Author : Gaylord CHERENCEY

Supervisor : Marco Clemencic

Department : PH/LCB

Project Title:

Analysis of access to databases for the implementation of a Squid proxy on LHCb experiment

Table of Contents

Thanks

I would like to thank Marco Clemencic the supervisor of this project. He has been really welcoming and pedagogical with me during the realization of the project by explaining step by step all the features I had to implement. Moreover I would like to highlight the efficiency of all the Summer Student staff in order to keep us updated about the lectures, visits and the events happening during the summer. Finally I would like to thank all the summer students for the moral support and the punctual help with my code, especially Ben Burdick for his editing of this report.

Table of figures

Introduction

The LHCb experiment collects collision event data from the detector to be then translated into a form that is suitable for physics analysis. This translation process, called *reconstruction*, requires information on the status of the detector. The status of the detector is described in *condition data* which are stored on SQL databases.

Currently, we distribute the condition data to the Grid Tier-1 Computing Centres, where the reconstruction processes run, in the form of SQLite files. The big advantage of SQLite files over SQL servers is that there is no configuration needed to access those files. The disadvantage is that those files are growing in size (because of new condition data added to them), and soon we will reach the point where they are too big for the current distribution model.

An alternative approach to SQLite is to use a central Oracle Database server that we access through standard web proxy servers and a Frontier [1] server, an HTTP server developed by CMS and Fermilab to allow HTTP access to Oracle databases. The Frontier-based set-up allows for a more dynamic and efficient and scalable distribution of the condition data, than the plain Oracle or SQLite.

The aim of the project is to extract, from logfiles of reconstruction jobs run on the grid, the access patterns to the condition data at the Tier-1s. These access patterns are then used to estimate the best parameters to be used in the configuration of the HTTP proxy servers that will be used in conjunction with Frontier/Oracle.

The LHCb Computing Model [2] prescribes that reconstruction jobs are to be run at the 7 Tier-1 Centres collaborating with the LHCb experiment: CERN, NIKHEF, IN2P3, PIC, CNAF, and GRIDKA. The reconstruction jobs require access to the condition data that describe the status of the detector, which are currently distributed as SQLite files as shown in Figure 1.

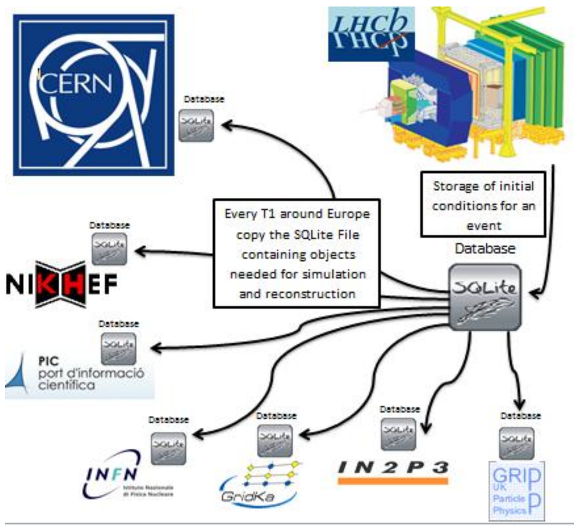

CERN
LHCb
Database
Storage of initial conditions for an event
Database
Every T1 around Europe copy the SQLite File containing objects needed for simulation and reconstruction
SQLite
NIKHEF
PIC port d'informació científica
INFN
GridKa
IN2P3
GRIDP UK Particle Physics

Figure 1 : Scheme of the deployment of condition data via SQLite files.

The SQLite files we use are growing in size, and soon it will not be practical anymore to distribute them. We are planning to use an alternative approach is based on the Frontier technology

[1], developed by the CMS experiment and the Fermilab laboratory, which allows access to an Oracle Database server via HTTP requests.

Although the conversion between SQL queries and HTTP requests implies some overhead, it's easy to associate regular proxy servers (like Squid [3]) to the Frontier server to cache the queries and the results (see Figure 2).

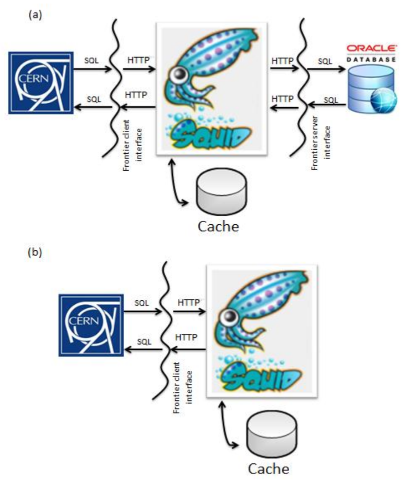

(a)
SQL
HTTP
HTTP
SQL
HTTP
SQL
HTTP
SQL
Frontier client interface
Frontier server interface
ORACLE DATABASE
Cache
(b)
SQL
HTTP
SQL
HTTP
Frontier client interface
Cache

Figure 2 : Access to an Oracle Database server via Frontier+Squid. If the result of the query is not in the Squid cache, the query is forwarded to Oracle (a), but when the data is available in the cache, it is returned immediately (b).

Using Frontier with a network of Squid proxy servers, we can replace the scheme in Figure 2 with the one in Figure 3.

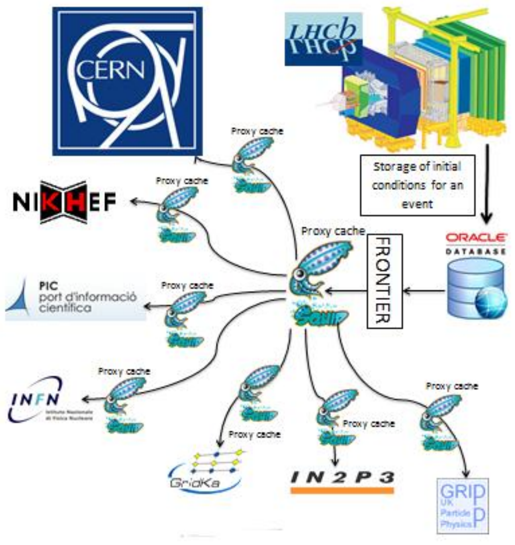

CERN
LHCb
Storage of initial conditions for an event
FRONTIER
ORACLE DATABASE
Proxy cache
NIKHEF
PIC port d'informació científica
INFN
GridKa
IN2P3
GRIDPP

Figure 3 : Scheme of the deployment of condition data using Frontier and Squid servers at all Tier-1s.

For an optimal use of the Frontier+Squid setup, we need the best tuning of two main parameters in the configuration: the granularity of the query (or *grouping factor*, i.e. how many jobs will make the same request to the proxy) and the maximum time the result of a query can stay in the proxy cache.

We used the following approach to find the optimal tuning. First we analyzed the log files of real reconstruction jobs run at the Tier-1s in order to extract the access pattern to the database. Then we used the collected information simulating the behavior of the proxy servers with several different configurations, to find the best parameters.

Presentation of the company

CERN, the European Organization for Nuclear Research, is one of the world's largest and most respected centers for scientific research. Its business is fundamental physics, finding out what the Universe is made of and how it works. At CERN, the world's largest and most complex scientific instruments are used to study the basic constituents of matter — the fundamental particles. By studying what happens when these particles collide, physicists learn about the laws of Nature.

The instruments used at CERN are particle accelerators and detectors. Accelerators boost beams of particles to high energies before they are made to collide with each other or with stationary targets. Detectors observe and record the results of these collisions. [4]

Three majors experiments are installed on the LHC, the largest accelerator, (27 kilometers in circumference), which are working on different fields of physics. CMS [5] and ATLAS [6] are looking for new particles such as the Higgs Boson to prove the coherency of the Standard Model, LHC-b [7] measures certain B-hadron qualities such as asymmetries or CP violations and ALICE [8] is producing quark–gluon plasma.

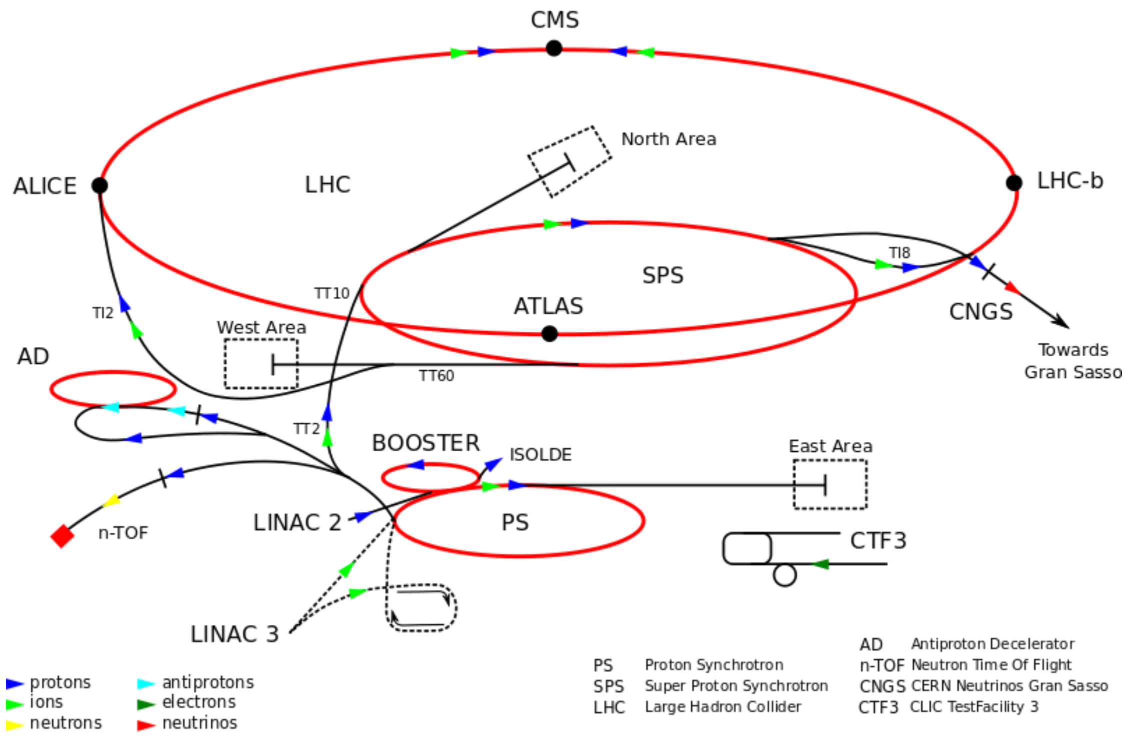


Figure 4 : Map of the CERN accelerator complex [9]

Founded in 1954, the CERN Laboratory sits astride the Franco–Swiss border near Geneva. It was one of Europe's first joint ventures and now has 20 Member States.

