ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΕΦΑΡΜΟΣΜΕΝΩΝ ΜΑΘΗΜΑΤΙΚΩΝ ΚΑΙ ΦΥΣΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΟΜΕΑΣ ΦΥΣΙΚΗΣ
ΕΡΓΑΣΤΗΡΙΟ ΠΕΙΡΑΜΑΤΙΚΗΣ ΦΥΣΙΚΗΣ ΥΨΗΛΩΝ ΕΝΕΡΓΕΙΩΝ

Live Video Streaming με χρήση της υποδομής του GRID

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Δημητρίου Κ. Καρακασίλη

Επιβλέπων: Θεόδωρος Αλεξόπουλος
Καθηγητής

Αθήνα, Μάρτιος 2011

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 21η Μαρτίου 2011.

.....		.....
Γεώργιος Τσιπολίτης	.....	Θεόδωρος Αλεξόπουλος
Αναπληρωτής Καθηγητής	Ευάγγελος Γαζής Καθηγητής	Καθηγητής

Περίληψη

Στις 29 Μαρτίου του 2006 όσοι βρέθηκαν στο Καστελόριζο είχαν την τύχη να παρακολουθήσουν μία ολική έκλειψη ηλίου. Ήταν η πρώτη ολική έκλειψη που έγινε ορατή στην Ελλάδα μετά από 70 χρόνια. Για όσους δεν μπορούσαν να είναι στο Καστελόριζο, έγινε μία προσπάθεια από το Πανεπιστήμιο Αθηνών, με την συνδρομή ερασιτεχνών αστρονόμων, να μεταδοθεί η έκλειψη μέσω διαδικτύου. Το βίντεο μεταφερόταν σε video servers σε διάφορες αναλύσεις και ήταν διαθέσιμο για άμεση παρακολούθηση. Οι απαιτήσεις τελικά σε εύρος ζώνης ξεπέρασαν τα αναμενόμενα, με προφανή αποτελέσματα στην ποιότητα του βίντεο. Τότε γεννήθηκε η ιδέα, να χρησιμοποιηθούν οι πόροι του GRID, για την μετάδοση βίντεο μέσω διαδικτύου [1].

Το GRID λόγω της κατανομής του σε όλο τον κόσμο και λόγω του πλήθους των υπολογιστών και των συνδέσεων που το αποτελούν, θα αποτελούσε ιδανική πηγή πόρων για μία τέτοια εφαρμογή. Υλοποίηση αυτής της ιδέας αποτελεί αυτή η διπλωματική. Σκοπός της διπλωματικής ήταν να δείξει, ότι το GRID μπορεί να χρησιμοποιηθεί για δυναμική εκχώρηση πόρων, για μετάδοση ροής βίντεο. Στα επόμενα κεφάλαια έγινε προσπάθεια να δωθούν στον ενδιαφερόμενο όλες τις πληροφορίες που χρειάζονται για να "τρέξει" των κώδικα, αλλά περισσότερο να μπορέσει να τον βελτιώσει και να τον επεκτείνει. Η γλώσσα της διπλωματικής είναι τα ελληνικά, αλλά εξ'αιτίας της φύσης του θέματος και για να γνωρίζει ο ενδιαφερόμενος αναγνώστης τις κατάλληλες λέξεις κλειδιά, δεν είναι δυνατόν να παραληφθούν οι διεθνείς όροι. Έτσι όπου χρειάζεται δίνεται σε παρένθεση δίπλα από τον ελληνικό όρο και ο αντίστοιχος στα αγγλικά ή κάποιες φορές, μόνο ο αγγλικός. Η πληροφορίες οργανώθηκαν σε τρία κεφάλαια.

Το πρώτο περιλαμβάνει οδηγίες χρήσης του GRID και απευθύνεται κυρίως σε νέους χρήστες χωρίς προηγούμενη εμπειρία. Δεν αποτελεί πλήρες φυλλάδιο οδηγιών, αλλά θα πρέπει να είναι επαρκές, για να γίνουν κατανοητά τα επόμενα κεφάλαια. Το δεύτερο κεφάλαιο συνοψίζει τις υπάρχουσες τεχνολογίες μετάδοσης ροής βίντεο καθώς και πληροφορίες για τα πρωτόκολλα μετάδοσης και τις μεθόδους συμπίεσης. Ούτε εδώ οι πληροφορίες είναι εξαντλητικές, αλλά είναι αρκετές για να δικαιολογηθούν οι επιλογές που έγιναν, αλλά και να παρακολουθήσει ο ενδιαφερόμενος στο κεφάλαιο 3 την ανάπτυξη του κώδικα. Τέλος, το τρίτο κεφάλαιο είναι ουσιαστικά η τεκμηρίωση (documentation) της εφαρμογής, μαζί με τα συμπεράσματα και προτάσεις για πιθανές εφαρμογές και επεκτάσεις της διπλωματικής αυτής.

Στην σύζυγό μου, Νικολέττα

Ευχαριστώ τους γονείς μου για την χωρίς όρους στήριξή τους, τον καθηγητή μου κ. Θεόδωρο Αλεξόπουλο για την άμεση βοήθειά του και την εμπιστοσύνη του στην εκπόνηση της διπλωματικής, τον Φώτη Γεωργάτο για τις στοχευμένες παρατηρήσεις και διορθώσεις και τον Νίκο Βιδιαδάκη για την βοήθεια του στο στήσιμο της εφαρμογής.

Περιεχόμενα

Κεφάλαιο 1

Το GRID

1.1 Από τους Υπερυπολογιστές στο GRID

Οι υπολογιστές από την αρχή της ιστορίας τους ανέλαβαν την επίλυση προβλημάτων που απαιτούσαν πολλές πράξεις και επαναλήψεις. Από την παραγωγή γραφικών σε παιχνίδια μέχρι την επίλυση δύσκολων μαθηματικών προβλημάτων, οι σημερινοί προσωπικοί υπολογιστές τα καταφέρνουν ικανοποιητικά σε πλήθος εφαρμογών. Όμως οι απαιτήσεις κάποιων εφαρμογών σε επεξεργαστική ισχύ οδήγησαν τις εταιρίες την δεκαετία του 1960 να δημιουργήσουν μοναδικά πρωτότυπα υπολογιστών, με πλήθος επεξεργαστών, μεγάλες απαιτήσεις σε ενέργεια και ψύξη και ανάλογα ψηλό κόστος. Η εξέλιξη της τεχνολογίας, έκανε τους υπερυπολογιστές του παρελθόντος να μην εντυπωσιάζουν πλέον με τις επιδόσεις τους και φτάσαμε έτσι στον γρηγορότερο σήμερα υπερυπολογιστή Tianhe-I με ταχύτητα 2.566 petaFLOPS (10^{15} FLOPS).

Το κόστος και η πολύπλοκη αρχιτεκτονική των υπερυπολογιστών, οδήγησε στην αναζήτηση εναλλακτικής λύσης. Αποτέλεσμα ήταν η ιδέα του GRID computing. Ο συνδυασμός δηλαδή της επεξεργαστικής ισχύος, πολλών κοινών υπολογιστών. Το όνομα GRID οφείλεται στην αναλογία με το δίκτυο της ηλεκτρικής ενέργειας. Η παρακάτω σύγκριση δικαιολογεί την επιλογή αυτή:

Ηλεκτρικό δίκτυο	GRID
Δεν ενδιαφέρει τον καταναλωτή από που έρχεται η ηλεκτρική ενέργεια. Απλά συνδέει την συσκευή στην πρίζα και κάνει την δουλεία του.	Δεν ενδιαφέρει τον χρήστη σε ποιόν υπολογιστή τρέχει η εφαρμογή του. Απλά συνδέεται στο internet και κάνει την δουλειά του.
Υπάρχει μία υποδομή που συνδέει τις διάφορες μονάδες παραγωγής ενέργειας, μέσω διαφόρων σταθμών και γραμμών μεταφοράς. Η υποδομή φέρνει την ηλεκτρική ενέργεια σχεδόν παντού.	Υπάρχει μία υποδομή που φέρνει την επεξεργαστική ισχύ από διάφορους σταθμούς (υπολογιστές), σχεδόν παντού (Μέρος της υποδομής είναι το Internet)
Η Ηλεκτρική ενέργεια παρέχεται ως υπηρεσία. Την ζητάς, σου παρέχεται, την πληρώνεις.	Το GRID παρέχεται ως υπηρεσία. Ζητάς πρόσβαση,σου παρέχεται, την πληρώνεις.

Ο παραπάνω πίνακας είναι μία υπεραπλούστευση της σημερινής εικόνας του GRID, αφού προς το παρόν και τα τρία χαρακτηριστικά είναι πιο περιορισμένα στο GRID από το ηλεκτρικό δίκτυο.Παρ'όλα αυτά σχεδιάστηκε και εξελίσσεται με αυτήν την φιλοσοφία και κάποια στιγμή η αναλογία θα είναι πιο ρεαλιστική.

Για να γίνουν κατανοητά τα σχετικά με το θέμα της διπλωματικής αυτής, παρακάτω δίνεται μία συνοπτική περιγραφή των στοιχείων που συνθέτουν το GRID.

Σχήμα 1.1: Το GRID σε επίπεδα

1.2 Απο τι φτιάχνεται το GRID

Η αρχιτεκτονική του GRID περιγράφεται με την χρήση επιπέδων (layers) (Σχήμα 1.1[1]).

- Στο χαμηλότερο επίπεδο βρίσκεται το δίκτυο, που είναι αυτό που συνδέει τους διάφορους πόρους.

- Παραπάνω βρίσκεται το επίπεδο των πόρων,όπως είναι οι υπολογιστές, τα μέσα αποθήκευσης, αισθητήρες και τηλεσκόπια τα οποία είναι συνδεδεμένα στο δίκτυο.

- Το επίπεδο του middleware παρέχει τα εργαλεία που επιτρέπουν στα διάφορα στοιχεία (servers, μέσα αποθήκευσης, δίκτυα κ.τ.λ.) να συμμετέχουν σε ένα GRID. Το επίπεδο αυτό είναι ουσιαστικά ο εγκέφαλος πίσω από το GRID.

- Το ψηλότερο επίπεδο είναι αυτό της εφαρμογής, στο οποίο περιλαμβάνονται εφαρμογές επιστημονικές, μηχανικής, οικονομικές, και πολλές άλλες, καθώς επίσης και εργαλεία για την υποστήριξη των εφαρμογών.Αυτό είναι το επίπεδο με το οποίο αλληλεπιδρούν οι χρήστες.

[1] http://www.interactions.org/sgtw/2005/0914/images/grid_layers_400.jpg

Τι χρειάζεται όμως για να τρέξει κάποιος ένα πρόγραμμα στο GRID; Αρχικά θα πρέπει να γραφτεί σε ένα εικονικό οργανισμό (VO). Οι εικονικοί οργανισμοί είναι ομάδες χρηστών που δουλεύουν πάνω στο ίδιο project και χρησιμοποιούν της ίδιες εφαρμογές [18]. Τα μέλη διαφορετικών VO, μπορεί να έχουν πρόσβαση σε διαφορετικές υπηρεσίες του GRID. Στην συνέχεια, αφού αποκτήσει τα απαραίτητα πιστοποιητικά, χρειάζεται ένα λογαριασμό σε κάποιο User Interface (UI). Το UI είναι η "πύλη" του χρήστη στο GRID. Έχει εγκατεστημένα όλα τα προγράμματα που χρειάζονται για να εκτελεστεί ένα πρόγραμμα, να ακυρωθεί η εκτέλεση του, να λάβει ο χρήστης το αποτέλεσμα της εκτέλεσης και πολλά άλλα. Συνήθως ο χρήστης αποκτά λογαριασμό σε ένα υπάρχον UI, κατόπιν αίτησης. Μετά από αυτά, ο χρήστης μπορεί να τρέξει την εφαρμογή του στο GRID. Η εκτέλεση παρ'όλα αυτά μίας εφαρμογής στο GRID διαφέρει αρκετά από την εκτέλεση ενός προγράμματος σε ένα προσωπικό υπολογιστή. Ας δούμε την διαδικασία μέσα από ένα παράδειγμα.

Έστω ότι θέλουμε να τρέξουμε το παρακάτω απλό πρόγραμμα :

```
#!/usr/bin/python
print "Hello Grid!"
```

hello.py

Για να εκτελεστεί το πρόγραμμα αυτό σε linux χρειάζεται απλά να τρέξουμε την εντολή "./hello.py". Για να τρέξει το πρόγραμμα στο GRID χρειάζονται τα παρακάτω βήματα :

1. Συνδεόμαστε στο UI με τον λογαρισμό μας. Συνήθως αυτό γίνεται μέσω ssh είτε από το τερματικό, είτε με κάποιο γραφικό πρόγραμμα (π.χ. putty). Θα πρέπει να έχουμε δημιουργήσει τα απαραίτητα πιστοποιητικά και να τα έχουμε αντιγράψει στους κατάλληλους φακέλους για να μπορέσουμε να εκτελέσουμε τα επόμενα βήματα.

2. Δημιουργούμε τα παρακάτω δύο αρχεία :

```
#!/usr/bin/python
print "Hello Grid!"
```

hello.py

```
Executable = "hello.py";
StdOutput = "std.out";
StdError = "std.err";
InputSandbox = {"hello.py"};
```

```
OutputSandbox = {"std.out", "std.err"};
VirtualOrganisation = "see";
```

hello.jdl

Το πρώτο αρχείο είναι το εκτελέσιμο script. Το δεύτερο είναι ένα αρχείο jdl (Job Description Language). Η JDL είναι μία γλώσσα που χρησιμοποιείται για να περιγράψουμε την εφαρμογή μας στο middleware, το οποίο θα αποφασίσει τελικά που θα τρέξει η εφαρμογή βασιζόμενο στις πληροφορίες στο jdl αρχείο. Στο παραπάνω ορίσαμε το εκτελέσιμο αρχείο, τα αρχεία που θα γραφτεί η έξοδος και τα σφάλματα του προγράμματος, τα αρχεία που πρέπει να μεταφερθούν στον υπολογιστή ή στους υπολογιστές που θα τρέξει η εφαρμογή και τα αρχεία που θέλουμε να μεταφερθούν στο UI μετά την εκτέλεση. Η τελευταία παράμετρος δηλώνει σε ποιο εικονικό οργανισμό (VO) ανήκουμε.

3. Τρέχουμε την εντολή :

```
glite-wms-job-list-match -a hello.jdl
```

Η εντολή θα επιστρέψει όσα Computing Elements ταιριάζουν με τα κριτήρια της εφαρμογής μας και ανήκουν στο δικό μας VO. Στην δική μας περίπτωση το πιο σημαντικό είναι ότι θα ελέγξει το αρχείο jdl για λάθη.

4. Αν η προηγούμενη εντολή έτρεξε χωρίς να βρει κάποιο λάθος, υποβάλουμε το job (job submission), δηλαδή την προς εκτέλεση εφαρμογή, μαζί με όλα τα αρχεία που χρειάζεται για να τρέξει, με την παρακάτω εντολή :

```
glite-wms-job-submit -o jobID -a hello.jdl
```

Η εντολή θα επιστρέψει ένα ID για το συγκεκριμένο job και θα δημιουργήσει ένα αρχείο με το όνομα jobID, που μπορούμε να χρησιμοποιούμε για να παρακολουθούμε το job.

5. Τρέχουμε την εντολή :

```
glite-wms-job-status -i jobID
```

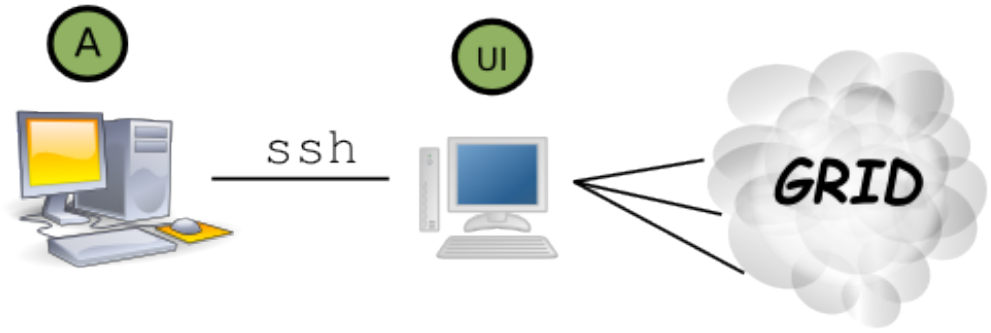
Όταν το αποτέλεσμα γίνει Done(Success) τότε μπορούμε να πάρουμε τα αποτελέσματα της εκτέλεσης της εφαρμογής μας με την εντολή :

```
glite-wms-job-output -i jobID --dir ./
```

Θα δημιουργηθεί ένας φάκελος μέσα στον τρέχοντα φάκελο με τα αρχεία std.out και std.err που δηλώσαμε στο jdl αρχείο. Το αρχείο std.out θα περιέχει το κείμενο "Hello Grid!" και αν δεν υπήρξαν σφάλματα το std.err θα είναι άδειο.

Τα παραπάνω είναι τα ελάχιστα που χρειάζεται να κάνουμε για να τρέξει η εφαρμογή μας. Η πολυπλοκότητά τους, σε σχέση με την εκτέλεση του ίδιου script στον υπολογιστή μας, δείχνει πως το GRID δεν φτιάχτηκε για να τρέχει το hello.py, παρ'όλη την χρησιμότητα του. Κατάλληλες εφαρμογές για το GRID είναι όσες χρειάζονται και μπορούν να τρέξουν σε περισσότερους από έναν υπολογιστές όπως στην περίπτωση αυτής της διπλωματικής. Υπάρχουν διάφορα γραφικά περιβάλλοντα και εργαλεία που κάνουν πιο φιλική την παραπάνω διαδικασία (JSMC, CoG Kit, Ganga κ.α.). Ένα τέτοιο εργαλείο δημιουργήθηκε και για τις ανάγκες τις παρούσας εργασίας.

Ένα σημείο που μπορεί να προκαλέσει σύγχυση στον νέο χρήστη του GRID, είναι ο ρόλος του κάθε υπολογιστή που συμμετέχει στην όλη διαδικασία. Στο Σχήμα 1.2 ο υπολογιστής A είναι ο προσωπικός υπολογιστής του χρήστη. Σε αυτόν τρέχει μόνο ο ssh client για την σύνδεση με το UI. Όλες οι εντολές στα παραπάνω βήματα τρέχουν στο UI. Μόλις τρέξει η : glite-wms-job-submit , το middleware θα ψάξει να βρει κάποιον κατάλληλο υπολογιστή στο GRID (από τους πόρους που διαχειρίζεται το δικό μας VO), και θα βάλει την εφαρμογή μας στην ουρά για εκτέλεση, μεταφορτώνοντας όσα αρχεία χρειάζεται από το UI στο Worker Node (τον υπολογιστή που θα τρέξει το πρόγραμμα). Όταν ολοκληρωθεί η εκτέλεση, με την εντολή : glite-wms-job-output , μεταφορτώνουμε τα αρχεία με τα αποτελέσματα στο UI. Από εκεί μπορούμε να τα διαβάσουμε, μέσω κάποιου text editor για τερματικό (π.χ. vi , nano) ή να τα αντιγράψουμε στον προσωπικό υπολογιστή (υπολογιστής A) μέσω της εντολής : scp



Σχήμα 1.2: Υπολογιστές και ρόλοι

1.3 Σύγκριση με παρεμφερείς τεχνολογίες

Το Cloud computing και το GRID computing είναι έννοιες σχετικά καινούργιες στον κλάδο της πληροφορικής. Πολλές φορές θεωρείται ότι αναφέρονται στο ίδιο πράγμα παρ'ότι αυτό δεν ισχύει.

Και οι δύο τεχνολογίες είναι δίκτυα που "κρύβουν" (abstract) την διαδικασία της επεξεργασίας από τον τελικό χρήστη και του παρουσιάζουν μία απλοποιημένη διεπαφή, την οποία να χειρίζεται εύκολα. Επίσης και στις δύο, τα δεδομένα μπορούν να βρίσκονται σε διαφορετικούς servers και η επεξεργασία τους να ολοκληρώνεται σε περισσότερους από ένα υπολογιστές.

