρίτη γενιά των P2P και ιδιαίτερα στις ιδιότητες του DHT. Παρόλο που η τρίτη γενιά έχει κάνει θετικά βήματα σε αρκετούς τομείς, υπάρχουν και κάποιοι τομείς που χρειάζονται περαιτέρω προσοχή και ανάπτυξη. Δύο από αυτούς τους τομείς είναι η ασφάλεια και η υποστήριξη μιας τυποποιημένης υποδομής για ανάπτυξη P2P συστημάτων. Σε αυτούς τους δύο τομείς, όλες οι προηγούμενες εφαρμογές που αναφέραμε υστερούν αισθητά. Το JXTA [18, 19] και το XtremWeb[17] είναι από τις σοβαρότερες προσπάθειες που αναπτύσσονται προκειμένου να υποστηρίξουν επαρκώς αυτούς τους δύο τομείς. Το JXTA θα αναλυθεί περισσότερο στην επόμενη ενότητα και είναι η βάση του TAP.

2.2.2 **JXTA**

Το Project JXTA[18] είναι ανοιχτού κώδικα και άρχισε από την Sun Microsystems το 2001. Σκοπός του η ανάπτυξη μιας γενικής πλατφόρμας, με τυποποιημένα πρωτόκολλα, σχεδιασμένη να υποστηρίζει την ανάπτυξη εφαρμογών P2P δικτύων. Αυτήν την περίοδο, υλοποιήσεις των JXTA πρωτοκόλλων είναι διαθέσιμες σε C και Java, ενώ εφαρμογές βασισμένες πάνω του σε C, Object C, Perl, Smaltalk και Java. Τα JXTA πρωτόκολλα παρουσιάζουν μια πλειάδα από αφηρημένους όρους όπως peers, peers groups, communication pipes and advertisments για να διευκολύνουν και να επιταχύνουν την ανάπτυξη τέτοιων εφαρμογών. Είναι έτσι ορισμένα, ώστε να μπορούν να περνούν από όλα τα υπάρχοντα φυσικά δίκτυα και τους μηχανισμούς μεταγωγής τους.

Το JXTA προσπαθεί να τυποποιήσει την ανταλλαγή μηνυμάτων μεταξύ των peer-to-peer συστημάτων, τυποποιώντας πρωτόκολλα αντί για εφαρμογές. Τα μηνύματα δρομολογούνται διαφανώς (transparently), διεισδύοντας μέσα από firewalls και NAT, ενώ χρησιμοποιούν διάφορα πρωτόκολλα μεταφοράς (HTTP, TCP/IP) για να φτάσουν στους κόμβους. Μοιάζει με σύστημα μηνυμάτων, αλλά δεν είναι αυτό ακριβώς, αν και έχει αρκετές αναλογίες. Τα πρωτόκολλα [19] εφαρμόζονται από κόμβους για να επικοινωνήσουν και να συνεργαστούν με άλλους κόμβους που τα εφαρμόζουν. Οι peers μπορούν να συνδέονται ή να εγκαταλείπουν το δίκτυο οποιαδήποτε στιγμή, ή να αλλάζουν τη φυσική διεύθυνση δικτύου (π.χ. IP) χωρίς πρόβλημα.

Για να επιτευχθεί αυτό, το JXTA έχει υλοποιήσει δικό του σχήμα διευθυνσιοδότησης (addressing scheme), δημιουργώντας ένα εικονικό δίκτυο με τους peers, πάνω από το φυσικό δίκτυο. Έτσι το εικονικό δίκτυο επιτρέπει στους peers να διατηρούν τις ταυτότητες (IDs) τους καθώς αλλάζουν φυσικές διευθύνσεις. Εκτός από τους peers, ID έχουν και όλοι οι υπόλοιποι πόροι του JXTA, όπως τα peer-group και τα pipes. Όλοι οι πόροι του δικτύου JXTA (peers, groups, services, pipes) γνωστοποιούνται μέσω advertisements, οι οποίες δημοσιεύονται από τους κόμβους στο δίκτυο. Τα advertisements είναι XML documents που περιέχουν στοιχεία (elements) με πληροφορία σχετική με τον πόρο που τα δημοσίευσε.

Οι peers στο εικονικό δίκτυο διακρίνονται σε τρεις κατηγορίες, ανάλογα με το ρόλο που έχουν στην εφαρμογή των πρωτοκόλλων και είναι οι εξής:

- *Edge peers*: Κόμβοι που δεν έχουν καμία ευθύνη για τη λειτουργία του JXTA, αλλά είναι οι πιο διαδεδομένοι και παίζουν σημαντικό ρόλο για τη λειτουργία των εφαρμογών.

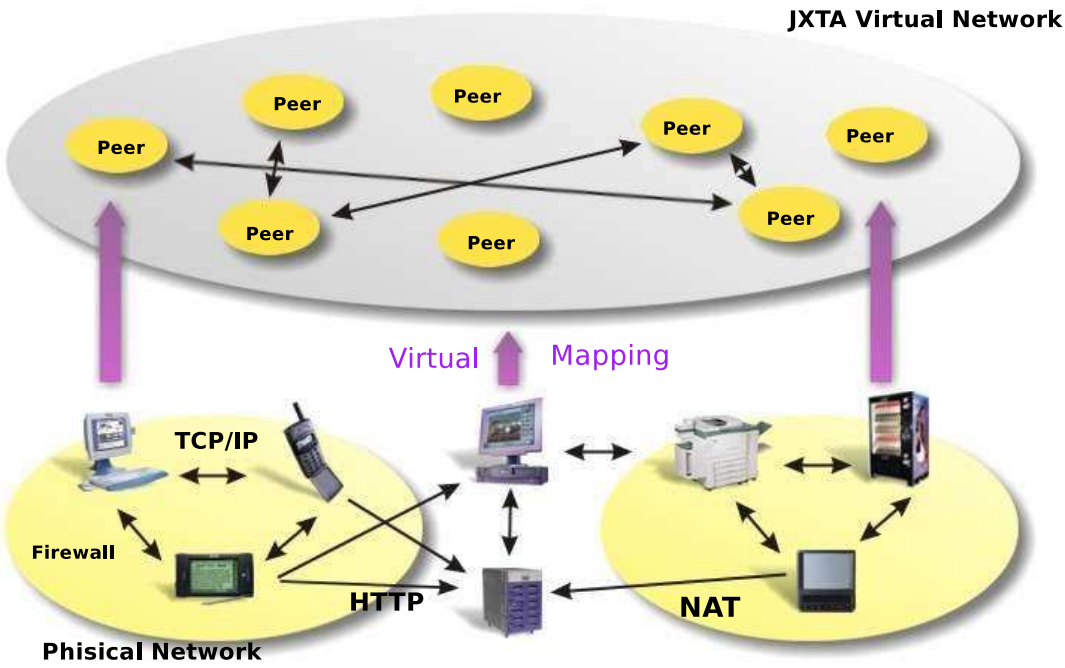
- *Rendezvous peers*: Κόμβοι με ιδιαίτερο ρόλο στην «επίλυση» των discovery queries, σχηματίζοντας ένα ειδικό δίκτυο από "super-peers".

- *Relay peers*: Κόμβοι με σημαντικό ρόλο, που λειτουργούν ως γέφυρες επικοινωνίας μεταξύ peers που είναι απομονωμένοι μεταξύ τους λόγω των τοπικών δικτύων τους.

Αυτοί οι κόμβοι οργανώνονται, οι ίδιοι, σε ιεραρχικά εικονικά τμήματα, που ονομάζονται peer groups. Τα peer-groups εφοδιάζουν το JXTA δίκτυο με εικονικά όρια, που περιορίζουν την επικοινωνία μεταξύ των κόμβων εντός των peer-groups. Ωστόσο, κάθε κόμβος μπορεί να συμμετέχει σε πολλές ομάδες (peer-groups). Εκμεταλλευόμενοι τα πλεονεκτήματα του εικονικού δικτύου, οι κόμβοι μπορούν να συμμετέχουν σε κάποια ομάδα, ανεξάρτητα από το φυσικό δίκτυο στο οποίο υπόκειται το κάθε μέλος. Ένα παράδειγμα από ένα JXTA network φαίνεται στο Σχ. 2.8

Το JXTA παρέχει τρεις βασικούς μηχανισμούς επικοινωνίας, ο καθένας σε διαφορετικό αφαιρετικό (abstract) επίπεδο.

- Το *Endpoint service* είναι στο χαμηλότερο επίπεδο επικοινωνίας και εμπεριέχει διάφορες τεχνικές δικτυακής επικοινωνίας. Απαλλάσσει τους κόμβους από τους περιορισμούς επικοινωνίας του φυσικού δικτύου, καθώς επικοινωνούν με "JXTA messages". Παράλληλα, αποθηκεύει πληροφορία δρομολόγησης και ελέγχει δυναμικά την τοπολογία του δικτύου.



Σχήμα 2.8: Παράδειγμα ενός JXTA network.

- Το *Pipe service*, που συμπληρώνει το Endpoint service ενσωματώνοντας την έννοια του εικονικού καναλιού επικοινωνίας (virtual communication channel). Το pipe έχει και αυτό ID και κόμβους προέλευσης και προορισμού.

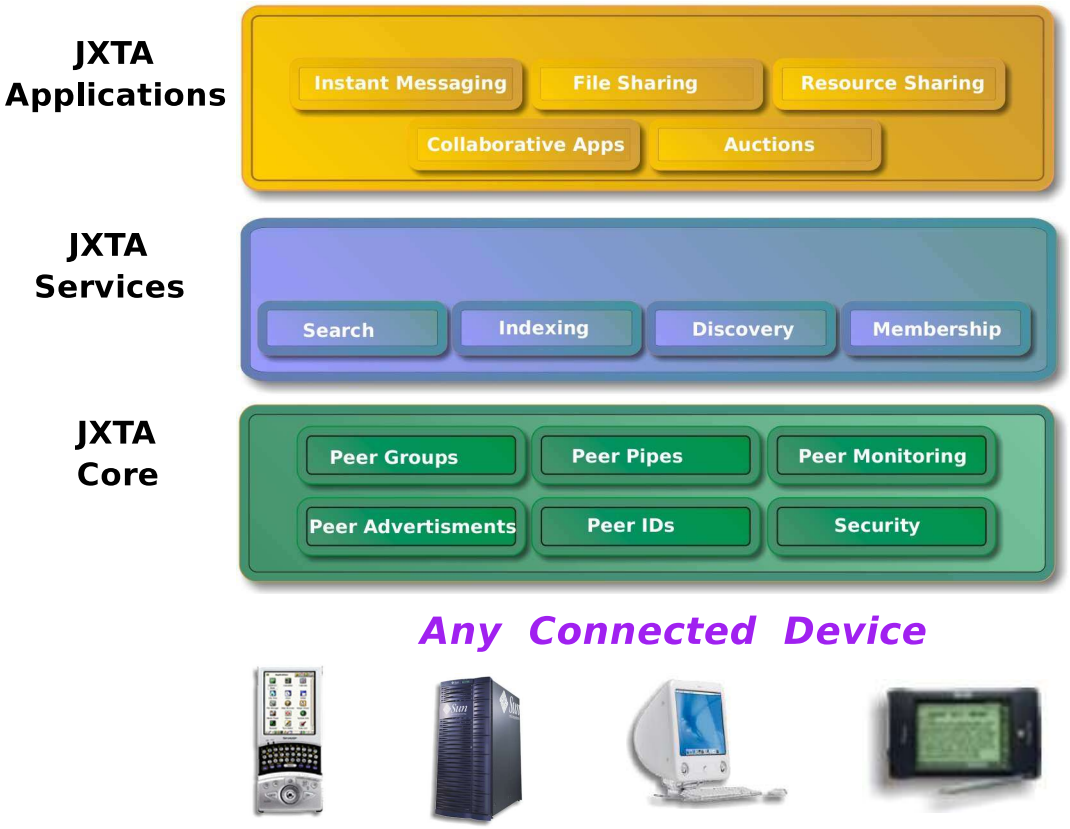
- Το *JXTA Socket* παρέχει άλλον έναν μηχανισμό επικοινωνίας που είναι χτισμένος πάνω από τα προηγούμενα δύο και το interface του μοιάζει πολύ με τα Unix network sockets.

Στο θέμα της ασφάλειας μεταξύ των peer groups, αλλά και μεταξύ των peers γενικότερα, το JXTA χρησιμοποιεί ένα entry-level trust model. Αυτό είναι βασισμένο στο Trasport Layer Security (TLS) του IETF. Το JXTA παρέχει ένα εικονικό επίπεδο μεταφοράς (virtual transport layer) βασισμένο στο TLS, με προεπιλεγμένη κρυπτογράφηση το RSA 1024 με 3DES και SHA-1.

Μια αναπαράσταση της αρχιτεκτονικής του JXTA φαίνεται στο Σχ. 2.9.

2.2.3 GISP

Το GISP (Global Information Sharing Protocol) [20] είναι χτισμένο πάνω στο JXTA και προσφέρει ένα πρωτόκολλο για δημιουργία Distributed Hash Table. Το GISP έχει αρκετά κοινά



Σχήμα 2.9: Η αρχιτεκτονική του JXTA.

χαρακτηριστικά με άλλα συστήματα που προσφέρουν κατανεμημένη δεικτοδότηση. Κάθε κόμβος έχει μια m-bit τιμή, κάθε κλειδί έχει μια m-bit τιμή, και το πιο κόμβος είναι υπεύθυνος για πιο κλειδί εξαρτάται από αυτές τις τιμές. Λεπτομέρειες για την υλοποίηση και τον αλγόριθμο που ακολουθεί το GISP μπορούν να βρεθούν στο [21].

Οι λειτουργίες που προσφέρει το GISP είναι δύο. Η πρώτη είναι εισαγωγή δεδομένων, όπου κάθε δεδομένο αναγνωρίζεται από ένα κλειδί. Το περιεχόμενο των δεδομένων μπορεί να είναι είτε κάποια συμβολοσειρά (string), είτε κάποιο XML document. Η δεύτερη λειτουργία είναι η ερώτηση (query) για δεδομένα. Το κάθε δεδομένο μπορεί να αναζητηθεί είτε μέσω του κλειδιού του, είτε μέσω του κλειδιού του και ενός XPath expression (όταν αναφέρεται σε XML document).