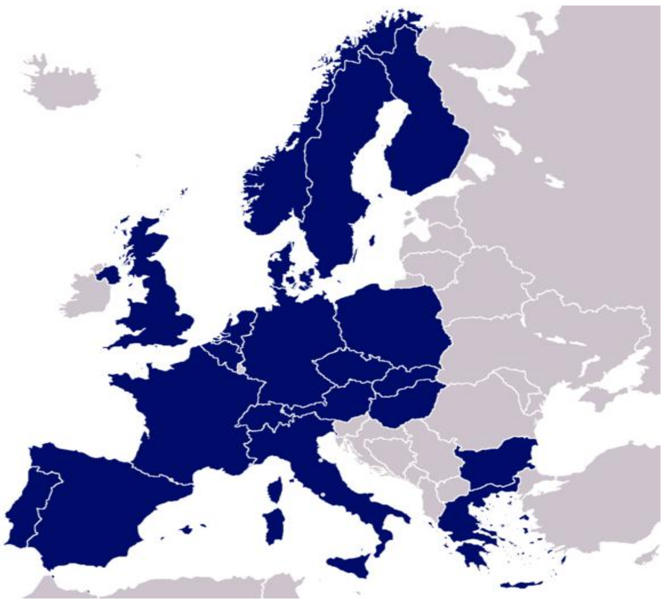


Figure 5 : CERN member states in 1999 (map borders from 2008) [10]

CERN employs fewer than 2400 people. Some 10,000 visiting scientists, half of the world's particle physicists, come to CERN for their research. They represent 608 universities and 113 nationalities.

Scientists from some 608 institutes and universities around the world use CERN's facilities. See below the international repartition of scientists using CERN's facilities.

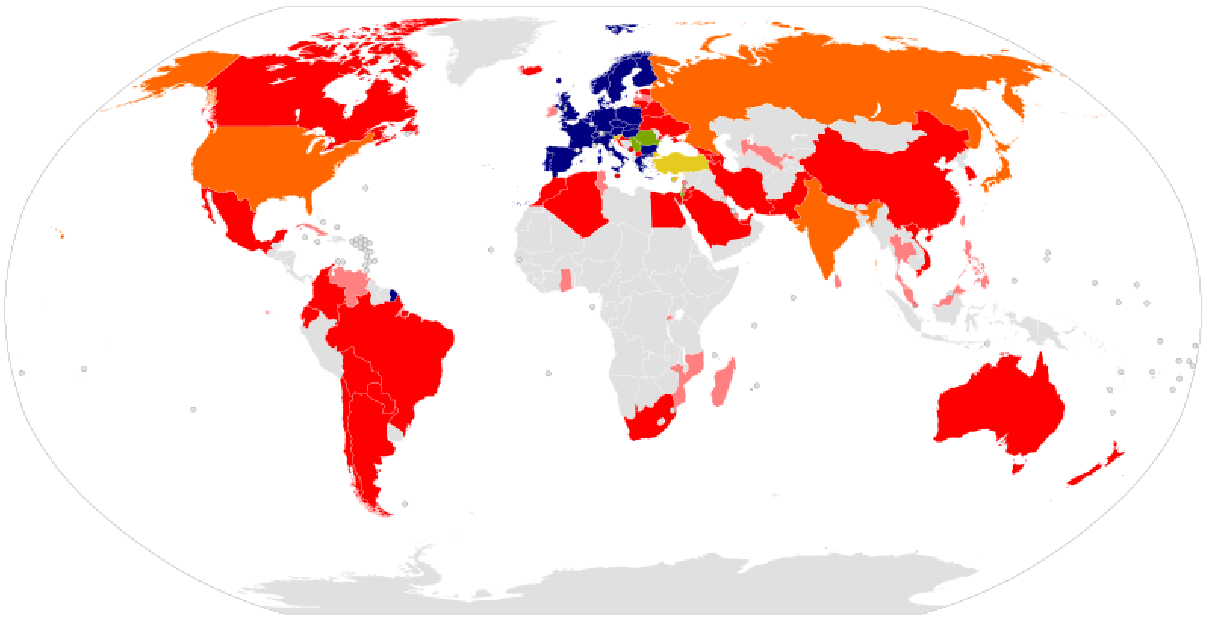


Figure 6 : Modified BlankMap-World-Microstates to show international relations of CERN. [11]

Several important achievements in particle physics have been made during experiments at CERN. They include:

- 1973: The discovery of neutral currents in the Gargamelle bubble chamber.
- 1983: The discovery of W and Z bosons in the UA1 and UA2 experiments.
- 1989: The determination of the number of light neutrino families at the Large Electron–Positron Collider (LEP) operating on the Z boson peak.
- 1995: The first creation of antihydrogen atoms in the PS210 experiment.
- 1999: The discovery of direct CP violation in the NA48 experiment.
- 2010: The isolation of 38 atoms of antihydrogen
- 2011: Maintaining antihydrogen for over 15 minutes
- 2012: A boson with mass around 125 GeV consistent with long-sought Higgs boson. [12]

CERN is divided into several departments working in synergy to keep a level of innovation and high accuracy in the results.

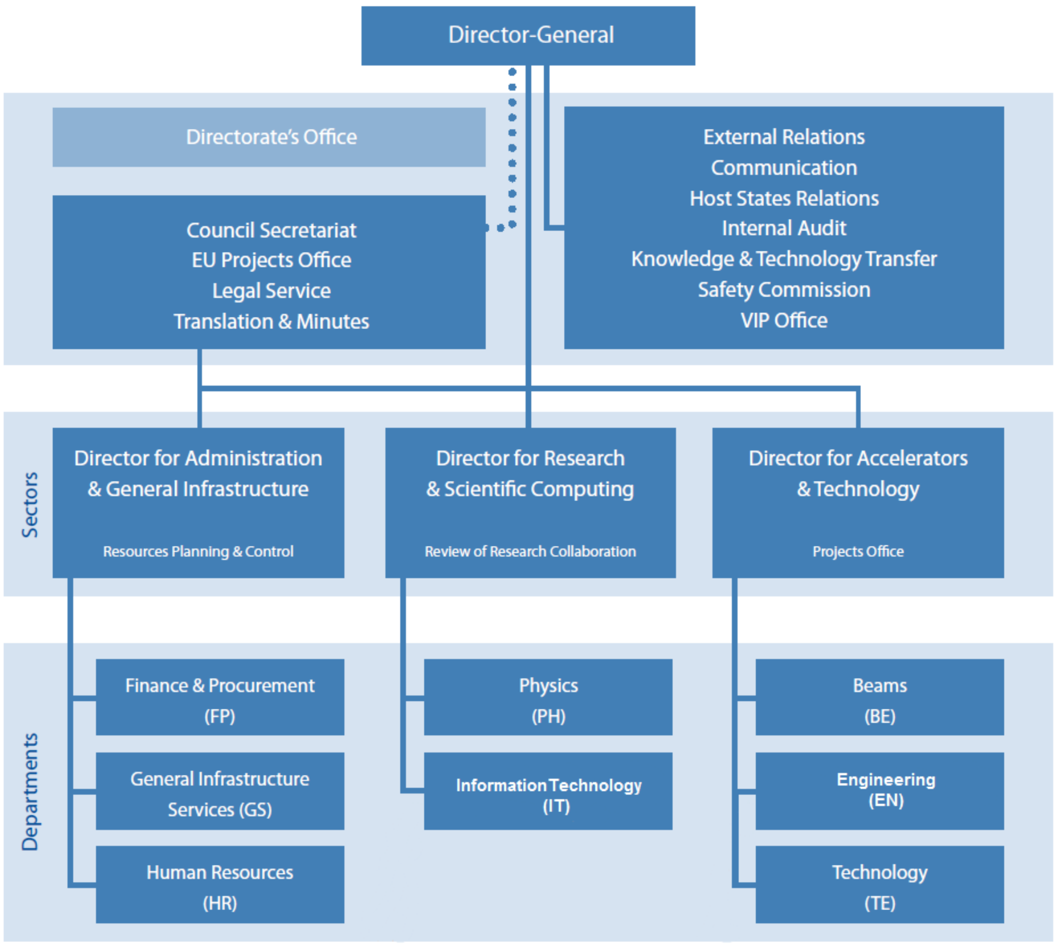

Director-General
Directorate's Office
Council Secretariat
EU Projects Office
Legal Service
Translation & Minutes
External Relations
Communication
Host States Relations
Internal Audit
Knowledge & Technology Transfer
Safety Commission
VIP Office
Sectors
Director for Administration & General Infrastructure
Resources Planning & Control
Director for Research & Scientific Computing
Review of Research Collaboration
Director for Accelerators & Technology
Projects Office
Departments
Finance & Procurement (FP)
General Infrastructure Services (GS)
Human Resources (HR)
Physics (PH)
Information Technology (IT)
Beams (BE)
Engineering (EN)
Technology (TE)

Figure 7 : CERN's structure

This project took place in the PH department, and more precisely in the LHCb branch of this department.

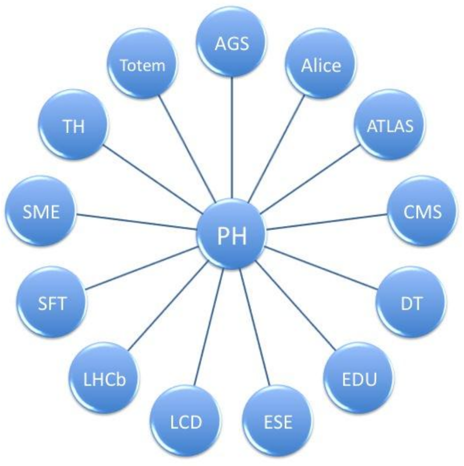


Figure 8 : PH department branches [13]

It was realized over a period of 8 weeks in the summer student program. The main idea comes from LHCb core software group and was developed in Python [14] 2.6 with Eclipse [15].

Work done

Extraction of the access pattern

The log files of all the LHCb reconstruction jobs run on the grid can be found in compressed archives stored on the CASTOR storage system at CERN, each archive containing several archives, one per job (see Figure 4).

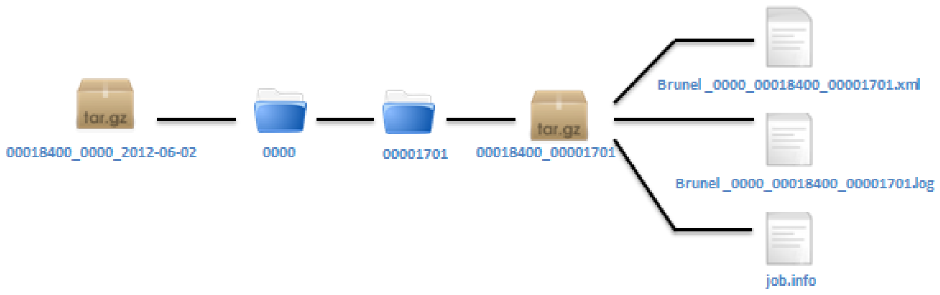


Figure 9 : Scheme of the content of the archives of the job logfiles. "Brunel" is the name of the reconstruction application used by LHCb.