Η βασική τους διαφορά είναι ότι το Cloud εστιάζει στην παροχή των πόρων, ενώ το GRID στον διαμοιρασμό αυτών (resource provisioning vs resource sharing). Το Cloud μπορεί να συνδυάσει υπηρεσίες, για να παρουσιάσει στον χρήστη ένα ομοιογενές βελτιστοποιημένο αποτέλεσμα. Ο κύριος στόχος στο GRID είναι η αποδοτική χρήση τις επεξεργαστικής δύναμης και η σημαντική μείωση του χρόνου που χρειάζεται η εκτέλεση μίας εργασίας. [9]

Η ζωή του World Wide Web ξεκίνησε το 1990 και σήμερα επιτρέπει την μεταφορά πληροφορίας σε όλα τα μέρη του κόσμου. Το GRID μπορεί να θεωρηθεί η εξέλιξη αυτής της τεχνολογίας, που επιτρέπει εκτός από την πληροφορία, τον διαμοιρασμό αποθηκευτικού χώρου και επεξεργαστικής ισχύος[10]. Στην πραγματικότητα το GRID συμπληρώνει τον παγκόσμιο ιστό, προσφέροντας δομημένες υπηρεσίες για χρήση από εφαρμογές αντί για ανθρώπους.

Τέλος σε σύγκριση με τα clusters που συναθροίζουν και αυτά υπολογιστικούς πόρους, το GRID διαφέρει ως προς την δυνατότητά του να συνα-

θροίζει πόρους που είναι διεσπαρμένοι τόσο διοικητικά όσο και γεωγραφικά [11].

1.4 Το GLUE Schema

Οι διαθέσιμοι πόροι του GRID είναι οργανωμένοι ιεραρχικά και οι πληροφορίες για αυτούς παρέχονται μέσα από τοπικά ευρετήρια (local index) και κεντρικά ευρετήρια (central index). Κάθε site έχει τουλάχιστον ένα τοπικό ευρετήριο που παρέχει πληροφορίες για το συγκεκριμένο site. Οι πληροφορίες είναι δύο ειδών:

- Στατικές πχ

  Τι τύπου/πόσοι επεξεργαστές υπάρχουν σε κάθε κόμβο

  Πόση μνήμη έχει κάθε κόμβος

  Τι τύπου λειτουργικό σύστημα υπάρχει στους κόμβους

  Τι είδους αποθηκευτικός χώρος υπάρχει (δίσκοι/βιβλιοθήκη ταινιών tape library)

- Δυναμικές

  Πόσοι επεξεργαστές είναι διαθέσιμοι;

  Πόσο μέγεθος του αποθηκευτικού χώρου είναι διαθέσιμο;

  Πόσες εργασίες εκτελούνται αυτή τη στιγμή στους κόμβους του site;

Όλη αυτή η πληροφορία είναι δομημένη. Δηλαδή, χρησιμοποιείται ένα σχήμα, ώστε να γίνεται εύκολη η αποθήκευση/παρουσίαση/αναζήτηση σε αυτή. Το σχήμα που χρησιμοποιείται ευρέως είναι το GLUE Schema. Υπάρχουν δύο υλοποιήσεις του σχήματος που μπορούν να χρησιμοποιηθούν. Μια σε XML και μια σε LDAP [12].

Στο επίπεδο του χρήστη του GRID σημαντικές είναι οι παράμετροι GLUE, που περνάνε ως φίλτρα στο αρχείο JDL. Όπως θα δούμε στο τέλος του τρίτου κεφαλαίου, μπορούμε να κάνουμε submit μία εργασία με συγκεκριμένες απαιτήσεις (Requirements), οι οποίες καθορίζονται μέσω παραμέτρων GLUE [13]. Για παράδειγμα για να επιβάλουμε στην εφαρμογή μας να τρέξει μόνο σε ένα συγκεκριμένο Computing Element (CE), μπορούμε να συμπεριλάβουμε στο jdl αρχείο, την γραμμή:

```
Requirements = other.GlueCEName == "myfavoriteCE";
```

όπου φυσικά το myfavoriteCE θα είναι το Computing Element της αρεσκείας μας ή την γραμμή:

```
Requirements = RegExp("cern.ch", other.GlueCEUniqueId);
```

η οποία θα περιορίσει την εκτέλεση της εφαρμογής μας, μόνο στα CEs που το όνομά τους έχει μέσα το "cern.ch". [14]

Άλλες παράμετροι που μπορούμε να ελέγξουμε είναι ο αριθμός των ελεύθερων CPUs, ο αριθμός των εργασιών που περιμένουν να εκτελεστούν, οι εγκατεστημένες εφαρμογές, αλλά και πολύ συγκεκριμένα στοιχεία όπως η IP, η συνδεσιμότητα η εγκατεστημένη RAM κ.α.

Εκτός από χρήσιμες στην διαχείριση ενός CE, αυτές οι πληροφορίες μας επιτρέπουν να "διαλέξουμε" το κατάλληλο περιβάλλον για την εκτέλεση της εφαρμογής μας.

Κεφάλαιο 2

VIDEO STREAMING (συνεχής ροή βίντεο)

Στο κεφάλαιο αυτό, θα γίνει μία σύντομη περιγραφή της τεχνολογίας μετάδοσης βίντεο μέσω δικτύου, για να γίνουν κατανοητές κάποιες επιλογές που έγιναν. στην ανάπτυξη της διπλωματικής αυτής.

2.1 Τρόποι μετάδοσης

Στις πρώτες μέρες των δικτύων, η ταχύτητες μετάδοσης των δεδομένων, επέτρεπαν στους χρήστες να ανταλλάσσουν κυρίως αρχεία κειμένου. Με την αύξηση της ταχύτητας, την διείσδυση των προσωπικών υπολογιστών στην αγορά και την δημιουργία του internet δημιουργήθηκαν νέες δυνατότητες και νέες απαιτήσεις. Εικόνες, ήχος και βίντεο είναι πλέον αναπόσπαστα κομμάτια της εμπειρίας χρήσης του internet. Η διαδικασία με την οποία βλέπουμε μία σελίδα είναι συνήθως η ακόλουθη. Ζητάμε από έναν υπολογιστή (server) μία σελίδα και ο server μας στέλνει την σελίδα και όλα της τα περιεχόμενα (κείμενο, εικόνες). Στην περίπτωση όμως του ήχου και του βίντεο η διαδικασία περιπλέκεται. Τα αρχεία είναι μεγάλα, αλλά για να δούμε ένα βίντεο ή να ακούσουμε ένα τραγούδι, δεν είναι ανάγκη να περιμένουμε, μέχρι να έχουμε ολόκληρο το αρχείο. Στην πραγματικότητα η αναπαραγωγή ξεκινάει, όταν έχουμε μεταφορτώσει μόλις ένα μέρος του αρχείου. Μέχρι όμως να δούμε ή να ακούσουμε αυτό το κομμάτι, ένα άλλο μέρος του αρχείου έχει φτάσει στον υπολογιστή μας.

Η μετάδοση περιεχομένου βίντεο γίνεται με δύο βασικούς τρόπους. Ζωντανά (live) ή κατ' απαίτηση (on demand). Ένα βίντεο που μεταδίδεται κατ' απαίτηση είναι συνήθως αποθηκευμένο, για μεγάλο χρονικό διάστημα σε κάποιον υπολογιστή (server). Οι χρήστες μπορούν να δουν το βίντεο

όποια στιγμή θέλουν και ανάλογα την εφαρμογή μπορούν να μεταπηδούν σε οποιοδήποτε σημείο του βίντεο ή να το σταματάνε και να το ξεκινούν σε όποιο σημείο θέλουν. Παραδείγματα μετάδοσης κατ'απαίτηση είναι οι υπηρεσίες των youtube, dailymotion, vimeo κ.α.

Ένα βίντεο που μεταδίδεται ζωντανά, φτάνει συνήθως στον χρήστη σε μικρό χρονικό διάστημα από την στιγμή που έγινε η σύλληψη (capture). Όλοι η χρήστες που παρακολουθούν το βίντεο, το βλέπουν με μικρή χρονική διαφορά μεταξύ τους. Παράδειγμα ζωντανής μετάδοσης είναι η κλασική τηλεόραση, το ραδιόφωνο, και τα διαδικτυακά αντίστοιχά τους. Στην ζωντανή μετάδοση, το βίντεο συνήθως δεν αποθηκεύεται σε κάποιον υπολογιστή, μετά την αρχική μετάδοσή του. Οι χρήστες δεν μπορούν να δουν κάποιο προηγούμενο στιγμιότυπο, ούτε και να σταματήσουν το βίντεο και να το συνεχίσουν από εκεί που το σταμάτησαν. Το τελευταίο, στην περίπτωση αναπαραγωγής σε υπολογιστή μπορεί να επιτευχθεί με προσωρινή αποθήκευση. Παραδείγματα ζωντανής μετάδοσης μέσω internet είναι οι υπηρεσίες των qik, skype, sopcast.

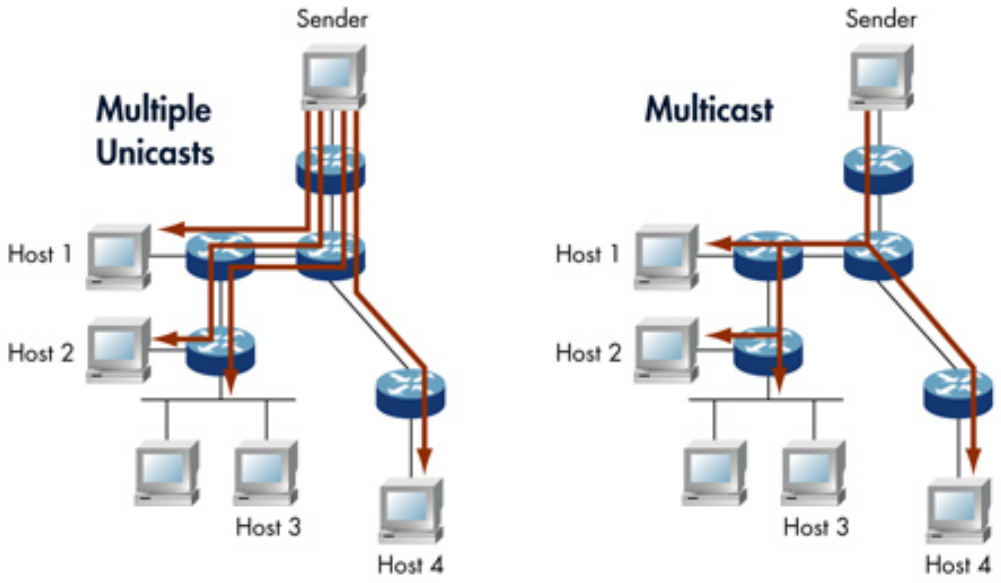
Οι δύο αυτοί τρόποι μετάδοσης, δημιουργούν διαφορετικές απαιτήσεις σε αποθηκευτικό χώρο και εύρος ζώνης (bandwidth), γι'αυτό και η τεχνολογία που χρησιμοποιείται είναι διαφορετική. Στην περίπτωση της μετάδοσης κατ'απαίτηση κάθε χρήστης βλέπει διαφορετικό stream έτσι χρειάζεται μία ξεχωριστή σύνδεση. Σε περιπτώσεις πολύ μεγάλης ζήτησης, όπως στην περίπτωση του youtube, η μετάδοση είναι αδύνατον να γίνει από ένα υπολογιστή. Έτσι η μετάδοση γίνεται από clusters και με χρήση πολλών διαφορετικών συνδέσεων στον ιστό.

Η διπλωματική αυτή, έχει θέμα την ζωντανή μετάδοση βίντεο, έτσι παρακάτω θα δούμε ποιες λύσεις υπάρχουν, για τον συγκεκριμένο τρόπο μετάδοσης.

2.1.1 multicasting

Στην ζωντανή μετάδοση, όλοι οι χρήστες βλέπουν το ίδιο βίντεο, την ίδια χρονική στιγμή. Στην περίπτωση μικρών δικτύων, των οποίων η αρχιτεκτονική μπορεί να επιλεχθεί, η λύση είναι το multicasting. Στο multicasting τα πακέτα των δεδομένων μεταδίδονται μία φορά αλλά καταλήγουν σε περισσότερους από ένα παραλήπτες (Σχήμα 2.1[1]). Για να επιτευχθεί αυτό πρέπει το πακέτο να φέρει κάποιες παραπάνω πληροφορίες (XCAST) ή οι routers να γνωρίζουν το σύνολο των παραληπτών μέσω routing tables. Σε μεγάλα και πολύπλοκα δίκτυα αυτό μπορεί να οδηγήσει σε μεγάλα headers στα πακέτα (στο header αποθηκεύεται η λίστα με τους παραλήπτες) ή σε

[1] http://www.net130.com/CMS/Files/Uploadimages/Multicast_Multi-vs-Unicast.jpg



Σχήμα 2.1: Multicast vs Unicast

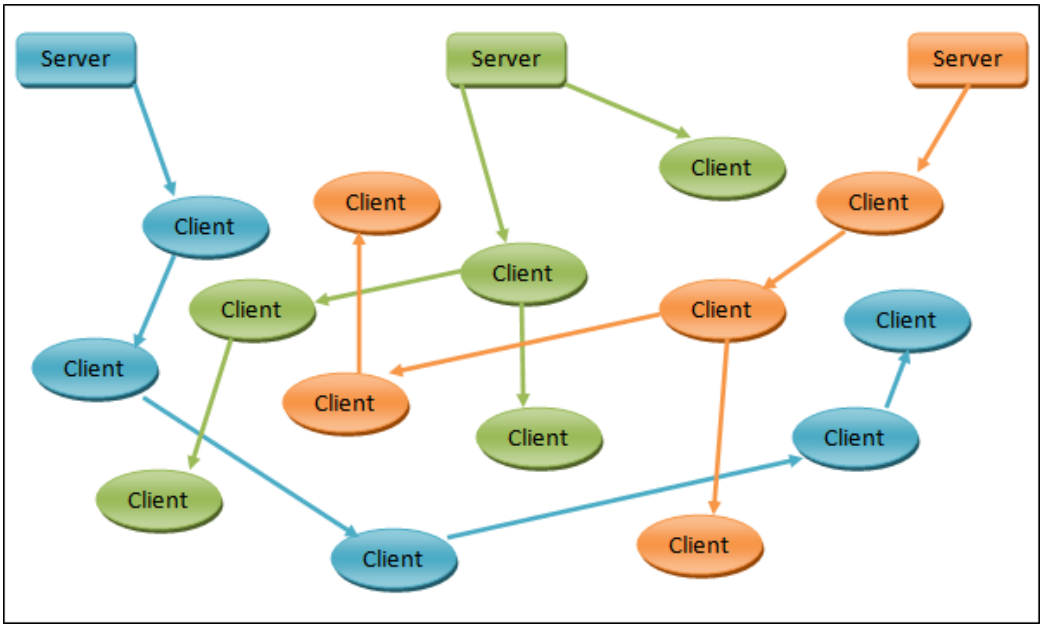
απαγορευτικές απαιτήσεις από τους routers. Επιπλέον, multicasting μπορεί να γίνει μόνο πάνω στο πρωτόκολλο UDP, γενονός που αποκλείει από τις πιθανές πηγές κάποιες εφαρμογές (video streaming servers) που μεταδίδουν βίντεο μέσω tcp (π.χ. QuickTime Streaming Server).

2.1.2 p2ptv

Μία λύση είναι η μετάδοση του βίντεο peer to peer. Βασίζεται στην ιδέα πίσω από το bittorrent πρωτόκολλο και συνοψίζεται στο εξής : Κάθε χρήστης που βλέπει το βίντεο (download), θα το στέλνει ταυτόχρονα σε τουλάχιστον ένα άλλο χρήστη (upload)(Σχήμα 2.2[2]). Έτσι, αν υποθέσουμε ότι όλοι οι χρήστες έχουν αρκετό εύρος ζώνης (bandwidth) για να κατεβάσουν και να ανεβάσουν το βίντεο ταυτόχρονα, μπορεί θεωρητικά να συνδεθεί ένας άπειρος αριθμός χρηστών.

Πολλές εταιρίες (κυρίως Κινέζικες), έχουν αναπτύξει εφαρμογές που υλοποιούν την παραπάνω ιδέα όπως οι TVUPlayer, PPLive, QQLive, PPStream και το sopcast, που επιτρέπει στους χρήστες να δημιουργούν τα δικά τους streams. Λύσεις ανοιχτού κώδικα υπάρχουν επίσης όπως οι :Goalbit, vidtorrent, streamtorrent και το stream2stream (συνδυασμός p2p και multicasting).

[2]http://geek.coolstreaming.us/wp-content/uploads/2011/03/P2ptv.png



Σχήμα 2.2: P2P μετάδοση

Το βασικότερο πρόβλημα της μετάδοσης μέσω p2p είναι ο αδυναμία ελέγχου της αναμετάδοσης. Κάθε χρήστης μπορεί να επέμβει στο βίντεο, εκούσια ή ακούσια. Μπορεί για παράδειγμα, να λαμβάνει το βίντεο και προτού το μεταδώσει να παρεμβάλει διαφημίσεις ή λόγω μειωμένης ταχύτητας ανεβάσματος, όλοι οι χρήστες που ακολουθούν, να δέχονται το βίντεο με χαμηλή ταχύτητα. Οι αναφερόμενες εταιρίες λύνουν το πρόβλημα της εκούσιας παρεμβολής, κάνοντας χρήση εφαρμογών κλειστού κώδικα, αλλά αυτό δεν μπορεί να αποτρέψει τον αποφασισμένο προγραμματιστή ούτε να λύσει το πρόβλημα της ταχύτητας. Το τελευταίο, μπορεί να αντιμετωπιστεί με "σπάσιμο" του αρχικού stream σε περισσότερα με μικρότερο ρυθμό μετάδοσης δεδομένων (bitrate), μετάδοσής τους από διαφορετικές πηγές και ανασύνθεσής τους στον τελικό χρήστη, για αναπαραγωγή του αρχικού βίντεο με το υψηλό bitrate. Μία υλοποίηση αυτής της λύσης είναι το gridcasting, της εταιρίας Octoshape.

Έτσι η λύση αυτή, παρ'ότι μπορεί να είναι ιδανική για την μετάδοση ενός τοπικού αγώνα μπάσκετ ή ακόμα και αναμετάδοσης του προγράμματος τηλεοπτικών καναλιών, δεν μπορεί να διασφαλίσει την ποιότητα που απαιτεί μία πληρωμένη υπηρεσία ή κάλυψη της αναμετάδοσης ενός επιστημονικού συνεδρίου χωρίς κάποια μέθοδο διασφάλισης ποιότητας. Οι διπλωματική αυτή, μπορεί ουσιαστικά να θεωρηθεί μία παραλλαγή της μετάδοσης p2p, με τον περιορισμό οι peers να είναι υπολογιστές στο grid

των οποίων τον έλεγχο έχει αποκλειστικά ο δημιουργός του stream. Έτσι διατηρούνται όλα τα χαρακτηριστικά της p2p μετάδοσης και παράλληλα αποτρέπεται η εκούσια παρεμβολή στο μεταδιδόμενο βίντεο. Επίσης η δυναμική του συστήματος δεν εξαρτάται από την συμμετοχή των χρηστών (οπως στην περίπτωση του p2p), αφού η δέσμευση πόρων από το grid μπορεί να γίνει ανεξάρτητα από τον αριθμό των χρηστών.

2.1.3 TCP vs UDP

Το TCP (Transmission Control Protocol) είναι το πιο συχνά χρησιμοποιούμενο πρωτόκολλο στο Internet. Ο λόγος είναι ότι προσφέρει αυτόματη διόρθωση σφαλμάτων. Εγγυάται δηλαδή ότι τα πακέτα δεδομένων θα φτάσουν όλα στον προορισμό τους, χωρίς αλλοιώσεις και στην σωστή σειρά. Για τον λόγο αυτό το TCP χρησιμοποιείται στην μετάδοση αρχείων, email, ssh και γενικά σε κάθε περίπτωση που είναι σημαντική η ακεραιότητα των μεταδιδόμενων δεδομένων.

Το UDP (User Datagram Protocol) είναι ένα άλλο πολύ διαδεδομένο πρωτόκολλο στο Internet. Σε αντίθεση με το TCP δεν χρησιμοποιείται για μετάδοση "σημαντικών" δεδομένων. Η κύρια εφαρμογή του είναι στην μετάδοση βίντεο και ήχου (άλλες εφαρμογές είναι tftp, DNS, online multiplayer games). Ο λόγος ύπαρξης του είναι η μεγαλύτερη ταχύτητά του σε σχέση με το TCP. Το UDP δεν παρέχει διόρθωση σφαλμάτων. Από την στιγμή που στέλνει ένα πακέτο δεν ασχολείται ξανά με αυτό (fire and forget). Πετυχαίνει έτσι μεγαλύτερη ταχύτητα μετάδοσης. Στην περίπτωση μετάδοσης βίντεο είναι συνήθως πιο σημαντικό να έρχονται τα δεδομένα γρήγορα, από το να έρχονται όλα σωστά, αφού μερικά αλλοιωμένα πακέτα δεν αλλάζουν δραματικά την εμπειρία του χρήστη.