Κάθε κόμβος του GISP έχει έναν πίνακα με πληροφορία που αναφέρεται στους ενεργούς κόμβους. Κάθε εγγραφή έχει χρόνο εκπνοής, που όταν εκπνεύσει ο αναφερόμενος κόμβος σβήνεται από τον πίνακα. Οι κόμβοι στέλνουν μηνύματα περιοδικά σε όλους τους κόμβους που έχουν στον πίνακα τους, για να ελέγχουν αν είναι ενεργοί ή όχι.

Κάθε κόμβος έχει μια τοπική βάση δεδομένων για να αποθηκεύει τα δεδομένα. Όταν θέλει να εισάγει ένα δεδομένο το αποθηκεύει στη βάση του και το στέλνει και σε κόμβους που έχει στον πίνακα των ενεργών κόμβων και είναι μέσα στους χ (όπου χ παράγοντας μικρότερης απόστασης) «κοντινότερους». Όταν ένας κόμβος λάβει το δεδομένο εισαγωγής το αποθηκεύει στην τοπική του βάση δεδομένων. Τα δεδομένα δεν σβήνονται με κάποια εξωτερική κλήση, αλλά εκπνέουν (μετά το πέρασμα του χρόνου Time-To-Live) και μετά με ένα εσωτερικό μηχανισμό, που λειτουργεί περιοδικά για κάθε κόμβο, διαγράφονται από την τοπική βαση.

2.3 Σχετική έρευνα και εφαρμογές

Το L* [22] είναι μια τεχνική για το χτίσιμο μιας κατανεμημένης δομής δεδομένων, που να υποστηρίζει πολλούς χρήστες (multi-user), σε αναξιόπιστα peer-to-peer distributed hash tables.

Το L* χρησιμοποιεί πολλαπλά ημερολόγια (logs), ένα ημερολόγιο για τον κάθε συμμετέχοντα, για να αποθηκεύει τις αλλαγές στη δομή δεδομένων. Κάθε συμμετέχων βρίσκει τα δεδομένα που θέλει συμβουλευόμενος όλα τα ημερολόγια, αλλά όταν κάνει αλλαγές στη δομή τις προσθέτει μόνο στο δικό του ημερολόγιο. Το L* αποθηκεύει όλα τα ημερολόγια στο DHash [23], που είναι ένα κατανεμημένο peer-to-peer hash table. Κάθε εγγραφή στο ημερολόγιο (log-record) έχει ένα κλειδί, ενώ με κάθε αλλαγή (modification) του ιστορικού

(που αναφέρεται στο ίδιο στοιχείο της δομής) δημιουργείται καινούργιο ημερολόγιο. Κάθε ημερολόγιο έχει τέσσερα πεδία, το *prev*, που είναι το κλειδί του προηγούμενου ιστορικού, το *seq*, που είναι αυξοντας ακολουθιακος αριθμός για κάθε ημερολόγιο, το *version*, που είναι ένα version vector [24], και τέλος το *head*, που είναι το κλειδί του log-head. Το log-head είναι ένα ζευγάρι κλειδί-τιμή, όπου το κλειδί είναι μοναδικό, και πάντα το ίδιο για κάθε συμμετέχοντα, και η τιμή είναι το κλειδί του πιο πρόσφατου ημερολογίου του αντίστοιχου συμμετέχοντα.

Σε κάθε διαμοιραζόμενη δομή (ή κομμάτι της) που μοιράζεται από έναν αριθμό από συμμετέχοντες αντιστοιχεί και ένα view block. Κάθε view block περιέχει όλα τα κλειδιά των log-head των αντίστοιχων συμμετεχόντων. Μέσω του view επιτυγχάνεται συνδυασμός των επιμέρους ημερολογίων και το αποτέλεσμα για κάθε ερώτηση του χρήστη. Αυτό γίνεται μέσω μιας συνάρτησης, της order(), η οποία βάζει σε σειρά όλα τα σχετιζόμενα ημερολόγια, με βάση το version vector. Το interface που προσφέρει το L* είναι δυο συναρτήσεις, η traverse() και η append(). Η πρώτη χρησιμοποιείται για αναζήτηση (και χρησιμοποιεί την order), ενώ η δεύτερη για αλλαγές στη δομή (αφού όμως καλέσει την πρώτη).

Η διαφορά του TAP με το L* είναι ότι το L* χρησιμοποιεί ένα ημερολόγιο για κάθε πελάτη, έτσι αναγκάζεται να συνδυάζει όλα τα ημερολόγια για κάθε ερώτηση, ενώ για κάθε αλλαγή δημιουργεί καινούργιο ημερολόγιο. Το TAP, χρησιμοποιώντας ένα ενιαίο ημερολόγιο δεν χρειάζεται να κάνει αυτή τη διαδικασία. Επιπλέον, το L* χρησιμοποιεί version vectors ενώ το TAP τα Lamport clocks για να συγχρονίζουν τις αλλαγές στη δομή. Εκεί που φανερά διαφέρει το TAP είναι στην ασφάλεια, λόγω της χρησιμοποίησής των πρωτοκόλλων του Globus αλλά και του JXTA, που εξασφαλίζουν πολύ ευκολότερη ασφάλεια από αυτή που προσφέρει το DHash (μέσω του SFS [25]).

Το Sprite LFS [26] παρουσιάζει ένα file system δομημένο από τα ημερολόγια των εντολών, σύμφωνα με ένα στιγμιότυπο από i-number σε i-node. Το LFS χρησιμοποιεί ένα μόνο ημερολόγιο, διαχειριζόμενο από μόνο έναν server, με σκοπό να επιταχύνει την ταχύτητα των μικρών εγγραφών. Το TAP χρησιμοποιεί και αυτό μόνο ένα ημερολόγιο, αλλά δεν βασίζεται σε έναν μοναδικό server, οπότε ξεπερνάει όλα τα μειονεκτήματα που προκύπτουν από αυτό (ασφάλεια, αποτυχία, κ.α.).

Το σύστημα Bayou [27] παρουσιάζει τις αλλαγές σε μια βάση δεδομένων ως ένα ημερολόγιο ενημερώσεων. Κάθε ενημέρωση περιλαμβάνει μια συγκεκριμένη *merge procedure*, ανάλογα με την εφαρμογή, για να επιλύσει τις συγκρούσεις. Κάθε κόμβος διατηρεί ένα τοπικό ημερολό-

γιο για όλες τις ανανεώσεις που γνωρίζει, τις δικές του και άλλες. Οι κόμβοι διάγουν κυρίως σε αποσυνδεδεμένη κατάσταση, και συγχωνεύουν τα ημερολόγια τους όταν συνδέονται. Το ημερολόγιο και οι merge procedures επιτρέπουν σε έναν κόμβο να ξαναχτίσει την βάση του αφού προστεθούν ανανεώσεις που έγιναν από άλλους κόμβους. Καθώς οι ανανεώσεις φθάνουν σε «ειδικό» κόμβο, αυτός αποφασίζει την τελική-μόνιμη κατάσταση των εγγραφών στο ημερολόγιο. Το TAP διαφέρει από το Bayou στο ότι χρησιμοποιεί ένα κατανεμημένο αλγόριθμο για να βάλει σε σειρά τις ημερολογιακές εγγραφές, ενώ το Bayou χρησιμοποιεί αυτό τον «ειδικό» κόμβο. Το TAP διαβεβαιώνει ότι η δομή θα είναι συνεπής (consistent) μετά από μια ανανέωση, ενώ το Bayou βασίζεται στις merge procedures που προσφέρονται από την κάθε εφαρμογή.

Κεφάλαιο 3

Η υπηρεσία TAP

Σε αυτό το κεφάλαιο θα παρουσιάσουμε την υπηρεσία TAP. Η πρώτη ενότητα περιγράφει την αρχιτεκτονική του TAP και αναλύει τα τρία τμήματά του. Δίνονται λεπτομέρειες για τον πελάτη και τις δυνατότητές του, για το Grid Service, για το CDS και τελευταία για το τμήμα του P2P, το GISP. Στην δεύτερη ενότητα περιγράφεται αναλυτικά ο αλγόριθμος λειτουργίας του TAP. Στην τελευταία ενότητα δίνονται τα αποτελέσματα από τα πειράματα που έγιναν. Πριν προχωρήσουμε παρακάτω πρέπει να αναφερθεί ότι όλα τα μέρη του TAP είναι γραμμένα σε γλώσσα Java.

3.1 Η αρχιτεκτονική του TAP

3.1.1 Ο πελάτης του TAP

Έχοντας μια γενική εικόνα του TAP θα αναλύσουμε έναν πελάτη του, τις δυνατότητες που έχει αλλά και τις απαραίτητες «προϋποθέσεις» που θα πρέπει να «εκπληρώσει». Πριν προχωρήσουμε θα διαχωρίσουμε τους όρους «πελάτης» και «χρήστης». Ο πρώτος αντιστοιχεί στο λογισμικό που επικοινωνεί με το TAP και κάνει τη σύνδεση με το Globus, τηρώντας όλα τα πρωτόκολλα που είναι αναγκαία (security, κ.α.), ενώ ο δεύτερος αντιστοιχεί στον άνθρωπο ή το λογισμικό που δεν γνωρίζει τίποτα γι΄ αυτά και απλά χρησιμοποιεί το παραπάνω λογισμικό. Αρχικά θα πρέπει να βρει ένα Grid service instance, και να πάρει μια αναφορά-περιγραφή του, χρησιμοποιώντας το Grid Service Handle. Αφού γίνει αυτό, θα μπορεί να έχει πρόσβαση στο Grid και στην υπηρεσία μας, στο TAPService. Ο χρήστης υποβάλλεται σε μια σειρά από ερωτήσεις, για να διευκρινιστεί η απομακρυσμένη (remote) συνάρτηση, και οι τιμές των παραμέτρων που θα εκτελεσει. Οι κλήσεις που μπορεί να εκτελέσει δίνονται στον Πίνακα 3.1.1 και η ανάλυση κάθε μίας γίνεται παρακάτω. Όλες οι παράμετροι των κλήσεων καθώς

GET(path)	Ανάκτηση του URL, που αντιστοιχεί στο συγκεκριμένο 'path'.
PUTFILE(path, url)	Εισαγωγή, ότι το συγκεκριμένο 'path' βρίσκεται στο συγκεκριμένο 'url'.
DELFILE(path)	Διαγραφή του URL, που αντιστοιχεί στο συγκεκριμένο 'path'.
LS(path)	Ανάκτηση των περιεχομένων του καταλόγου που αντιστοιχεί στο συγκεκριμένο 'path'.
MKDIR(path, url)	Εισαγωγή, ότι το συγκεκριμένο 'path' (που είναι φάκελος) βρίσκεται στο συγκεκριμένο 'url'.
DELDIR(path)	Διαγραφή του URL (αναφέρεται σε φάκελο), που αντιστοιχεί στο συγκεκριμένο 'path'.
GetGroups(client)	Ανάκτηση όλων των ομάδων που είναι μέλος ο χρήστης 'client'.
AddUserToGroup(client, group)	Προσθέτουμε τον πελάτη 'client' στην ομάδα 'group'. Σημείωση: Αυτή η εντολή μπορεί να εκτελεστεί μονό από τον root.
DelUserFromGroup(client, group)	Διαγραφή του πελάτη 'client' από την ομάδα 'group'. Σημείωση: Αυτή η εντολή μπορεί να εκτελεστεί μονό από τον root.

Πίνακας 3.1: Οι κλήσεις που παρέχει το TAPService και μπορεί να εκτελέσει ο πελάτης.

και τύποι επιστροφής όλων των συναρτήσεων είναι τύπου String (java.lang.String).

Η κλήση GET αναφέρεται και σε αρχεία και σε καταλογους. Οι κλήσεις PUTFILE, DELFILE αναφέρονται σε αρχεία. Αντίθετα, οι κλήσεις LS, MKDIR, DELDIR αναφέρονται σε καταλόγους. Η παράμετρος 'path' πρέπει να έχει τη μορφή απόλυτης διεύθυνσης (διαδρομής) σε UNIX filesystem, για παράδειγμα : "/absolute/path/of/file/adrian.ps". Πρέπει να αρχίζει με ′/′ και κάθε directory πρέπει να τελειώνει με ′/′. Η GET είναι κλήση ανάκτησης και επιστρέφει το URL που αντιστοιχεί στο συγκεκριμένο 'path', αν υπάρχει τέτοια εγγραφή και δεν υπάρχει πρόβλημα εξουσιοδότησης (permission problem), αλλιώς μια τιμή λάθους. Η LS είναι και αυτή κλήση ανάκτησης και επιστρέφει όλες τις εγγραφές που υπάρχουν μέσα σε αυτόν τον κατάλογο (τα περιεχόμενα του καταλόγου). Οι κλήσεις PUTFILE και MKDIR είναι για ανανέωση-εισαγωγή πληροφορίας, δημιουργία αρχείου/φακέλου αντίστοιχα. Για παράδειγμα, ο πελάτης δηλώνει ότι το 'path' έχει τιμή το 'url', π.χ. ότι το path="/home/adrian/mirrors/" αντιστοιχεί στο url="ftp://ftp.softnet.tuc.gr/software/mirrors/". Σε επιτυχία ή αποτυχία επιστρέφεται το κατάλληλο μήνυμα. Οι DELFILE και DELDIR εκτελούν τη διαγραφή του αρχείου/καταλόγου αντίστοιχα.

Οι τρεις τελευταίες κλήσεις σχετίζονται με τις ομάδες (groups) των πελατών. Η GetGroups επιστρέφει όλες τις ομάδες που είναι μέλος ο πελάτης 'client' (π.χ. client="adrian"). Οι κλήσεις AddUserToGroup και DelUserFromGroup ανανεώνουν-εισάγουν πληροφορία στο σύστημα, προσθέτουν (διαγράφουν) τον πελάτη 'client' στην (από την) ομάδα 'group' (π.χ. group="softnet"). Αυτές οι δύο κλήσεις μπορούν να εκτελεστούν μόνο από τον root ή από πελάτη που ανήκει στην ομάδα με επωνυμία "root". Η πληροφορία για το ποιος είναι ο πελάτης root, που μπορεί να μην είναι μόνο ένας, παρέχεται από ένα config file που διαβάζεται από τον Grid server όταν ξεκινάει.