From the archives of the individual jobs, we have to extract three pieces of information:

- in which Tier-1 site the job was run (to know which proxy server it would have contacted)
- which input data the object used (for which condition data it used from the database)
- when it connected to the database (for the time of the request to the proxy)

These details have to be extracted from different files inside a compressed archive that is inside another compressed archive a few GB in size.

The simplest approach would be to unpack the outer archives, then unpack the inner archives to extract the data from the files inside. This approach would need several tens of GB of temporary disk space, because the compression ratio is of the order of a factor 10-20. Moreover, the extraction of the data cannot start until the lengthy unpacking is completed.

The slightly more complex approach we chose was to write a script in Python [4] that extracts the data we need directly from the compressed archives, unpacking in memory only the small chunks we need to access. The main advantages of this approach are that we do not need temporary disk space (except for the archives themselves) and we could test the extraction process quickly on only a small part of the archive.

To process it we choose to use a python module called Tarfile. The tarfile module includes several objects (like TarFile) which provide an interface to a tar archive. A tar archive is a sequence of blocks. An archive member (a stored file) is made up of a header block followed by data blocks. It is possible to store a file in a tar archive several times. Each archive member is represented by a TarInfo object [16]. Thanks to those objects we can go through a tarfile and see the detail of every TarInfo object. Aside from storing all required attributes of a file (like file type, size, time, permissions, owner etc.), it provides some useful methods to determine its type. It does not contain the file's data itself so if we loop over every TarInfo object contained in the tar archive, we can easily find the documents needed and *then* extract the data.

Like we said we need the information about where the job was run, which input data the object used and when it connected to the database.

The site and the time are extracted in the same way. After finding the needed files we extract them from the tar archive. Then we open the file and thanks to a regex (a function coming from the *re* module that looks at the line given and tries to match a certain pattern) (Annexe 1.1 and Annexe 1.2).

The case of the input file is more complicated because this information is stored in an XML (eXtensible Markup Language) file. We actually can get this information in two ways, because here we know that the information is an attribute of the file tag, which is between input tags like in the example below (see in italic the information needed)

<input>

<file status="full" name="LFN:/lhcb/data/2012/RAW/FULL/LHCb/COLLISION12/114751/114751_0000000031.raw" GUID="">60879

</file>

</input>

So we can use the XML Tree module. This module will go throw the entire file and will store in memory all the tags and attributes so it will be really easily to get the information (see the entire function at Annexe 1.3). The XML file is about 14000 lines of code so it takes a long time to go throw the file.

After some research we found that Python provides a module called *sax.xml.* In this module (Annexe 1.4) you just have to wrote a class (here called `MySaxDocumentHandler:` Annexe 2) in order to define in which tag you have to look for. Here we know the layout so the `MySaxDocumentHandler` will go throw all the tag and will find the special layout given, then we will be able to get the attribute from the file tag. This method will definitely save a lot of execution time because we already know that this information is always at the beginning of the file so we get all the information about input files in no time. We need the have an efficient code because here we talking about the extraction of information coming from thousands of files.

Since the extraction is anyway a long operation, the extracted data is stored in a SQLite database (with the help of the library SQLAlchemy [117]) to be used as input for the analysis step. The schema of the database is represented in Figure 10.

SQLAlchemy has some tools which help us to map objects with a database table. Object-relational mapping (ORM) is a computer programming technique that creates the illusion of an object oriented database from a relational database, by defining correspondences between the database and the objects of the language used. (Annexe 3).

This type of storage permits us to use much simpler requests to the database. Moreover if the user only wants to extract the data and then write his own analysis script, he can, because the extraction and the analysis are done in two separate scripts.

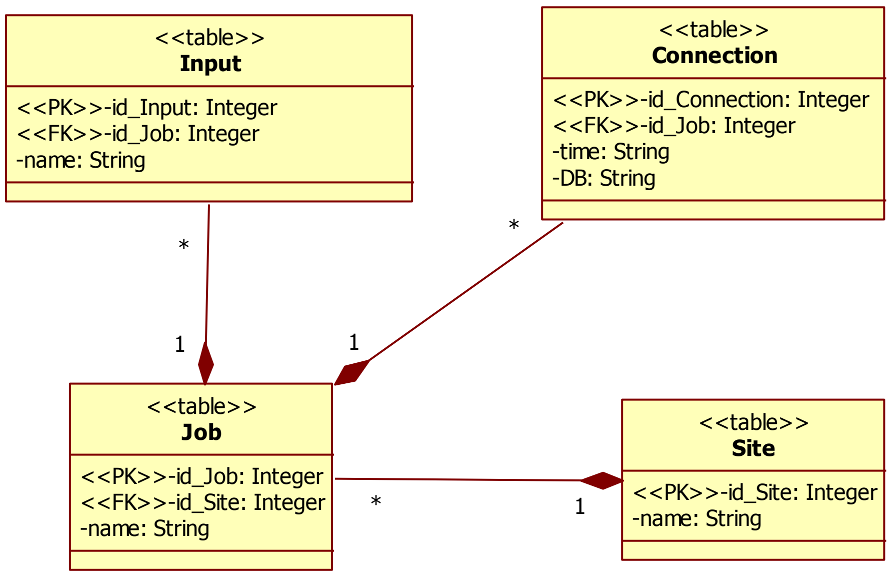


Figure 10 : Schema of the SQLite database used to store the extracted data.

Analysis of the behavior of simulated proxy servers

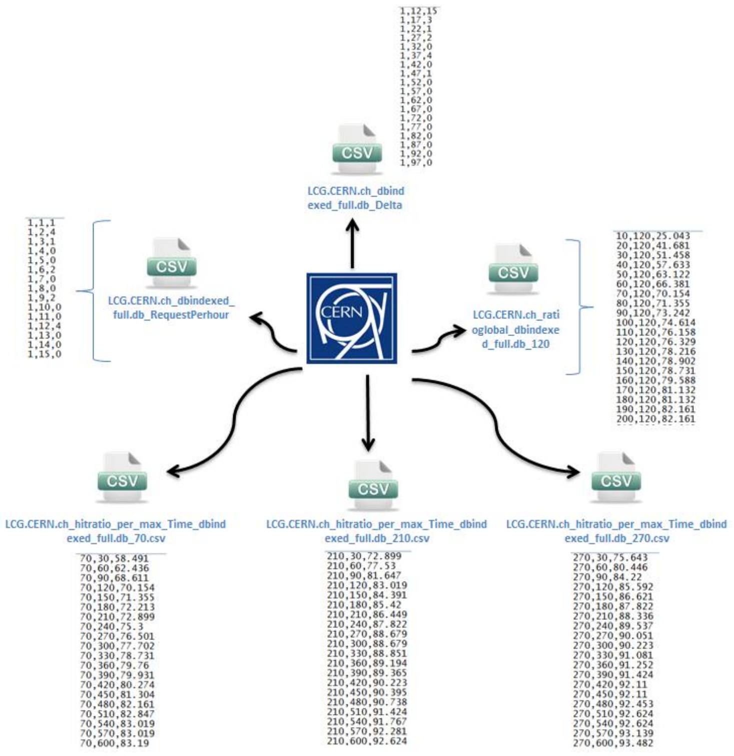
The data collected allows us to study different aspects of the access patterns.

The main aspect we analyzed is the *hit ratio* of the Squid proxy server, defined as the number of requests served from the cache divided by the total number of requests, for different values of the configuration parameters. Then to better understand the impact of the requests on the proxy, we also measured the number of requests per hour per site and the time between two identical requests on each site, for a fixed grouping factor. The requests per hour can be used to estimate the amount of data the proxy server needs to be able to serve every hour; so that we can decide how many servers we should foresee using. Finally the time between two identical requests helps us to better understand, together with the hit ratio, how likely it is that two jobs are making the same request within a given time.

To determine the access pattern and calculate the indicators described below we developed a generic proxy class (Annexe 4). This class will simulate the behavior of a real proxy (rules of object's replacement).

As you can see, for the time between two requests and the number of requests per hour, we used histograms in order to store data and create the CSV files more easily. This is the purpose of the Histogram class gets the data bin and create CSV files.

Finally we just have to create several instances of proxies in order to match the structure presented on Figure 3 (one per site plus a global one). When we run the script several CSV files will be created for every site (Figure 11) and we will have a global view of the pattern.


CSV
LCG.CERN.ch_dbindexed_full.db_Delta
CSV
LCG.CERN.ch_dbindexed_full.db_RequestPerhour
CSV
LCG.CERN.ch_ratioglobal_dbindexed_full.db_120
CERN
CSV
LCG.CERN.ch_hitratio_per_max_Time_dbindexed_full.db_70.csv
CSV
LCG.CERN.ch_hitratio_per_max_Time_dbindexed_full.db_210.csv
CSV
LCG.CERN.ch_hitratio_per_max_Time_dbindexed_full.db_270.csv

```
1,12,15
1,17,3
1,22,1
1,27,2
1,32,0
1,37,4
1,42,0
1,47,1
1,52,0
1,57,0
1,62,0
1,67,0
1,72,0
1,77,0
1,82,0
1,87,0
1,92,0
1,97,0
```

```
1,1,1
1,2,4
1,3,1
1,4,0
1,5,0
1,6,2
1,7,0
1,8,0
1,9,2
1,10,0
1,11,0
1,12,4
1,13,0
1,14,0
1,15,0
```

```
10,120,25.043
20,120,41.681
30,120,51.458
40,120,57.633
50,120,63.122
60,120,66.381
70,120,70.154
80,120,71.355
90,120,73.242
100,120,74.614
110,120,76.158
120,120,76.329
130,120,78.216
140,120,78.902
150,120,78.731
160,120,79.588
170,120,81.132
180,120,81.132
190,120,82.161
200,120,82.161
```

```
70,30,58.491
70,60,62.436
70,90,68.611
70,120,70.154
70,150,71.355
70,180,72.213
70,210,72.899
70,240,75.3
70,270,76.501
70,300,77.702
70,330,78.731
70,360,79.76
70,390,79.931
70,420,80.274
70,450,81.304
70,480,82.161
70,510,82.847
70,540,83.019
70,570,83.019
70,600,83.19
```

```
210,30,72.899
210,60,77.53
210,90,81.647
210,120,83.019
210,150,84.391
210,180,85.42
210,210,86.449
210,240,87.822
210,270,88.679
210,300,88.679
210,330,88.851
210,360,89.194
210,390,89.365
210,420,90.223
210,450,90.395
210,480,90.738
210,510,91.424
210,540,91.767
210,570,92.281
210,600,92.624
```

```
270,30,75.643
270,60,80.446
270,90,84.22
270,120,85.592
270,150,86.621
270,180,87.822
270,210,88.336
270,240,89.537
270,270,90.051
270,300,90.223
270,330,91.081
270,360,91.252
270,390,91.424
270,420,92.11
270,450,92.11
270,480,92.453
270,510,92.624
270,540,92.624
270,570,93.139
270,600,93.482
```

Figure 11 : Example of the CSV created and needed for the plots (here only for CERN)

Then we plot the data in the CSV files thanks to GNUPlot [18] and a bash script (Annexe 5). The main goal is to choose the best parameters to set into the proxies, so a plot is always a good way to see the behavior.