Βέβαια σε περιπτώσεις που η μετάδοση γίνεται με μεγάλο αριθμό σφαλμάτων και εφ'όσον η ταχύτητα του δικτύου είναι επαρκής, η επιλογή του πρωτοκόλλου TCP θα μπορούσε να είναι καλύτερη επιλογή. Στον πίνακα 2.1.3 φαίνονται συνοπτικά οι διαφορές ανάμεσα στα δύο πρωτόκολλα.Στην διπλωματική αυτή δοκιμάστηκαν και τα δύο πρωτόκολλα. Τα αποτελέσματα της σύγκρισης αναπτύσσονται στο επόμενο κεφάλαιο.

TCP	UDP
Το TCP δημιουργεί σύνδεση μεταξύ των δύο άκρων. Ένα μήνυμα θα μεταδοθεί σωστά εκτός αν διακοπεί η σύνδεση. Σε αυτήν την περίπτωση το χαμένο μέρος θα μεταδοθεί ξανά. Δεν υπάρχει αλλοίωση κατά την μετάδοση των μηνυμάτων.	Στο UDP δεν υπάρχει σύνδεση (connectionless protocol). Όταν στέλνονται τα δεδομένα υπάρχει πιθανότητα να μην φτάσουν στον προορισμό τους ή να φτάσουν αλλοιωμένα.
Όταν σταλούν δύο μηνύματα το ένα μετά το άλλο, θα φτάσουν με την ίδια σειρά στον παραλήπτη.	Όταν σταλούν δύο μηνύματα δεν είναι σίγουρο με ποια σειρά θα φτάσουν στον παραλήπτη.
Όταν τα μέρη ενός stream φτάνουν με λάθος σειρά, πρέπει να σταλούν αιτήματα επαναποστολής και όλα τα κομμάτια πρέπει να μπουν στην σωστή σειρά. Απαιτείται λίγη δουλειά για να γίνουν όλα αυτά.	Δεν τηρείται σειρά στα πακέτα, ανοιχτές συνδέσεις κτλ. Στέλνεις και ξεχνάς! Έτσι είναι πιο γρήγορο και η κάρτα δικτύου και το λειτουργικό έχουν πολύ λίγη δουλειά να κάνουν για να πάρουν τα δεδομένα από τα πακέτα.
Τα δεδομένα μεταδίδονται σαν "stream" έτσι ώστε τίποτα να μην ξεχωρίζει το που τελειώνει ένα πακέτο και που αρχίζει το επόμενο. Κάθε εντολή read μπορεί να περιλαμβάνει περισσότερα από ένα πακέτα.	Τα πακέτα στέλνονται ξεχωριστά και αν φτάσουν θα φτάσουν ολόκληρα. Σε κάθε εντολή read διαβάζεται μόνο ένα πακέτο.

Πίνακας 2.1: Σύγκριση TCP - UDP

2.2 Προγράμματα μετάδοσης

Τα βίντεο που είναι αποθηκευμένα σε ένα υπολογιστή ή που αποθηκεύονται στην μνήμη μίας ψηφιακής κάμερας είναι συμπιεσμένα με κάποια μέθοδο συμπίεσης. Το πρόγραμμα που συμπιέζει και αποσυμπιέζει ένα βίντεο ονομάζεται codec (compressor-decompressor). Κάθε μέθοδος συμπίεσης (h.264/MPEG4 , MPEG2, Dirac, wmv) χρειάζεται έναν codec για την δημιουργία και αναπαραγωγή του βίντεο. Παραδείγματα codecs είναι οι theora,wax, Schrödinger. Όταν το βίντεο πρόκειται να μεταδοθεί μέσω διαδικτύου, χρειάζεται μία εφαρμογή να δημιουργήσει με βάση κάποιο πρωτόκολλο, τα πακέτα προς αποστολή (video streaming server). Ανάλογα το πρωτόκολλο που θα χρησιμοποιηθεί, τα πακέτα εκτός από το βίντεο θα περιέχουν πληροφορίες, όπως η διεύθυνση αποστολής.

Υπάρχουν διάφορα πρωτόκολλα για την μετάδοση βίντεο. Κάποια βασίζονται στο πρωτόκολλο TCP (π.χ. τα RTSPT,MMST,HTTP) και κάποια (συνήθως παραλλαγές των πρώτων) στο πρωτόκολλο UDP (π.χ. RTSPU, MMSU). Οι διάφορες πηγές μετάδοσης βίντεο, είτε πρόκειται για προγράμματα είτε για συσκευές χρησιμοποιούν συνήθως ένα από τα γνωστά πρωτόκολλα μετάδοσης ή ακόμα και κάποιο δικό τους πρωτόκολλο, του οποίου τα ακριβή χαρακτηριστικά μπορεί να μην είναι γνωστά. Είναι σημαντικό στην ανάπτυξη μίας λύσης για την μετάδοση πολυμέσων, να μην υπάρξει δέσμευση σε κάποιο συγκεκριμένο πρωτόκολλο ή πρόγραμμα, ώστε να είναι δυνατή η εφαρμογή σε πλήθος περιπτώσεων.

Από τα πολλά προγράμματα μετάδοσης βίντεο (streaming servers), στην συγκεκριμένη διπλωματική χρησιμοποιήθηκε ο VLC. Πρόκειται για ένα πρόγραμμα ανοιχτού κώδικα με δυνατότητα streaming, transcoding και αναπαραγωγής, σε πλήθος μορφών βίντεο και με διάφορα πρωτόκολλα μετάδοσης. Παρ'όλα αυτά η επιλογή αυτή δεν είναι δεσμευτική και μπορεί να χρησιμοποιηθεί οποιοσδήποτε συνδυασμός server και κατάλληλου προγράμματος αναπαραγωγής, με λίγες ή και καθόλου τροποποιήσεις. Έγινε προσπάθεια δημιουργίας μία εφαρμογής, που να κάνει την διαδικασία μετάδοσης του βίντεο, όσο δυνατόν πιο διάφανη για τον τελικό χρήστη. Στο παρακάτω κεφάλαιο εξηγούνται οι διάφορες επιλογές που έγιναν και αναλύεται ο κώδικας και η διαδικασία "εγκατάστασης" της εφαρμογής.

Κεφάλαιο 3

Η εφαρμογή

Στο κεφάλαιο αυτό θα δωθεί η τεκμηρίωση του κώδικα και οδηγίες για την εκτέλεσή του για μία επιτυχή αναμετάδοση από το GRID.

3.1 Δομή

Η εφαρμογή μας θα τρέξει σε περισσότερους από έναν υπολογιστές. Στο Σχήμα 3.1 φαίνονται οι υπολογιστές και μέσα τα πλαίσια τα αρχεία που πρέπει να έχει ο κάθε ένας. Η πηγή στέλνει στον distributor το αρχικό stream. Αυτός διαβάζει σε μία βάση (gvr.db), την λίστα με τους worker nodes που τρέχουν την εφαρμογή μας και τους στέλνει το stream. Η βάση μπορεί να βρίσκεται στον ίδιο ή σε άλλο υπολογιστή. Οι worker nodes λαμβάνουν το stream και το στέλνουν ο καθένας σε μία λίστα από χρήστες. Τέλος οι χρήστες βλέπουν το βίντεο μέσω μίας σελίδας που αναλαμβάνει να ενημερώσει την βάση για την ύπαρξή τους.

Στο Σχήμα υπάρχουν δύο ακόμα υπολογιστές. Ο ένας είναι ο υπολογιστής του διαχειριστή που ελέγχει πόσοι worker nodes τρέχουν την εφαρμογή σε κάθε στιγμή. Η εκκίνηση ενός νέου gvr (= grid video reflector) θα μπορούσε να γίνεται αυτόματα, με κριτήριο το πλήθος τον χρηστών που εξυπηρετεί ο κάθε gvr. Ο υπολογιστής UI, είναι το User Interface μέσω του οποίου γίνεται το job submission (βλέπε Κεφάλαιο 1). Ο ρόλος του κάθε αρχείου θα φανεί παρακάτω.

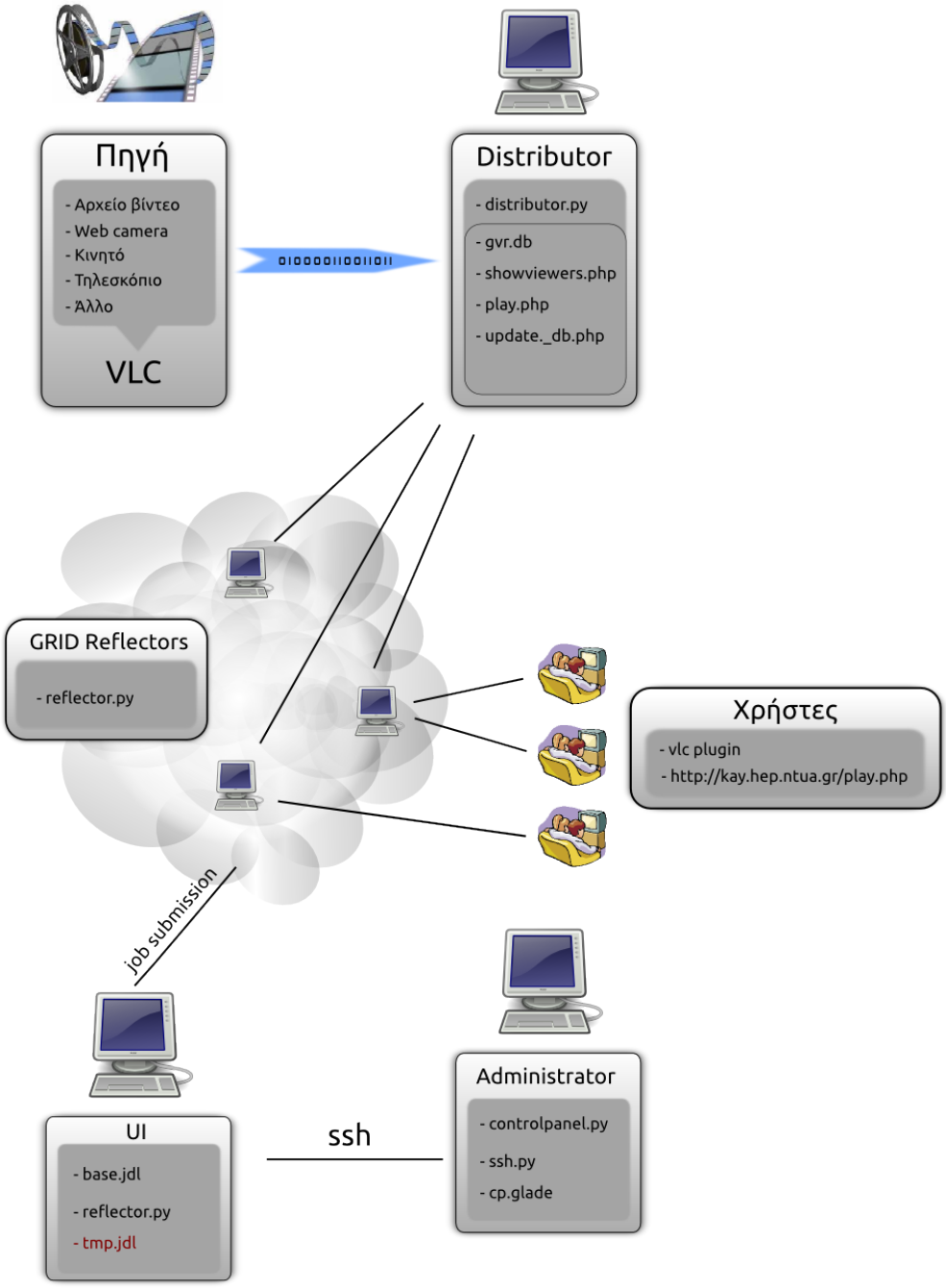
Για την ανάπτυξη του κώδικα επιλέχθηκε η γλώσσα προγραμματισμού python [1] . Είναι μία interpreted γλώσσα προγραμματισμού που τρέχει σε πολλά διαφορετικά περιβάλλοντα. Κάποια από τα χαρακτηριστικά της που ήταν καθοριστικά για την επιλογή της στην διπλωματική αυτή είναι :

- Είναι ελεύθερο λογισμικό.
- Έτοιμες βιβλιοθήκες για όλες τις ανάγκες τις διπλωματικής.
- Οργάνωση του κώδικα σε modules. Ευκολότερη επέκταση και επαναχρησιμοποίηση του κώδικα.
- Καθαρός κώδικας. Η σωστή στοίχιση του κώδικα είναι υποχρεωτική για την σωστή μεταγλώττιση.

Για όσα scripts τρέχουν στον web server χρησιμοποιήθηκε η γλώσσα php. Σαν βάση δεδομένων χρησιμοποιήθηκε η sqlite3 που υποστηρίζεται και από την python και από την php. Είναι πιο απλή σε σχέση με άλλες βάσεις δεδομένων αφού δεν χρειάζεται εγκατάσταση database server και δημιουργεί μόνο ένα αρχείο. Έτσι η μεταφορά της εφαρμογής από ένα υπολογιστή σε άλλο είναι ζήτημα μίας απλής αντιγραφής αρχείων.

Για το γραφικό περιβάλλον του διαχειριστή χρησιμοποιήθηκε η βιβλιοθήκη GTK μέσω του προγράμματος σχεδίασης περιβάλλοντος glade. Το τελικό αρχείο glade είναι ένα απλό αρχείο xml. Έτσι είναι δυνατή αλλαγή του GUI ή και η δημιουργία ενός τελείως διαφορετικού χωρίς να αλλάξει ο κώδικας της εφαρμογής.

[1] Τα scripts τρέχουν σε έκδοση 2.7 της python.



Σχήμα 3.1: Δομή

3.2 Ο κώδικας

Θα παρακολουθήσουμε την πορεία του stream μέσα από τα διάφορα scripts. Ο κώδικας ολοκληρωμένος δίνεται στο παράρτημα. Όλα τα σχόλια έχουν αφαιρεθεί για ευκολότερη ανάγνωση. Υπάρχουν όμως στα πηγαία αρχεία. Οι αριθμοί των γραμμών στις παρενθέσεις είναι αυτοί του παραρτήματος και όχι των πηγαίων αρχείων (διαφέρουν λόγω της αφαίρεσης των σχολίων). Όλες οι εντολές εκτελούνται στο τερματικό του linux.

3.2.1 Η πηγή

Θα χρησιμοποιήσουμε τον vlc για να δημιουργήσουμε το αρχικό stream. Θέλουμε να μεταδώσουμε μέσω udp. Η εντολή που δημιουργεί το stream είναι :

```
1 vlc seriousmovie.mpeg --loop --sout '#duplicate{dst=
    std{access=udp,mux=ts,dst=147.223.124.12:55100}}'
```

όπου 147.223.124.12 είναι η ip του υπολογιστή που τρέχει το distributor.py και seriousmovie.mpeg είναι το αρχείο βίντεο που θέλουμε να μεταδώσουμε.

3.2.2 distributor.py

Το αρχείο distributor.py πρέπει να ακούει στο port 55100 που του στέλνουμε το stream και να διαβάζει τα δεδομένα που έρχονται. Επίσης πρέπει να κρατάει μία λίστα με τους ενεργούς reflectors. Ως ενεργοί θεωρούνται οι reflectors που έχουν στείλει ένα μήνυμα αναφοράς όχι παλιότερο από την μεταβλητή TIME_LIMIT σε seconds. Πρέπει λοιπόν να υπάρχει μία διαδικασία (thread) που δέχεται συνεχώς τις αναφορές από τους reflectors. Τέλος πρέπει να στέλνει, μέσω ενός άλλου thread, τα δεδομένα που δέχεται στο port 55100, στους ενεργούς reflectors.

Έτσι στον κορμό του προγράμματος (149 - 174) δημιουργούνται τρεις διεργασίες (threads). Και οι τρεις εκτελούνται συνεχώς και ταυτόχρονα. Η report_receiver_thread εκτελεί την συνάρτηση report_receiver_func η οποία τρέχει συνεχώς μέχρι το event ev1 που αντιστοιχεί στο thread να γίνει set (167). Όμοια η get_reflectors_thread εκτελεί την get_reflectors_func και ελέγχεται από το ev2. Τέλος η stream_data_thread εκτελεί την συνάρτηση stream_data_func που εκτελείται, μέχρι το ev3 να γίνει set (169). Αφού ξεκινήσουμε τα τρία threads (158-160) ζητάμε από τον χρήστη να δώσει μία από τις συμβολοακολουθίες τερματισμού, που ορίζονται στην γραμμή 16,

για να τερματιστεί η εφαρμογή αφού πρώτα κλείσουμε τα ανοιχτά threads. Ας δούμε τι κάνουν μία μία οι συναρτήσεις.

Η report_receiver_func ανοίγει ένα socket τύπου SOCK_DGRAM (δηλαδή για πρωτόκολλο UDP) και το συνδέει στο port που ορίζεται από την μεταβλητή REPORT_PORT (14). Συνδέεται στην βάση της sqlite gvr.db (56). Στην συνέχεια διαβάζει συνεχώς μέσα σε μία select [8] όλες τις αναφορές που έρχονται στο report_socket. Οι αναφορές όπως θα δούμε στα αντίστοιχα scripts έρχονται πακεταρισμένες με χρήση της βιβλιοθήκης pickle. Η pickle είναι μία βιβλιοθήκη που κάνει data serialization, παίρνει δηλαδή οποιοδήποτε αντικείμενο της python και το κάνει ένα απλό string έτοιμο για μετάδοση ή αποθήκευση. Με την pickle.loads παίρνουμε από το string το αρχικό αντικείμενο μέσα σε ένα try - except block για τυχόν αλλοιωμένα πακέτα. Παρ'ότι η ασφάλεια της εφαρμογής για λόγους απλότητας δεν μας απασχόλησε σε μεγάλο βαθμό οφείλουμε να πούμε ότι κάποιες εναλλακτικές της pickle, όπως η json, θεωρούνται πιο ασφαλείς [7] αλλά δεν χρησιμοποιήθηκαν λόγω περιορισμένης υποστήριξης από προηγούμενες εκδόσεις της python και περιορισμού στον τύπο των αντικειμένων που δέχονται ως όρισμα.

Οι αναφορές που έρχονται στο REPORT_PORT είναι δύο τύπων. Ο πρώτος τύπος έρχεται από τους reflectors και είναι απλά μία λίστα με ένα στοιχείο : το ID του reflector που στέλνει το report (Περισσότερα για το ID στο script controlpanel.py). Ο άλλος τύπος είναι τα reports από το controlpanel.py και είναι μία λίστα με 2 στοιχεία. Το job_id της εργασίας που έγινε submit και το id του reflector που αντιστοιχεί στην διεργασία αυτή. Έτσι μπορεί να γίνεται αντιστοίχιση των jobs με τους reflectors. Η πληροφορία αυτή μεταδίδεται μόνο μία φορά, όταν γίνεται job submission από το controlpanel. Δεν γίνεται έλεγχος για το αν έφτασε στον distributor αφού αρκεί ένα report από τον reflector για να τον προσθέσουμε στην βάση. Η αντιστοίχιση job_id και id του reflector (my_id στην βάση) δεν είναι απαραίτητη.

Τα δεδομένα από τις αναφορές χρησιμοποιούνται είτε για να προσθέσουν ένα reflector στην βάση είτε για να ενημερώσουν την βάση με την ώρα τελευταίας αναφοράς του. Έτσι μπορούμε να ξέρουμε ποιοί reflectors είναι ακόμα ενεργοί. Το middleware παρέχει πληροφορίες για την κατάσταση της εκτέλεσης του job, αλλά στην πράξη η ενημέρωση που έχουμε από το gLite v3.1/v3.2 είναι και αργή και ανεπαρκής για τις ανάγκες της εφαρμογής μας.

Η βάση ενημερώνεται μέσω της συνάρτησης update_db η οποία δέχεται τρία ορίσματα. Το πρώτο είναι το report (μετά το unpickling). Το δεύτερο είναι η ip απο την οποία λάβαμε το report. Ελέγχουμε αν ένας reflector υπάρχει ήδη με βάση το id του έτσι η ip του απλά ενημερώνεται, για την περίπτωση που ένας reflector λάβει το id κάποιου παλιότερου και μη ενερ-

γού πλέον reflector. Αυτό μπορεί να συμβεί αφού τα ids δημιουργούνται με βάση τον αριθμό των jobs που υπάρχουν στο αρχείο jobID στο UI. Αν κάποια στιγμή σβήσουμε το αρχείο, τα ids θα ξεκινήσουν πάλι από το 1. Έτσι οι reflectors θα λαμβάνουν ids παλιότερων. Το τρίτο όρισμα της update_db είναι το αντικείμενο cursor της βάσης. Ουσιαστικά είναι η ανοιχτή σύνδεσή μας με την βάση. Κάθε thread πρέπει να δημιουργεί καινούργια σύνδεση. Οι cursors δεν μεταβιβάζονται από το ένα thread στο άλλο.