3.1.2 Λεπτομέρειες για το **Grid Service**

Την υπηρεσία που υλοποιήσαμε στο Globus την ονομάσαμε TAPService και με αυτή συνεργάζεται ο κάθε χρήστης-πελάτης του TAP. Το TAPService παρέχει στους πελάτες τις κλήσεις του Πίνακα 3.1.1. Αυτές περιγράφονται από το TAPService με τη γλώσσα WSDL και ο κώδικας που τις περιγράφει δίνεται στο Παράρτημα Α΄.

... Να βάλω τον κώδικα σε παράρτημα····

Για κάθε κλήση που καλείται να εκτελέσει το TAPService, γίνεται πρώτα μια επεξεργασία στα δεδομένα που έλαβε από τον πελάτη μαζί με κάποια τοπικά πεδία, και μετά γίνεται μια κλήση στο CDS. Το TAPService με το CDS επικοινωνούν μέσω μηνυμάτων τύπου String. Η επεξεργασία που γίνεται είναι η κατασκευή ενός String, που είναι το μήνυμα που θα στείλει το TAPService στο CDS και πρέπει να έχει συγκεκριμένη μορφή. Το μήνυμα αποτελείται από τα έξι πεδία που αναφέρονται στον Πίνακα 3.1.2. Η σειρά που απαιτείται να έχουν είναι από πάνω προς τα κάτω και στο τέλος κάθε πεδίου προσθέτεται το ειδικό πεδίο SEPARATOR. Τα πεδία που σχετίζονται άμεσα με τον εκάστοτε πελάτη και οι τιμές τους εξαρτώνται από αυτόν, είναι τα πρώτα τέσσερα. Τα τοπικά πεδία που παίρνουν μέρος στο μήνυμα είναι δύο και εξαρτώνται απολύτως από τον Grid server. Το πρώτο είναι το Argument5 που είναι σταθερό και μοναδικό για κάθε Grid server, και είναι η προσωπική του ταυτότητα μέσα σε όλο το Grid. Το δεύτερο είναι το Argument6 και αντιστοιχεί στην τιμή του Lamport clock τι στιγμή που εκτελείται η απομακρυσμένη κλήση στον Grid server. Κάθε Grid server έχει ένα Lamport clock, βάση των οποίων πετυχαίνεται ο συγχρονισμός όλων των εγγραφών του TAP.

Θυμίζουμε ότι η δομή δεδομένων του TAP προκύπτει από τις εισαγωγές εγγραφών από τα ημερολόγια (logs). Οπότε αυτό που γίνεται για κάθε πράξη (εισαγωγή-ανανέωση-διαγραφή) είναι η εισαγωγή μιας εγγραφής στο DHT. Ακόμα και η πράξη της διαγραφής μιας εγγραφής (π.χ. η διαγραφή ενός αρχείου από τη δομή) είναι μια ανανέωση της προηγούμενης εισαγωγής,

Argument1	Η απομακρυσμένη κλήση που επιθυμεί να εκτελέσει ο χρήστης.
Argument2	Το μοναδικό ID που έχει ο πελάτης. Κάθε πελάτης πρέπει να τηρεί τους κανόνες ασφάλειας του Grid, οπότε και κατέχει ένα μοναδικό ID.
Argument3	Η τιμή της πρώτης παραμέτρου της απομακρυσμένης κλήσης, που μπορεί να είναι είτε το 'path' είτε ο 'client', ανάλογα με την κλήση.
Argument4	Η τιμή της δεύτερης παραμέτρου της απομακρυσμένης κλήσης, που μπορεί να είναι είτε το 'url' είτε το 'group' είτε τίποτα.
Argument5	Το μοναδικό ID του Grid server, από τον οποίο ο πελάτης θα τρέξει το TAPService.
Argument6	Την τιμή του τοπικού ρολογιού που έχει ο Grid server.
SEPARATOR	"$$$"

Πίνακας 3.2: Τα πεδία του μηνύματος που στέλνει το TAPService στο CDS και απορρέουν από τις κλήσεις του interface του TAPService.

που έγινε κατά την δημιουργίαή ανανέωση του.

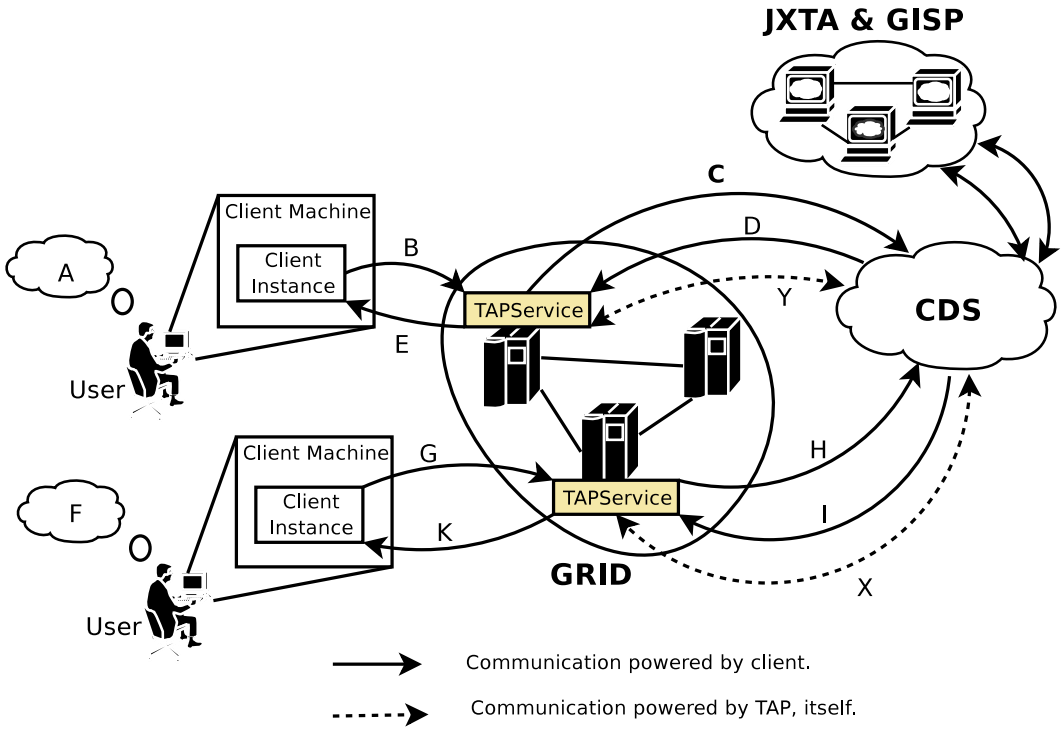
Όπως αναφέρθηκε προηγουμένως στην ενότητα 2.2.3 το GISP χρησιμοποιεί ένα Time-To-Live (TTL),τιμή που θέτουμε εμείς, για κάθε εγγραφή του, όπου μετά το πέρασμα αυτού του χρόνου η εγγραφή λήγει και σβήνεται αυτόματα. Αυτό στην περίπτωση μας έχει δύο συνέπειες. Η πρώτη συνέπεια είναι ότι σβήνονται αυτόματα οι εγγραφές που είναι μετά από κάποιο σημείο άχρηστες, οπότε δεν υπάρχει άχρηστη πληροφορία στο σύστημα μετά από ένα χρονικό διάστημα (περαιτέρω λεπτομέρειες στην ενότητα 3.2). Τέτοιες εγγραφές είναι αυτές που ανανεώθηκαν με εισαγωγή καινούργιας πληροφορίας για το ίδιο κλειδί ή με εισαγωγή μιας εγγραφής που δηλώνει τη διαγραφή αυτού του κλειδιού (εγγραφής).

Η δεύτερη συνέπεια είναι ότι οι εγγραφές που είναι έγκυρες σβήνονται πάλι αυτόματα μετά το πέρασμα του TTL. Για την αντιμετώπιση αυτής της συμπεριφοράς του GISP δημιουργήσαμε ένα μηχανισμό (ένα instance του σε κάθε Grid server) που ανανεώνει-επαναλαμβάνει (renew) μονάχα τις έγκυρες εγγραφές, ανά χρονικό διάστημα λίγο μικρότερο του TTL αλλά ικανό ώστε να έχει γίνει γνωστό οτι αυτή η εγγραφή είναι έγκυρη για άλλο ένα TTL. Για να το πετύχει αυτό, ο μηχανισμός κρατάει στοιχεία για όλες τις εισαγωγές και ανανεώσεις που γίνονται από ίδιο τον Grid server που υπάρχει κι αυτός. Για τις ανανεώσεις που δηλώνουν διαγραφή μιας εγγραφής, ο μηχανισμός δεν κρατάει στοιχεία και ούτε τις ανανεώνει-επαναλαμβάνει, αφού δεν είναι απαραίτητο αν εκμεταλλευτούμε έξυπνα τη λειτουργία του TTL. Εκτός από τα στοιχεία που πρέπει να κρατάει, πρέπει και να επικοινωνεί με το CDS ώστε να γίνονται οι απαραίτητες εισαγωγές επανάληψης (renew) στο GISP. Αυτού του είδους η επικοινωνία του TAPService με

το CDS έχει άλλου είδους χαρακτηριστικά από οτι αυτή που αναλύσαμε στον Πίνακα 3.1.2. Σε αυτή δεν παίζει κανέναν ρόλο ο πελάτης, και ούτε καν χρειάζεται να συνεργάζεται το TAPService με κάποιον πελάτη εκείνη τη στιγμή για να πραγματοποιηθεί. Λεπτομέρειες για την λειτουργία του μηχανισμού θα δοθούν στην ενότητα 3.2.

Συνοψίζοντας, αναφέρουμε οτι το TAPService επικοινωνεί με τους πελάτες του σύμφωνα με το πρωτόκολλο που περιγράφεται με τη WSDL, ενώ επικοινωνεί με το CDS με δύο τρόπους διαφορετικής φύσεως. Ο πρώτος είναι κατ΄ εντολή του πελάτη και ο δεύτερος με δική του πρωτοβουλία, για να διατηρήσει συνεπή (consistent) τη δομή δεδομένων. Ένα σενάριο λειτουργίας του TAP απεικονίζεται στο Σχ. 3.1 και τα μηνύματα που ανταλλάσονται φαίνονται παρακάτω:

- A: I want to store that "/mirrors/" is at "ftp://softnet.tuc.gr/mirrors".
- B: MKDIR("/mirrors/", "ftp://softnet.tuc.gr/mirrors")
- C: SendToCDS("MKDIR$$$clientID$$$/mirrors/ $$$ftp://softnet.tuc.gr/mirrors$$$GridID$$$429$$$")
- D: return "OK";
- E: return "Your entry is stored";
- F: I would like to know the groups of client "adrian".
- G: GetGroups("adrian")
- H: SendToCDS("GetGroups$$$clientID$$$adrian$$$GridID$$$429$$$")
- I: return "softnet$$$distributed$$$"
- K: return "Groups of adrian are: softnet, distributed."
- X: renew that "/mirrors/" is at "ftp://softnet.tuc.gr/mirrors".
- Y: renew that "/home/nikos/" is at "http://softnet.tuc.gr/users/nikos/".



Σχήμα 3.1: Παράδειγμα επικοινωνίας μεταξύ χρήστη-πελάτη-TAPService-CDS-GISP.

3.1.3 Λεπτομέρειες για το **CDS**

Ο ρόλος του CDS είναι να διαμορφώνει τη δομή δεδομένων που χτίζουν οι χρήστες του TAP και βρίσκεται μέσα στο GISP. Το CDS δέχεται κάποιες κλήσεις (αιτήσεις) από το TAPService, τις επεξεργάζεται κατάλληλα, και αφού τις μετατρέψει στην κατάλληλη μορφή, κάνει και αυτό με τη σειρά του κάποιες κλήσεις στο GISP ώστε να ικανοποιήσει τις αιτήσεις του TAPService.

Οι κλήσεις που μπορεί να εκτελέσει το CDS προς το GISP είναι μόνο δύο. Η πρώτη είναι αποθήκευσης ενός ζευγαριού, δηλαδή ενός κλειδιού και της τιμής του, και η δεύτερη ανάκτησης της τιμής ενός συγκεκριμένου κλειδιού.

Επειδή η επικοινωνία του TAPService με το CDS έχει δύο μορφές, η επεξεργασία που επιδέχεται κάθε μορφή είναι διαφορετική. Στην επικοινωνία που εκκινείται από τον πελάτη ακολουθεί διαφορετικό πρωτόκολλο (Πρωτόκολλο-1) από οτι στην επικοινωνία που εκκινείται από το ίδιο το TAPService (Πρωτόκολλο-2). Το Πρωτόκολλο-1 δρομολογεί τις απαραίτητες διαδικασίες και ελέγχους για αλληλεπίδραση με πελάτη. Μέσω του Πρωτόκολλο-1 οι πελάτες διαμορφώνουν τη δομή δεδομένων που εμπεριέχεται στο GISP. Μετά διακλαδώνεται, σε περισσότερα υποπρωτόκολλα του αρχικού, σύμφωνα με την απαίτηση του πελάτη, δηλαδή το πια κλήση του Πίνακα 3.1.1 έχει καλέσει. Εδώ γίνονται έλεγχοι της μορφής αν αυτός ο πελάτης είναι αρμόδιος να δημιουργήσει ένα αρχείο μέσα σε κάποιο κατάλογο, ελέγχοντας αν υπάρχει αυτός ο κατάλογος και αν έχει την κατάλληλη εξουσιοδότηση (έλεγχος στα permission "owner:group:all") να γράψει μέσα σε αυτόν τον κατάλογο.

Το Πρωτόκολλο-2 είναι πολύ πιο απλό, οι έλεγχοι που γίνονται δεν αφορούν κανέναν πελάτη, παρά μόνο τις ίδιες τις ίδιες τις εγγραφές και κατά πόσο είναι έγκυρες ή όχι. Οι ενέργειες που γίνονται δεν προκαλούν καμία αλλαγή στη δομή δεδομένων παρά μόνο τη συντηρούν συνεπή (consistent) και βελτιώνουν την απόδοση του GISP, σβήνοντας τις μη έγκυρες (πλέον) εγγραφές. Οι ενέργειες είναι της μορφής: έλεγξε αν κάποιος έχει διαγράψει ή πανωγράψει αυτή την εγγραφή, αν έχει γίνει τουλάχιστον ένα από τα δύο μην επικοινωνήσεις καθόλου με το GISP και «πες» στο TAPService να μην κρατάει πλέον στοιχεία για αυτή την εγγραφή. Αν δεν έχει γίνει τίποτα από τα δύο επικοινώνησε με το GISP, ώστε να γίνει επανάληψη (renew) της εγγραφής, και «πες» στο TAPService ότι θα χρειαστεί να διατηρήσει τα στοιχεία αυτής της εγγραφής.