Firstly let's analyze the results we have for the hit ratio:

Here we just change the max_Time, that is, the maximum time an object stays in the cache before being considered "stale".

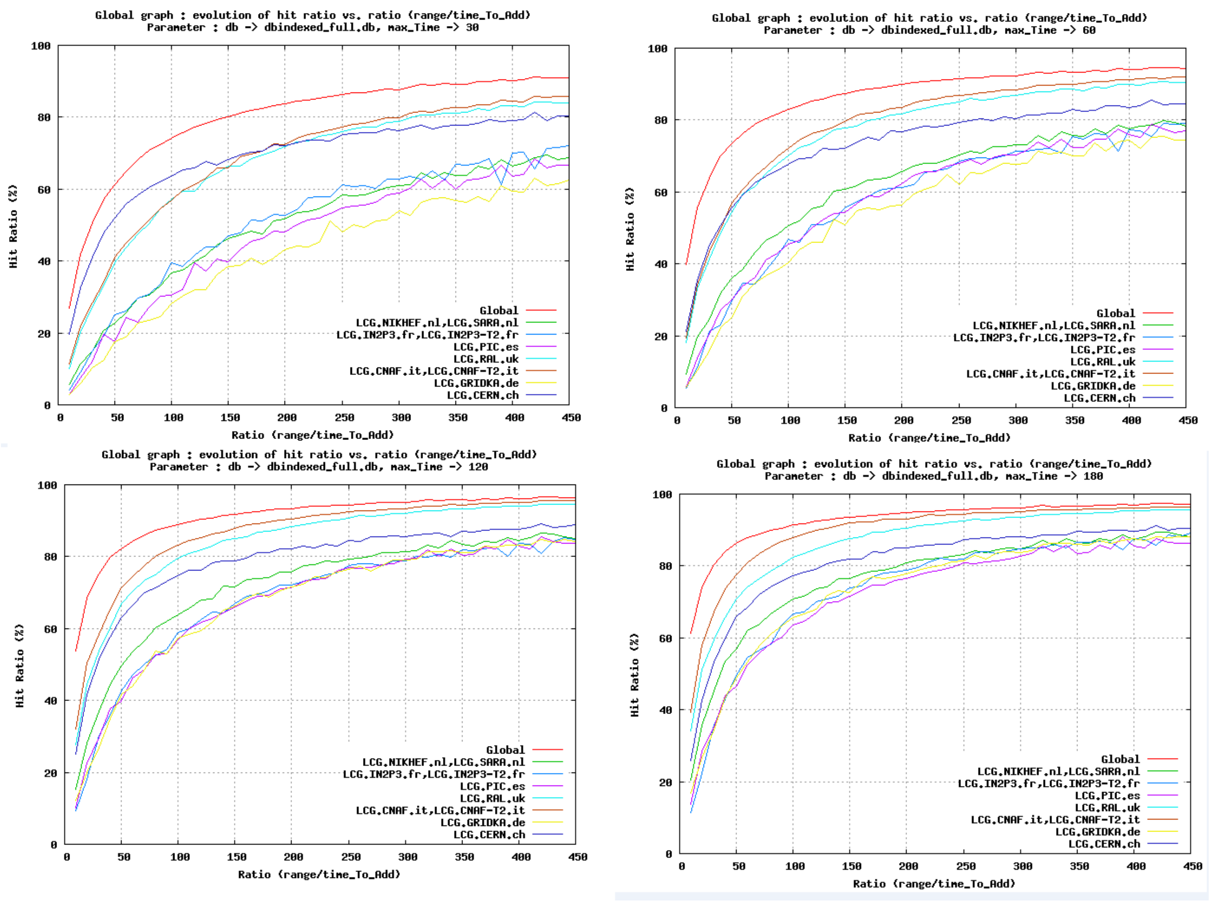


Figure 12 : Plots representing the number of requests served from the cache divided by the total number of requests for different max_time

You see in horizontal axis on the Figure 12 that the ratio is the range divided by the time to add. We also call it the grouping factor because this permits us to group several objects together which is more realistic regarding the actual operation.

List of inputs	Time (30s)	Range (1h)
N. 31	0	0 - 3599
N. 12	30	0 - 3599
N. 43	60	0 - 3599
…	…	…
N. 3	4350	3599 - 7199

Here we know empirically that the range, which means the time during which the objects are requested, is about an hour and the time to add is about 30 seconds.

So after shuffling the inputs list, in order to keep a more realistic simulation, we map the time with the range by saying that if the time is in a one-hour interval then this interval becomes the URL (Uniform Resource Locator) representing a single group object.

To have a more efficient proxy we must have a hit ratio above 80% and a grouping factor not too high. So if we look at the plots we can eliminate those with a time_max of 30 and 60 minutes for the reasons given above. Moreover the difference between a max_time of 120 and 180 isn't really important. In fact in this interval the value of the hit ratio evolves very little so it's better to choose a max_time of 120 in order to keep a high accuracy during the user's simulation.

For the ratio we see that the hit ratio increases rapidly at the beginning but at the end it stabilizes so it's cleverer to choose the ratio which is above the curve. If we do this analyze on the plot with the max_time equal to 120 we see that the best ratio is 210.

We confirm this result by plotting the evolution of hit ratio vs max_time. So on the Figure 12 we fix three different ratios and then we look at the evolution of the hit ratio regarding the time_max. The three different values chosen are 70, 210 and 270. As we said in the last paragraph we determine that the most relevant time_max to choose is 120 so we just have to look this particular value.

As a conclusion on this plot we confirm that the best ratio to choose is 210 because the gain of hit ratio between the two is very small.

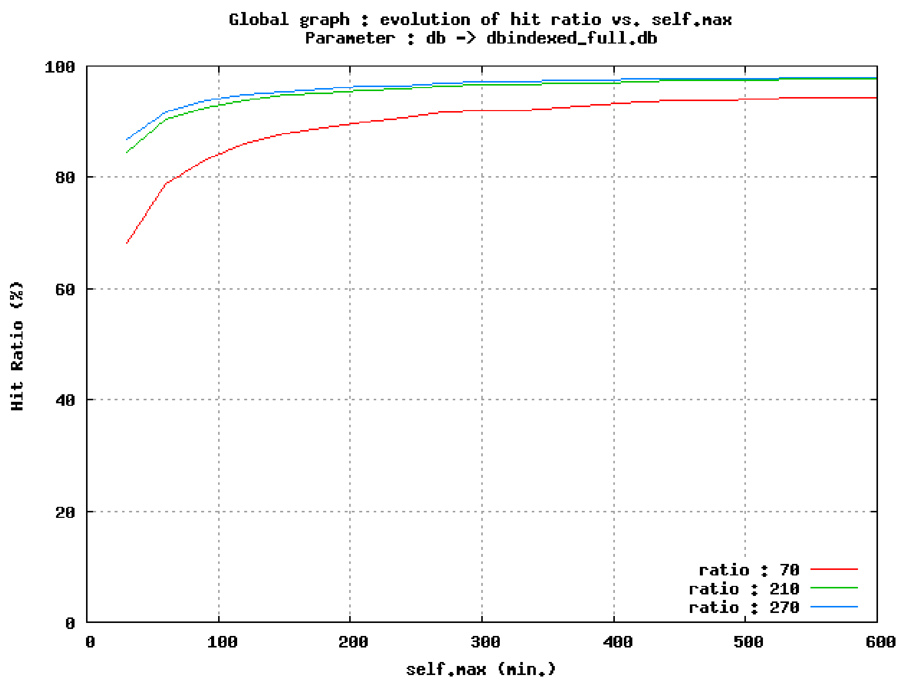


Figure 12: Plot representing the evolution of the hit ratio for different values of the configuration

The other aspect we look for is the number of requests per hour (Figure 13). Here we see that most of the requests are between 70 and 90 hours after the first request so we now know that the server have to be efficient during this particular period.

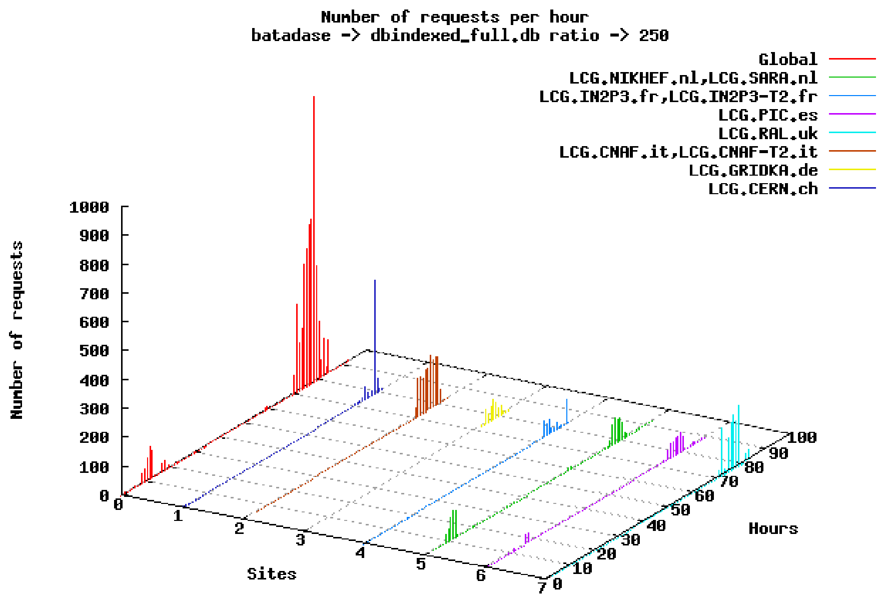


Figure 13 : Plot representing the number of requests per hour per site

Finally we take a look at the time between two identical requests (Figure 14). This helps us to better understand, together with the hit ratio, how likely it is that two jobs are making the same request within a given time. We see that the frequency of requesting is very important in a short period of time because the time between two requests is more likely every 10 minutes.