Η συνάρτηση get_reflectors_func συνδέεται στην βάση και ενημερώνει την λίστα reflectors (global μεταβλητή) με τις ip των reflectors που έχουν στείλει report νεώτερο από TIME_LIMIT seconds (Η TIME_LIMIT ορίζεται στην γραμμή 18). Έτσι έχουμε μία λίστα με τους reflectors που πρέπει να λάβουν το βίντεο. Την λιστα αυτή χρησιμοποιεί η stream_data_func. Ο λόγος που δεν ενημερώνουμε την λίστα reflectors με την report_receiver_func είναι ότι θέλουμε να κάνουμε την συχνότητα ενημέρωσης της λίστας ανεξάρτητη από τις αναφορές. Άλλωστε αν κάποια στιγμή δεν λαμβάνουμε για πολλή ώρα reports αυτό θα σήμαινε ότι πρέπει να αδειάσουμε την λίστα. Αυτό όμως δεν θα συνέβαινε από την report_receiver_func που θα περίμενε κάποιο report για να κάνει το οτιδήποτε.

Έτσι φτάνουμε στην stream_data_func. Η συνάρτηση αυτή δημιουργεί ένα καινούργιο socket τύπου datagram (UDP) και ακούει με αυτό στο port που ορίζεται από την μεταβλητή LISTEN_PORT (20). Επίσης δημιουργεί ένα socket για την αποστολή του βίντεο στους reflectors. Μετά την δημιουργία των sockets μπαίνει σε ένα while loop που τερματίζεται όταν το ev3 γίνει set())(169). Μέσα στο loop διαβάζει από το listen_sock όταν υπάρχουν δεδομένα (ο έλεγχος γίνεται με την συνάρτηση select) και τα στέλνει μέσω του talk_sock στην λίστα με τους ενεργούς reflectors, η οποία όπως είδαμε ενημερώνεται από άλλο thread. Η αποστολή γίνεται μέσω της συνάρτησης fullsend (132).

Η fullsend είναι ουσιαστικά μία while loop που τρέχει την sendto μέχρι να στείλει όλα τα δεδομένα στην IP που ορίζεται από την παράμετρο addr στο port REFLECTORS_LIST_PORT (21). Ο λόγος που χρειάζεται να γίνει η αποστολή με αυτόν τον τρόπο είναι ότι η sendto μπορεί να μην καταφέρει να στείλει όλα να δεδομένα με μία κλίση. Το αποτέλεσμα που επιστρέφει είναι το μέγεθος των δεδομένων που έστειλε σε bytes. Μία πιθανή αιτία που μπορεί να συμβεί αυτό, είναι όταν το buffer του socket είναι γεμάτο με δεδομένα από προηγούμενη κλίση τα οποία δεν έχουν αποσταλεί ακόμα (για non blocking sockets).

3.2.3 reflector.py

Το script αυτό τρέχει στους reflectors. Δηλαδή στους Worker Nodes του GRID. Θα πρέπει να τρέχει σε διαφορετικές εκδόσεις της python (version agnostic), να έχει όσο το δυνατόν λιγότερες απαιτήσεις από το σύστημα που τρέχει και να χρησιμοποιεί μόνο βασικά modules της python έτσι ώστε να τρέχει χωρίς πρόβλημα σε οποιοδήποτε Worker Node. Με βάση αυτό το σκεπτικό η δουλειά του reflector.py είναι σχεδόν η ίδια με του distributor.py . Θα ακούει σε ένα port για δεδομένα από τον distributor και θα τα στέλνει σε μία λίστα από χρήστες που του έχουν ανατεθεί. Έτσι χρειαζόμαστε μία συνάρτηση που θα ενημερώνει τον distributor ότι ο reflector ενεργός και έτοιμος να λάβει δεδομένα (reporting_func). Μία άλλη συνάρτηση θα διαβάζει από τον distributor την λίστα με τους χρήστες στους οποίους πρέπει να στείλει το βίντεο (get_viewers_func). Την αποστολή θα αναλάβει μία άλλη συνάρτηση (streaming_funct). Όπως και στο προηγούμενο script, κάθε συνάρτηση είναι ένα thread. Σε αυτό το script όλα τα threads τερματίζονται από ένα event και όχι από διαφορετικά. Το ίδιο μπορούσε να γίνει και στο προηγούμενο script αλλά εδώ υπάρχει ένας λόγος παραπάνω αφού μία συνάρτηση αναλαμβάνει να τερματίσει όλα τα threads όταν λάβει το μήνυμα "shut" στο SHUT_SOCK (15). Αυτό χρειάζεται γιατί στο WORKER NODE δεν μπορούμε να δώσουμε απ'ευθείας input για τον τερματισμό. Θα μπορούσε να χρησιμοποιηθεί η ακύρωση του job από το middleware αλλά αυτό δεν θα επέτρεπε να λάβουμε output από το ακυρωμένο job. Ας δούμε όμως κάθε συνάρτηση χωριστά.

Η reporting_func() είναι η συνάρτηση που εκτελείται από την διεργασία reporting_thread (111). Χρησιμοποιεί το report_sock που δημιουργείται στον κορμό του προγράμματος (100) για να στέλνει μία λίστα με το job_id του reflector στον distributor. Η διεύθυνση του distributor (SERVER_ADDRESS) ορίζεται στην γραμμή 23. Η λίστα "πακετάρεται" πρώτα με την pickle. Θα μπορούσαμε να στείλουμε το job_id χωρίς την pickle και χωρίς να το βάλουμε σε λίστα. Αλλά η επιλογή μας, θα επιτρέψει να στείλουμε περισσότερες πληροφορίες σε κάθε report, αν χρειαστεί στο μέλλον. Τα reports, στέλνονται σε χρονικό διάστημα που καθορίζεται από την REPORT_INTERVAL (60).

Η get_viewers_func γεμίζει την global μεταβλητή viewers_list (95) με τις IP των χρηστών στους οποίους πρέπει να στείλει το video. Για να διαβάσει την λίστα, δημιουργείται ένα instance της κλάσης FirefoxOpener() που ουσιαστικά είναι "παιδί" της κλάσης URLOpener της βιβλιοθήκης urllib. Στην γραμμή 28 ορίζεται η έκδοση του περιηγητή. Αυτό γίνεται γιατί κάποιοι servers ελέγχουν την έκδοση για να αποτρέπουν τα bots από το να ανοίγουν τις σελίδες τους. Βέβαια αυτό δεν συμβαίνει αν η σελίδα είναι σηκω-

μένη σε δικό μας server. Η σελίδα που επιστρέφει την λίστα με τους viewers είναι η showviewers.php στην διεύθυνση που ορίζεται από την μεταβλητή SRV_URL (12). Για να δούμε τους viewers που αντιστοιχούν στον δικό μας reflector περνάμε την GET μεταβλητή refl με τιμή το job_id του δικού μας reflector. Η σελίδα δίνει την λίστα σαν απλό κείμενο, χωρίζοντας την κάθε IP με μία καινούργια γραμμή. Αυτό δεν παράγει μία ιδιαίτερα εμφανίσιμη σελίδα, αλλά είναι αρκετό για να διαβάσει ο reflector τις IP. Σε περίπτωση που οι πληροφορίες που επέστρεφε η σελίδα ήταν πιο πολύπλοκες, λύσεις όπως το CORBA ή το πρωτόκολλο SOAP θα έπρεπε να εξεταστούν. Για λόγους απλότητας αυτό δεν έγινε στην ανάπτυξη της διπλωματικής αυτής.

Η συνάρτηση streaming_func χρησιμοποιεί μία select (66) για να διαβάσει από το listening_sock (101) τα δεδομένα που έρχονται στο LIST_PORT (14). Μόλις διαβάσει δεδομένα μεγέθους DATA_SIZE (17) το στέλνει σε κάθε ένα από τους viewers στην λίστα viewers_list. Η αποστολή γίνεται και πάλι μέσω της συνάρτησης fullsend (75). Η διαφορά σε αυτό το script είναι ότι το socket talking_sock που χρησιμοποιείται για την αποστολή δεν περνάει ως παράμετρος αλλά είναι global μεταβλητή (102).

Τέλος η συνάρτηση shutdown_func δέχεται μέσω του shut_sock (103) δεδομένα στο port που ορίζεται από την μεταβλητή SHUT_PORT (15). Όταν λάβει το μύνημα "shut" τερματίζει όλα τα threads και το κυρίως πρόγραμμα βγαίνει από το loop (121). Με την συνάρτηση αυτή μπορούμε να τερματίζουμε την εφαρμογή απλά στέλνοντας το μήνυμα shut στο κατάλληλο port του reflector.

3.2.4 Οι χρήστες

Το αρχικό video stream δημιουργήθηκε από τον vlc, όπως είδαμε στην παράγραφο 3.2.1. Η αναπαραγωγή μπορεί να γίνει από οποιοδήποτε πρόγραμμα υποστηρίζει αναπαραγωγή udp stream. Θα πρέπει όμως, με κάποιο τρόπο να ενημερωθεί ο distributor για κάθε καινούργιο viewer και να αναθέσει σε κάποιον reflector την μετάδοση του βίντεο. Αυτό επιτυγχάνεται με δύο scripts σε php που βρίσκονται στον distributor όπως θα δούμε παρακάτω. Το σύστημα του χρήστη πρέπει να έχει εγκατεστημένο στον browser, το plugin του vlc. Το plugin προς το παρόν τρέχει μόνο στον browser mozilla για linux. Εναλλακτικά θα μπορούσε να ενημερώνεται η βάση μέσω ενός script που θα αναλάμβανε να τρέξει και το πρόγραμμα αναπαραγωγής της επιλογής του χρήστη αλλά η λύση μέσω του plugin κάνει την όλη διαδικασία πιο διάφανη για τον τελικό χρήστη. Αν όμως ο χρήστης δεν το έχει εγκατεστημένο, μπορεί και πάλι να δει το βί-

ντεο. Θα πρέπει να αφήσει ανοιχτή την σελίδα που κανονικά θα έτρεχε το plugin και να ανοίξει τον vlc, δίνοντάς του για αναπαραγωγή το stream udp://@:55102 όπου το 55102 είναι το port που ορίζεται από την μεταβλητή REMOTE_PORT στο script reflector.py (16).

Επίσης ο υπολογιστής του χρήστη πρέπει να είναι απ'ευθείας συνδεδεμένος στο δίκτυο ή αλλιώς να έχει προωθήσει όλη την UDP κίνηση που έρχεται στο παραπάνω port στον δικό του υπολογιστή (μέσω NAT). Ο λόγος είναι ότι δεν χρησιμοποιούμε συνδεδεμένα sockets και τα πακέτα στέλνονται στον χρήστη μόνο με την IP του. Σε ένα τοπικό δίκτυο όμως πολλοί υπολογιστές μπορεί να μοιράζονται την ίδια εξωτερική IP. Η διαδικασία προώθησης των δεδομένων σε ένα υπολογιστή πίσω από ένα router μπορεί να αυτοματοποιηθεί μέσω κάποιου πρωτοκολλου NAT traversal όπως το SOCKS ή το UPnP. Αυτές οι λύσεις ξεφεύγουν από τους σκοπούς της διπλωματικής αυτής.

Έχοντας ικανοποιήσει τις παραπάνω απαιτήσεις ο χρήστης, για να δει το βίντεο ανοίγει την σελίδα play.php στον server που τρέχει ο distributor.

3.2.5 Distributor - showviewers.php

Όπως είδαμε οι reflectors δημιουργούν την λίστα με τους viewers στους οποίους πρέπει να στείλουν το βίντεο, διαβάζοντας τα δεδομένα που επιστρέφει η σελίδα showviewers.php . Η σελίδα αυτή, για λόγους απλότητας, θεωρούμε ότι βρίσκεται στον ίδιο φάκελο με το script distributor.py και το αρχείο της βάσης gvr.db . Ο φάκελος αυτός αποτελεί το document root του web server που τρέχει ο Distributor. Στην ανάπτυξη της διπλωματικής χρησιμοποιήθηκε ο server lighttpd. Είναι ελεύθερο λογισμικό, πολύ ελαφρύς και είναι ιδιαίτερα καλός στο να χειρίζεται πολλές ταυτόχρονες συνδέσεις. Ανάλογα τις ρυθμίσεις του web server ίσως δεν είναι καλή ιδέα τα scripts της python να βρίσκονται εντός του document root. Στην πραγματικότητα δεν υπάρχει λόγος να βρίσκονται ούτε στον ίδιο υπολογιστή αρκεί να έχουν πρόσβαση στην βάση (Γι'αυτό στο σχήμα 3.1 τα αρχεία της php βρίσκονται σε πλαίσιο). Ας δούμε όμως τι κάνει το script.

Όλος ο κώδικας βρίσκεται μέσα σε ένα block if - else που ελέγχει αν στο URL υπάρχει η μεταβλητή refl. Κάθε reflector βάζει στην μεταβλητή αυτή το δικό του id για να δει τους viewers που του έχουν ανατεθεί από το update_db.php όπως θα δούμε παρακάτω. Μέσα στην if γίνεται η σύνδεση στην βάση μέσω της επέκτασης PDO της PHP. Η PDO κάνει εύκολη την μετάβαση από μία βάση δεδομένων σε μία άλλη αν αυτό χρειαστεί (πχ από sqlite σε mysql) αλλάζοντας συνήθως μόνο μία γραμμή κώδικα (5). Στην συνέχεια δημιουργούμε την μεταβλητή $reflector από την GET μεταβλητή

refl. Κανονικά όταν διαβάζουμε μεταβλητές που μπορεί να δημιουργήσει ο χρήστης είναι καλό τα χρησιμοποιούμε συναρτήσεις που αφαιρούν ειδικούς χαρακτήρες (htmlspecialchars) για να αποφύγουμε injection attacks. Συνήθως οι προεπιλεγμένες ρυθμίσεις της php δυσκολεύουν αρκετά τα injections.

Στην συνέχεια ζητάμε από την βάση τις IP των viewers που έχουν ως reflector_id την τιμή της refl που διαβάσαμε από το URL και που το τελευταίο τους report (last_report) δεν είναι παλιότερο από την μεταβλητή $TIME_LIMIT (3). Έτσι οι viewers αφαιρούνται αυτόματα από την λίστα του reflector όταν δεν στείλουν report για κάποιο χρόνο. Τα reports των viewers τα αναλαμβάνει μία συνάρτηση javascript στην σελίδα play.php με χρήση ajax, όπως θα δούμε. Στην συνέχεια δίνεται ως απλό κείμενο η λίστα με τις IP χωρισμένες με χαρακτήρα καινούργιας γραμμής. Ο χαρακτήρας αυτός δεν φαίνεται όταν γίνεται render η σελίδα από κάποιο browser, αλλά με τον τρόπο που την διαβάζει ο reflector είναι αρκετός γιά να ξεχωρίσει τις IP.

3.2.6 Distributor - play.php

Όλα όσα βλέπει και ξέρει για την εφαρμογή μας ο τελικός χρήστης, είναι ουσιαστικά η σελίδα play.php . Αν εξαιρέσουμε τα κομμάτια html που είναι απλές πληροφορίες η λειτουργικότητα εξασφαλίζεται από δύο κομμάτια κώδικα μέσα στην σελίδα.

Από την γραμμή 68 ως την 74, καλείται το plugin του vlc να παίξει το udp stream που φτάνει το port 55102. Το port αυτό πρέπει να είναι το ίδιο με το REMOTE_PORT του reflector.py .

Από την γραμμή 4 ως την γραμμή 53 μέσα στο script tag είναι ο κώδικας της javascript που καλεί μέσω ajax την σελίδα update_db.php με interval που καθορίζεται στην γραμμή 51. Η συνάρτηση update() δουλεύει ως εξής. Δημιουργεί το αντικείμενο XMLHttpRequest μέσα σε try - catch block. Αυτό γίνεται γιατί ο τρόπος δημιουργίας του στους διάφορους browsers είναι διαφορετικός. Στην συνέχεια ορίζεται η συνάρτηση που θα τρέξει, όταν η σελίδα update_db.php που καλέσαμε (6) επιστρέψει την έξοδό της. Η έξοδος είναι σε μορφή xml και αποθηκεύεται στην μεταβλητή docx (36). Διαβάζουμε από αυτήν δύο τιμές. Την ip του reflector και το ID (37-38) και ενημερώνουμε τα δύο κατάλληλα spans στην σελίδα με αυτές τις τιμές (39-40). Οι γραμμές 42-44 εκτελούνται όταν το αποτέλεσμα της σελίδας δεν είναι XML. Αυτό μπορεί να συμβεί όταν επιστρέψει κάποιο μήνυμα λάθους. Η σελίδα update_db.php καλείται στις γραμμές 47-48.

3.2.7 Distributor - update_db.php

Το script αυτό καλείται από την javascript συνάρτηση του play.php. Όλη η λογική πίσω από την ανάθεση των viewers στους reflectors βρίσκεται σε αυτό το αρχείο. Όλος ο κώδικας βρίσκεται μέσα σε ένα try - catch block που προσπαθεί να συνδεθεί στην βάση (gvr.db). Η σύνδεση και σε αυτό το script γίνεται με χρήση του PDO extension. Εφ'όσον η σύνδεση επιτευχθεί (5) αποθηκεύουμε την IP του χρήστη στην μεταβλητή $ip (6). Στην γραμμή 8 ζητάμε από την βάση το id του reflector που έχει αναλάβει τον χρήστη. Το query επιστρέφει αποτέλεσμα μόνο αν έχει ξανατρέξει το script. Το αποτέλεσμα αποθηκεύεται στην μεταβλητή $reflector_id (12). Σε περίπτωση που ο viewer (χρήστης) δεν υπάρχει καθόλου στην βάση το query δεν θα επιστρέψει αποτέλεσμα και η $_reflector_id θα πάρει την τιμή -1 (10).

Αν ο χρήστης υπάρχει στην βάση και το query επέστρεψε αποτέλεσμα, τότε η συνθήκη στην γραμμή 14 γίνεται TRUE έτσι μπαίνουμε μέσα στο block του if. Πρέπει τώρα να ελέγξουμε αν ο reflector του χρήστη είναι "ζωντανός".

Αυτό συμβαίνει όταν το τελευταίο report του, δεν είναι παλιότερο από $TIME_LIMIT δευτερόλεπτα. Είναι λογικό ότι η τιμή αυτή πρέπει να είναι ίδια με την τιμή της TIME_LIMIT του script distributor.py . Όσες τιμές "μοιράζονται" μεταξύ των scripts, θα μπορούσαν να αποθηκεύονται στην βάση, για να εξασφαλίσουμε ότι θα είναι ίδιες σε όλα. Προτιμήθηκαν οι τοπικές μεταβλητές για μεγαλύτερη ευελιξία και πειραματισμό. Το query που ελέγχει αν ο reflector είναι ζωντανός, βρίσκει τον reflector του viewer με την IP του χρήστη και συγκρίνει τον χρόνο από το τελευταίο report με την $TIME_LIMIT (15-18). Αν βρει αποτέλεσμα (20) απλά ενημερώνει την τελευταία αναφορά του viewer με την ώρα εκείνης της στιγμής (21-22).

Αν ο reflector κριθεί "νεκρός" από την παραπάνω διαδικασία μπαίνουμε το else block στην γραμμή 24. Πρέπει να βρεθεί ένας "ζωντανός" reflector και να ενημερωθεί η βάση για τις αλλαγές. Καλύτερος reflector θεωρείται αυτός με τους λιγότερους viewers (έτσι επιτυχάνεται το balancing). Μπορεί να φανεί λογικό στον αναγνώστη, να εκτελείται μόνο ένα query που θα επιλέγει τον καλύτερο "ζωντανό" reflector και θα αναθέτει σε αυτόν το stream χωρίς να ελέγχει αν υπάρχει ήδη άλλος. Ένα από τα πολλά προβλήματα που δημιουργεί αυτή η λογική είναι ότι από την στιγμή που γράφεται ένας viewer στην βάση, μέχρι να ενημερωθεί ο reflector, μεσολαβεί κάποιος χρόνος. Έτσι συμφέρει τον viewer να μένει στον reflector που είναι ήδη, αν αυτός είναι ζωντανός, παρά να γραφτεί σε έναν άλλο και να περιμένει ξανά μέχρι ο νέος reflector να ενημερώσει την λίστα του. Άλλο πρόβλημα είναι ότι ο φόρτος των reflectors δεν θα ανταποκρίνεται στην πραγματικότητα αφού η λίστα τους, θα περιλαμβάνει viewers που τους "εγκατέλειψαν".