Περισσότερες λεπτομέρειες για τον γενικό αλγόριθμο, και των πρωτοκόλλων που αποτελείται, θα δοθούν στην ενότητα 3.2.

insert(String key, String value)	Εισεγαγε ότι το κλειδί key έχει την τιμή value.
query(String key)	Επέστρεψε την τιμή που αντιστοιχεί στο κλειδί key.

Πίνακας 3.3: Οι κλήσεις του GISP.

3.1.4 Λεπτομέρειες για το τμήμα του P2P και το **GISP**

Το τρίτο τμήμα είναι ένα DHT (συγκεκριμένα το GISP) πάνω σε ένα P2P δίκτυο, υλοποιημένο με το JXTA.

Κάθε εισαγωγή δεδομένων στο GISP έχει ένα χρόνο εκπνοής (TTL), που μετά το πέρασμά του σβήνεται αυτόματα. Όταν γίνεται μια ερώτηση (query) για την τιμή ενός κλειδιού, κάθε κόμβος που έχει στην τοπική του βάση την τιμή για αυτό το κλειδί απαντάει επιστρέφοντας αυτο. Ο κόμβος που έκανε την ερώτηση μπορεί να πάρει πολλές απαντήσεις (τιμές), που μπορεί να είναι ίδιες μπορεί και όχι, ανάλογα με τις εισαγωγές που έχουν γίνει για το συγκεκριμένο κλειδί. Ο τρόπος που παίρνει, ή που του στέλνουν, τις απαντήσεις είναι ασυγχρονος. Ο κόμβος εκτελεί μια κλήση ερώτησης και επιστρεφει. Οι απαντήσεις έρχονται ως String και ο κόμβος πρέπει με κάποιο τρόπο να τις συλλέξει ασυγχρονα. Έτσι, φτιάξαμε έναν μηχανισμό που κάνει αυτή τη δουλειά, περιμένει για ένα χρονικό διάστημα t απαντήσεις για το συγκεκριμένο κλειδί a που θα του υποδειχθεί. Σε κάθε ερώτηση (query) που κάνει το CDS στο GISP, πρέπει να περιμένει για χρονικό διάστημα ίσο με t ώστε να φτάσουν σε αυτό όλες οι πιθανές απαντήσεις που υπάρχουν στο GISP. Για να αποφύγουμε αυτή την χρονική καθυστέρηση t για κάθε μία από τις ερωτήσεις, τις εκτελούμε όλες παράλληλα, οπότε ο χρόνος που απαιτείται για να πάρουμε απάντηση για (πιθανως) πέντε ερωτήσεις, είναι t συν το χρόνο (έστω t2) που απαιτείται ώστε να βρεθεί η πιο πρόσφατη εγγραφή για την κάθε ερωτηση. Το t2 είναι πολύ μικρο, διότι το parsing είναι πάνω σε String που γίνεται με «κανονικές εκφράσεις» (regular expressions) και είναι αρκετά γρηγορο.

Για να εκτελέσει οποιαδήποτε κλήση το TAPService, πρέπει να εκτελέσει τουλάχιστον δυο κλήσεις το GISP. Το GISP μπορεί να εκτελέσει μόνο δύο ειδών κλήσεις, ερώτησης και αποθήκευσης. Οι κλήσεις δίνονται στον Πίνακα 3.3.

Πριν προχωρήσουμε στην ανάλυση της "insert" θα πρέπει να ξεκαθαρίσουμε το τι είδους εισαγωγές μπορεί να γίνουν στο GISP. Οι προφανείς είναι αυτές που προκύπτουν από τον Πίνακα 3.1.1. Για παράδειγμα, αν θέλουμε να εισάγουμε (PUT) ότι το αρχείο "/etc/profile", άρα κλειδί="/etc/profile", βρίσκεται σε κάποιο URL (π.χ. "ftp://ftp.softnet.tuc.gr/conf/profile"), τότε η τιμή αυτου του κλειδιού θα είναι ίση με "URL$ClientID$group$permission$GridID$LamClock$". Αυτό, όμως, που πραγματικά γίνεται είναι ότι για κάθε μία από αυτές γίνεται μία επιπλέον εισα-

Κλήση TAPService	key	value
PUTFILE	"/dirs/to/file/A"	"url$owner$group$perm$GIDLC"
Επιπλεον εισαγωγη	"DIR$/dirs/to/file/"	"PrevEntries$A:file$GIDLC"
DELFILE	"/dirs/to/file/A"	"DELETED$owner$group$perm$GIDLC"
Επιπλέον εισαγωγή	"DIR$/dirs/to/file/"	"PrevEntriesWithoutAGIDLC$"
MKDIR	"/dirs/to/dir/Y/"	"url$owner$group$perm$GIDLC"
Επιπλέον εισαγωγή	"DIR$/dirs/to/dir/"	"PrevEntries$Y:dir$GIDLC"
DELDIR	"/dirs/to/dir/y/"	"DELETED$owner$group$perm$GIDLC"
Επιπλέον εισαγωγή	"DIR$/dirs/to/dir/"	"PrevEntriesWithoutYGIDLC$"
AddUserToGroup	"group$client"	"newGroup$PrevGroups$GIDLC"
DelUserFromGroup	"group$client"	"PrevGroupsWithoutTheDeletedGIDLC$"

Πίνακας 3.4: Οι παράμετροι της κλήσης "insert" του GISP, ανάλογα με την κάθε περίπτωση.

Είδος της ερώτησης	Η αντίστοιχη μορφή του "key"
Ερώτηση για FILE (GET)	"/dirs/to/file/x"
Ερώτηση για DIRECTORY (LS)	"DIR$/dirs/to/dir/y/"
Ερώτηση για τα GROUP (GetGroups) κάποιου "client"	"GROUP$client"

Πίνακας 3.5: Η μορφή της παραμέτρου "key" της κλήσης "query" του GISP.

γωγή. Στην παραπάνω περίπτωση, θα γίνει η εισαγωγή που αναφέραμε, αλλά επιπλεόν θα γίνει και άλλη μία εισαγωγή, με κλειδί το "DIR$/etc/" και με τιμή το "X$profile#file$GridID$LamClock$". Όπου το X είναι τα περιεχόμενα του καταλόγου "/etc/" πριν αυτή την εισαγωγή. Επειδή σε κάθε δημιουργία (όπου δεν υπάρχει ήδη) ή διαγραφή ενός αρχείου/καταλόγου αλλάζουν και τα περιεχόμενα του καταλόγου όπου αυτό περιέχεται αλλάζουν, πρέπει να ενημερωθεί και αυτός (ο κατάλογος που το περιέχει) ότι έγινε μια αλλαγή (δημιουργία/διαγραφή). Αυτό γίνεται γιατί για κάθε κατάλογο υπάρχει άλλη μια εγγραφή στο GISP που περιέχει τα περιεχόμενα του, και το τι είναι το καθένα (αρχείο/κατάλογος). Χρησιμοποιήσαμε αυτή την τεχνική για να υλοποιήσουμε την κλήση LS με όσο το δυνατόν λιγότερες κλήσεις στο GISP, αφού μαζί με την GET είναι οι δυο πιο συχνές κλήσεις σε συστήματα που προσφέρουν Directory Services.

Η συνάρτηση "insert" εκτελείται από τις συναρτήσεις του Πίνακα 3.1.1 που κάνουν κάποια εισαγωγή, και τα "key" και "value" έχουν συγκεκριμένη μορφή σε κάθε περίπτωση. Αυτές οι κλήσεις και η αντίστοιχη μορφή των παραμέτρων της "insert" δίνονται στον Πίνακα 3.4.

Αντίθετα, η κλήση "query" εκτελείται από κάθε συνάρτηση του Πίνακα 3.1.1, μία ή και περισσότερες φορές για καθεμιά, όπως θα δούμε στην επόμενη ενότητα. Η μορφή του της παραμέτρου "key", της "query", είναι ανάλογη του τι «ρωτάμε» (αρχείο ή κατάλογο ή ομάδα),

και φαίνεται στον Πίνακα 3.5. Η τιμή επιστροφής της αντιστοιχεί στην τιμή του κλειδιού ("key") που έχει εισαχθεί (μπορεί και όχι) προηγουμένως από την "insert".

3.2 Ο αλγόριθμος

Αν και τα περισσότερα από τα βασικά κομμάτια του αλγορίθμου έχουν αναλυθεί από την προηγούμενη ενότητα, δεν είναι εκτενής και ολοκληρωμένη η ανάλυση του. Σε αυτή την ενότητα, θα αναλύσουμε κάθε διαδικασία ξεχωριστά, ξεκινώντας από το πρώτο μέχρι το τελευταίο βήμα που περνάει μέχρι να ολοκληρωθεί. Οι διαδικασίες, δηλαδή ο τρόπος που λειτουργεί το TAP, είναι δύο ειδών και ανήκουν είτε στο Πρωτόκολλο-1, είτε στο Πρωτόκολλο-2, όπως έχουν ήδη αναφερθεί στην ενότητα 3.1.3. Στο Πρωτόκολλο-1 ανήκουν οι διαδικασίες που εκτελούνται όταν το TAP αλληλεπιδρά με κάποιον χρήστη. Ενώ στο Πρωτόκολλο-2 ανήκουν οι διαδικασίες που εκτελούνται από το TAP αυτόνομα, αποσκοπούν στην σωστή συνέχιση της λειτουργίας του συστήματος και ο χρήστης δεν έχει καμία επίδραση πάνω τους.

Πριν μπούμε στις λεπτομέρειες του κάθε είδους διαδικασιών, θα αναφέρουμε κάποια λειτουργικά στοιχεία του TAP που χρησιμοποιούνται και από τα δύο είδη. Το TAP προσφέρει Directory Services και είναι μια τεχνική για να χτίζεις multi-user distributed data structures. Λειτουργεί ανάλογα με το UNIX filesystem, δηλαδή κάθε αρχείο/κατάλογος χαρακτηρίζεται από ιδιοκτήτη, ομάδα, κοινό και αδειών διαβασμα-γραψιμο για το κάθε ένα (owner:group:all , rw:rw:rw). Κάθε Grid server τηρεί τα πρωτόκολλα που απαιτεί το Globus, και έχει μια ταυτότητα (ID) που είναι μοναδική μέσα στο δίκτυο Grid, την οποία παρέχει το Globus. Κάθε πελάτης του Grid server τηρεί και αυτός όλα τα απαραίτητα πρωτόκολλα, και υπόκειται στις διαδικασίες (λόγω Grid) που ευθύνονται για την ασφάλεια (authorization and authedication), οπότε και αυτός έχει μια μοναδική ταυτότητα (ID). Κάθε Grid server έχει ένα πεδίο που αντιστοιχεί στο Lamport clock του. Σε κάθε κλήση του πελάτη για ερώτηση ή εισαγωγή, ο server ελέγχει αν κάποια άλλη εγγραφή (πάνω στο ίδιο κλειδί) έχει μεγαλύτερη τιμή στο Lamport clock, αν όχι τότε η τιμή του πεδίου αυξάνεται κατά μία μονάδα, αν ναι τότε παίρνει την μεγαλύτερη τιμή συν μία μονάδα.

3.2.1 Το Πρωτόκολλο-1

Οι διαδικασίες που ανήκουν στο Πρωτόκολλο-1 είναι οι λειτουργίες που γίνονται για κάθε κλήση που επιθυμεί να εκτελέσει ο χρήστης. Ο χρήστης μπορεί μόνο να καλέσει τις κλήσεις του Πίνακα 3.1.1, άρα τόσες και ανάλογες της κάθε μίας είναι και οι διαδικασίες αυτού του

πρωτοκόλλου. Παρακάτω θα αναλύσουμε κάθε μία από τις εννιά κλήσεις.

PUTFILE

Ο χρήστης καλεί την PUTFILE όταν θέλει να κάνει εισαγωγή (δημιουργία ή ανανέωση) ενός URL που αναφέρεται σε κάποιο αρχείο. Πρέπει να δώσει το path που θέλει να δημιουργήσει (ανανεώσει) και το URL όπου θα υπάρχει. Ο χρήστης λέει στον πελάτη ότι θέλει να εκτελέσει την κλήση PUTFILE με τις αντίστοιχες παραμέτρους. Ο πελάτης το λέει στο TAPService, αυτό κατασκευάζει το ανάλογο String με τα ανάλογα ορίσματα του Πίνακα 3.1.2 και το στέλνει στο CDS. Το CDS κάνει parse το String και βλέπει ότι πρόκειται για εντολή εισαγωγής (δημιουργίας/ανανέωσης) ενός αρχείου. Πριν κάνει την εισαγωγή πρέπει να πιστοποιήσει ότι αυτός ο πελάτης μπορεί να δημιουργήσει (ή να ανανεώσει) αυτό το αρχείο στον συγκεκριμένο κατάλογο.