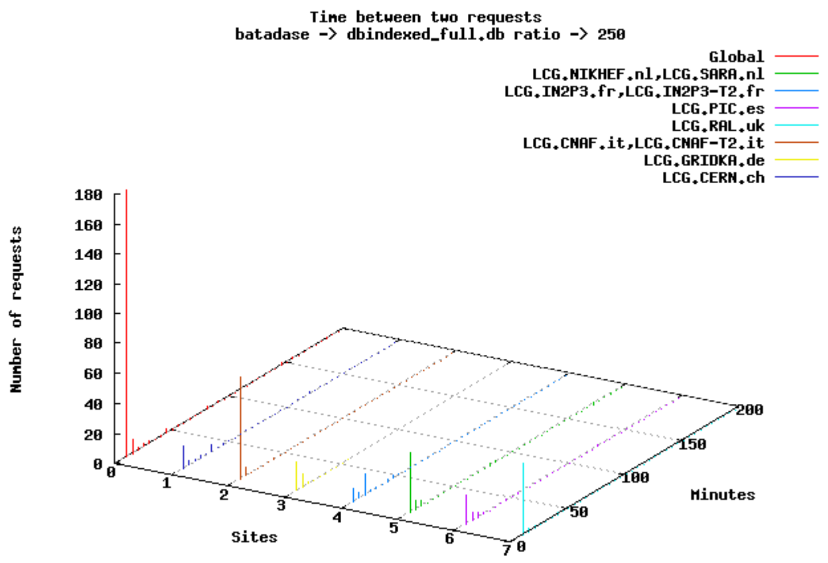


Figure 14 : Plot representing the time between two identical requests on each site and for a fixed grouping factor

To sum up everything, we run the script graph.sh which takes some parameters as input:

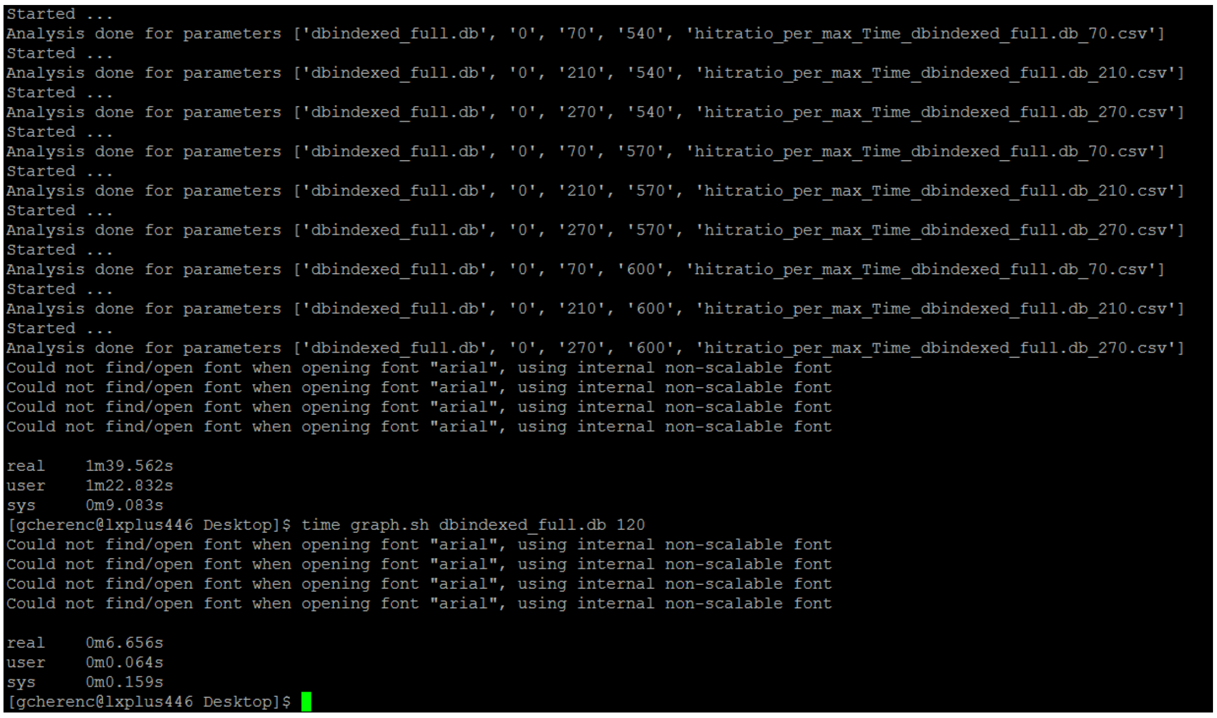
```
Usage: graph.sh db max_Time

       db -> the name of your database file
       max_time -> the maximum time you want your object into the proxy cache
```

Then the script will loop over the analysis.py script and change the ratio parameter at each iteration. This script also takes several parameters as entry:

```
Usage: analysis.py db seed_Number ratio_Number max_Time ratio_CSV [options]: Use
the -h option to get help

       db -> the name of your database file

       seed_Number -> certain way to shuffle the input data

       ratio_Number -> parameter to group the inputs

       max_time -> the maximum time you want your object into the proxy cache

       ratio_CSV -> name of the CSV File created
```

So the looping over the analysis script will give us all the points needed for the plotting part. In order to improve the process we add a condition saying that if the user wants to execute the script for a second time the script won't do the loop but it will just open the previously created graphics .

You see on the Figure 15 that we save a lot of computing time. It take 1 minute and 39 seconds to run the script for the first time but it only takes 6 seconds for the second time.



```
Started ...
Analysis done for parameters ['dbindexed_full.db', '0', '70', '540', 'hitratio_per_max_Time_dbindexed_full.db_70.csv']
Started ...
Analysis done for parameters ['dbindexed_full.db', '0', '210', '540', 'hitratio_per_max_Time_dbindexed_full.db_210.csv']
Started ...
Analysis done for parameters ['dbindexed_full.db', '0', '270', '540', 'hitratio_per_max_Time_dbindexed_full.db_270.csv']
Started ...
Analysis done for parameters ['dbindexed_full.db', '0', '70', '570', 'hitratio_per_max_Time_dbindexed_full.db_70.csv']
Started ...
Analysis done for parameters ['dbindexed_full.db', '0', '210', '570', 'hitratio_per_max_Time_dbindexed_full.db_210.csv']
Started ...
Analysis done for parameters ['dbindexed_full.db', '0', '270', '570', 'hitratio_per_max_Time_dbindexed_full.db_270.csv']
Started ...
Analysis done for parameters ['dbindexed_full.db', '0', '70', '600', 'hitratio_per_max_Time_dbindexed_full.db_70.csv']
Started ...
Analysis done for parameters ['dbindexed_full.db', '0', '210', '600', 'hitratio_per_max_Time_dbindexed_full.db_210.csv']
Started ...
Analysis done for parameters ['dbindexed_full.db', '0', '270', '600', 'hitratio_per_max_Time_dbindexed_full.db_270.csv']
Could not find/open font when opening font "arial", using internal non-scalable font
Could not find/open font when opening font "arial", using internal non-scalable font
Could not find/open font when opening font "arial", using internal non-scalable font
Could not find/open font when opening font "arial", using internal non-scalable font

real    1m39.562s
user    1m22.832s
sys     0m9.083s
[gcherenc@lxplus446 Desktop]$ time graph.sh dbindexed_full.db 120
Could not find/open font when opening font "arial", using internal non-scalable font
Could not find/open font when opening font "arial", using internal non-scalable font
Could not find/open font when opening font "arial", using internal non-scalable font
Could not find/open font when opening font "arial", using internal non-scalable font

real    0m6.656s
user    0m0.064s
sys     0m0.159s
[gcherenc@lxplus446 Desktop]$ 
```

Figure 15 : Improvement of the graph.sh script

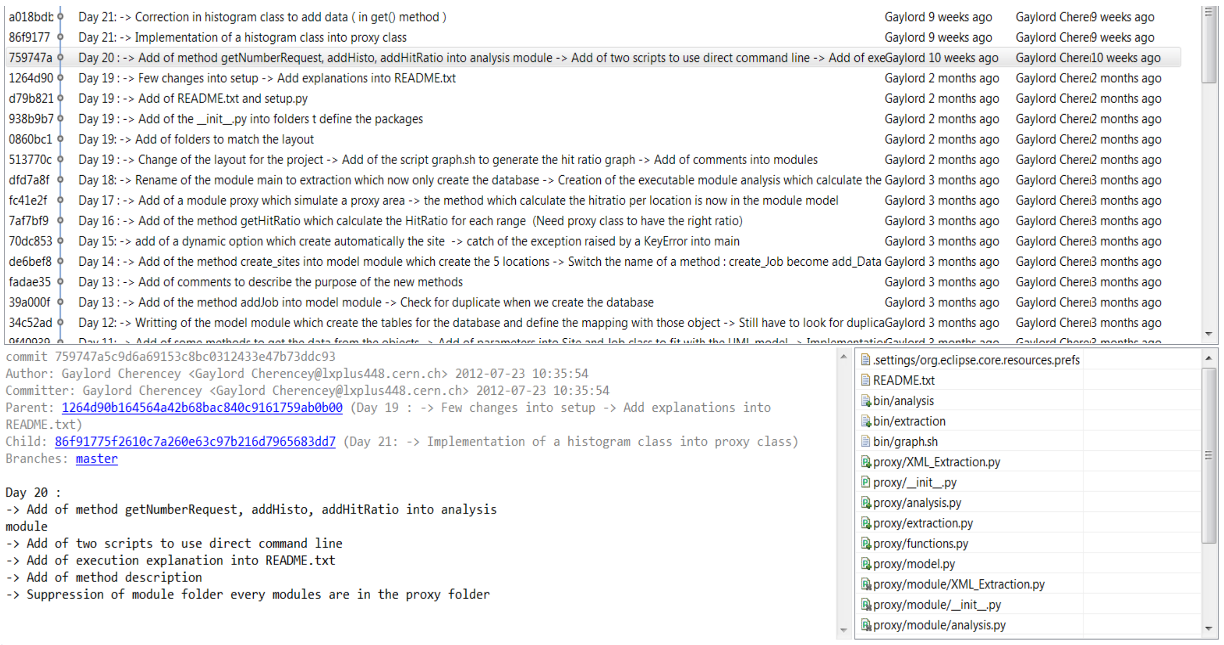
Documentation and versioning

Documentation is vital to any business, especially when your code has to be read or executed by another person. That's why writing a clear documentation is a key point when you work in a group with several members.

A "readme" file (Annexe 6) is here to explain how to run the script or set the environment. But the documentation is not only for users; it's also for other developers who want to add new features in the current code. So, it's compulsory to have clear comments in the code in order to help the future developer understand how the code works. You can see that the code developed in this project contains comments which explain the more difficult parts.

Another important aspect when you have to develop a project with a large number of scripts is to keep track of your different versions. For this project Git [19] was the main versioning plugin.