Η επιλογή του καλύτερου "ζωντανού" reflector γίνεται με δύο queries. Πρώτα βρίσκουμε όλους τους ζωντανούς (25-26). Αρχικοποιούμε την μεταβλητή $reflector_id στην τιμή -1 (28) (αφού ξέρουμε ότι ο παλιός είναι νεκρός). Για κάθε ένα από τα αποτελέσματα μετράμε τους viewers που έχουν το ID του reflector (30) (Για κατανόηση των queries, η δομή της βάσης παρουσιάζεται παρακάτω). Το αποτέλεσμα συγκρίνεται με την μεταβλητή $last_workload που αρχικοποιήθηκε στο μηδέν (27). Αν είναι μικρότερο ή ίσο ο reflector είναι ο καλύτερος μέχρι στιγμής και ενημερώνουμε την $reflector_id και την $last_workload με τις δικές του τιμές (37-38). Όταν ολοκληρωθεί το loop θα έχουμε σε αυτές τις μεταβλητές τον καλύτερο ζωντανό reflector. Σε περίπτωση "ισοπαλίας" προφανώς επιλέγεται ο τελευταίος σε σειρά καλύτερος.

Μετά το foreach loop (29) η $reflector_id δεν μπορεί να είναι -1 εκτός αν δεν υπάρχει κανένας ζωντανός reflector και το loop δεν έτρεξε ποτέ. Σε αυτήν την περίπτωση η $reflector_id μένει -1, τιμή η οποία επιστρέφεται τελικά και στην XML (76). Άν όμως δεν είναι -1 ενημερώνουμε την βάση με την ώρα του report του viewer (44) και τον καινούργιο reflector του.

Ο κώδικας από την γραμμή 50 μέχρι την 70 εκτελείται αν ο viewer δεν υπάρχει ήδη στην βάση. Στην περίπτωση αυτή βρίσκουμε, όπως στα προηγούμενα, τον καλύτερο ζωντανό reflector (51-65) και αν βρεθεί έστω και ένας εισάγουμε μία νέα καταχώρηση στην βάση για τον viewer (68-69). Αν δεν βρεθεί κανένας ζωντανός reflector η $reflector_id συνεχίζει να έχει την τιμή -1 από την γραμμή 10.

Από την γραμμή 72 ως την γραμμή 77 δημιουργείται η έξοδος του script σε XML. Ουσιαστικά είναι δύο τιμές. Η ip του reflector και το ID του. Οι πληροφορίες αυτές είναι απαραίτητες στο play.php και περνάνε απλά ως πληροφορία. Η ουσιαστική δουλειά του update_db.php είναι η ενημέρωση της βάσης με την ώρα του τελευταίου report του viewer και το δίκαιο μοιρασμα των viewers στους reflectors. Στην γραμμή 84 γίνεται η αποσύνδεση από την βάση.

3.2.8 Distributor - gvr.db

Η βάση αποτελείται από δύο πίνακες. Στον πίνακα reflectors αποθηκεύονται οι παρακάτω τιμές.

Το ID του reflector (my_id). Αυτό περνάει ως παράμετρος μέσω του jdl αρχείου όπως θα δούμε παρακάτω και είναι μοναδικό για κάθε reflector. Ο λόγος που δεν χρησιμοποιούμε το job id για τα reports είναι ότι αυτό δεν μπορεί να το γνωρίζει το script αφού μας επιστρέφεται μετά το job submission. Αντί λοιπόν να στείλουμε αυτήν την πληροφορία στον reflector,

του περνάμε ως παράμετρο ένα ID. Μόλις γίνει το submission του job, αποθηκεύουμε και το job id για μπορούμε να συνδέουμε τα reports από του reflectors με τα jobs. Η σύνδεση αυτή δεν χρειάστηκε στο επίπεδο της διπλωματικής αυτής, γι'αυτό και δεν έγινε προσπάθεια να αποθηκευτεί οπωσδήποτε αυτή η πληροφορία. Υπάρχει για πιθανή μελλοντική χρήση. Τέλος στον πίνακα αποθηκεύουμε την ip του reflector (την οποία παίρνουμε κατ'ευθείαν από τα πακέτα των reports, μέσω της συνάρτησης report_receiver_funt του disrtibutor.py) και την ώρα του τελευταίου report.

Στον πίνακα viewers αποθηκεύονται το id του viewer, το οποίο αυξάνεται αυτόματα με κάθε καταχώρηση, το id του reflector που του στέλνει το stream, η ip του και η ώρα της τελευταίας αναφοράς του.

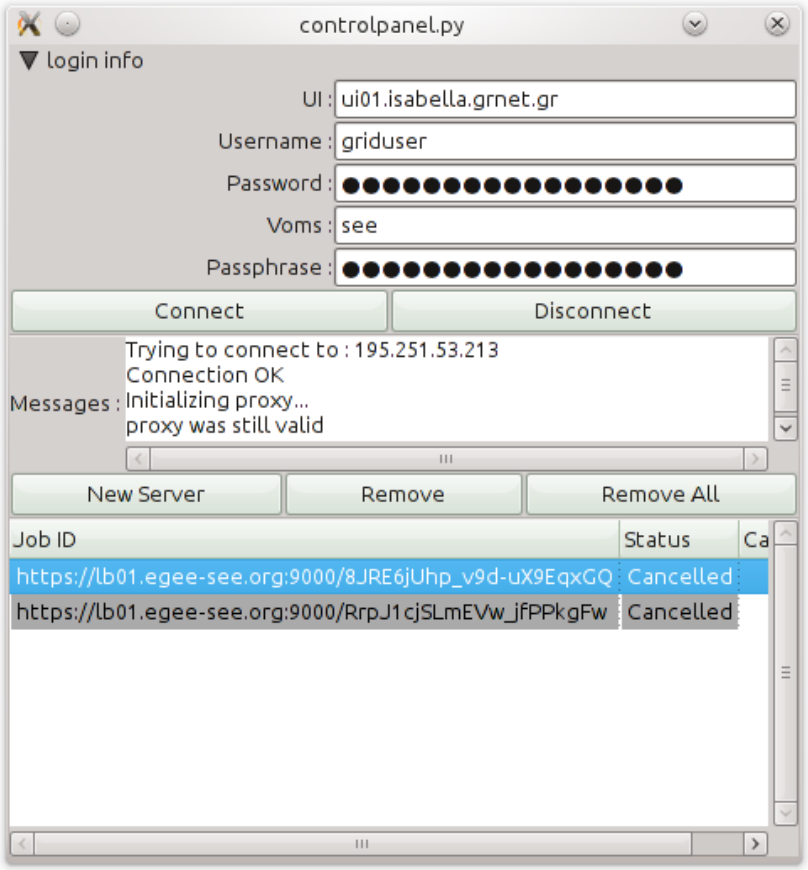
Η παραπάνω δομή φαίνεται στο παράρτημα, στο script db_create.sql που δημιουργεί την βάση.

3.2.9 Administrator - controlpanel.py

Είδαμε τα scripts που σχετίζονται με την μετάδοση του βίντεο. Ίσως έγινε φανερό ήδη ότι δεν είναι δυνατόν να σηκώνουμε τους reflectors με την παραδοσιακή μέθοδο που περιγράφεται στο πρώτο κεφάλαιο. Πρέπει να υπολογίζουμε κάθε φορά το id του reflector και να δημιουργούμε το αρχείο jdl που θα έχει ως παράμετρο αυτό το id. Είναι άλλωστε πιο εύκολο για τον μέσο χρήστη να χρησιμοποιεί ένα γραφικό περιβάλλον αντί για την κονσόλα. Για τις ανάγκες της διπλωματικής αυτής δημιουργήθηκε ένα πρόγραμμα διαχείρησης σε python. Για το γραφικό περιβάλλον χρησιμοποιήθηκε η βιβλιοθήκη GTK. Είναι έτσι δυνατόν να τρέξει η εφαρμογή σε όλα τα δημοφιλή λειτουργικά αν και δοκιμάστηκε μόνο σε linux κατά την ανάπτυξη της. Η εφαρμογή περιορίστηκε στο να κάνει μόνο τα απαραίτητα για τις ανάγκες τις διπλωματικής. Αν ήταν απαραίτητα περισσότερα χαρακτηριστικά θα έπρεπε να εξεταστεί η χρήση κάποιου CoG Kit (CoG = Commodity Grid) όπως τα pyGlobus, Java CoG Kit ή οι εφαρμογές Ganga και P-GRADE.

Η εφαρμογή είναι ουσιαστικά ένα γραφικό περιβάλλον για εκτέλεση εντολών στο UI, στο οποίο θεωρείται ότι έχει πρόσβαση ο administrator και από το οποίο μπορεί να κάνει επιτυχημένα job submissions. Για την σύνδεση στο UI δημιουργήθηκε ένα module (ssh.py) που συνδέεται μέσω ssh με χρήση του module paramiko.

Ο σχεδιασμός του γραφικού περιβάλλοντος έγινε με την εφαρμογή glade. Το τελικό αρχείο είναι απλά ένα αρχείο XML, του οποίου τα αντικείμενα συνδέονται σε callback συναρτήσεις στο πρόγραμμά μας. Έτσι η μορφή του γραφικού περιβάλλοντος μπορεί να αλλάξει εύκολα, χωρίς επέμβαση



Σχήμα 3.2: controlpanel

στον κώδικα. Η μορφή του προγράμματος φαίνεται στην εικόνα 3.2. Ας δούμε όμως πως δημιουργείται το γραφικό περιβάλλον και πως παίρνει ζωή μέσα από την python.

Στην αρχή του script (3-20) γίνονται import όλα τα απαραίτητα modules. Το module ssh είναι δικό μας και θα το εξηγήσουμε παρακάτω.

Η JOB_FILE (22), καθορίζει το όνομα του αρχείου στο οποίο αποθηκεύονται τα job ids στο UI και χρησιμοποιείται στο job submission αλλά και για την δημιουργία μοναδικών ids για τους reflectors.

Η UPDATE_INTERVAL, καθορίζει κάθε πότε να ενημερώνεται η λιστα με τα jobs (το κουτί στο κάτω μέρος του παραθύρου) σε milliseconds. Η τιμή της, θα πρέπει να είναι μεγαλύτερη από ένα λεπτό αφού δεν προσφέρει κάποια πληροφορία που απαιτεί άμεση αντίδραση και δεν πρέπει να ξεχνάμε ότι ουσιαστικά τρέχει εντολές σε ένα κοινόχρηστο UI.

Η SRV_URL καθορίζει το URL του distributor, για να ξέρει η κατάλληλη

συνάρτηση που να στείλει τα reports.

Η SRV_PORT καθορίζει το port στο οποίο δέχεται τα reports το script distributor.py και πρέπει προφανώς να είναι ίδια με την REPORT_PORT αυτού του script.

Η κλάση Gui αποτελεί την εφαρμογή μας με όλες τις συναρτήσεις callback και όλη την λειτουργικότητα. Στον κύριο κορμό του προγράμματος (200-201) δημιουργείται το instance της κλάσης και ξεκινάει το main loop της εφαρμογής.

Η συνάρτηση say (32-35) χρησιμοποιείται για να στέλνει κείμενο τόσο στην κονσόλα (33), όσο και στο gui (34-35). Χρησιμοποιείται αντί της απλής print, από όλες τις συναρτήσεις που πρέπει να πληροφορήσουν τον χρήστη για κάποιο γενονός. Το κείμενο γράφεται στο παράθυρο με την ετικέτα Messages.

Η συνάρτηση updater (37-48) χρησιμοποιεί το αντικείμενο myui (59) για να διαβάσει από το middleware την κατάσταση των submitted jobs. Η μέθοδος που το κάνει αυτό είναι η getjobs του module ssh που θα δούμε παρακάτω. Με βάση τα δεδομένα που επιστρέφει η getjobs (39) χτίζουμε την λίστα servers_list που φαίνεται στο πλαίσιο με τα jobs (46). Η συνάρτηση αυτή καλείται από τις συναρτήσεις, όταν συμβεί κάποιο γεγονός που αλλάζει την λίστα, αλλά και αυτόματα από τον timer στην γραμμή 193.

Η getHostFromGui (50-55) διαβάζει από το drop down πλαίσιο "login info" τα στοιχεία σύνδεσης του χρήστη στο UI και τα αποθηκεύει σε μεταβλητές. Όπως φαίνεται και στο Σχήμα 3.2 τα στοιχεία αυτά είναι το url του UI, το username και το password του χρήστη για το login στον λογαριασμό του.

Η connect_to_ui (57-74) χρησιμοποιεί το αποτέλεσμα της προηγούμενης συνάρτησης για να συνδεθεί το UI και να κάνει το proxy initialization (69). Για το proxy χρησιμοποιεί τα πεδία Voms και Passphrase εφ'όσον δεν είναι κενά.

Η sendreport (76-78) στέλνει την μεταβλητή data (ότι και αν είναι αυτή) στην διεύθυνση και το port που καθορίζονται από την server_address (27). Το socket για την αποστολή περνάει ως παράμετρος στην συνάρτηση.

Η newServer (81-90) συνδέεται με το κλικ του κουμπιού New Server (137) και σηκώνει ένα νέο reflector κάνοντας job submission. Το jdl που χρησιμοποιείται είναι το tmp.jdl το οποίο δημιουργείται από την updatejdl του module ssh. Η διαφορά του με το base.jdl είναι η μία γραμμή, που περνά ως παράμετρο το reflector id. Το id δημιουργείται από τον αριθμό των jobs που έχουν γίνει submit συν ένα (83). Μόνο τα jobs που βρίσκονται στο αρχείο JOB_FILE καταμετρώνται. Αν το αρχείο σβηστεί, η μέτρηση αρχίζει από το ένα για τον επόμενο reflector. Αυτό μπορεί να δώσει σε reflectors, id παλιότερων. Το αντιμετωπίσαμε ήδη αυτό το θέμα παραπάνω.

Η removeServer (92-100) κάνει ακύρωση του επιλεγμένου job. Αυτός είναι ο ένας τρόπος για να κλείσει ένας reflector και ο πιο σίγουρος. Αν μας ενδιαφέρει το output του reflector.py μπορούμε να κλείσουμε το script πιο "καθαρά" στέλνοντας το μήνυμα shut στο SHUT_PORT του reflector.py . Είδαμε παραπάνω πως ακριβώς δουλεύει αυτός ο τρόπος. Υλοποιήθηκε η μέθοδος με το job cancellation, γιατί κατά την ανάπτυξη της διπλωματικής, δεν είχαν όλες οι εκδόσεις του reflector.py την δυνατότητα απομακρυσμένου τερματισμού (μέσω απόστολής μηνύματος στο SHUT_PORT). Η δυνατότητα αυτή προστέθηκε για λόγους debugging (γιατί μας επιτρέπει να λάβουμε το αρχείο εξόδου του σταματημένου reflector).

Η disconnect συνδέεται με το κλικ του κουμπιού Disconnect και κλείνει την σύνδεση με το UI.

Με το ίδιο τρόπο η destroy (107-111) τρέχει όταν κλείσει το κεντρικό παράθυρο της εφαρμογής (132) και αφού αποσυνδεθεί από το UI (110) κλείνει το main loop της εφαρμογής (111).

Η remove_all (113-114) πρέπει να συνδεθεί με το κλικ του κουμπιού "Remove All" και να κάνει cancel όλα τα ανοιχτά jobs. Δεν υλοποιήθηκε γιατί δεν χρειάστηκε για τις ανάγκες τις διπλωματικής αυτής. Το κουμπί και η συνάρτηση υπάρχουν για μελλοντική υλοποίηση.

Η default_me χρησιμοποιήθηκε για να γεμίζουν αυτόματα τα πεδία του κουτιού login info. Δεν είναι καλή ιδέα να αποθηκεύονται τα στοιχεία του χρήστη σε απλό αρχείο χωρίς κρυπτογράφηση. Η συνάρτηση αυτή χρησιμοποιήθηκε μόνο για να διευκολύνει την ανάπτυξη και να μην απαιτείται να εισάγονται συνεχώς τα ίδια στοιχεία κατά τις δοκιμές.

Η _init_ τρέχει κατά την δημιουργία του instance της κλάσης Gui (200). Εισάγεται το αρχείο της διεπαφής του χρήστη (127-128). Όπως ήδη αναφέραμε το αρχείο δημιουργήθηκε με το πρόγραμμα glade και είναι στην ουσία ένα αρχείο xml. Για να διαβάσουμε ένα αντικείμενο από το αρχείο (για παράδειγμα ένα κουμπί ή το κεντρικό παράθυρο της εφαρμογής) χρησιμοποιούμε την εντολή wTree.get_widget() (130,141,142). Στην γραμμή 142 συνδέουμε το κλείσιμο του κεντρικού παραθύρου με την συνάρτηση destroy. Άλλος τρόπος για την σύνδεση των signals με τις callback συναρτήσεις είναι με χρήση ενός dictionary της python και την εντολή signal_autoconnect (137-139). Στην 135 ορίζουμε το μέγεθος του παραθύρου. Στις γραμμές 141 ως 152 δημιουργούμε το παράθυρο με τα μηνύματα και κρύβουμε το περιεχόμενο των πεδίων Password και Passphrase (141-142).

Στις γραμμές 156 ως 189 δημιουργείται η λίστα με τα jobs που φαίνεται στο κάτω κουτί της εφαρμογής. Χωρίς να μπούμε σε λεπτομέρειες θα πούμε ότι το αντικείμενο Treeview της βιβλιοθήκης GTK, υλοποιεί το μοντέλο MVC (Model-View-Controller). Το Model μας είναι το αντικείμενο servers_list της κλάσης ListStore (157), το View είναι το treeview,οι στήλες

και τα κελιά (158,160,165). Ο Controller είναι οι συναρτήσεις που καθορίζουν πως τα δεδομένα στο Model παρουσιάζονται στο View. Η λίστα έχει τρείς στήλες. Η πρώτη είναι το Job ID (165-168). Η δεύτερη είναι η κατάσταση του job (170-174). Η τελευταία περιλαμβάνει ένα checkbox για κάθε job και έμεινε για πιθανή μελλοντική χρήσης της.

Στην γραμμή 191 εμφανίζονται όλα τα controls. Στην 193 δημιουργείται ο timer που τρέχει την updater όπως είδαμε και παραπάνω. Τέλος στην γραμμή 196 δημιουργείται το socket τύπου DATAGRAM (UDP)για την αποστολή των reports. Το socket αυτό περνάει ως παράμετρος στην συνάρτηση sendreport (πχ από την newServer στην γραμμή 86).

3.2.10 Administrator - ssh.py

Το script ssh.py χρησιμοποιείται ως module από το controlpanel.py και αποτελείται από δύο κλάσεις. Η μία είναι η Jobobg που χρησιμοποιείται από την δεύτερη για να αποθηκεύεται κάθε job και την κατάστασή του ως αντικείμενο. Η δεύτερη είναι η MyUi και παρέχει μεθόδους για την εκτέλεση εντολών στο UI. Η MyUi κληρονομεί όλες τις ιδιότητες της paramiko.SSHClient (14). Ας δούμε και εδώ τις συναρτήσεις μία μία.

Η __init__ εκτελείται όταν δημιουργείται το αντικείμενο. Δέχεται μία παράμετρο την host_info (η self περνάει αυτόματα από την python). Η παράμετρος αυτή είναι ένα dictionary της python της μορφής :

```
{"host":"host","un":"username","passwrd":"password"}
```

Τα δεδομένα του dictionary αποθηκεύονται σε μεταβλητές της κλάσης (17-20). Στις γραμμές 22 ως 25 βρίσκουμε από το url του UI την ip του και την αποθηκεύουμε στην ui_ip. Η γραμμή 27 χρειάζεται για να δέχεται αυτόματα το key του UI όταν δεν είναι στην τοπική λίστα με τα επιτρεπόμενα κλειδιά.

Η συνάρτηση connect_me πραγματοποιεί την σύνδεση με το UI με χρήση της συνάρτησης connect της κλάσης SSHClient (31).

Η proxy_valid εκτελεί την εντολή voms-proxy-info στο UI για να δει αν υπάρχει ενεργό proxy. Στις γραμμές 47 - 48 χρησιμοποιούμε μία regular expression για να βρούμε το string "0:00:00" που επιστρέφεται όταν το proxy έχει λήξει. Κάποιες φορές, το output σε περίπτωση που δεν υπάρχει ενεργό proxy είναι διαφορετικό και η συνάρτηση θα αποτύχει. Σε αυτήν την περίπτωση, θα πρέπει να δημιουργήσουμε το proxy μόνοι μας από το UI, προτού να μπορέσουμε να χρησιμοποιήσουμε το script ή εναλλακτικά

η συνάρτηση μπορεί να επεκταθεί για να "καταλαβαίνει" όλα τα πιθανά outputs της εντολής voms-proxy-info.