Παρακάτω ακολουθεί ο ψευδοκώδικας της συνάρτησης PUT :

PUTFILE(path, url)

Listing 3.1: Command "PUTFILE"

```
1  int count = countDirs(path);
2  String[] all_keys = String[count + 3];
3
4  all_keys[0] = "groups:" + user; //user_group
5  all_keys[1] = path;
6  all_keys[count + 2] = "dirs:" + delLastField(path);
7              //Previous directory contents
8
9  /* Going to do queries */
10
11 gisp.query.(all_keys[0]);
12 gisp.query.(all_keys[1]);
13 gisp.query.(all_keys[count + 2]);
14
15 String pathtemp = path;
16 for(int i = 2; i < (count + 2); i++) {
17 while (!pathtemp.equals("/") ) {
18   pathtemp = delLastField(pathtemp);
19   query(pathtemp);
20   all_keys[i] = pathtemp;
```

```
21 }
22
23 gisp.wait(WAIT_TIME_UPDATE);
24
25 /* Going to do cheks */
26
27 if (EntryExist(MostRecentRecordOf(all_keys[0]) ) {
28   if (userNotInGroup(MostRecentRecordOf(all_keys[0]) ) ) {
29     return "ERROR: User not exist in this group.";
30   }
31 }
32
33 String prv_dir_data = MostRecentRecordOf(all_keys[2]);
34                       //Previous directory of initial path
35 if (EntryExist(prv_dir_data) ) {
36   if (!HaveReadWritePermissions(prv_dir_data) ) {
37     return "ERROR: Operation not permitted";
38   }
39 } else {
40   return "ERROR: Previous directory not exist";
41 }
42
43 String result;
44 for (int i = (count + 1); i > 2; i--) {
45   result = MostRecentRecordOf(all_keys[i]);
46   if (EntryExist(result) ) {
47     if (!HaveReadWritePermissions(result) ) {
48       return "ERROR: Operation not permitted";
49     }
50   } else {
51     return "ERROR: Previous directory not exist";
52   }
53 }
54
55 String initial_path_data = MostRecentRecordOf(all_keys[1]);
56                          //Our initial path
57 if (!initial_path_data.equals("") ) {
58   if (!HaveReadWritePermissions(initial_path_data) ) {
59     return "ERROR: Operation not permitted";
60   }
61 }
62
63
```

```
64 /* Going to insert the new data */
65
66 String prv_dir_data_dir = MostRecentRecordOf(all_keys[count + 2]);
67 String prv_dir_contents = "";
68 if (EntryExist(prv_dir_data_dir) ) {
69   prv_dir_contents = GetOnlyContents(prv_dir_data_dir);
70   syncSetNewTime(prv_dir_data_dir);
71 } else {
72   syncSetNewTime(prv_dir_data);
73 }
74 gisp.insert(all_keys[count + 2], prv_dir_contents + \\
75       getLastField(path) + GridID + LTime);
76 gisp.insert(path, url + owner + group + permissions + \\
77       GridID + LTime);
```

MKDIR

Ο χρήστης καλεί την MKDIR όταν θέλει να κάνει εισαγωγή (δημιουργία ή ανανέωση) ενός URL που αναφέρεται σε κάποιο κατάλογο. Πρέπει να δώσει το path που θέλει να δημιουργήσει (ανανεώσει) και το URL όπου θα υπάρχει. Ο χρήστης λέει στον πελάτη ότι θέλει να εκτελέσει την κλήση MKDIR με τις αντίστοιχες παραμέτρους. Ο πελάτης το λέει στο TAPService, αυτό κατασκευάζει το ανάλογο String με τα ανάλογα ορίσματα του Πίνακα 3.1.2 και το στέλνει στο CDS. Το CDS κάνει parse το String και βλέπει ότι πρόκειται για εντολή εισαγωγής (δημιουργίας ή ανανέωσης) ενός καταλόγου. Πριν κάνει την εισαγωγή πρέπει να πιστοποιήσει ότι αυτός ο πελάτης μπορεί να δημιουργήσει έναν κατάλογο μέσα στον κατάλογο που επιθυμεί.

Παρακάτω ακολουθεί ο ψευδοκώδικας της συνάρτησης MKDIR :

MKDIR(path, url)

Listing 3.2: Command "MKDIR"

```
1 int count = countDirs(path);
2 String[] all_keys = String[count + 4];
3
4 all_keys[0] = "groups:" + user; //user_group
5 all_keys[1] = path;
6 all_keys[count + 2] = "dirs:" + delLastField(path);
7             //Previous directory contents
8 all_keys[count + 3] = "dirs:" + path;
9             //Initial directory contents
```

```
10
11 /* Going to do queries */
12
13 gisp.query.(all_keys[0]);
14 gisp.query.(all_keys[1]);
15 gisp.query.(all_keys[count + 2]);
16 gisp.query.(all_keys[count + 3]);
17
18 String pathtemp = path;
19 for(int i = 2; i < (count + 2); i++) {
20 while (!pathtemp.equals("/") ) {
21   pathtemp = delLastField(pathtemp);
22   query(pathtemp);
23   all_keys[i] = pathtemp;
24 }
25
26 gisp.wait(WAIT_TIME_UPDATE);
27
28 /* Going to do cheks */
29
30 if (EntryExist(MostRecentRecordOf(all_keys[0]) ) {
31   if (userNotInGroup(MostRecentRecordOf(all_keys[0]) ) ) {
32     return "ERROR: User not exist in this group.";
33   }
34 }
35
36 String initial_dir_data_dir = MostRecentRecordOf(all_keys[count + 3]);
37                     //Contents of initial directory
38 if (EntryExist(initial_dir_data_dir) ) {
39   return "ERROR: Operation not permitted, \\
40 ␣␣␣␣␣␣␣␣␣␣because directory is not empty.";
41 }
42
43 String prv_dir_data = MostRecentRecordOf(all_keys[2]);
44                     //Previous directory of initial path
45 if (EntryExist(prv_dir_data) ) {
46   if (!HaveReadWritePermissions(prv_dir_data) ) {
47     return "ERROR: Operation not permitted";
48   }
49 } else {
50   return "ERROR: Previous directory not exist";
51 }
52
```

```
53 String initial_path_data = MostRecentRecordOf(all_keys[1]);
54                         //Our initial path
55 if (EntryExist(initial_path_data) ) {
56   if (!HaveReadWritePermissions(initial_path_data) ) {
57     return "ERROR: Operation not permitted";
58   }
59 }
60
61 String result;
62 for (int i = (count + 1); i > 2; i--) {
63   result = MostRecentRecordOf(all_keys[i]);
64   if (EntryExist(result) ) {
65     if (!HaveReadWritePermissions(result) ) {
66       return "ERROR: Operation not permitted";
67     }
68   } else {
69     return "ERROR: Previous directory not exist";
70   }
71 }
72
73
74 /* Going to insert the new data */
75
76 String prv_dir_data_dir = MostRecentRecordOf(all_keys[count + 2]);
77 String prv_dir_contents = "";
78 if (EntryExist(prv_dir_data_dir) ) {
79   prv_dir_contents = GetOnlyContents(prv_dir_data_dir);
80   syncSetNewTime(prv_dir_data_dir);
81 } else {
82   syncSetNewTime(prv_dir_data);
83 }
84 gisp.insert(all_keys[count + 2], prv_dir_contents + \\
85       getLastField(path) + GridID + LTime);
86 gisp.insert(path, url + owner + group + permissions + \\
87       GridID + LTime);
```

GET

Ο χρήστης καλεί την GET οταν θέλει να ανακτήσει την τιμή (δηλαδή το URL) ενός κλειδιού (δηλαδή μιας διεύθυνσης π.χ. "/etc/profile") που αναφέρεται σε κάποιο αρχείο ή κατάλογο. Πρέπει να δώσει το path που θέλει να μάθει ώστε να αντίστοιχο URL. Ο χρήστης λέει στον

πελάτη ότι θέλει να εκτελέσει την κλήση GET με το αντίστοιχο path . Ο πελάτης το λέει στο TAPService, αυτό κατασκευάζει το ανάλογο String με τα ανάλογα ορίσματα του Πίνακα 3.1.2 και το στέλνει στο CDS. Το CDS κάνει parse το String και βλέπει ότι πρόκειται για εντολή ανάκτησης μιας εγγραφής. Πριν του απαντήσει με το ανάλογο URL, θα πρέπει να πιστοποιήσει ότι αυτός ο πελάτης έχει τα κατάλληλα permission για να το διαβάσει, αλλά και ότι η εγγραφή υπάρχει.

Παρακάτω ακολουθεί ο ψευδοκώδικας της συνάρτησης GET :

GET(path)

Listing 3.3: Command "GET"

```
1  int count = countDirs(path);
2  String[] all_keys = String[count + 3];
3
4  all_keys[0] = "groups:" + user; //user_group
5  all_keys[1] = path;
6  all_keys[count + 2] = "dirs:" + delLastField(path);
7              //Previous directory contents
8
9  /* Going to do queries */
10
11 gisp.query.(all_keys[0]);
12 gisp.query.(all_keys[1]);
13 gisp.query.(all_keys[count + 2]);
14
15 String pathtemp = path;
16 for(int i = 2; i < (count + 2); i++) {
17 while (!pathtemp.equals("/") ) {
18   pathtemp = delLastField(pathtemp);
19   query(pathtemp);
20   all_keys[i] = pathtemp;
21 }
22
23 gisp.wait(WAIT_TIME_UPDATE);
24
25 /* Going to do cheks */
26
27 if (EntryExist(MostRecentRecordOf(all_keys[0]) ) {
28   if (userNotInGroup(MostRecentRecordOf(all_keys[0]) ) ) {
29     return "ERROR: User not exist in this group.";
```

```
30    }
31  }
32
33  String initial_path_data = MostRecentRecordOf(all_keys[1]);
34                          //Our initial path
35  if (EntryExist(initial_path_data) ) {
36    if (!HaveReadPermissions(initial_path_data) ) {
37      return "ERROR: Operation not permitted";
38    }
39  } else {
40    return "ERROR: Record not exist.";
41  }
42
43  String result;
44  for (int i = (count + 1); i > 1; i--) {
45    result = MostRecentRecordOf(all_keys[i]);
46    if (EntryExist(result) ) {
47      if (!HaveReadPermissions(result) ) {
48        return "ERROR: Operation not permitted";
49      }
50    } else {
51      return "ERROR: Previous directory not exist";
52    }
53  }
54
55  String prv_dir_data_dir = MostRecentRecordOf(all_keys[count + 2]);
56  syncSetNewTime(prv_dir_data_dir);
57
58  return getUrlOf(initial_path_data);
```

LS

Ο χρήστης καλεί την LS όταν θέλει να ανακτήσει τα περιεχόμενα ενός καταλογου. Πρέπει να δώσει το path (που είναι κατάλογος) για το οποίο θέλει να μάθει, ώστε τους επιστραφούν οι αντίστοιχες εγγραφές (περιεχόμενα). Τα οποία όμως είναι τα κλειδιά και όχι οι τιμές τους, π.χ. τα περιεχόμενα του καταλόγου "/opt/" μπορεί να είναι τα "dir:java, dir:ant, file:list.cpp". Ο χρήστης λέει στον πελάτη ότι θέλει να εκτελέσει την κλήση LS με το αντίστοιχο path. Ο πελάτης το λέει στο TAPService, αυτό κατασκευάζει το ανάλογο String με τα ανάλογα ορίσματα του Πίνακα 3.1.2 και το στέλνει στο CDS. Το CDS κάνει parse το String και βλέπει ότι πρόκειται για εντολή ανάκτησης περιεχομένων ενός καταλόγου. Πριν του απαντήσει με την

ανάλογη λίστα από κλειδιά, θα πρέπει να πιστοποιήσει ότι αυτός ο πελάτης έχει τα κατάλληλα permission για να το διαβάσει αυτόν τον κατάλογο, αλλά και ότι ο κατάλογος υπάρχει.

Παρακάτω ακολουθεί ο ψευδοκώδικας της συνάρτησης LS :

LS(path)

Listing 3.4: Command "LS"

```
1  int count = countDirs(path);
2  String[] all_keys = String[count + 3];
3
4  all_keys[0] = "groups:" + user; //user_group
5  all_keys[1] = path;
6  all_keys[count + 2] = "dirs:" + path;
7              //Initial directory contents
8
9  /* Going to do queries */
10
11 gisp.query.(all_keys[0]);
12 gisp.query.(all_keys[1]);
13 gisp.query.(all_keys[count + 2]);
14
15 String pathtemp = path;
16 for(int i = 2; i < (count + 2); i++) {
17 while (!pathtemp.equals("/") ) {
18   pathtemp = delLastField(pathtemp);
19   query(pathtemp);
20   all_keys[i] = pathtemp;
21 }
22
23 gisp.wait(WAIT_TIME_UPDATE);
24
25 /* Going to do cheks */
26
27 if (EntryExist(MostRecentRecordOf(all_keys[0]) ) {
28   if (userNotInGroup(MostRecentRecordOf(all_keys[0]) ) ) {
29     return "ERROR: User not exist in this group.";
30   }
31 }
32
33 String initial_path_data = MostRecentRecordOf(all_keys[1]);
34                         //Our initial path
35 if (EntryExist(initial_path_data) ) {
```

```
36    if (!HaveReadPermissions(initial_path_data) ) {
37      return "ERROR: Operation not permitted";
38    }
39  } else {
40    return "ERROR: Record not exist.";
41  }
42
43  String result;
44  for (int i = (count + 1); i > 1; i--) {
45    result = MostRecentRecordOf(all_keys[i]);
46    if (EntryExist(result) ) {
47      if (!HaveReadPermissions(result) ) {
48        return "ERROR: Operation not permitted";
49      }
50    } else {
51      return "ERROR: Previous directory not exist";
52    }
53  }
54
55  String initial_dir_data_dir = MostRecentRecordOf(all_keys[count + 2]);
56  if (EntryExist(initial_dir_data_dir) ) {
57    syncSetNewTime(initial_dir_data_dir);
58    return getContentsOf(initial_path_data);
59  } else {
60    syncSetNewTime(initial_path_data);
61    return "This directory is empty";
62  }
```

DELFILE

Ο χρήστης καλεί την DELFILE όταν θέλει να κάνει διαγραφή μιας εγγραφής που αναφέρεται σε κάποιο αρχείο. Όπως έχουμε ήδη αναφέρει ακόμα και η διαγραφή μιας εγγραφής γίνεται με εισαγωγή μιας εγγραφής, που δηλώνει την διαγραφή της προηγούμενης. Ο χρήστης πρέπει να δώσει το path που θέλει να διαγράψει. Ο χρήστης λέει στον πελάτη ότι θέλει να εκτελέσει την κλήση DELFILE με τις αντίστοιχες παραμέτρους. Ο πελάτης το λέει στο TAPService, αυτό κατασκευάζει το ανάλογο String με τα ανάλογα ορίσματα του Πίνακα 3.1.2 και το στέλνει στο CDS. Το CDS κάνει parse το String και βλέπει ότι πρόκειται για εντολή εισαγωγής (διαγραφής) ενός αρχείου. Πριν κάνει την εισαγωγή πρέπει να πιστοποιήσει ότι αυτός ο πελάτης μπορεί να διαγράψει αυτό το αρχείο στον συγκεκριμένο κατάλογο.