Git allows us to keep track of the various changes made to the code during at every saves. On the Figure 16 you see that we created a branch containing every version of the code with a short description of the changes.



```
a018bdb  Day 21: -> Correction in histogram class to add data ( in get() method )                                   Gaylord 9 weeks ago    Gaylord Chere9 weeks ago
86f9177  Day 21: -> Implementation of a histogram class into proxy class                                           Gaylord 9 weeks ago    Gaylord Chere9 weeks ago
759747a  Day 20 : -> Add of method getNumberRequest, addHisto, addHitRatio into analysis module -> Add of two scripts to use direct command line -> Add of exeGaylord 10 weeks ago   Gaylord Chere10 weeks ago
1264d90  Day 19 : -> Few changes into setup -> Add explanations into README.txt                                    Gaylord 2 months ago   Gaylord Chere2 months ago
d79b821  Day 19 : -> Add of README.txt and setup.py                                                                Gaylord 2 months ago   Gaylord Chere2 months ago
938b9b7  Day 19 : -> Add of the __init__.py into folders t define the packages                                     Gaylord 2 months ago   Gaylord Chere2 months ago
0860bc1  Day 19: -> Add of folders to match the layout                                                              Gaylord 2 months ago   Gaylord Chere2 months ago
513770c  Day 19 : -> Change of the layout for the project -> Add of the script graph.sh to generate the hit ratio graph -> Add of comments into modules   Gaylord 2 months ago   Gaylord Chere2 months ago
dfd7a8f  Day 18: -> Rename of the module main to extraction which now only create the database -> Creation of the executable module analysis which calculate the Gaylord 3 months ago   Gaylord Chere3 months ago
fc41e2f  Day 17 : -> Add of a module proxy which simulate a proxy area -> the method which calculate the hitratio per location is now in the module model   Gaylord 3 months ago   Gaylord Chere3 months ago
7af7bf9  Day 16 : -> Add of the method getHitRatio which calculate the HitRatio for each range  (Need proxy class to have the right ratio)   Gaylord 3 months ago   Gaylord Chere3 months ago
70dc853  Day 15: -> add of a dynamic option which create automatically the site  -> catch of the exception raised by a KeyError into main   Gaylord 3 months ago   Gaylord Chere3 months ago
de6bef8  Day 14 : -> Add of the method create_sites into model module which create the 5 locations -> Switch the name of a method : create_Job become add_Data Gaylord 3 months ago   Gaylord Chere3 months ago
fadae35  Day 13 : -> Add of comments to describe the purpose of the new methods                                     Gaylord 3 months ago   Gaylord Chere3 months ago
39a000f  Day 13 : -> Add of the method addJob into model module -> Check for duplicate when we create the database  Gaylord 3 months ago   Gaylord Chere3 months ago
34c52ad  Day 12: -> Writting of the model module which create the tables for the database and define the mapping with those object -> Still have to look for duplicaGaylord 3 months ago   Gaylord Chere3 months ago
9f40939  Day 11: -> Add of some methods to get the data from the objects -> Add of parameters into Site and Job class to fit with the UML model -> ImplementatioGaylord 3 months ago   Gaylord Chere3 months ago
```

```
commit 759747a5c9d6a69153c8bc0312433e47b73ddc93
Author: Gaylord Cherencey <Gaylord Cherencey@lxplus448.cern.ch> 2012-07-23 10:35:54
Committer: Gaylord Cherencey <Gaylord Cherencey@lxplus448.cern.ch> 2012-07-23 10:35:54
Parent: 1264d90b164564a42b68bac840c9161759ab0b00 (Day 19 : -> Few changes into setup -> Add explanations into
README.txt)
Child: 86f91775f2610c7a260e63c97b216d7965683dd7 (Day 21: -> Implementation of a histogram class into proxy class)
Branches: master

Day 20 :
-> Add of method getNumberRequest, addHisto, addHitRatio into analysis
module
-> Add of two scripts to use direct command line
-> Add of execution explanation into README.txt
-> Add of method description
-> Suppression of module folder every modules are in the proxy folder
```

```
.settings/org.eclipse.core.resources.prefs
README.txt
bin/analysis
bin/extraction
bin/graph.sh
proxy/XML_Extraction.py
proxy/__init__.py
proxy/analysis.py
proxy/extraction.py
proxy/functions.py
proxy/model.py
proxy/module/XML_Extraction.py
proxy/module/__init__.py
proxy/module/analysis.py
```

Figure 16 : Versionning of the code with Git

All this work may seem constraining, but it is essential if we want our project to last over time and be as adaptable as possible. But above all this work can make scripts very easy to maintain since you just have to reopen the old versions and the problem.

Tests

The design of a software project rime very often a phase of painful and tedious debugged. To overcome these phases, often repulsive, it is important to realize unit tests on the newly implemented functions. That's why we developed tests thanks to Python init-tests (Annexe 7) in order to see what happens in the code and what the outputs of the methods are.

Review

Technical

The challenge on this project was huge because my coding background was mostly in Java. In this project I had to use Python so the first step was to learn all the syntax and how to implement complex features. We usually say that a good developer is someone who manages to develop applications in different languages so the outcomes on this project were really important. Learning a new way to develop is always an opportunity to get good coding habits. Like my supervisor said to me early in the project "You can't hammer something if you don't know how to use a hammer". Those things were really interesting because I learned how to develop in a professional way (syntax, comments, documentation, tests, and versioning). Those habits will be useful in future internships. Conceiving an application always follows a certain pattern:

- Understand the needs of the client
- Look for different solutions
- Implement this solution and discuss it with the client
- Find a possible way to make the code more efficient and understandable to everyone in case of future updates
- Write the documentation

To sum up: when you develop an application, you don't develop it for yourself but you design it in a way that every user or future developer on this project can understand how it works. This is why the documentations and the comments have to be clear and precise. So I learned during this internship how to write documentation and how to keep a history of the different versions of the code (using Git). This habit permits other developers to implement easily new functions.

Another important aspect of this project is that CERN is a research center, so the state of mind is completely different than in a company. In fact a single resolution for an issue doesn't exist so it's our job to find different solutions and then choose the best one regarding the efficiency and the compatibility with the existing framework. It's really appreciable because I am really interested by research. This project was a good way to know more about research center and their way to keep a high level of innovation and accuracy.

Cultural

Like I highlighted in the section “Presentation of the company”, CERN is really a major place for every scientists around the world. Even if I worked alone on this project the intercultural interactions were something I had to deal with every day. In fact, the main language at CERN is English so it was important to be understood by everyone. This aspect was really important when I had to explain a technical issue about the project with my supervisor or just to present some results. Technical terms were things I had to learn quickly in order to make my point and write understandable documentation. In addition the summer students come from Europe, Asia or America so English was compulsory. Thanks to this internship I practiced my English a lot (regular or technical) and I discovered so many different cultures and ways of life. This definitely gave me the taste of traveling, and more importantly the taste of working for an international company.

Finally, CERN is the place where you can meet a physicist, a medical engineer, a mathematician or a computer scientist so you have to adapt your speech to the person you talking to. This aspect is really important for an engineer because we will have to explain to our future client which functionalities we can implement by using non-technical terms.

Conclusion

From our analysis, we can conclude that an optimal grouping factor is 210 (i.e. 210 input files will need the same request to the server), keeping the result of the query in the cache for 2 hours. These parameters will give us a hit ratio above 80% in the type of production analyzed.

This analysis doesn't cover every aspect of a proxy server. In fact the object replacement policy can also have the size of the object as a parameter. After defining the maximum size of an object allowed in the cache, if the size of the object is above this parameter, the object will be consider as "stale".

The results obtained will be used to configure the proxy servers in the next months, and then we will run test productions to validate the setup before finally switching the infrastructure from SQLite to Frontier.

References

[1] http://frontier.cern.ch "Frontier home page"

[2] http://cdsweb.cern.ch/record/811089 Brook N, "LHCb Computing Model", LHCb Note, CERN-LHCb-2004-119

[3] http://www.squid-cache.org "Squid Proxy Server home page"

[4] http://public.web.cern.ch/public/en/About/About-en.html "About page of CERN website"

[5] http://cms.cern.ch/iCMS/ "CMS experiment home page"

[6] http://atlas.web.cern.ch/Atlas/Collaboration/ "ATLAS experiment home page"

[7] http://lhcb.web.cern.ch/lhcb/"LHC-b experiment home page"

[8] http://aliweb.cern.ch/ "ALICE experiment home page"

[9] http://en.wikipedia.org/w/index.php?title=File:Cern-accelerator-complex.svg&page=1 "Wikipedia"

[10] http://en.wikipedia.org/wiki/File:CERN1999.png "Wikipedia"

[11] http://en.wikipedia.org/w/index.php?title=File:CERN_international_relations_map.svg&page=1 "Wikipedia"

[12] http://en.wikipedia.org/wiki/CERN#History "Wikipedia"

[13] http://ph-dep.web.cern.ch/ph-dep/ "CERN PH department website"

[14] http://www.eclipse.org/ "Eclipse home page"

[15] http://www.python.org "Python home page"

[16] http://docs.python.org/library/tarfile.html "Python documentation on tarfile module''

[17] http://www.sqlalchemy.org "SQLAlchemy home page"

[18] http://en.wikipedia.org/wiki/Gnuplot "Wikipedia"

[19] http://git-scm.com/ "Git home page"

Annexes

Annexe 1.1: In functions.py the getSite method

```
def getSite(member, tarFileObject):
    """Extract the site from the .info file"""

    logging.debug("Find a info file : %s", member.name)
    dataFile = tarFileObject.extractfile(member)
    for line in dataFile:

        result = re.match(r'/Site = (\S*)', line)

        if result:
            site = result.group(1)
            logging.debug("Find the Location : %s", site)
            return site
```

Annexe 1.2: In functions.py the getConnections method

```
def getConnections(member, tarFileObject):
    """Extract the Ligns in the Log files which contains the words : 'Connected to
database'"""
    data = []

    dataFile = tarFileObject.extractfile(member)
    logging.debug("Extraction of *.log file...")
    logging.debug("Research in the file %s :", member.name)

        #We extract all the log file and go throw them to find the lines
containing 'Requested to process'
        #From those lines we keep the date, the time and the purpose of the
connection
        #Ex : 2012-05-19  16:40:58 UTC    BrunelInit
    for line in dataFile:
        result = re.match(r'(\d{4}-\d{2}-\d{2}) *(\d{2}:\d{2}:\d{2} \S*) (\S*)
*\S* *Requested to process', line)

        #If the result is not None we return it as a proper string
        if result:
            resultAsString = (r"{0}  {1}    {2}").format(result.group(1),
result.group(2), result.group(3))
            data.append(resultAsString)

    logging.debug("Give back data : %s", data)
    return data
```