Η init_proxy ελέγχει με της προηγούμενη συνάρτηση αν υπάρχει ενεργό proxy (54). Αν δεν βρει, το δημιουργεί με χρήση της εντολής:

```
voms-proxy-init -voms see
```

όπου see το VO στο οποίο ανήκει ο χρήστης (administrator). Στην γραμμή 58 περνάμε το passphrase για την δημιουργία του proxy.

Η συνάρτηση update_jdl αντιγράφει το αρχείο base.jdl στο tmp.jdl και προσθέτει στο τελευταίο την γραμμή :

```
Arguments = 134
```

όπου 134 είναι η τιμή της παραμέτρου arg. Η παράμετρος αυτή περνάει από το controlpanel.py και είναι το id του reflector. Το jdl που χρησιμοποιείται για το job submission είναι το tmp.jdl. Το base.jdl χρειάζεται για να μην φτιάχνουμε όλο το αρχείο από τον κώδικα, αφού αυτό που αλλάζει κάθε φορά είναι μόνο η γραμμή με τα Arguments.

Η submit_job τρέχει την εντολή :

```
glite-wms-job-submit -o jobID -a  tmp.jdl
```

όπου jobID είναι το αρχείο στο οποίο αποθηκεύονται τα ids των jobs και το tmp.jdl είναι το αρχείο που φτιάξαμε παραπάνω. Και τα δύο αρχεία περνάνε ως παράμετροι στην συνάρτηση. Στις γραμμές 97-99 εξάγεται από το ouput της εντολής το id του job. Όπως είδαμε αυτό θα χρησιμοποιηθεί από το controlpanel.py για το report, που θα συνδέσει το job id με το id του reflector.

Η συνάρτηση getjobs είναι αυτή που χρησιμοποιεί η συνάρτηση update του controlpanel, για να διαβάσει την λίστα με τα jobs και την κατάσταση του καθενός. Η εντολή του UI είναι η :

```
glite-wms-job-status -i jobID
```

όπου jobID το αρχείο που χρησιμοποιήσαμε και στην πιο πάνω συνάρτηση. Στις γραμμές 111 ως 117 εξάγεται από το ouput της εντολής το job id και η κατάσταση του job και αποθηκεύνται σε ένα instance της κλάσης

Jobobg. Η συνάρτηση επιστρέφει μία λίστα με τέτοια αντικείμενα ένα για κάθε job.

Η συνάρτηση printjobs είναι ακριβώς ίδια με την getjobs με την διαφορά ότι τυπώνει την λίστα με τα jobs και την κατάστασή τους (139-140). Η συνάρτηση χρησιμοποιείται όταν το ssh.py δεν τρέχει ως module αλλά ως εφαρμογή (178-195) ή σε κάθε περίπτωση που αυτό κρίνεται χρήσιμο.

Η cancel_job εκτελείται από το controlpanel.py όταν ακυρώνετια ένα job. Η εντολή του UI είναι η :

```
glite-wms-job-cancel job.id
```

όπου job_id το id του job που θέλουμε να ακυρώσουμε[1].

Η job_file_delete ακυρώνει όλα τα jobs στο αρχείο job_file που περνάει ως παράμετρος και σβήνει το αρχείο αυτό. Δεν χρησιμοποιείται από το controlpanel.py αλλά θα ήταν χρήσιμη σε μελλοντική υλοποίηση της λειτουργίας "Remove All" όπως είδαμε παραπάνω. Ουσιαστικά καθαρίζει την λίστα όταν έχει γεμίσει σταματημένα jobs ή όταν χρειάζεται να κλείσουμε όλους τους reflector με μία εντολή.

Η γραμμές γραμμές 178 ως 195 εκτελούνται μόνο όταν ισχύει η συνθήκη __name__ == "__main__" το οποίο συμβαίνει όταν το script εκτελείται μόνο του και όχι ως module [2]. Χρησιμοποιήθηκαν για το debugging κατά την ανάπτυξη του module και μπορούν να αφαιρεθούν.

[1] της μορφής https://lb01.egee-see.org:9000/QL8djVzqjiUJCM_NGJOtKw

[2] http://diveintopython.org/getting_to_know_python/testing_modules.html

3.2.11 UI - tmp.jdl

Το tmp.jdl είναι το αρχείο με το οποίο γίνεται το job submission. Είναι ίδιο με το base.jdl εκτός από την τελευταία γραμμή του πρώτου, που καθορίζει το id του reflector.

```
1 Executable = "reflector.py";
2 StdOutput = "std.out";
3 StdError = "std.err";
4 InputSandbox = {"reflector.py"};
5 OutputSandbox = {"std.out", "std.err"};
6 VirtualOrganisation = "see";
7 Requirements = RegExp("hellasgrid",other.
    GlueCEUniqueId) && (other.
    GlueHostNetworkAdapterInboundIP ==true) &&
8 (other.GlueHostNetworkAdapterOutboundIP ==true);
```

tmp.jdl

Όπως είδαμε στο πρώτο κεφάλαιο, στις τρεις πρώτες γραμμές καθορίζονται το εκτελέσιμο και τα αρχεία της εξόδου και των σφαλμάτων. Στις γραμμές 4 και 5 καθορίζουμε ποια αρχεία πρέπει να μεταφορτωθούν στο Worker Node για την εκτέλεση της εφαρμογής και ποια αρχεία θέλουμε μετά την εκτέλεση, να μεταφορτωθούν πίσω στο UI. Στην γραμμή 6 καθορίζεται το UI στο οποίο ανήκουμε. Οι παράμετρος Requirements αξίζει σχολιασμό.

Κατά την ανάπτυξη του reflector.py έγινε προσπάθεια να ελαχιστοποιηθούν οι απαιτήσεις από το σύστημα που θα τρέξει το script. Ελάχιστη απαίτηση όμως, είναι να μπορεί να λάβει και να στείλει δεδομένα σε κάποια ελεύθερα ports. Στην υλοποίηση σε TCP, που περιγράφεται παρακάτω, η εφαρμογή μας βρήκε πιο φιλόξενο περιβάλλον με ελέυθερα TCP ports. Στην UDP υλοποίηση που αναπτύξαμε πιο πάνω η πράξη έδειξε, ότι οι ρυθμίσεις κάποιων Computing Elements, δεν επέτρεπαν στην εφαρμογή μας να διακινεί δεδομένα. Γι'αυτό στις γραμμές 7 και 8 περιορίσαμε την εκτέλεση σε όσα WNs επιτρέπουν την αποστολή και λήψη δεδομένων μέσω δικτύου και όσα ανήκουν το hellasgrid αφού η πράξη έδειξε, ότι σε αυτά οι πιθανότητα να τρέξει με επιτυχία ένα reflector ήταν μεγαλύτερη. Οι παράμετροι Glue που πετυχαίνουν το "φιλτράρισμα" των CEs αναλύθυκαν στο πρώτο κεφάλαιο.

3.3 Παρατηρήσεις - Συμπεράσματα

3.3.1 TCP vs UDP

Στα παραπάνω είδαμε με λεπτομέρεια πως επιτύχαμε την μετάδοση του βίντεο, μέσα από τον κώδικα. Όπως σε όλες τις εφαρμογές, η πρώτη έκδοση δέχεται πολλές βελτιώσεις και διορθώσεις, σε σημείο πολλές φορές, μετά από μερικές εκδόσεις να έχει μείνει ίδια, μόνο η αρχική ιδέα. Στην πραγματικότητα κατά την ανάπτυξη του κώδικα δοκιμάστηκαν και άλλες λύσεις, όπως θα δούμε αμέσως, που μας επιτρέπουν να βγάλουμε και κάποια χρήσιμα συμπεράσματα.

Στην πρώτη έκδοση της εφαρμογής, χρησιμοποιήθηκε το πρωτόκολλο TCP. Το σκεπτικό ήταν ότι επειδή ο τελικός χρήστης τις περισσότερες φορές βρίσκεται πίσω από κάποιο router, η χρήση συνδεδεμένων TCP sockets θα αφαιρούσε την απαίτηση της ρύθμισης του PORT FORWARDING από τον χρήστη, για την παρακολούθηση του βίντεο. Πραγματικά αποφεύχθηκε αυτή η ρύθμιση, όμως ο χρήστης θα έπρεπε να τρέξει στον υπολογιστή του δικό μας κώδικα, κάτι που κανονικά δεν θα έπρεπε να προτιμά, σε σχέση με την ρύθμιση του router του. Άλλωστε με την έλευση της IPv6 η ρύθμιση αυτή δεν θα χρειάζεται πλέον, αφού θα μπορεί κάθε υπολογιστής να έχει την δική του IP.

Ο κυριότερος όμως λόγος που δοκιμάστηκαν και τα δύο πρωτόκολλα είναι για να μπορέσει να γίνει μία σύγκριση των επιδόσεών τους, στην συγκεκριμένη εφαρμογή. Είναι εντυπωσιακό, ότι παρ'ότι το UDP θεωρείται το καταλληλότερο για μετάδοση βίντεο πραγματικού χρόνου, η έκδοση σε TCP δεν υστερούσε σε επιδόσεις, βασιζόμενοι πάντα στην ανάλυση και τον ρυθμό των δεδομένων (bitrate) του βίντεο που καταφέραμε να μεταδώσουμε.

Βέβαια οι δύο εκδόσεις διέφεραν σε πολλά σημεία και στα συμπεράσματα θα πρέπει να είμαστε συγκρατημένοι, αλλά μπορούμε να τα δικαιολογήσουμε εν μέρει, ως εξής :

- Τα WNs επέβαλαν πιθανώς λιγότερους περιορισμούς στα TCP ports, αφού οι πιο συχνές εφαρμογές του GRID κάνουν χρήση αυτών. Αυτό επέτρεψε στην TCP έκδοση να τρέχει με επιτυχία σε περισσότερα sites. Όπως είδαμε αυτό ήταν πρόβλημα στην UDP έκδοση.

- Το πλεονέκτημα του UDP, είναι ότι δεν διορθώνει τα σφάλματα στην μετάδοση, ούτε κάνει έλεγχο για την σειρά που φτάνουν τα δεδομένα. Αυτό επιτρέπει στα δεδομένα να στέλνονται πιο γρήγορα και χρησιμοποιείται σε περιπτώσεις που μερικά λάθη στην μετάδοση,

δεν αλλοιώνουν το αποτέλεσμα σημαντικά. Στην δική μας περίπτωση όμως, το βίντεο δεν στέλνεται κατ'ευθείαν από την πηγή στον τελικό χρήστη, αλλά μεσολαβούν πολλές αναμεταδόσεις. Έτσι τα σφάλματα είναι τόσα, που δημιουργούν σημαντική αλλοίωση στο βίντεο. Την μικρότερη ταχύτητα του TCP μπορούμε να αντιμετωπίσουμε, μειώνοντας τον ρυθμό μετάδοσης των δεδομένων. Τα σφάλματα στην μετάδοση όμως δεν μπορούμε να τα διορθώσουμε τόσο απλά.

- Τέλος, αν και άσχετο με το πρωτόκολλο μετάδοσης, θα πρέπει να αναφέρουμε το γεγονός ότι όταν δημιουργήθηκε και δοκιμάστηκε η TCP έκδοση, η δημοφιλέστερη έκδοση της python ήταν η 2.7 . Η UDP έκδοση της εφαρμογής βρήκε πολλά sites αναβαθμισμένα στην έκδοση 3 , με αποτέλεσμα τα scripts να μην δουλεύουν καθόλου σε αυτές τις περιπτώσεις, λόγω μερικής ασυμβατότητας των προηγούμενων εκδόσεων με την καινούργια. Η μετατροπή των scripts σε έκδοση 3 εξαλείφει αυτόν τον παράγοντα [1] .

Τα παραπάνω δεν καθιστούν το UDP απαραίτητα ακατάλληλο, για την συγκεκριμένη εφαρμογή. Μόνο όταν ο κώδικας βελτιστοποιηθεί σε μεγάλο βαθμό, μπορούμε να βγάλουμε πιο σίγουρα συμπεράσματα για κάτι τέτοιο. Είναι σημαντικό όμως το συμπέρασμα, ότι στην περίπτωση του ιδιόμορφου "περιβάλλοντος" του GRID, η καταλληλότερη λύση δεν είναι προφανής. Χρειάζονται δοκιμές σε πραγματικές συνθήκες, καθώς και σαφής κατανοήση των περιορισμών που επιβάλει η χρήση κάθε πρωτοκόλλου.

3.3.2 Ιδέες για πιθανές εφαρμογές

Η διπλωματική αυτή έδειξε ότι το GRID μπορεί να χρησιμοποιηθεί για μετάδοση βίντεο πραγματικού χρόνου. Το αν είναι καλή λύση και σε ποιες περιπτώσεις δεν είναι κάτι που μας απασχόλησε μέχρι τώρα. Η διαδικασία της ανάπτυξης της εφαρμογής έδειξε, ότι το GRID επιβάλει πολλούς περιορισμούς σε εφαρμογές που απευθύνονται στον μέσο χρήστη. Μεγάλο μέρος του έτοιμου κώδικα από τις desktop εφαρμογές δεν μπορεί να χρησιμοποιηθεί και πολλές φορές πρέπει να επιλεγούν λύσεις πολύ διαφορετικές. Οι νέες όμως δυνατότητες που δίνει η χρήση του GRID, δικαιολογούν την προσπάθεια. Ας δούμε κάποιες εφαρμογές που ταιριάζουν στην λύση του GRID.

[1] Η ασυμβατότητα προκύπτει κυρίως από την εντολή print και τα try - except blocks. Η μετατροπή σε έκδοση 3 μπορεί να γίνει αυτόματα με το πρόγραμμα 2to3 που διανέμεται μαζί με την τρίτη έκδοση της python

Στις συνηθισμένες μεταδόσεις ενός αγώνα μοτοσυκλέτας ο κάθε θεατής βλέπει, ό,τι αποφασίσει ο σκηνοθέτης να δείξει σε κάθε στιγμή. Έτσι όταν ένα σημείο του αγώνα έχει περισσότερο ενδιαφέρον, η προσπάθεια των άλλων οδηγών χάνεται και ο θεατής δεν έχει την δυνατότητα της επιλογής.

Με την βοήθεια του GRID θα μπορούσε σε κάθε κάμερα που χρησιμοποιείται για την κάλυψη του αγώνα, να ανατεθεί ένας αριθμός reflectors, ο οποίος θα αλλάζει δυναμικά ανάλογα με τον αριθμό των χρηστών που συνδέονται. Έτσι η κατανομή των πόρων, θα προσαρμόζεται αυτόματα στα δεδομένα του αγώνα, κάνοντας ευκολότερη την κάλυψη του γεγονότος, δίνοντας στους χρήστες περισσότερες δυνατότητες. Με παρόμοιο τρόπο, οι χρήστες θα μπορούσαν να διαλέγουν την κάμερα προτίμησής τους σε ένα αγώνα ποδοσφαίρου ή σε οποιοδήποτε γεγονός καλύπτεται από πολλές κάμερες.

Μία άλλη εφαρμογή που το GRID μπορεί να κάνει δυνατή, είναι η ζωντανή μετάδοση πανεπιστημιακών διαλέξεων και γενικότερα μαθημάτων ή σεμιναρίων. Η διαδικτυακή προσέλευση των ενδιαφερομένων, θα καθορίζει και τον αριθμό των reflectors που θα αναλαμβάνουν την μετάδοση. Έτσι κάθε μάθημα, θα δεσμεύει τους πόρους που πραγματικά του αναλογούν, δίνοντας και στους χρήστες την δυνατότητα απομακρυσμένης παρακολούθησης.

Στην περίπτωση τις διαδικτυακής τηλεόρασης, το GRID θα μπορούσε να διευρύνει τις δυνατότητες. Ένα κανάλι θα μπορούσε να εκπέμπει δύο ή και περισσότερα προγράμματα, δεσμεύοντας για το κάθε ένα, πόρους ανάλογους της ακροαματικότητας. Έτσι παράλληλα με το κυρίως πρόγραμμα, θα μπορούσε να μεταδίδει επαναλήψεις δημοφιλών σειρών ή εκπομπών, αυξάνοντας έτσι τον διαφημιστικό χρόνο χωρίς επιπλέον κόστος.

Η μετάδοση έκτακτων γεγονότων με παγκόσμιο ή γενικά μεγάλο ενδιαφέρον θα μπορούσε να επωφεληθεί της χρήσης του GRID. Τα έκτακτα γεγονότα, λόγω της φύσης τους, δεν επιτρέπουν τον οργανωμένο σχεδιασμό της μετάδοσής τους. Με το GRID οι πόροι δεσμεύονται γρήγορα και αποδεσμεύονται το ίδιο γρήγορα όταν δεν χρειάζονται πλέον.

Μία ακόμα εφαρμογή, θα μπορούσε να χρησιμοποιεί το GRID για την μετάδοση βίντεο από πηγές με περιορισμένη σε όγκο σύνδεση, όπως για παράδειγμα ένα κινητό. Τα προγράμματα χρέωσης για πρόσβαση στον internet από κινητό, αλλά και οι προσφερόμενες ταχύτητες, δεν επιτρέπουν την μετάδοση βίντεο σε πολλούς χρήστες. Η εφαρμογή Movino που αναπτύχθηκε στο Abo Akademi University, μπορεί να φανεί χρήσιμη στην μετάδοση βίντεο από κινητό.

Τα παραπάνω παραδείγματα κάνουν χρήση, όχι μόνο των πόρων του GRID αλλά και της δυνατότητας δυναμικής εκχώρησής τους, ανάμεσα σε ανταγωνιστικά σε πόρους γεγονότα, με βάση τις απαιτήσεις. Φυσικά το

GRID μπορεί να χρησιμοποιηθεί και σε κάθε περίπτωση μετάδοσης ενός βίντεο όπως στην περίπτωση μίας έκλειψης ή μίας συναυλίας.

Σε μία εξέλιξη της εφαρμογής, η μετάδοση μέσω του GRID θα μπορούσε να συνδυαστεί με peer to peer μετάδοση από τους χρήστες. Έτσι θα ήταν δυνατόν να μεταδοθεί ακόμα μεγαλύτερος όγκος δεδομένων. Αν για παράδειγμα είχαμε ένα περιορισμένο αριθμό reflectors και κάθε ένας από αυτούς είχε φτάσει το όριο των χρηστών που μπορεί να εξυπηρετήσει, θα μπορούσε η μετάδοση στους καινούργιους χρήστες να γίνεται από τους ήδη συνδεδεμένους, peer to peer. Έτσι θα λειτουργούσε ως εφεδρικό σύστημα (fallback) στην περίπτωση υπερφόρτωσης.

Κάτι που θα αύξανε σημαντικά τις δυνατότητες της εφαρμογής μας, θα ήταν η δημιουργία περισσότερων επιπέδων με reflectors. Στη παρούσα υλοποίηση, υπάρχει ένας distributor που στέλνει τα δεδομένα σε όλους του reflectors. Αν κάθε υπολογιστής μπορεί να στείλει το βίντεο το πολύ σε τρεις άλλους, ο μέγιστος αριθμός των viewers θα είναι εννιά, μια όχι και τόσο εντυπωσιακή επίδοση. Αν όμως ο distributor έστελνε σε τρεις reflectors-distributors, οι οποίοι με την σειρά τους έστελναν σε τρεις reflectors ο κάθε ένας, τότε οι τελικοί χρήστες θα ήταν 27. Ο τελικός αριθμός αυξάνει εκθετικά με των αριθμό των επιπέδων που θα υλοποιηθούν. Η υλοποίηση μπορεί να γίνει αρκετά πολύπλοκη, αλλά τα αποτελέσματα μάλλον θα δικαιώσουν την προσπάθεια.

Σχετικά με την εμπορική ετοιμότητα της εφαρμογής που αναπτύχθηκε, πρέπει να παραδεχτούμε ότι, για να μπορέσει να στηρίξει με αξιοπιστία την μετάδοση κάποιου γεγονότος, χρειάζεται πολύ δουλειά και μεγάλες ίσως αλλαγές στον κώδικα. Ο σκοπός όμως της διπλωματικής ήταν να αποδείξει, ότι η επιπλέον αυτή δουλειά θα έχει το επιθυμητό αποτέλεσμα και να δώσει την εμπιστοσύνη στις μελλοντικές προσπάθειες ότι το GRID μπορεί να μεταδώσει βίντεο.

Ελπίζουμε τελειώνοντας την διπλωματική αυτή οι παραπάνω ιδέες, να δώσουν την αρχή για περισσότερες εναλλακτικές χρήσεις του GRID. Έχουμε στην διάθεσή μας ένα καινούργιο εργαλείο, που το έχουμε εξοπλίσει με πλήθος δυνατοτήτων. Οι χρησιμότητά του τελειώνει, εκεί που τελειώνει η φαντασία των χρηστών του. Η μετάδοση βίντεο, φέρνει το GRID πιο κοντά σε καθημερινές εφαρμογές. Η ρύση του δημιουργού του World Wide Web, ας ελπίσουμε να επαληθευτεί και για το GRID :

''The Web as I envisaged it, we have not seen it yet. The future is still so much bigger than the past.''