Παρακάτω ακολουθεί ο ψευδοκώδικας της συνάρτησης DELFILE :

DELFILE(path)

Listing 3.5: Command "DELFILE"

```
1  int count = countDirs(path);
2  String[] all_keys = String[count + 3];
3
4  all_keys[0] = "groups:" + user; //user_group
5  all_keys[1] = path;
6  all_keys[count + 2] = "dirs:" + delLastField(path);
7              //Previous directory contents
8
9  /* Going to do queries */
10
11 gisp.query.(all_keys[0]);
12 gisp.query.(all_keys[1]);
13 gisp.query.(all_keys[count + 2]);
14
15 String pathtemp = path;
16 for(int i = 2; i < (count + 2); i++) {
17 while (!pathtemp.equals("/") ) {
18   pathtemp = delLastField(pathtemp);
19   query(pathtemp);
20   all_keys[i] = pathtemp;
21 }
22
23 gisp.wait(WAIT_TIME_UPDATE);
24
25 /* Going to do cheks */
26
27 if (EntryExist(MostRecentRecordOf(all_keys[0]) ) {
28   if (userNotInGroup(MostRecentRecordOf(all_keys[0]) ) ) {
29     return "ERROR: User not exist in this group.";
30   }
31 }
32
33 String initial_path_data = MostRecentRecordOf(all_keys[1]);
34                         //Our initial path
35 if (EntryExist(initial_path_data) ) {
36   if (!HaveReadWritePermissions(initial_path_data) ) {
37     return "ERROR: Operation not permitted";
38   }
39 } else {
```

```
40    return "ERROR: This files does not exist";
41  }
42
43  String result;
44  for (int i = (count + 1); i > 1; i--) {
45    result = MostRecentRecordOf(all_keys[i]);
46    if (EntryExist(result) ) {
47      if (!HaveReadWritePermissions(result) ) {
48        return "ERROR: Operation not permitted";
49      }
50    } else {
51      return "ERROR: Previous directory not exist";
52    }
53  }
54
55
56  /* Going to insert the new data */
57
58  String prv_dir_data_dir = MostRecentRecordOf(all_keys[count + 2]);
59  String prv_dir_contents = GetOnlyContents(prv_dir_data_dir);
60  prv_dir_contents = removeField(prv_dir_contents, getLastField(path) );
61  syncSetNewTime(prv_dir_data_dir);
62
63  gisp.insert(all_keys[count + 2], prv_dir_contents + GridID + LTime);
64  gisp.insert(path, url + owner + group + permissions + \\
65        "DELETED" + GridID + LTime);
```

DELDIR

Ο χρήστης καλεί την DELDIR όταν θέλει να κάνει διαγραφή μιας εγγραφής που αναφέρεται σε κατάλογο. Όπως έχουμε ήδη αναφέρει ακόμα και η διαγραφή μιας εγγραφής γίνεται με εισαγωγή μιας εγγραφής, που δηλώνει την διαγραφή της προηγούμενης. Πρέπει να δώσει το path που θέλει να διαγράψει. Ο χρήστης λέει στον πελάτη ότι θέλει να εκτελέσει την κλήση DELDIR με τις αντίστοιχες παραμέτρους. Ο πελάτης το λέει στο TAPService, αυτό κατασκευάζει το ανάλογο String με τα ανάλογα ορίσματα του Πίνακα 3.1.2 και το στέλνει στο CDS. Το CDS κάνει parse το String και βλέπει ότι πρόκειται για εντολή εισαγωγής (διαγραφής) ενός καταλόγου. Πριν κάνει την εισαγωγή πρέπει να πιστοποιήσει ότι αυτός ο πελάτης μπορεί να διαγράψει έναν κατάλογο από τον κατάλογο που επιθυμεί.

Παρακάτω ακολουθεί ο ψευδοκώδικας της συνάρτησης DELDIR :

DELDIR(path)

Listing 3.6: Command "DELDIR"

```
1  int count = countDirs(path);
2  String[] all_keys = String[count + 4];
3
4  all_keys[0] = "groups:" + user; //user_group
5  all_keys[1] = path;
6  all_keys[count + 2] = "dirs:" + delLastField(path);
7              //Previous directory contents
8  all_keys[count + 3] = "dirs:" + path;
9              //Initial directory contents
10
11 /* Going to do queries */
12
13 gisp.query.(all_keys[0]);
14 gisp.query.(all_keys[1]);
15 gisp.query.(all_keys[count + 2]);
16 gisp.query.(all_keys[count + 3]);
17
18 String pathtemp = path;
19 for(int i = 2; i < (count + 2); i++) {
20 while (!pathtemp.equals("/") ) {
21   pathtemp = delLastField(pathtemp);
22   query(pathtemp);
23   all_keys[i] = pathtemp;
24 }
25
26 gisp.wait(WAIT_TIME_UPDATE);
27
28 /* Going to do cheks */
29
30 if (EntryExist(MostRecentRecordOf(all_keys[0]) ) {
31   if (userNotInGroup(MostRecentRecordOf(all_keys[0]) ) ) {
32     return "ERROR: User not exist in this group.";
33   }
34 }
35
36 String initial_path_data = MostRecentRecordOf(all_keys[1]);
37                         //Our initial path
38 if (EntryExist(initial_path_data) ) {
39   if (!HaveReadWritePermissions(initial_path_data) ) {
```

```
40      return "ERROR: Operation not permitted";
41    }
42  } else {
43    return "ERROR: This directory does not exist";
44  }
45
46
47
48  String initial_dir_data_dir = MostRecentRecordOf(all_keys[count + 3]);
49                        //Contents of initial directory
50  if (EntryExist(initial_dir_data_dir) ) {
51    return "ERROR: Operation not permitted, \\
52  ..........because directory is not empty.";
53  }
54
55  String result;
56  for (int i = (count + 1); i > 1; i--) {
57    result = MostRecentRecordOf(all_keys[i]);
58    if (EntryExist(result) ) {
59      if (!HaveReadWritePermissions(result) ) {
60        return "ERROR: Operation not permitted";
61      }
62    } else {
63      return "ERROR: Previous directory not exist";
64    }
65  }
66
67
68  /* Going to insert the new data */
69
70  String prv_dir_data_dir = MostRecentRecordOf(all_keys[count + 2]);
71  String prv_dir_contents = GetOnlyContents(prv_dir_data_dir);
72  prv_dir_contents = removeField(prv_dir_contents, getLastField(path));
73  syncSetNewTime(prv_dir_data_dir);
74
75  gisp.insert(all_keys[count + 2], prv_dir_contents + GridID + LTime);
76  gisp.insert(path, url + owner + group + permissions + \\
77            "DELETED" + GridID + LTime);
```

AddUserToGroup

Ο χρήστης καλεί την AddUserToGroup όταν θέλει να κάνει εισαγωγή, ώστε να προσθέσει έναν πελάτη σε μια ομάδα. Πρέπει να δώσει το ID του πελάτη και την ομάδα που θέλει να τον κάνει μέλος. Ο χρήστης λέει στον πελάτη ότι θέλει να εκτελέσει την κλήση AddUserToGroup με τις αντίστοιχες παραμέτρους. Ο πελάτης το λέει στο TAPService, αυτό κατασκευάζει το ανάλογο String με τα ανάλογα ορίσματα του Πίνακα 3.1.2 και το στέλνει στο CDS. Το CDS κάνει parse το String και βλέπει ότι πρόκειται για εντολή εισαγωγής (πρόσθεσης ενός πελάτη σε μια ομάδα). Πριν κάνει την εισαγωγή πρέπει να πιστοποιήσει ότι αυτός ο πελάτης που εκτελεί αυτή την εντολή έχει προνόμια root, δηλαδή είναι ο root ή ανήκει στο group του root.

Παρακάτω ακολουθεί ο ψευδοκώδικας της συνάρτησης AddUserToGroup :

AddUserToGroup(user, newgroup)

Listing 3.7: Command "AddUserToGroup"

```
1  if (!thisClientHasRootPermissions() ) {
2    return "ERROR: This operation not permitted";
3  }
4
5  String mygroups = "groups:" + user;
6
7  gisp.query(mygroups);
8
9  gisp.wait(WAIT_TIME_UPDATE);
10
11 String result = MostRecentRecordOf(mygroups);
12 if (EntryExist(result) {
13   String cur_groups = getOnlyGroups(result);
14   syncSetNewTime(result);
15   if (isUserInGroup(cur_groups, newgroup) ) {
16     return "ERROR: User already exists in this group.";
17   }
18   gisp.insert(mygroups, cur_groups + newgroup + GridID + LTime);
19 } else {
20   syncSetNewTime(""); //No changes at time;
21   gisp.insert(mygroups, newgroup + GridID + LTime);
22 }
```

DelUserFromGroup

Ο χρήστης καλεί την DelUserFromGroup όταν θέλει να κάνει εισαγωγή, ώστε να διαγράψει έναν πελάτη από μια ομάδα. Πρέπει να δώσει το ID του πελάτη και την ομάδα που θέλει να τον διαγράψει από μέλος. Ο χρήστης λέει στον πελάτη ότι θέλει να εκτελέσει την κλήση AddUserToGroup με τις αντίστοιχες παραμέτρους. Ο πελάτης το λέει στο TAPService, αυτό κατασκευάζει το ανάλογο String με τα ανάλογα ορίσματα του Πίνακα 3.1.2 και το στέλνει στο CDS. Το CDS κάνει parse το String και βλέπει ότι πρόκειται για εντολή εισαγωγής (διαγραφής ενός πελάτη από μια ομάδα). Πριν κάνει την εισαγωγή πρέπει να πιστοποιήσει ότι αυτός ο πελάτης που εκτελεί αυτή την εντολή έχει προνόμια root, δηλαδή είναι ο root ή ανήκει στο group του root.

Παρακάτω ακολουθεί ο ψευδοκώδικας της συνάρτησης DelUserFromGroup :

DelUserFromGroup(user, delgroup)

Listing 3.8: Command "DelUserFromGroup"

```
1  if (!thisClientHasRootPermissions() ) {
2    return "ERROR: This operation not permitted";
3  }
4
5  String mygroups = "groups:" + user;
6
7  gisp.query(mygroups);
8
9  gisp.wait(WAIT_TIME_UPDATE);
10
11 String result = MostRecentRecordOf(mygroups);
12 if (EntryExist(result) {
13   String cur_groups = getOnlyGroups(result);
14   syncSetNewTime(result);
15   if (!isUserInGroup(cur_groups, delgroup) ) {
16     return "ERROR: User does not exist in this group.";
17   }
18   gisp.insert(mygroups, delFieldFrom(delgroup, cur_groups) + GridID + LTi
19 } else {
20   syncSetNewTime(""); //No changes at time;
21   return "ERROR: User does not exist in this group.";
22 }
```

GetGroups

Ο χρήστης καλεί την GetGroups όταν θέλει να κάνει μάθει τις ομάδες που είναι μέλος ενας πελατης. Πρέπει μονάχα να δώσει το ID του πελάτη που τον ενδιαφερει. Ο χρήστης λέει στον πελάτη ότι θέλει να εκτελέσει την κλήση AddUserToGroup με τις αντίστοιχες παραμέτρους. Ο πελάτης το λέει στο TAPService, αυτό κατασκευάζει το ανάλογο String με τα ανάλογα ορίσματα του Πίνακα 3.1.2 και το στέλνει στο CDS. Το CDS κάνει parse το String και βλέπει ότι πρόκειται για εντολή ανάκτησης (ανάκτησης των ομάδων ενός πελάτη).

Παρακάτω ακολουθεί ο ψευδοκώδικας της συνάρτησης GetGroups :

GetGroups(user)

Listing 3.9: Command "GetGroups"

```
1  String mygroups = "groups:" + user;
2
3  gisp.query(mygroups);
4
5  gisp.wait(WAIT_TIME_UPDATE);
6
7  String result = MostRecentRecordOf(mygroups);
8  if (EntryExist(result) {
9    String cur_groups = getOnlyGroups(result);
10   syncSetNewTime(result);
11   return cur_groups;
12 } else {
13   syncSetNewTime(""); //No changes at time;
14   return "ERROR: Not any groups for this user.";
15 }
```

3.2.2 Το Πρωτόκολλο-2

Όπως έχουμε ήδη αναφέρει, το GISP λειτουργεί με την τεχνική του Time-To-Live (TTL), κάθε εγγραφή μετά το πέρασμα αυτού του χρόνου σβήνεται αυτομάτα. Για να κρατάμε μόνο τις έγκυρες εγγραφές και να διατηρούμε τη δομή συνεπή, έχουμε χτίσει μέσα στο TAPService έναν μηχανισμό (DoRenew) που ανανεώνει μόνο τις έγκυρες εγγραφές.

Κάθε Grid server έχει δύο σταθερά πεδία, το πρώτο είναι ένας ακέραιος (TTL_RENEW) που είναι το χρονικό διάστημα, βάση του οποίου ανανεώνεται περιοδικά μια έγκυρη εγγραφή, και είναι λίγο μικρότερο από το TTL της εγγραφής. Ενώ το δεύτερο είναι μια ακέραια

τιμή (MAX_COUNT_RENEW) που δηλώνει ότι κάθε ¨τόσες' φορές ανανέωσης μιας έγκυρης εγγραφής θα γίνεται έλεγχος (μέσω του CDS καλώντας την query) αν αυτή η εγγραφή έχει ανανεωθεί ή διαγραφεί από κάποιον άλλον Grid server.

Από το Πρωτόκολλο-1 είδαμε ότι υπάρχουν εννέα κλήσεις, από τις οποίες οι τρεις είναι ανάκτησης και οι έξι αποθήκευσης. Αυτές τις έξι (PUT, MKDIR, DELFILE, DELDIR, AddUserToGroup, DelUserFromGroup) θα τις χωρίσουμε σε δυο κατηγορίες. Κάθε μία από αυτές για να εκτελεστεί πλήρως καλεί την *insert* (του Πίνακα 3.3) τουλάχιστον μία φορά. Στην πρώτη κατηγορία είναι οι εξής: *PUT, MKDIR, AddUserToGroup, DelUserFromGroup*. Το TTL που περνάει στην *insert* όταν καλείται από αυτή την κατηγορία έχει μια τιμή, έστω Vttl. Στην δεύτερη κατηγορία ανήκουν οι εξής: *DELFILE, DELDIR*. Το TTL που περνάει στην *insert* όταν καλείται από αυτή την κατηγορία έχει τιμή ίση με MAX_COUNT_RENEW φορές την προηγούμενη συν μια άλλη τιμή (αρκετά μικρότερη του Vttl), έστω Tmin. Άρα το TTL της δεύτερης κατηγορίας είναι ίσο με (MAX_COUNT_RENEW*Vttl + Tmin).