Annexe 1.3: In functions.py the getInputs method

```
def getInputs(member, tarFileObject):
    """Extract the inputs from the XML file using xml.tree

        This method take too much execution time so we will use getInputs2 which
use xml.sax
    """

    logging.debug("Find a XML file : %s", member.name)

    inputFile = tarFileObject.extractfile(member)

    tree = ET.parse(inputFile)
    elems = tree.findall("input/file")

    for elem in elems:
        logging.debug("Find a input : %s", elem.attrib["name"])
        return elem.attrib["name"]
```

Annexe 1.4: In functions.py the getInputs2 method

```
def getInputs2(member, tarFileObject):
    """Extract the inputs from the XML file using xml.sax"""

    logging.debug("Find a XML file : %s", member.name)

    handler = MySaxDocumentHandler()

    parser = make_parser()

    parser.setContentHandler(handler)
    inputFile = tarFileObject.extractfile(member)

    parser.parse(inputFile)
    inputFile.close()

    inputs = handler.get_inputsList()

    logging.debug("Find a input : %s", inputs)

    return inputs
```

Annexe 2: The MySaxDocumentHandler class

```python
'''
Created on Jun 25, 2012

To get the input from the XML File using xml.sax

@author: Gaylord Cherencey
'''
from xml.sax import handler

class MySaxDocumentHandler(handler.ContentHandler):
    '''Class which go through the XML file a extract the input'''

    def __init__(self):
        self.level = 0
        self.inInput = False
        self.inputsList = []

    def get_inputsList(self):
        return self.inputsList

    def startDocument(self):
        pass

    def endDocument(self):
        pass

    def startElement(self, name, attrs):

        #we go through the tags into the XML File
        #we get name attribute if we find this layout:
        #<input>
        #    <file  name="">

        self.level += 1

        if name == 'input':
            self.inInput = True

        elif self.inInput and name == 'file':
            for attrName in attrs.keys():
                if attrName == 'name':
                    self.inputsList.append(attrs.get(attrName))

    def endElement(self, name):
        if name == 'input':
            self.inInput = False

        self.level -= 1
```

Annexe 3: model.py module

```
'''
Created on Jun 28, 2012

Module which create the object and map them for the database
(Job,Site,Connection,Input)

@author: Gaylord Cherencey
'''

from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship
from sqlalchemy.schema import ForeignKey, Column, Sequence
from sqlalchemy.types import Integer, String
import re

Base = declarative_base()

class Input(Base):
    '''Class which describe a input :
    - <<PK>> id_Input : Integer
    - <<FK>> id_Job : Integer
    - name : String
    '''

    __tablename__ = 'inputs'
    id_Input = Column(Integer, Sequence('input_id_seq'), primary_key=True)
    id_Job = Column(Integer, ForeignKey('jobs.id_Job'),index=True)
    name = Column(String)

    def __init__(self, name):

        self.name = name

class Connection(Base):
    '''Class which describe a connection :
    - <<PK>> id_Connection : Integer
    - <<FK>> id_Job : Integer
    - time : String
    - DB : String
    '''

    __tablename__ = 'connections'
    id_Connection = Column(Integer, Sequence('connection_id_seq'),
primary_key=True)
    id_Job = Column(Integer, ForeignKey('jobs.id_Job'),index=True)
    time = Column(String)
    DB = Column(String)

    def __init__(self, time, DB):

        self.time = time
        self.DB = DB

    def __repr__(self):

        return "Site: id_Job({0}), time({1}))".format(
                self.id_Job, self.time)
```

```
class Job(Base):
    '''Class which describe a job :
    - id_Job <<PK>>
    - id_Site <<FK>>
    - name : String
    - relationship with Input table
    - relationship with Connection table
    '''

    __tablename__ = 'jobs'
    id_Job = Column(Integer, Sequence('job_id_seq'), primary_key=True)
    id_Site = Column(Integer, ForeignKey('sites.id_Site'),index=True)
    name = Column(String)

    inputs = relationship("Input", backref="jobs")
    connections = relationship("Connection", backref="jobs")

    def __init__(self, name):

        self.name = name

    def __repr__(self):

        return "Job: id_Job({0}), id_Site({1}))".format(
                self.id_Job, self.id_Site)

class Site(Base):
    '''Class which describe a site :
    - <<PK>> id_Site : Integer
    - name : String
    - relationship with Job table
    '''

    __tablename__ = 'sites'
    id_Site = Column(Integer, Sequence('site_id_seq'), primary_key=True)
    name = Column(String,index=True)

    jobs = relationship("Job", backref="sites")

    def __init__(self, name):
        self.name = name

class DataBase(object):
    '''Class which declare the database and contains few methods to add data into
it'''

    def __init__(self, filename):
        """Initialization of the database"""
        from sqlalchemy import create_engine
        from sqlalchemy.orm import sessionmaker
        self.filename = filename
        self.engine = create_engine('sqlite:///%s' % self.filename, echo=False)
        Base.metadata.create_all(self.engine)
        self._sessionMaker = sessionmaker(bind=self.engine)
        self.session = self._sessionMaker()
        self.all_site = {}

    def site(self, name):
        """Creation of the site object and add it to the database"""

        site = self.session.query(Site).filter(Site.name==name).first()
        if site is None:
            site = Site(name)
            self.session.add(site)
        return site

        self.session.commit()

    def add_Data(self, k, v):
        """"Add of the data into the database

            k -> name of the job
            v dictionary with the data ( { 'connections':[], 'inputs': [], 'site':
'' } )
        """

        if v['site'] not in self.all_site.values():
            self.all_site[self.all_site.__len__()] = v['site']

        current_site = self.site(v['site'])

        if self.session.query(Job).filter(Job.name==k).first() is None :
            current_job = Job(k)

            for input_Element in v["inputs"]:
                #Check if this input object is already in the database or not
                if
self.session.query(Input).filter(Input.name==input_Element).first() is None :
                    #if not we add it
                    current_input = Input(input_Element)
                    current_job.inputs.append(current_input)

            for connection in v["connections"]:
                result = re.split(r'(\S*)$', connection)
                #Check if this connection object is already in the database or not
                if
self.session.query(Connection).filter(Connection.time==result[0]).first() is None
:
                    #if not we add it
                    current_connection = Connection(result[0],result[1])
                    current_job.connections.append(current_connection)

            current_site.jobs.append(current_job)

        self.session.add(current_site)

        self.session.commit()
```

Annexe 4: proxy class in proxy.py module

```
'''
Created on Jul 10, 2012

Proxy simulation

@author: Gaylord Cherencey
'''

import csv
import datetime

class Proxy:
    '''Class which simulate a proxy area with a simple policy :
            If the object it's in the cache since more than maxTime we refresh the
object by asking the server
    '''

    def __init__(self, maxTime):
        '''Method which initialize the proxy
            - self.cache contains url:time
            - self.histogramm contains every Histogram object
            - self.max represent the time (minutes) that a object can stay in the
cache
            - self.hit you add +1 if the object is in the cache
            - self.miss you add +1 if the object is not in the cache
            - self.formatTime permit to precise how to transform a string into a
datetime object
            - self.timeref is the reference date to calculate delta
        '''

        self.cache = {}
        self.histograms = {}

        #Creation of the two histogram: Delta and RequestPerHour
        self.histograms["Delta"] = Histogram(0 , 200 , 40)
        self.histograms["RequestPerHour"] = Histogram(190740, 190840 , 100)

        self.max = datetime.timedelta(minutes = maxTime)
        self.min = datetime.timedelta(hours = 0)
        self.hit = 0.00
        self.miss = 0.00
        self.formatTime = '%Y-%m-%d  %H:%M:%S'
        self.timeref = datetime.datetime.strptime("1990-08-07 00:00:00",
self.formatTime)


    def request(self, url, time):
        '''Method which simulate a request to the proxy to get a object :
            the proxy check it the object (url) is already in the cache or not
regarding the time
        '''

        formatedTime = datetime.datetime.strptime(time, self.formatTime)

        self.fillRequestPerHourHisto(formatedTime - self.timeref)

        #if the object is already in the cache
        if url in self.cache:

            cache_time = self.cache[url]

            #if the time between this request and the previous one is under
self.max we increment self.hit
            if formatedTime - cache_time < self.max:
                self.cache[url] = cache_time
                self.hit += 1

            #otherwise we increment self.miss and add the delta time at the delta
histogram
            else:
                self.miss += 1
                self.cache[url] = formatedTime

                self.fillDeltaHisto(formatedTime - cache_time)

        #we add the url if it not in the cache
        else :
            self.cache[url] = formatedTime
            self.miss += 1

    def hitRatio(self):
        '''Method which calculate the hit cache ratio : hitRatio = self.hit /
(self.hit + self.miss)'''

        hitRatio = self.hit / (self.hit + self.miss)

        return round(hitRatio * 100, 3)

    def createHitRatioCSV(self, name_CSV, ratio_Number, max_Time):
        '''Method which create the hit ratio CSV file

            name_CSV will be the name of the CSV file
            ratio_number is the ratio use to calculate this hit ratio
            max_Time is the max_time coming from the proxy used to calculate the
cache hit ratio
        '''

        list_Ratio_CSV = []

        list_Ratio_CSV.append(ratio_Number)
        list_Ratio_CSV.append(max_Time)

        #We open the file in ab mode to add every hit ratio regarding the max_time
and ratio
        ratio_file_CSV = csv.writer(open(name_CSV, "ab"))

        list_Ratio_CSV.append(self.hitRatio())

        ratio_file_CSV.writerow(list_Ratio_CSV)

    def fillDeltaHisto(self,delta):
        '''Method which add the delta to the delta histogram and transform it into
seconds'''

        if delta :
            deltagroup = (delta.seconds + (delta.days *86400))/60
            self.histograms["Delta"].addData(deltagroup)

    def fillRequestPerHourHisto(self, data):
        '''Method which add the data to the requestperhour histogram and transform
it into hours'''

        if data :
            dataHour = (data.seconds + (data.days * 86400))/3600
            self.histograms["RequestPerHour"].addData(dataHour)
```