Παράρτημα Α

Ο κώδικας

Παρακάτω δίνεται ο κώδικας, πάνω στον οποίο βασίστηκε το τρίτο κεφάλαιο αυτής της εργασίας. Οι τελευταίες εκδόσεις των πηγαίων αρχείων βρίσκονται στην διεύθυνση : http://sourceforge.net/projects/grivider/

A .1 distributor.py

```
1  # -*- coding: utf-8 -*-
2  """
3  distributor.py
4  """
5  from socket import *
6  import sys
7  import pickle
8  import select
9  import sqlite3
10 import threading
11
12
13
14 REPORT_PORT = 55000
15 DB_FILE = "gvr.db"
16 EXIT_KEY = ["Y","y"]
17 REPORT_SIZE = 100
18 TIME_LIMIT = 180
19 GET_REFLECTORS_INTERVAL = 120
20 LISTEN_PORT = 55100
21 REFLECTORS_LIST_PORT = 55101
```

```
22 DATA_SIZE = 1316
23
24
25 def update_db(report,ip,c) :
26   """
27   update_db doc
28   """
29
30   if len(report)==1 :
31     t = (report[0],)
32     c.execute("SELECT COUNT(*) AS NUM FROM reflectors
          WHERE my_id=?" , t)
33     num_of_results = c.fetchone()
34
35     if num_of_results[0]>0 :
36       c.execute("UPDATE reflectors SET ip=:ip,
            last_report=datetime() WHERE my_id=:id" ,{"ip
            ":ip,"id":report[0]})
37       print "reflector updated"
38     else :
39       c.execute("INSERT INTO reflectors values(:id,
            NULL,:ip,datetime())" ,{"id":report[0],"ip":
            ip})
40       print "new reflector added"
41
42   elif len(report)==3 :
43     print "here comes a control panel report"
44
45
46
47 def report_receiver_func(event):
48   """
49   report_receiver_func doc
50   """
51
52   report_sock = socket(AF_INET, SOCK_DGRAM)
53   report_sock.bind(("",REPORT_PORT))
54
55   try:
56     conn = sqlite3.connect('gvr.db')
57   except:
```

```
58      print "Could not connect to database."
59      exit
60    c = conn.cursor()
61
62    while not event.isSet():
63      try :
64        readlist,writelist,xlist = select.select([
             report_sock,],[],[],0)
65      except :
66        print "Select gone wrong!!"
67
68      if report_sock in readlist :
69        print "Here comes a report !"
70        data,addr = report_sock.recvfrom(REPORT_SIZE)
71        try:
72          report = pickle.loads(data)
73        except:
74          print "Can't read report. Bad package maybe?
               Dropped it"
75          break
76
77        print "Got %s from %s" % (report,addr[0])
78        update_db(report,addr[0],c)
79        conn.commit()
80    c.close()
81    conn.close()
82
83
84
85  def get_reflectors_func(event):
86    """
87     get_reflectors_func doc
88    """
89    try:
90      conn = sqlite3.connect('gvr.db')
91    except:
92      print "Could not connect to database."
93      exit
94    c = conn.cursor()
95
96    while not event.isSet():
```

```
97      global reflectors
98      reflectors = []
99      c.execute("SELECT ip FROM reflectors WHERE
           strftime('%s','now') - strftime('%s',
           last_report)< ?" , (TIME_LIMIT,))
100     for row in c:
101       reflectors.append(row[0])
102     print "Sending to ",reflectors
103     event.wait(GET_REFLECTORS_INTERVAL)
104
105   c.close()
106   conn.close()
107
108 def stream_data_func(event):
109   """
110   stream_data_func doc
111   """
112   listen_sock = socket(AF_INET, SOCK_DGRAM)
113   listen_sock.bind(("",LISTEN_PORT))
114
115   talk_sock = socket(AF_INET, SOCK_DGRAM)
116
117   while not event.isSet():
118     try :
119       readlist,writelist,xlist = select.select([
             listen_sock,],[talk_sock],[],0)
120     except :
121       print "Select gone wrong!!"
122       break
123
124     data = ""
125     if listen_sock in readlist :
126       data,addr = listen_sock.recvfrom(DATA_SIZE)
127     if talk_sock in writelist and data:
128       for r in reflectors :
129         fullsend(data,r,talk_sock)
130
131
132 def fullsend(msg,addr,sock):
133   totalsent = 0
134   while totalsent < len(msg):
```

```
135     try :
136       sent = sock.sendto(msg[totalsent:],(addr,
            REFLECTORS_LIST_PORT))
137     except error, e:
138       print "exception raised"
139       break
140     except IOError, e:
141       print "Got IOError: ", e
142       break
143     totalsent = totalsent + sent
144     sent = 0
145   return totalsent
146
147
148
149 reflectors = []
150
151 ev1 = threading.Event()
152 ev2 = threading.Event()
153 ev3 = threading.Event()
154 report_receiver_thread = threading.Thread(target=
      report_receiver_func,args=(ev1,))
155 get_reflectors_thread = threading.Thread(target=
      get_reflectors_func,args=(ev2,))
156 stream_data_thread = threading.Thread(target=
      stream_data_func,args=(ev3,))
157
158 report_receiver_thread.start()
159 get_reflectors_thread.start()
160 stream_data_thread.start()
161
162
163 while raw_input("Give the key to stop the distributor
      :") not in EXIT_KEY:
164   pass
165
166
167 ev1.set()
168 ev2.set()
169 ev3.set()
170 print "Exiting script..."
```

```
171 report_receiver_thread.join()
172 get_reflectors_thread.join()
173 stream_data_thread.join()
174 print "Bye"
```

distributor.py

A .2 reflector.py

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  import socket
5  import sys
6  import select
7  import urllib
8  import pickle
9  import threading
10 import errno, time
11
12 SRV_URL = "kay.hep.ntua.gr"
13 SRV_PORT = 55000
14 LIST_PORT = 24200
15 SHUT_PORT = 24201
16 REMOTE_PORT = 55102
17 DATA_SIZE = 1316
18 REPORT_INTERVAL = 60
19 UPDATE_VIEWERS_INTERVAL = 30
20
21 EXIT_KEY = ["Y","y"]
22
23 SERVER_ADDRESS = ((socket.gethostbyname_ex(SRV_URL)
      [2][0],SRV_PORT))
24
25
26 class FirefoxOpener(urllib.URLopener):
27   """Documentation"""
28   version = 'Mozilla/5.0 (X11; U; Linux i686; en-US;
       rv:1.9.2.12) Gecko/20101027 Firefox/3.6.12'
29
```

```
30
31 def shutdown_func(event):
32   """Documentation"""
33   while not event.isSet():
34     try:
35       msg,addr  = shut_sock.recvfrom(DATA_SIZE)
36     except :
37       print "Problem with shutdown recvfrom"
38       continue
39     if msg=="shut\n":
40       event.set() # this is the global ev1 that stops
            all threads
41
42
43 def reporting_func(event):
44   while not event.isSet():
45     try:
46       report_sock.sendto(pickle.dumps([job_id,]),
            SERVER_ADDRESS)
47     except :
48       print "Couldn't send report"
49     event.wait(REPORT_INTERVAL)
50
51 def get_viewers_func(event):
52   """Documentation"""
53   global viewers_list
54   myfirefox = FirefoxOpener()
55   while not event.isSet():
56     resp = myfirefox.open("http://"+SRV_URL+"/
          showviewers.php?refl="+str(job_id))
57     html = resp.read()
58     rows = html.splitlines()
59     viewers_list = rows
60     event.wait(UPDATE_VIEWERS_INTERVAL)
61
62
63 def streaming_func(event):
64   global total_sent
65   while not event.isSet():
66     readlist,writelist,b = select.select([
          listening_sock,],[],[],0)
```

```
67      data = ""
68      if listening_sock in readlist :
69        data,addr  = listening_sock.recvfrom(DATA_SIZE)
70        for viewer in viewers_list:
71          total_sent += fullsend(data,viewer)
72
73
74
75  def fullsend(msg,addr):
76    totalsent = 0
77    while totalsent < len(msg):
78      try :
79        sent = talking_sock.sendto(msg[totalsent:],(addr
             ,REMOTE_PORT))
80      except socket.error, e:
81        print "exception raised"
82        break
83      except IOError, e:
84        print "Got IOError: ", e
85        break
86      totalsent = totalsent + sent
87      sent = 0
88    return totalsent
89
90
91  if __name__ == "__main__":
92
93    job_id = int(sys.argv[1])
94
95    viewers_list = []
96
97    total_sent = 0
98
99
100   report_sock = socket.socket(socket.AF_INET, socket.
        SOCK_DGRAM)
101   listening_sock = socket.socket(socket.AF_INET,
        socket.SOCK_DGRAM)
102   talking_sock = socket.socket(socket.AF_INET, socket.
        SOCK_DGRAM)
```

```
103   shut_sock = socket.socket(socket.AF_INET, socket.
        SOCK_DGRAM)
104
105
106   listening_sock.bind(('',LIST_PORT))
107
108   shut_sock.bind(('',SHUT_PORT))
109
110   ev1 = threading.Event()
111   reporting_thread = threading.Thread(target=
        reporting_func,args=(ev1,))
112   get_viewers_thread = threading.Thread(target=
        get_viewers_func,args=(ev1,))
113   streaming_thread = threading.Thread(target=
        streaming_func,args=(ev1,))
114   shutdown_thread = threading.Thread(target=
        shutdown_func,args=(ev1,))
115
116   reporting_thread.start()
117   get_viewers_thread.start()
118   streaming_thread.start()
119   shutdown_thread.start()
120
121   while not ev1.isSet():
122     pass
123
124   print "Sent a total of "+str(total_sent/1000)+"kb to
        " + str(viewers_list)
125   print "Exiting script..."
126   reporting_thread.join()
127   get_viewers_thread.join()
128   streaming_thread.join()
129   shutdown_thread.join()
130   print "Bye"
```

reflector.py

A .3 showviewers.php

```php
<?php
  if(isset($_GET["refl"])){
    $TIME_LIMIT = 500;

    $db = new PDO('sqlite:gvr.db');
    $reflector = $_GET["refl"];
    $res = $db->query("SELECT ip FROM viewers WHERE
      reflector_id =$reflector AND
      strftime('%s','now') - strftime('%s',viewers.
         last_report)<$TIME_LIMIT"
      );

    foreach($res as $row){
      print $row[0]."\n";
    }
  }
  else{
    print "select a reflector";
  }
?>
```

showviewers.php

A .4 play.php

```html
<html>
<head>
<title>Grid Video Reflector - Karakasilis Dimitris</
   title>
<script language="javascript" type="text/javascript">
  function update(){
    var url='update_db.php';
      var xmlHttp;
      try
      {
        // Firefox, Opera 8.0+, Safari
        xmlHttp=new XMLHttpRequest();
      }
      catch (e)
      {
```

```
15          // Internet Explorer
16          try
17        {
18              xmlHttp=new ActiveXObject("Msxml2.XMLHTTP"
                    );
19        }
20          catch (e)
21        {
22          try
23              {
24                  xmlHttp=new ActiveXObject("Microsoft.
                        XMLHTTP");
25              }
26          catch (e)
27              {
28              alert("Your browser does not support AJAX!
                    ");
29                return false;
30              }
31        }
32        }
33      xmlHttp.onreadystatechange=function(){
34          if(xmlHttp.readyState==4 && xmlHttp.status ==
                200){
35            if(xmlHttp.responseXML!=null){
36            docx = xmlHttp.responseXML;
37            ip = docx.getElementsByTagName('ip')[0].
                  firstChild.nodeValue;
38            ref_id = docx.getElementsByTagName('ref_id')
                  [0].firstChild.nodeValue;
39            document.getElementById('ip').innerHTML = ip
                  ;
40            document.getElementById('ref_id').innerHTML
                  = ref_id;
41          }
42          else if(xmlHttp.responseText!=null){
43                alert(xmlHttp.responseText);
44            }
45          }
46        }
47      xmlHttp.open("GET",url,true);
```

```
48      xmlHttp.send(null);
49    }
50
51    window.setInterval('update()',10000);
52    window.onload = update;
53 </script>
54
55 <link href="style.css" rel="stylesheet" type="text/css
      ">
56
57 </head>
58
59 <body>
60 <div id ="main">
61 <h1>Grid Video Reflector</h1>
62 <h3>A project by Dimitris Karakasilis</h3>
63 <h5>You are <span id="ip"></span> and
64 you are watching reflector <span id="ref_id"></span></
      h5>
65 <h5>To see the video all UDP traffic through port
66  20101 should be coming to your computer.(Check your
      router settings)</h5>
67
68 <embed type="application/x-vlc-plugin"
69        name="video1"
70        autoplay="yes"
71        loop="yes"
72        width="800"
73        height="600"
74        target="udp://@:55102" />
75 </div>
76 </body>
77 </html>
```

play.php

A .5 update_db.php

```
1 <?php
2 $TIME_LIMIT = 180;
```

```
3
4 try{
5   $db = new PDO('sqlite:gvr.db');
6   $ip=$_SERVER['REMOTE_ADDR'];
7
8   $res = $db->query("SELECT reflector_id FROM viewers
       WHERE ip = '$ip'");
9
10  $reflector_id = -1;
11  foreach($res as $row){
12    $reflector_id = $row["reflector_id"];
13  }
14  if($reflector_id!=-1){
15    $res = $db->query("SELECT COUNT(*) as num FROM
         viewers,reflectors
16      WHERE viewers.ip = '$ip'
17      AND viewers.reflector_id = reflectors.my_id
18      AND strftime('%s','now') - strftime('%s',
          reflectors.last_report)<$TIME_LIMIT");
19    $a = $res->fetch();
20    if($a[0]>0){
21        $res = $db->query("UPDATE viewers SET
22          last_report = datetime('now') WHERE ip = '
              $ip' ");
23    }
24    else{
25      $res = $db->query("SELECT my_id  FROM
           reflectors WHERE
26        strftime('%s','now') - strftime('%s',
            reflectors.last_report)<$TIME_LIMIT");
27      $last_workload = 0;
28      $reflector_id = -1;
29      foreach($res as $row){
30        $id = $row["my_id"];
31        $res2 = $db->query("select COUNT(viewers.ip)
             FROM reflectors
32          LEFT OUTER JOIN viewers ON reflectors.my_id
              = $id
33          AND reflectors.my_id=viewers.reflector_id
              GROUP BY reflectors.my_id");
34
```

```
35          $workload = $res2->fetchColumn();
36            if($workload<=$last_workload){
37                $reflector_id = $id;
38                $last_workload = $workload;
39            }
40        }
41
42        if($reflector_id!=-1){
43          $res = $db->query("UPDATE viewers SET
44            last_report = datetime('now'),
45            reflector_id = $reflector_id WHERE ip = '$ip
                ' ");
46        }
47
48      }
49    }
50    else{
51      $res = $db->query("SELECT my_id  FROM  reflectors
52      WHERE strftime('%s','now') - strftime('%s',
           reflectors.last_report)<$TIME_LIMIT");
53      $last_workload = 0;
54      foreach($res as $row){
55        $id = $row["my_id"];
56        $res2 = $db->query("select COUNT(viewers.ip)
             FROM reflectors
57        LEFT OUTER JOIN viewers ON reflectors.my_id =
             $id
58        AND reflectors.my_id=viewers.reflector_id
59        GROUP BY reflectors.my_id");
60        $workload = $res2->fetchColumn();
61          if($workload<=$last_workload){
62              $reflector_id = $id;
63              $last_workload = $workload;
64          }
65      }
66
67      if($reflector_id!=-1){
68        $res = $db->exec("INSERT INTO viewers
69        VALUES(NULL,$reflector_id,'$ip',datetime('now'))
            ");
70      }
```

```
71   }
72   header('Content-Type: application/xml; charset=UTF-8
        ');
73   print "<?xml version=\"1.0\" encoding=\"UTF-8\"
        standalone=\"yes\" ?>
74       <data>
75         <ip>$ip</ip>
76         <ref_id>$reflector_id</ref_id>
77       </data>";
78
79 }
80 catch(PDOException $e){
81   echo $e->getMessage();
82 }
83
84 $db = null;
85 ?>
```

update_db.php

A .6 db_create.sql

```
1  create table reflectors(
2  my_id INTEGER PRIMARY KEY,
3  job_id TEXT,
4  ip TEXT,
5  last_report DATETIME
6  );
7
8  create table viewers(
9  viewer_id INTEGER PRIMARY KEY AUTOINCREMENT,
10 reflector_id INTEGER,
11 ip TEXT,
12 last_report DATETIME
13 );
```

db_create.sql

A .7 controlpanel.py

```
#!/usr/bin/env python

import sys
import glib
import ssh

try:
  import pygtk
    pygtk.require("2.0")
except:
    pass
try:
  import gtk
    import gtk.glade
    import gobject
    import pickle
    import socket
except:
  sys.exit(1)


JOB_FILE = "jobID"
UPDATE_INTERVAL = 60000
SRV_URL = "jimmykarily.ath.cx"
SRV_PORT = 5500

server_address = ((socket.gethostbyname_ex(SRV_URL)
   [2][0],SRV_PORT))

class Gui:
  myui = None

  def say(self,words):
    print words
    self.messages_buffer.insert_at_cursor(words+"\n")
    self.window.show_all()

  def updater(self):
```

```
38      if self.myui :
39        self.jobs = self.myui.getjobs(JOB_FILE)
40
41        self.servers_list.clear()
42
43        if self.jobs != -1 :
44          for j in self.jobs :
45            rem_but = gtk.Button()
46            self.servers_list.append([j.id,j.status,
                rem_but,"#000","#AAA"])
47          self.window.show_all()
48          return True
49
50    def getHostFromGui(self):
51      url_field = self.wTree.get_widget("ui_url").
          get_text()
52      un_field = self.wTree.get_widget("username_tb").
          get_text()
53      pwd_field = self.wTree.get_widget("password_tb").
          get_text()
54      host = {"host":url_field,"un":un_field,"passwrd":
          pwd_field}
55      return host
56
57    def connect_to_ui(self,caller):
58      host = self.getHostFromGui()
59      self.myui = ssh.MyUi(host)
60      self.say("Trying to connect to : " + self.myui.
          ui_ip)
61      if self.myui.connect_me()!=-1:
62        self.say("Connection OK")
63        self.say("Initializing proxy...")
64
65        passphrase = self.wTree.get_widget("
            passphrase_tb").get_text()
66        vom =  self.wTree.get_widget("vom_tb").get_text
            ()
67
68        if passphrase and vom :
69          self.say(self.myui.init_proxy(vom,passphrase)
              and "proxy initialized" or "proxy was still
```

```
            valid")
70          self.updater() # Run it once manually because
               it may take some time until its
               automatically run.
71        else :
72          self.say("Fill The vom and the passphrase
               fields")
73      else:
74        self.say("Could not connect!")
75
76    def sendreport(self,data,sock):
77      sock.sendto(pickle.dumps(data),server_address)
78      return 1
79
80
81    def newServer(self,caller):
82      if self.myui :
83        my_job_id = str(len(self.jobs) +1)
84        self.myui.updatejdl(my_job_id)
85        grid_job_id = self.myui.submit_job(JOB_FILE,"tmp
             .jdl")
86        self.sendreport(["1",my_job_id,grid_job_id],self
             .report_sock)
87        self.say("The job %s was submitted identified by
              %s" % (grid_job_id, my_job_id))
88        self.updater()
89      else :
90        self.say("Not connected!")
91
92    def removeServer(self,caller):
93      if self.jobs :
94        select1, select2 = self.treeview.get_selection()
             .get_selected()
95        entry = select1.get_value(select2,0)
96        jobs_to_cancel = [j for j in self.jobs if j.id
             == entry]
97        for j in jobs_to_cancel:
98          self.say("Cancelling job "+entry)
99          self.myui.cancel_job(j)
100       self.updater()
101
```

```
102   def disconnect(self,caller):
103     self.say("Disconnecting from ui...")
104     self.myui.close()
105     self.say("Disconnected")
106
107   def destroy(self,caller):
108     if self.myui :
109       self.say("Disconnecting from ui...")
110       self.myui.close()
111     gtk.main_quit()
112
113   def remove_all(self,caller):
114     self.say("Removing all jobs")
115
116
117   def default_me(self):
118     self.wTree.get_widget("password_tb").set_text("
          default_password")
119     self.wTree.get_widget("passphrase_tb").set_text("
          default_passphrase")
120     url_field = self.wTree.get_widget("ui_url").
          set_text("ui01.isabella.grnet.gr")
121     un_field = self.wTree.get_widget("username_tb").
          set_text("default_username")
122     self.wTree.get_widget("vom_tb").set_text("
          default_vo")
123
124
125   def __init__(self):
126     #Set the Glade file
127     self.gladefile = "cp.glade"
128     self.wTree = gtk.glade.XML(self.gladefile)
129
130     self.window = self.wTree.get_widget("MainWindow")
131     if (self.window):
132       self.window.connect("destroy", self.destroy)
133
134
135     self.window.set_default_size(500, 200)
136
```