Το κριτήριο που διαχώρισε σε δυο κατηγορίες τις παραπάνω κλήσεις είναι ότι οι κλήσεις τις δεύτερης κατηγορίας προσπαθούν να διαγράψουν εντελώς ένα κλειδί. Για παράδειγμα η DELFILE προσπαθεί να σβήσει εντελώς από τη δομή το κλειδί π.χ. "/etc/profile", ομοίως και η DELDIR. Αντίθετα, οι κλήσεις της πρώτης κατηγορίας προσπαθούν είτε να ανανεώσουν την τιμή ενός κλειδιού είτε να το δημιουργήσουν αν δεν υπάρχει. Ακόμα και η DelUserFromGroup, δεν προσπαθεί να διαγράψει το κλειδί που αναφέρεται στις ομάδες ενός πελάτη, αλλά να του ανανεώσει την τιμή σβήνοντας μια ομάδα από αυτές. Δεδομένου ότι κάθε πελάτης με ένα ID ανήκει και στην ομώνυμη ομάδα, όπου δεν μπορεί να διαγράφει από αυτή, άρα η DelUserFromGroup κάνει πάντα ανανέωση της εγγραφής.

Κάθε Grid server έχει τον δικό του μηχανισμό που λειτουργεί αυτόνομα από τους άλλους, και ενημερώνεται μόνο έμμεσα από τους πελάτες του. Ο μηχανισμός έχει μια λίστα όπου κάθε κόμβος της περιέχει πληροφορία για κάθε κλήση της insert που έχει γίνει από τον δικό του Grid server, και προήλθε από την εκτέλεση μιας κλήσης της πρώτης κατηγορίας. Η πληροφορία είναι οι δύο παράμετροι τηςinsert (το κλειδί Gkey και η τιμή του Gvalue), η τοπική ωρολογιακή στιγμή (Gtime) που έγινε η κλήση της συνάρτησης που προκάλεσε την insert και άλλος ένας ακέραιος αριθμός (Gcount και αρχικοποιείται με μηδέν) που μετράει πόσες φορές έχει φτάσει στο τέλος της λίστας αυτός ο κόμβος.

Η λίστα λειτουργεί περίπου ως ουρά, δηλαδή κάθε καινούργιος κόμβος μπαίνει στην αρχή και σβήνουμε μόνο από το τέλος. Αυτό που δεν την κάνει ουρά είναι ότι δεν μπορεί να έχει δυο κόμβους ίδιους. Σε κάθε καινούργια εισαγωγή, ελέγχουμε αν υπάρχει ίδιος κόμβος, αν υπάρχει

τον σβήνουμε, και πάντα προσθέτουμε τον καινούργιο κόμβο στην αρχή της λίστας. Το κριτήριο ομοιότητας των κόμβων είναι το πεδίο με το κλειδί της insert, το Gkey. Καινούργιες εισαγωγές στη λίστα γίνονται κάθε φορά που εκτελείται μια από τις συναρτήσεις της πρώτης κατηγορίας, η οποία μπορεί να προκαλέσει και περισσότερες από μία (εισαγωγές στη λίστα), όλες όμως θα έχουν την ίδια τιμή στο Gtime.

Ο μηχανισμός λειτουργεί ως εξής: Κάθε κλήση μιας συνάρτησης της πρώτης κατηγορίας προκαλεί μία ή δύο κλήσεις της insert. Για κάθε κλήση της insert δημιουργούμε και έναν κόμβο και τον προσθέτουμε στην αρχή της λίστας. Αν η λίστα είναι άδεια, προσθέτουμε τον έναν (ή τους δυο) κόμβους στη λίστα και ο μηχανισμός κοιμάται μέχρι να περάσει χρόνος ίσος με αυτόν που δηλώνει το TTL_RENEW. Μόλις περάσει το TTL_RENEW ξυπνάει και ελέγχει τους δυο τελευταίους κόμβους της λίστας. Αν αυτοί έχουν το ίδιο Gtime πάει να πει ότι εισήχθησαν μαζί στη λίστα, οπότε τους σβήνει και τους δυο από το τέλος και τους ξαναεισαγει, αφού τους αλλάξει το Gtime με το τωρινό και αυξήσει και το Gcount του καθενός κατά ένα. Αν δεν είναι ίδιοι οι δυο τελευταίοι ακολουθείται και πάλι η ίδια διαδικασία. Έχοντας κρατήσει το παλιό Gtime, βρίσκει τη διαφορά του από το Gtime του κόμβου που είναι τώρα στο τέλος της λίστας. Αν υπάρχει και δεύτερος (ή τρίτος) κόμβος στη λίστα ο μηχανισμός θα κοιμηθεί χρόνο ίσο με αυτή τη διαφορά, αν δεν υπάρχει τέτοιος κόμβος ο μηχανισμός θα κοιμηθεί πάλι για TTL_RENEW. Μετά το τέλος της διαφοράς γίνεται πάλι το ίδιο. Αν κάποιος κόμβος εισαχθεί στη λίστα, ενώ δεν είναι άδεια, ο μηχανισμός κοιμάται ή ξυπνάει σύμφωνα με τις εντολές που έχει πάρει από τους κόμβους που είναι ήδη μέσα στη λίστα. Με αυτόν τον τρόπο κάθε κόμβος ανανεώνεται στη δομή περιοδικά, με περίοδο το χρόνο TTL_RENEW.

Κάθε φορά που ξυπνάει ο μηχανισμός ελέγχει τους δυο τελευταίους κόμβους. Ο έλεγχος γίνεται πάνω στο Gtime (για το TTL) αλλά και πάνω στο Gcount. Αν κάποιου, το πεδίο Gcount έχει φτάσει να είναι ίσο με MAX_COUNT_RENEW και πρεπει ξαναεισαχθεί (στην αρχή), τότε γίνεται πρώτα έλεγχος στο GISP αν αυτός είναι πραγματικά έγκυρος και μετά ξαναεισάγεται. Δηλαδή δεν έχει ανανεωθεί ή διαγραφεί από κάποιον άλλο Grid server. Αν έχει γίνει κάτι τέτοιο τότε απλά δεν ξαναεισάγεται και σβήνεται τελείως από τη λίστα.

Όπως αναφέραμε και πριν, οι συναρτήσεις της δεύτερης κατηγορίας, καλούν την insert με TTL=(MAX_COUNT_RENEW*TTL + Tmin). Με αυτό πετυχαίνουμε δυο στόχους. Ο πρώτος είναι ότι αποφεύγουμε την επεξεργασία αυτών των εγγραφών, άρα και την καθυστέρηση να ξαναστείλουμε (ανανεώσουμε) μια τέτοια εγγραφή στο GISP. Ο δεύτερος και πιο σημαντικός είναι ότι δεν χρειάζεται να πληρώσουμε της καθυστερήσεις μιας εκτέλεσης της query (στο GISP) για να μάθουμε, μετά από κάποιο χρόνο, αν η διαγραφή που κάναμε πριν από

Συνάρτηση
insert(key, value)
query(key)
update(key, value)
query(key)

Πίνακας 3.6: Η ακολουθία των εκτελέσεων.

κάποιο χρόνο είναι έγκυρη οπότε να την αναστείλουμε ή όχι. Υπερκαλύπτοντας τον χρόνο που ένας Grid server ελέγχει περιοδικά αν μια εισαγωγή του είναι ακόμα έγκυρη, γλιτώνουμε όλα τα παραπάνω.

3.3 Πειράματα

Σε αυτή την ενότητα θα ανάφερουμε τα αποτελέσματα των πειραμάτων που εκτελέσαμε, αλλά και τις συνθήκες βάση των οποίων έγιναν.

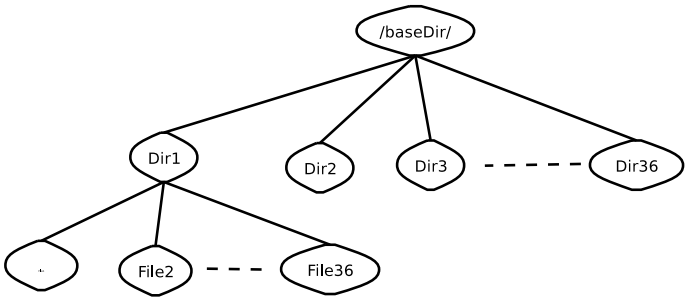
Στα πειράματα μετρήσαμε τον χρόνο που απαιτείται για να εκτελεστούν οι συναρτήσεις του Πίνακα 3.3 με τη σειρά που δίνονται.

Τα κλειδιά της σειράς εκτελέσεων της update και των δύο σειρών εκτελέσεων της query δείχνουν σε αρχεία. Τα κλειδιά της σειράς update και της δεύτερης σειράς της query είναι τα ίδια. Οι εισαγωγές (εκτελέσεις της insert) έγιναν με τη μορφή δέντρου, και όλες τους είχαν διαφορετικό κλειδί.

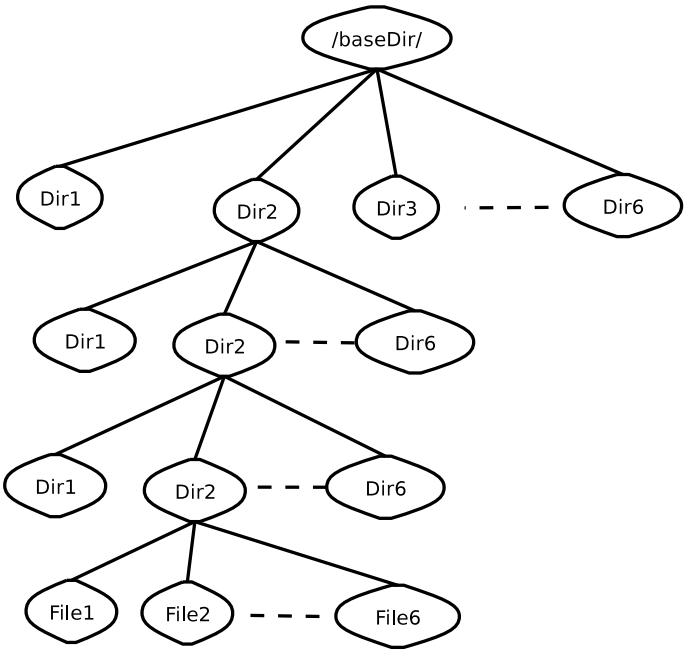
Ο αριθμός των κόμβων που έτρεχε το P2P κατά τη διάρκεια των πειραμάτων είχε τρείς τιμές, τρείς κόμβους, έξι κόμβους και εννέα κόμβους. Η ακολουθία εκτέλεσης του Πίνακα 3.3 πραγματοποιήθηκε δύο φορές για κάθε αριθμό κόμβων και με διαφορετικό δέντρο εισαγωγής.

Τα στοιχεία του τελευταίου επιπέδου του κάθε δέντρου εισαγωγής περιέχουν κλειδιά που δείχνουν σε αρχεία, ενω όλα τα υπόλοιπα περιέχουν κλειδιά που δείχνουν σε καταλόγους. Τα δύο δέντρα έχουν διαφορετικό ύψος και πλάτος, και φαίνονται στο Σχ. 3.2 και στο Σχ. 3.3.

Στον Πίνακα 3.3 αναφέρονται όλες οι περιπτώσεις με δέντρο εισαγωγής το δέντρο του Σχ. 3.2, αλλά και οι αντίστοιχοι χρόνοι που διάρκησε η κάθε περίπτωση. Ενώ στον Πίνακα 3.3 αναφέρονται όλες οι περιπτώσεις με δέντρο εισαγωγής το δέντρο του Σχ. 3.3 και οι αντίστοιχοι χρόνοι.



Σχήμα 3.2: Το δέντρο εισαγωγής με ύψος H=2 και πλάτος N=36.



Σχήμα 3.3: Το δέντρο εισαγωγής με ύψος H=4 και πλάτος N=6.

Κλήσεις	Αριθμός Κόμβων		
	9	6	3
1332 * insert	16783	16743	16716
500 * query	3853	3845	3842
500 * update	6296	6280	6272
500 * query	3859	3846	3844

Πίνακας 3.7: Οι χρόνοι εκτέλεσης σε δευτερόλεπτα, για H=2 και N=36.

Κλήσεις	Αριθμός Κόμβων		
	9	6	3
1554 * insert	19590.4	19559	19563
500 * query	3856	3845	3843
500 * update	6300	6291	6283
500 * query	3865	3849	3848

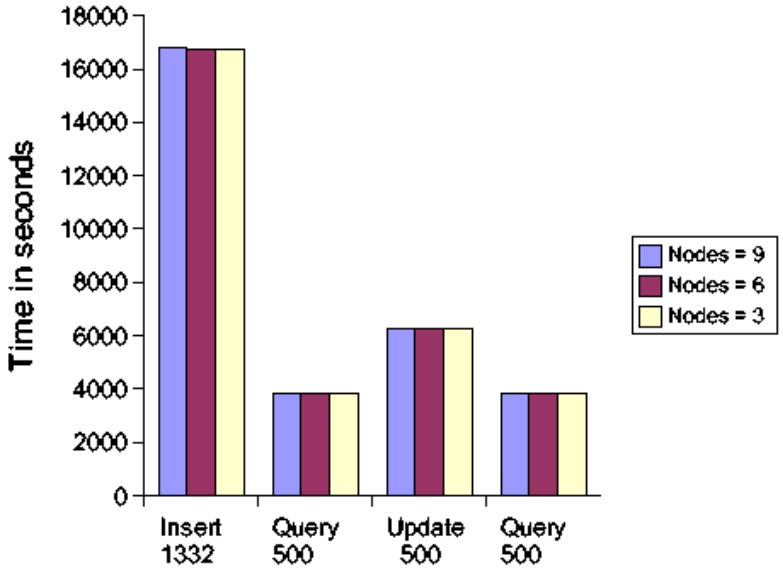
Πίνακας 3.8: Οι χρόνοι εκτέλεσης, σε δευτερόλεπτα, για H=4 και N=6.

Εκτός από τους δύο πίνακες, τα αποτελέσματα των πειραμάτων έχουν παρασταθεί και γραφικά με ραβδογράμματα και φαίνονται στο Σχ. 3.4 και στο Σχ. 3.5.