```
137     dic = { "connect_to_ui" : self.connect_to_ui,"
           newServer":self.newServer,
138          "disconnect":self.disconnect,"remove":self.
              removeServer}
139     self.wTree.signal_autoconnect(dic)
140
141     self.wTree.get_widget("password_tb").
           set_visibility(False)
142     self.wTree.get_widget("passphrase_tb").
           set_visibility(False)
143
144     self.messages_buffer = gtk.TextBuffer()
145     self.messages_box_tv = gtk.TextView()
146
147
148     self.scrolled_messages = gtk.ScrolledWindow(
           hadjustment= None, vadjustment= None)
149     self.scrolled_messages.set_size_request(400, 80)
150     self.wTree.get_widget("hbox3").add(self.
           scrolled_messages)
151     self.scrolled_messages.add(self.messages_box_tv)
152     self.messages_box_tv.set_buffer(self.
           messages_buffer)
153
154     self.default_me()
155
156     ############## Tree view stuff
           ################################
157     self.servers_list = gtk.ListStore(str, str, gtk.
           Button ,str,str)
158     self.treeview = gtk.TreeView(self.servers_list)
159
160     cell = gtk.CellRendererText()
161     cell.set_property('background-set' , True)
162     cell.set_property('foreground-set' , True)
163
164     #First column
165     col = gtk.TreeViewColumn("Job ID")
166     col.pack_start(cell, True)
167     col.set_attributes(cell,text=0, foreground=3,
           background=4)
```

```
168     self.treeview.append_column(col)
169
170     #Second column
171     col = gtk.TreeViewColumn("Status")
172     col.pack_start(cell, True)
173     col.set_attributes(cell,text=1, foreground=3,
           background=4)
174     self.treeview.append_column(col)
175
176     #Third column
177     check = gtk.CellRendererToggle()
178     col = gtk.TreeViewColumn("Cancel")
179     col.pack_start(check, True)
180     col.add_attribute(check, "active", 1)
181
182     self.treeview.append_column(col)
183
184     self.scrolled_window = gtk.ScrolledWindow(
           hadjustment=None, vadjustment=None)
185     self.scrolled_window.set_size_request(400, 200)
186     self.wTree.get_widget("vbox1").add(self.
           scrolled_window)
187     self.scrolled_window.add(self.treeview)
188
189     ######### End of treeview stuff
           ############################
190
191     self.window.show_all()
192
193     gobject.timeout_add(UPDATE_INTERVAL,self.updater)
194
195
196     self.report_sock = socket.socket(socket.AF_INET,
           socket.SOCK_DGRAM)
197
198
199 if __name__ == "__main__":
200   hwg = Gui()
201   gtk.main()
```

controlpanel.py

A .8 ssh.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import socket
import paramiko
import re


class Jobobg:
  def __init__(self,id):
    self.id=id
    self.status=""

class MyUi(paramiko.SSHClient):
  def __init__(self,host_info):
    paramiko.SSHClient.__init__(self)
    self.host = host_info["host"]
    self.usernm = host_info["un"]
    self.passwrd = host_info["passwrd"]
    self.connected = 0

    try :
      self.ui_ip = socket.gethostbyname_ex(self.host)
         [2][0]
    except :
      print "Could not resolve host ip"

    self.set_missing_host_key_policy(paramiko.
       AutoAddPolicy())

  def connect_me(self):
    try :
      self.connect(self.ui_ip,username=self.usernm,
         password=self.passwrd)
    except :
      return -1

  def proxy_valid(self):
```

```
36      command = "voms-proxy-info \n"
37      stdin,stdout,stderr = self.exec_command(command)
38      lines = stdout.readlines()
39
40      errors = stderr.readlines()
41      if len(errors) !=0 :
42        print "There was an error :"
43        for er in errors :
44          print er
45        return 0
46
47      pattern = re.compile(r'(\d{1,2}\:\d{1,2}\:\d{1,2})
          ')
48      if pattern.search(lines[-1]).groups()[0] != "
          0:00:00" :
49        return 1
50      else :
51        return 0
52
53    def init_proxy(self,vom,passphrase):
54      if not self.proxy_valid():
55        command = "voms-proxy-init -voms %s \n" % vom
56        passphrase = passphrase + "\n"
57        stdin,stdout,stderr = self.exec_command(command)
58        stdin.write(passphrase)
59        lines = stdout.readlines()
60        errors = stderr.readlines()
61        if len(errors) != 0 :
62          print "Proxy initialization failed. The error
              was :"
63          for er in errors :
64            print er
65          return 0
66
67        for l in lines :
68          print l
69
70        return 1
71      else :
72        return 0
73
```

```
74
75   def updatejdl(self,arg):
76     command = "cp base.jdl tmp.jdl"
77     stdin,stdout,stderr = self.exec_command(command)
78
79     command = "echo 'Arguments = \"%s\";' >> tmp.jdl"
         % arg
80     stdin,stdout,stderr = self.exec_command(command)
81     lines = stderr.readlines()
82
83     errors = stderr.readlines()
84     for er in errors :
85       print er
86       return 0
87
88     for l in lines :
89       print l
90     return lines
91
92   def submit_job(self,job_file,my_jdl):
93     if self.proxy_valid():
94       command = " glite-wms-job-submit -o %s -a  %s"
           % (job_file,my_jdl)
95       stdin,stdout,stderr = self.exec_command(command)
96       lines = stdout.readlines()
97       for l in lines :
98         if l[0:8] == "https://" :
99           return l.strip()                    #Remove
               whitespace and new line from the end
100    else :
101      return 0
102
103  def getjobs(self,jobfile):
104
105    if self.proxy_valid():
106      jobs =[]
107      command = " glite-wms-job-status -i %s "  %
           jobfile
108      stdin,stdout,stderr = self.exec_command(command)
109      stdin.write("a\n")
110      lines = stdout.readlines()
```

```
111       for i in range(len(lines)) :
112         job_label = "Status info for the Job : "
113         status_label = "Current Status: "
114         if lines[i].find(job_label)==0:
115           job_temp = Jobobg(lines[i].replace(job_label
                ,"").strip()) # id is a parameter
116           job_temp.status = lines[i+1].replace(
                status_label,"").strip()
117           jobs.append(job_temp)
118
119       return jobs
120     else :
121       return -1
122
123   def printjobs(self,jobfile):
124
125     if self.proxy_valid():
126       jobs =[]
127       command = " glite-wms-job-status -i %s "  %
            jobfile
128       stdin,stdout,stderr = self.exec_command(command)
129       stdin.write("a\n")
130       lines = stdout.readlines()
131       for i in range(len(lines)) :
132         job_label = "Status info for the Job : "
133         status_label = "Current Status: "
134         if lines[i].find(job_label)==0:
135           job_temp = Jobobg(lines[i].replace(job_label
                ,"").strip()) # id is a parameter
136           job_temp.status = lines[i+1].replace(
                status_label,"").strip()
137           jobs.append(job_temp)
138
139       for j in jobs :
140         print "%s = %s \n%s = %s \n\n" % ("Job".rjust
              (10),j.id,"Status".rjust(10),j.status)
141
142       return jobs
143     else :
144       return -1
145
```

```
146   def cancel_job(self,job):
147
148     if self.proxy_valid():
149       command = " glite-wms-job-cancel %s"  % job.id
150       stdin,stdout,stderr = self.exec_command(command)
151       stdin.write("y\n")
152       lines = stdout.readlines()
153       for l in lines :
154         print l
155       return 1
156     else :
157       return 0
158
159   def job_file_delete(self,job_file):
160
161     joblist = self.getjobs(job_file)
162     if joblist!=-1 :
163       for j in joblist :
164         self.cancel_job(j)
165     else :
166       print "Proxy not initialized! Not a valid proxy.
          "
167
168     if self.proxy_valid():
169       command = "rm %s"  % job_file
170       stdin,stdout,stderr = self.exec_command(command)
171       lines = stdout.readlines()
172       for l in lines :
173         print l
174       return 1
175     else :
176       return 0
177
178 if __name__ == "__main__" :
179   host = {"host":"ui01.isabella.grnet.gr","un":"
        griduser","passwrd":"default_password"}
180   myui = MyUi(host)
181
182   print "Trying to connect to : " + myui.ui_ip
183   if myui.connect_me()!=-1:
184     print "Connection OK"
```

```
185
186     print "Initializing proxy...\n"
187     print myui.init_proxy("see","passphrase") and "
           proxy initialized \n" or "proxy was still valid
            \n"  #This is the co and the passphrase
188
189
190     job_file = "jobID"
191     joblist = myui.printjobs(job_file)
192   else:
193     print "Could not connect!"
194
195   myui.close()
```

ssh.py

A .9 cp.glade

```
 1 <?xml version="1.0"?>
 2 <glade-interface>
 3   <!-- interface-requires gtk+ 2.16 -->
 4   <!-- interface-naming-policy project-wide -->
 5   <widget class="GtkWindow" id="MainWindow">
 6     <property name="visible">True</property>
 7     <property name="resizable">False</property>
 8     <child>
 9       <widget class="GtkVBox" id="vbox1">
10         <property name="visible">True</property>
11         <property name="orientation">vertical</
              property>
12         <child>
13           <widget class="GtkExpander" id="expander1">
14             <property name="visible">True</property>
15             <property name="can_focus">True</property>
16             <child>
17               <widget class="GtkHBox" id="hbox5">
18                 <property name="visible">True</
                      property>
19                 <child>
20                   <widget class="GtkVBox" id="vbox9">
```

```
21                      <property name="visible">True</
                           property>
22                      <property name="orientation">
                           vertical</property>
23                      <child>
24                        <widget class="GtkLabel" id="
                             ui_lb">
25                          <property name="visible">True
                             </property>
26                          <property name="xalign">1</
                             property>
27                          <property name="label"
                             translatable="yes">UI :</
                             property>
28                        </widget>
29                        <packing>
30                          <property name="position">0</
                             property>
31                        </packing>
32                      </child>
33                      <child>
34                        <widget class="GtkLabel" id="
                             username_lb">
35                          <property name="visible">True
                             </property>
36                          <property name="xalign">1</
                             property>
37                          <property name="label"
                             translatable="yes">Username
                              :</property>
38                        </widget>
39                        <packing>
40                          <property name="position">1</
                             property>
41                        </packing>
42                      </child>
43                      <child>
44                        <widget class="GtkLabel" id="
                             password_lb">
45                          <property name="visible">True
                             </property>
```

```
46                          <property name="xalign">1</
                               property>
47                          <property name="label"
                               translatable="yes">Password
                                :</property>
48                        </widget>
49                        <packing>
50                          <property name="position">2</
                               property>
51                        </packing>
52                      </child>
53                      <child>
54                        <widget class="GtkLabel" id="
                           vom_l">
55                          <property name="visible">True
                               </property>
56                          <property name="xalign">1</
                               property>
57                          <property name="label"
                               translatable="yes">Voms :</
                               property>
58                          <property name="justify">right
                               </property>
59                        </widget>
60                        <packing>
61                          <property name="position">3</
                               property>
62                        </packing>
63                      </child>
64                      <child>
65                        <widget class="GtkLabel" id="
                           passphrase_lb">
66                          <property name="visible">True
                               </property>
67                          <property name="xalign">1</
                               property>
68                          <property name="label"
                               translatable="yes">
                               Passphrase :</property>
69                        </widget>
70                        <packing>
```

```
71                          <property name="position">4</
                               property>
72                        </packing>
73                      </child>
74                    </widget>
75                    <packing>
76                      <property name="position">0</
                         property>
77                    </packing>
78                  </child>
79                  <child>
80                    <widget class="GtkVBox" id="vbox8">
81                      <property name="visible">True</
                         property>
82                      <property name="orientation">
                         vertical</property>
83                      <child>
84                        <widget class="GtkEntry" id="
                           ui_url">
85                          <property name="visible">True
                             </property>
86                          <property name="can_focus">
                             True</property>
87                          <property name="invisible_char
                             ">&#x25CF;</property>
88                          <property name="width_chars"
                             >20</property>
89                          <property name="shadow_type">
                             none</property>
90                        </widget>
91                        <packing>
92                          <property name="position">0</
                             property>
93                        </packing>
94                      </child>
95                      <child>
96                        <widget class="GtkEntry" id="
                           username_tb">
97                          <property name="visible">True
                             </property>
```

```
98                          <property name="can_focus">
                               True</property>
99                          <property name="invisible_char
                               ">&#x25CF;</property>
100                         <property name="width_chars"
                               >20</property>
101                       </widget>
102                       <packing>
103                         <property name="position">1</
                               property>
104                       </packing>
105                     </child>
106                     <child>
107                       <widget class="GtkEntry" id="
                            password_tb">
108                         <property name="visible">True
                               </property>
109                         <property name="can_focus">
                               True</property>
110                         <property name="invisible_char
                               ">&#x25CF;</property>
111                         <property name="width_chars"
                               >20</property>
112                       </widget>
113                       <packing>
114                         <property name="position">2</
                               property>
115                       </packing>
116                     </child>
117                     <child>
118                       <widget class="GtkEntry" id="
                            vom_tb">
119                         <property name="visible">True
                               </property>
120                         <property name="can_focus">
                               True</property>
121                         <property name="invisible_char
                               ">&#x25CF;</property>
122                         <property name="width_chars"
                               >20</property>
123                       </widget>
```

```
124                       <packing>
125                         <property name="position">3</
                               property>
126                       </packing>
127                     </child>
128                     <child>
129                       <widget class="GtkEntry" id="
                             passphrase_tb">
130                         <property name="visible">True
                               </property>
131                         <property name="can_focus">
                               True</property>
132                         <property name="invisible_char
                               ">&amp;#x25CF;</property>
133                       </widget>
134                       <packing>
135                         <property name="position">4</
                               property>
136                       </packing>
137                     </child>
138                   </widget>
139                   <packing>
140                     <property name="position">1</
                          property>
141                   </packing>
142                 </child>
143               </widget>
144             </child>
145             <child>
146               <widget class="GtkLabel" id="label1">
147                 <property name="visible">True</
                      property>
148                 <property name="label" translatable="
                      yes">login info</property>
149               </widget>
150               <packing>
151                 <property name="type">label_item</
                      property>
152               </packing>
153             </child>
154           </widget>
```

```
155             <packing>
156               <property name="position">0</property>
157             </packing>
158           </child>
159           <child>
160             <widget class="GtkHBox" id="hbox6">
161               <property name="visible">True</property>
162               <child>
163                 <widget class="GtkButton" id="connect">
164                   <property name="label">Connect</
                        property>
165                   <property name="visible">True</
                        property>
166                   <property name="can_focus">True</
                        property>
167                   <property name="receives_default">True
                        </property>
168                   <signal name="clicked" handler="
                        connect_to_ui"/>
169                 </widget>
170                 <packing>
171                   <property name="position">0</property>
172                 </packing>
173               </child>
174               <child>
175                 <widget class="GtkButton" id="disconnect
                      ">
176                   <property name="label" translatable="
                        yes">Disconnect</property>
177                   <property name="visible">True</
                        property>
178                   <property name="can_focus">True</
                        property>
179                   <property name="receives_default">True
                        </property>
180                   <signal name="clicked" handler="
                        disconnect"/>
181                 </widget>
182                 <packing>
183                   <property name="position">1</property>
184                 </packing>
```

```
185             </child>
186           </widget>
187           <packing>
188             <property name="expand">False</property>
189             <property name="position">1</property>
190           </packing>
191         </child>
192         <child>
193           <widget class="GtkHSeparator" id="
                hseparator1">
194             <property name="visible">True</property>
195           </widget>
196           <packing>
197             <property name="expand">False</property>
198             <property name="position">2</property>
199           </packing>
200         </child>
201         <child>
202           <widget class="GtkHBox" id="hbox3">
203             <property name="visible">True</property>
204             <child>
205               <widget class="GtkLabel" id="messages_lb
                    ">
206                 <property name="visible">True</
                      property>
207                 <property name="label" translatable="
                      yes">Messages : </property>
208               </widget>
209               <packing>
210                 <property name="expand">False</
                      property>
211                 <property name="position">0</property>
212               </packing>
213             </child>
214             <child>
215               <placeholder/>
216             </child>
217           </widget>
218           <packing>
219             <property name="expand">False</property>
220             <property name="position">3</property>
```

```
221             </packing>
222           </child>
223           <child>
224             <widget class="GtkHBox" id="hbox4">
225               <property name="visible">True</property>
226               <child>
227                 <widget class="GtkButton" id="
                      create_new_bt">
228                   <property name="label" translatable="
                        yes">New Server</property>
229                   <property name="visible">True</
                        property>
230                   <property name="can_focus">True</
                        property>
231                   <property name="receives_default">True
                        </property>
232                   <signal name="clicked" handler="
                        newServer"/>
233                 </widget>
234                 <packing>
235                   <property name="position">0</property>
236                 </packing>
237               </child>
238               <child>
239                 <widget class="GtkButton" id="remove_bt"
                      >
240                   <property name="label" translatable="
                        yes">Remove</property>
241                   <property name="visible">True</
                        property>
242                   <property name="can_focus">True</
                        property>
243                   <property name="receives_default">True
                        </property>
244                   <signal name="clicked" handler="remove
                        "/>
245                 </widget>
246                 <packing>
247                   <property name="position">1</property>
248                 </packing>
249               </child>
```

```
250             <child>
251               <widget class="GtkButton" id="remove_all
                     ">
252                 <property name="label" translatable="
                       yes">Remove All</property>
253                 <property name="visible">True</
                       property>
254                 <property name="can_focus">True</
                       property>
255                 <property name="receives_default">True
                       </property>
256               </widget>
257               <packing>
258                 <property name="position">2</property>
259               </packing>
260             </child>
261           </widget>
262           <packing>
263             <property name="expand">False</property>
264             <property name="position">4</property>
265           </packing>
266         </child>
267         <child>
268           <widget class="GtkHSeparator" id="
                 hseparator5">
269             <property name="visible">True</property>
270           </widget>
271           <packing>
272             <property name="expand">False</property>
273             <property name="position">5</property>
274           </packing>
275         </child>
276         <child>
277           <placeholder/>
278         </child>
279       </widget>
280     </child>
281   </widget>
282 </glade-interface>
```

cp.glade

Βιβλιογραφία

[1] Lambros Lambrinos and Fotis Georgatos, *A Case for Multimedia Streaming over the Grid Infrastructure* (Springer-Verlag MA, 2008),

[2] Petr Holub, *Network and Grid Support for Multimedia Distribution and Processing* http://frakira.fi.muni.cz/ hopet/thesis/thesis-screen.pdf

[3] Geoffrey Fox, Galip Aydin, Harshawardhan Gadgil, Shrideep Pallickara, Marlon Pierce and Wenjun Wu , *Management of Real-Time Streaming Data Grid Services* Springer Berlin / Heidelberg 2005)

[4] Brian Hall *Beej's Guide to Network Programming*

(http://beej.us/guide/bgnet)

[5] Mark Pilgrim *Dive Into Python* http://diveintopython.org

[6] E. Magli and P. Frossard, *An overview of network coding for multimedia streaming* (ISBN : 978-1-4244-4290-4)

[7] http://nadiana.com/python-pickle-insecure

[8] http://docs.python.org/library/select.html

[9] http://www.brighthub.com/environment/green-computing/articles/68785.aspx

[10] http://www.gridpp.ac.uk/docs/Gridpp.pdf

[11] http://artemis.cslab.ntua.gr/el_thesis/artemis.ntua.ece/DT2009-0316/DT2009-0316.pdf

[12] http://www.cslab.ntua.gr/diplom/files/grid-sec-06_07.pdf

[13] http://www-numi.fnal.gov/offline_software/srt_public_context /GridTools/docs/glue_schema.html

[14] http://www-numi.fnal.gov/offline_software/srt_public_context /GridTools/docs/jobs_jdl.html

[15] http://en.wikipedia.org/wiki/Supercomputer

[16] http://en.wikipedia.org/wiki/Grid_computing

[17] http://www.gridcafe.org/what-is-the-grid.html

[18] Ian Foster, Carl Kesselman, Steven Tuecke *The Anatomy of the Grid, Enabling Scalable Virtual Organizations* http://www.globus.org/alliance/publications/papers/anatomy.pdf