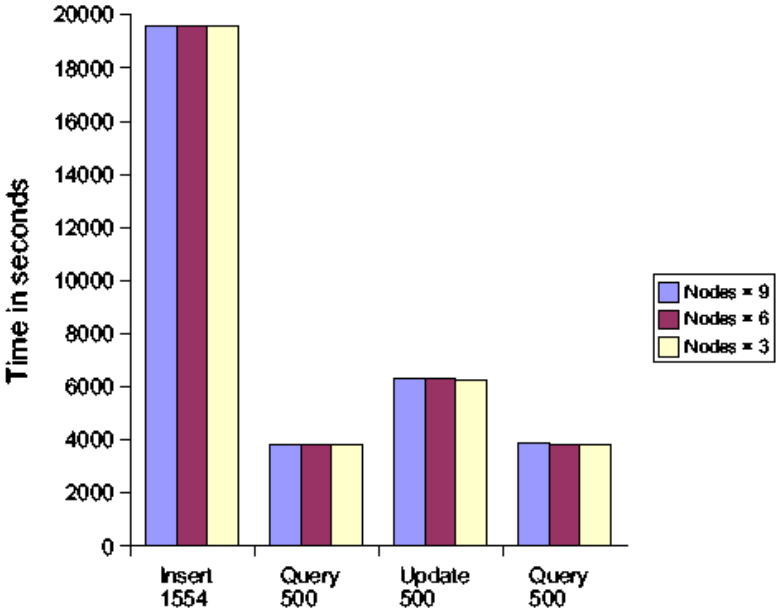
Εκτός από τους χρόνους που παρατίθενται για τη διάρκεια των εκτελέσεων, υπάρχει και άλλη μια χρονική σταθερά που είναι ο χρόνος που χρειάζεται, ένας πελάτης, για να κάνει login και να φορτώσει το αντίστοιχο Factory με τις ανάλογες υπηρεσίες. Ο χρήστης έχει την δυνατότητα να κάνει login-logout για να εκτελέσει κάθε μία κλήση, από τις κλήσεις του Πίνακα 3.1.1, αλλά και την δυνατότητα να εκτελέσει πολλές μαζί. Στην περίπτωση μας, για κάθε είδος κλήσης κάναμε μονάχα μια φορά login. Για παράδειγμα, για τις 500 κλήσεις τις update, έγινε μόνο μια φορά login, ομοίως και για τις υπόλοιπες.

Όπως φαίνεται από τα γραφήματα, αλλά και λεπτομερέστερα από τους πίνακες, οι χρόνοι που χρειάζεται η κάθε κλήση δεν έχει μεγάλη απόκλιση όταν αλλάζει ο αριθμός των κόμβων του P2P.

Τα πειράματα έγιναν στο εργαστήριο λογισμικού του Πολυτεχνείου.



Σχήμα 3.4: Οι χρόνοι εκτέλεσης σε δευτερόλεπτα, για δέντρο με H=2 και πλάτος N=36.



Σχήμα 3.5: Οι χρόνοι εκτέλεσης σε δευτερόλεπτα, για δέντρο με H=4 και πλάτος N=6.

Κεφάλαιο 4

Συμπεράσματα και Μελλοντική δουλειά

Έχοντας φτάσει στο τέλος της ολοκλήρωσης αυτής της διατριβής, κατάλαβα και γνώρισα αρκετές καινούργιες και πολύ ενδιαφέρον ιδέες και πράγματα που μου ήταν εντελώς άγνωστα. Επίσης, διαλεύκανα και πολλά πράγματα/ορισμούς που γνώριζα, αλλά δεν ήξερα ούτε τι ακριβώς είναι, ούτε που χρησιμοποιούνται στον πραγματικό κόσμο και στην επιστήμη.

Το TAP είναι μια εφαρμογή που συνδυάζει τις τεχνολογίες Grid και P2P, οι οποίες αν και πολλά τα κοινά τους σημεία τώρα γίνονται οι πρώτες προσπάθειες συνεργασίας των δύο τεχνολογιών. Το TAP είναι χτισμένο στην κατανεμημένη φύση των P2P δικτύων και στην ασφαλή δομή του Grid. Με αυτόν τον τρόπο παρέχει ασφάλεια και διαμιρασμό εργασίας και πληροφορίας.

Μελλοντική δουλειά που μπορεί να γίνει πάνω στο TAP είναι η βελτιστοποίηση του GSIP ώστε να παρέχει πιο γρήγορα αποτελέσματα, αλλά και βελτίωσης του αλγόριθμου δρομολόγησης των μηνυμάτων μεταξύ των κόμβων του. Επιπλέον, βελτίωση του TAP μπορεί να επιτευχθεί αν χρησιμοποιηθεί ασύγχρονη επικοινωνία μεταξύ του Grid και του CDS.

Μια πιθανή χρησιμοποίηση του TAP μπορεί να είναι σε μια κοινότητα, π.χ. στο Πολυτεχνείο, για να διαμοιράζονται έγγραφα, όπως δημοσιεύσεις, μεταξύ των μελών της κοινότητας, αλλά και έξω από αυτήν, αποκτώντας πρόσβαση μονάχα με την άδεια της αρμόδιου φορέα της κοινότητας. Άλλη μια εφαρμογή συγκεκριμένα για το Πολυτεχνείο, μπορεί να είναι η χρησιμοποίηση του TAP ως μιας υπηρεσίας παροχής των εκάστοτε URL όλων των μαθημάτων, των εργαστηρίων τους, σχετικά e-mails και ότι άλλο μπορεί να χρειάζεται.

Παράρτημα Α΄

Η δήλωση της υπηρεσίας **TAPService** σε **WSDL**

```
<types>
<xsd:schema targetNamespace="http://www.globus.org/␣\\
␣␣␣␣␣␣namespaces/2004/02/progtutorial/FSService"
attributeFormDefault="qualified"
elementFormDefault="qualified"
xmlns="http://www.w3.org/2001/XMLSchema">
<xsd:element name="get">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="ipath" type="xsd:string"/>
      <xsd:element name="iuser" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="getResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="res" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="put">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="ipath" type="xsd:string"/>
      <xsd:element name="iuser" type="xsd:string"/>
```

```
28        <xsd:element name="iurl" type="xsd:string"/>
29      </xsd:sequence>
30    </xsd:complexType>
31  </xsd:element>
32  <xsd:element name="putResponse">
33    <xsd:complexType>
34      <xsd:sequence>
35        <xsd:element name="res" type="xsd:string"/>
36      </xsd:sequence>
37    </xsd:complexType>
38  </xsd:element>
39
40  <xsd:element name="ls">
41    <xsd:complexType>
42      <xsd:sequence>
43        <xsd:element name="ipath" type="xsd:string"/>
44        <xsd:element name="iuser" type="xsd:string"/>
45      </xsd:sequence>
46    </xsd:complexType>
47  </xsd:element>
48  <xsd:element name="lsResponse">
49    <xsd:complexType>
50      <xsd:sequence>
51        <xsd:element name="res" type="xsd:string"/>
52      </xsd:sequence>
53    </xsd:complexType>
54  </xsd:element>
55
56  <xsd:element name="mkdir">
57    <xsd:complexType>
58      <xsd:sequence>
59        <xsd:element name="ipath" type="xsd:string"/>
60        <xsd:element name="iuser" type="xsd:string"/>
61        <xsd:element name="iurl" type="xsd:string"/>
62      </xsd:sequence>
63    </xsd:complexType>
64  </xsd:element>
65  <xsd:element name="mkdirResponse">
66    <xsd:complexType>
67      <xsd:sequence>
68        <xsd:element name="res" type="xsd:string"/>
69      </xsd:sequence>
70    </xsd:complexType>
```

```
71 </xsd:element>
72
73 <xsd:element name="delfile">
74   <xsd:complexType>
75     <xsd:sequence>
76       <xsd:element name="ipath" type="xsd:string"/>
77       <xsd:element name="iuser" type="xsd:string"/>
78     </xsd:sequence>
79   </xsd:complexType>
80 </xsd:element>
81 <xsd:element name="delfileResponse">
82   <xsd:complexType>
83     <xsd:sequence>
84       <xsd:element name="res" type="xsd:string"/>
85     </xsd:sequence>
86   </xsd:complexType>
87 </xsd:element>
88
89 <xsd:element name="deldir">
90   <xsd:complexType>
91     <xsd:sequence>
92       <xsd:element name="ipath" type="xsd:string"/>
93       <xsd:element name="iuser" type="xsd:string"/>
94     </xsd:sequence>
95   </xsd:complexType>
96 </xsd:element>
97 <xsd:element name="deldirResponse">
98   <xsd:complexType>
99     <xsd:sequence>
100      <xsd:element name="res" type="xsd:string"/>
101    </xsd:sequence>
102  </xsd:complexType>
103 </xsd:element>
104
105 <xsd:element name="getGroups">
106   <xsd:complexType>
107     <xsd:sequence>
108       <xsd:element name="iuser" type="xsd:string"/>
109     </xsd:sequence>
110   </xsd:complexType>
111 </xsd:element>
112 <xsd:element name="getGroupsResponse">
113   <xsd:complexType>
```

```
114       <xsd:sequence>
115         <xsd:element name="res" type="xsd:string"/>
116       </xsd:sequence>
117     </xsd:complexType>
118   </xsd:element>
119
120   <xsd:element name="addUTG">
121     <xsd:complexType>
122       <xsd:sequence>
123         <xsd:element name="ruser" type="xsd:string"/>
124         <xsd:element name="iuser" type="xsd:string"/>
125         <xsd:element name="igroup" type="xsd:string"/>
126       </xsd:sequence>
127     </xsd:complexType>
128   </xsd:element>
129   <xsd:element name="addUTGResponse">
130     <xsd:complexType>
131       <xsd:sequence>
132         <xsd:element name="res" type="xsd:string"/>
133       </xsd:sequence>
134     </xsd:complexType>
135   </xsd:element>
136
137   <xsd:element name="delUFG">
138     <xsd:complexType>
139       <xsd:sequence>
140         <xsd:element name="ruser" type="xsd:string"/>
141         <xsd:element name="iuser" type="xsd:string"/>
142         <xsd:element name="igroup" type="xsd:string"/>
143       </xsd:sequence>
144     </xsd:complexType>
145   </xsd:element>
146   <xsd:element name="delUFGResponse">
147     <xsd:complexType>
148       <xsd:sequence>
149         <xsd:element name="res" type="xsd:string"/>
150       </xsd:sequence>
151     </xsd:complexType>
152   </xsd:element>
153
154
155   </xsd:schema>
156   </types>
```

Βιβλιογραφία

[1] Foster, I., Kesselman, C. and Tuecke, S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15 (3). 200-222. 2001. Franklin, B., Letter to Jean-Baptiste Leroy, 1789.

[2] Foster, I. The Grid: A New Infrastructure for 21st Century Science. *Physics Today*, 55 (2). 42-47. 2002.

[3] Ian Foster, Carl Kesselman, Jeffrey M. Nick, Steven Tuecke - The Physiology of the Grid : An Open Grid Services Architecture for Distributed Systems Integration.

[4] Foster, I., Kesselman, C., Lee, C., Lindell, R., Nahrstedt, K. and Roy, A., A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation. In Proc. International Workshop on Quality of Service, (1999), 27-36.

[5] Gentzsch W. (editor), Special Issue on Metacomputing: From Workstation Clusters to Internet computing, Future Generation Computer Systems, No. 15, North Holland, 1999.

[6] Grid Computing Infoware - http://www.gridcomputing.com

[7] The Globus Project - http://www.globus.org

[8] The Global Grid Forum - http://www.gridforum.org

[9] Christensen, E., Curbera, F., Meredith, G. and Weerawarana., S. Web Services Description Language (WSDL) 1.1. W3C, Note 15, 2001, www.w3.org/TR/wsdl.

[10] Brittenham, P. An Overview of the Web Services Inspection Language. 2001, www.ibm.com/developerworks/webservices/library/ws-wsilover.

[11] Oram, A. (ed.), Peer-to-Peer: Harnessing the Power of Disruptive Technologies. O'Reilly, 2001.

[12] Shirky, C. What Is P2P... and What Isn't?
www.openp2p.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html,
2000.

[13] 5. Chien, A., Calder, B., Elbert, S. and Bhatia, K. Entropia: Architecture and Performance of an Enterprise Desktop Grid System. Journal of Parallel and Distributed Computing, To appear.

[14] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan, Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, Proc. of Conference on applications, technologies, architectures, and protocols for computer communications, 2001.

[15] Antony Rowstron and Peter Druschel, Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems, Proc. of IFIP/ACM, 2001.

[16] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker, A scalable Content addressable Network, Proc. of ACM SIGCOMM, 2001.

[17] Fedak, G., Germain, C., Niri, V. and Cappello, F., XtremWeb : A Generic Global Computing System. Workshop on Global Computing on Personal Devices (CCGRID2001), Berlin, Germany, 2001, IEEE Press.

[18] http://www.jxta.org

[19] Project JXTA: Protocol Specification V2.0, February 2003, http://spec.jxta.org

[20] http://gisp.jxta.org

[21] Ji Li, Ben Leong, and Karen Sollins, Implementing Aggregation/Broadcast over Distributed Hash Tables, Tech. report, MIT, 2003.

[22] Benjie Chen, Thomer M. Gil, Athicha Muthitacharoen, and Robert Morris, Building Data Structures on Untrusted Peer-to-Peer Storage. http://www.lcs.mit.edu/publications/pubs/pdf/MIT-LCS-TR-888.pdf

[23] F. Dabek, M. Frans Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In Proc. of the ACM Symposium on Operating System Principles, October 2001.

[24] D. Parker, G. Popek, G. Rudisin, A. Stoughton, B. Walker, E. Walton, J. Chow, D. Edwards, S. Kiser, and C. Kline. Detection of mutual inconsistency in distributed systems. In IEEE Transactions on Software Engineering, volume 9(3), pages 240●247, 1983.

[25] http://www.fs.net/sfswww/

[26] M. Rosenblum and J. Ousterhout. The design and implementation of a log-structured file system. ACM Transactions on Computer Systems, 10(1):26●52, 1992.

[27] D. Terry, M. Theimer, K. Petersen, A. Demers, M. Spreitzer, and C. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. In Proc. of the ACM Symposium on Operating System Principles, pages 172●183, December 1995.