Annexe 5: bash script called graph.sh

```
#!/bin/bash

args=("$@")

db=${args[0]}
max_Time=${args[1]}
i=0

if [ ${#args[*]} != 2 ]
then
	echo "
		Usage: graph.sh db max_Time

			db -> the name of your database file
			max_time -> the maximum time you want your object into the
proxy cache
				"
else

	####################################################
	# Calculate the ratio global and for each center

	if [ ! -e ALL_ratioglobal_${db}_${max_Time}.csv ]
	then
		for i in $(seq 10 10 450)
		do
			analysis ${db} 0 ${i} ${max_Time}
ratioglobal_${db}_${max_Time}.csv
		done
	fi

	#################################################################
	# Calculate the hit ratio per max_Time for a ratio of 70,210,270

	if [ ! -e ALL_hitratio_per_max_Time_${db}_70.csv ]
	then
		for i in $(seq 30 30 600)
		do
			analysis ${db} 0 70 ${i} hitratio_per_max_Time_${db}_70.csv
			analysis ${db} 0 210 ${i} hitratio_per_max_Time_${db}_210.csv
			analysis ${db} 0 270 ${i} hitratio_per_max_Time_${db}_270.csv
		done
	fi

	############################
	# Create the plot Delta.png
	if [ -e "ALL_${db}_Delta.csv" ];then

		gnuplot -persist << GPLOT

		set datafile separator ','

		set title "Time between two requests\n batadase -> ${db} ratio ->
250"
		set xlabel 'Sites'
		set ylabel 'Minutes'
		set zlabel 'Number of requests' rotate by 90
```

```
            set yrange [0:200]

            set ticslevel 0.0
            set grid

            splot "ALL_${db}_Delta.csv" u 1:2:3 with boxes title
"Global","LCG.NIKHEF.nl,LCG.SARA.nl_${db}_Delta.csv" u 1:2:3 with boxes title
"LCG.NIKHEF.nl,LCG.SARA.nl", "LCG.IN2P3.fr,LCG.IN2P3-T2.fr_${db}_Delta.csv" u
1:2:3 with boxes title "LCG.IN2P3.fr,LCG.IN2P3-T2.fr","LCG.PIC.es_${db}_Delta.csv"
u 1:2:3 with boxes title "LCG.PIC.es", "LCG.RAL.uk_${db}_Delta.csv" u 1:2:3 with
boxes title "LCG.RAL.uk", "LCG.CNAF.it,LCG.CNAF-T2.it_${db}_Delta.csv" u 1:2:3
with boxes title "LCG.CNAF.it,LCG.CNAF-T2.it", "LCG.GRIDKA.de_${db}_Delta.csv" u
1:2:3 with boxes title "LCG.GRIDKA.de", "LCG.CERN.ch_${db}_Delta.csv" u 1:2:3 with
boxes title "LCG.CERN.ch"

            set term png
            set out '${db}_Delta.png'

            replot

            quit


GPLOT

      fi

      ############################
      # Create the plot RequestPerHour.png

      if [ -e "ALL_${db}_RequestPerhour.csv" ];then

            gnuplot -persist << GPLOT

            set datafile separator ','

            set title "Number of requests per hour\n batadase -> ${db} ratio ->
250"
            set xlabel 'Sites'
            set ylabel 'Hours'
            set zlabel 'Number of requests' rotate by 90

            set ticslevel 0.0
            set grid

            splot "ALL_${db}_RequestPerhour.csv" u 1:2:3 with boxes title
"Global","LCG.NIKHEF.nl,LCG.SARA.nl_${db}_RequestPerhour.csv" u 1:2:3 with boxes
title "LCG.NIKHEF.nl,LCG.SARA.nl", "LCG.IN2P3.fr,LCG.IN2P3-
T2.fr_${db}_RequestPerhour.csv" u 1:2:3 with boxes title "LCG.IN2P3.fr,LCG.IN2P3-
T2.fr","LCG.PIC.es_${db}_RequestPerhour.csv" u 1:2:3 with boxes title
"LCG.PIC.es", "LCG.RAL.uk_${db}_RequestPerhour.csv" u 1:2:3 with boxes title
"LCG.RAL.uk", "LCG.CNAF.it,LCG.CNAF-T2.it_${db}_RequestPerhour.csv" u 1:2:3 with
boxes title "LCG.CNAF.it,LCG.CNAF-T2.it", "LCG.GRIDKA.de_${db}_RequestPerhour.csv"
u 1:2:3 with boxes title "LCG.GRIDKA.de", "LCG.CERN.ch_${db}_RequestPerhour.csv" u
1:2:3 with boxes title "LCG.CERN.ch"

            set term png
            set out '${db}_RequestPerHour.png'
            replot

            quit

GPLOT

      fi

      ##################################
      # Create the plot graph_per_ratio

      if [ -e "ALL_ratioglobal_${db}_${max_Time}.csv" ];then

            gnuplot -persist << GPLOT

            set datafile separator ','

            set title "Global graph : evolution of hit ratio vs. ratio
(range/time_To_Add)\nParameter : db -> ${db}, max_Time -> ${max_Time}"
            set xlabel 'Ratio (range/time_To_Add)'
            set ylabel 'Hit Ratio (%)'
            set xrange [0:450]
            set yrange [0:100]
            set key right bottom
            set grid

            plot "ALL_ratioglobal_${db}_${max_Time}.csv" using 1:3 w l title
"Global","LCG.NIKHEF.nl,LCG.SARA.nl_ratioglobal_${db}_${max_Time}.csv" using 1:3 w
l title "LCG.NIKHEF.nl,LCG.SARA.nl","LCG.IN2P3.fr,LCG.IN2P3-
T2.fr_ratioglobal_${db}_${max_Time}.csv" using 1:3 w l title
"LCG.IN2P3.fr,LCG.IN2P3-T2.fr","LCG.PIC.es_ratioglobal_${db}_${max_Time}.csv"
using 1:3 w l title "LCG.PIC.es","LCG.RAL.uk_ratioglobal_${db}_${max_Time}.csv"
using 1:3 w l title "LCG.RAL.uk","LCG.CNAF.it,LCG.CNAF-
T2.it_ratioglobal_${db}_${max_Time}.csv" using 1:3 w l title
"LCG.CNAF.it,LCG.CNAF-T2.it","LCG.GRIDKA.de_ratioglobal_${db}_${max_Time}.csv"
using 1:3 w l title "LCG.GRIDKA.de",
"LCG.CERN.ch_ratioglobal_${db}_${max_Time}.csv" using 1:3 w l title "LCG.CERN.ch"

            set term png
            set out 'graph_per_ratio_${db}_${max_Time}.png'

            replot

            quit

GPLOT

      fi


      #####################################
      # Create the plot graph_per_max_Time

      if [ -e "ALL_hitratio_per_max_Time_${db}_70.csv" ];then

            gnuplot -persist << GPLOT

            set datafile separator ','
```

```
             set title "Global graph : evolution of hit ratio vs.
self.max\nParameter : db -> ${db}"
             set xlabel 'self.max (min.)'
             set ylabel 'Hit Ratio (%)'
             set xrange [0:600]
             set yrange [0:100]
             set key right bottom
             set grid

             plot "ALL_hitratio_per_max_Time_${db}_70.csv" using 2:3 w l title
"ratio : 70","ALL_hitratio_per_max_Time_${db}_210.csv" using 2:3 w l title "ratio
: 210","ALL_hitratio_per_max_Time_${db}_270.csv" using 2:3 w l title "ratio : 270"

             set term png
             set out 'graph_per_max_Time_${db}.png'

             replot

             quit

GPLOT

      fi

fi
```

Annexe 6: README text

```
==============
Analysis Proxy
==============

Description :
=============
Analysis Proxy determinate which parameters you have to set into the proxy area to
get a good cache hit ration

In our case we want to implement a proxy area in every T1 working with LHCb such
as "GRIDKA.de" and so on.
So we have to analyze how this center behave with the database when it run jobs.

Installation :
==============

After getting the sources you have to extract them thanks to the command line tar
(see the help to set the right options)
When the files are extracted you have to get into the source folder (cd command):
Then do : /usr/bin/python2.6 setup.py build (to build everything)
               export PATH=$PWD/build/scripts-2.6:$PATH (to add the scripts to the
PATH)
               export PYTHONPATH=/usr/lib64/python2.6/site-packages/SQLAlchemy-
0.7.3-py2.6-linux-x86_64.egg:$PYTHONPATH
               export PYTHONPATH=$PWD/build/lib:$PYTHONPATH

Execution :
===========

To execute this you have to follow steps :

Step 1 : extraction from tarball
Step 2 : analyze the data

Step 1 : Extraction of data from tarball
----------------------------------------

See below the usage for this command line :

Usage: extraction.py name_of_the_tarball name_for_database [options] : Use the -h
option to get help

Options:
  -h, --help     show this help message and exit
  -v, --verbose  Print a message each time  a  module  is initialized
  -d, --debug    Print debug information

At the beginning you will see "Started ..." and at the end "Extraction done"

Remark :
--------
      name_of_the_tarball have to be a tgz file
      name_for_database will be the name of the database file

      ex : to extract data from test.tgz and call the database file test.db I have
to do -> extraction.py test.tgz test.db
```

```
       The input file has to by a back-up of the LHCbDirac log files of a
       reconstruction production (Brunel). These can be retrieved from CASTOR
       with something like:

       rfcp /castor/cern.ch/grid/lhcb/backup/log/00018310_0000_2012-06-04.tgz

Step 2 : Analyze the data from the database
-------------------------------------------

When you have the database file you can run analysis and get graphs

So you have to run the script called graph.sh

This script take some parameter such as : db, max_time

db -> the name of your database file
max_time -> the maximum time you want your object into the proxy cache

The script will create some csv file corresponding to every graph and png pictures
( "evolution of hit ratio vs. ratio (range/time_To_Add)", "evolution of hit ratio
vs. self.max" )
If you when to see again the plots you just have to run again the script with the
same parameters

General Remark :
================

You can use separately the analysis.py script to get the hit ratio for a special
ratio
The usage of this script is:

         Usage: %prog db seed_Number ratio_Number max_Time ratio_CSV [options] :
Use the -h option to get help

       db -> the name of your database file
       seed_Number -> certain way to shuffle the input data
       ratio_Number -> parameter to group the inputs
       max_time -> the maximum time you want your object into the proxy cache
       ratio_CSV -> name of the CSV File created """

You can change the number of bins, the minimum and maximum time for the histogram
by setting the parameters of Histogram object into the proxy.py script

ex : Histogram(190740, 190840 , 100) in line 36 will create a Histogram Object
with 100 bins, 190740 in minimum and 190840 in maximum
```

Annexe 7: test.py script into the test package

```
'''
Created on Jun 15, 2012

Unit-test of the method for the Unzip project

@author: Gaylord Cherencey
'''
from proxy import extraction, functions
import unittest

class Test(unittest.TestCase):

    def testBadFile(self):
        '''Test when a wrong file name is put as a argument in the main module'''

        pass
        print "\n***************Test with bad file********************\n"
        self.assertEqual(extraction.main(["filename.tgz", "mydb.db"]), 
extraction.FAILURE)
        print "Test with bad file... OK"

    def testVerboseMode(self):
        '''Test if the verbose mode goes well in the main module'''

        pass
        print "\n***************Test verbose mode********************\n"
        self.assertEqual(functions.defineParser(["-v", "00018400_0000_2012-06-
02.tgz", "mydb.db"])[1], 20)
        print "Extraction and verbose mode... OK"

    def testDebugMode(self):
        '''Test if the debug mode goes well in the main module'''

        pass
        print "\n***************Test debug mode********************\n"
        self.assertEqual(functions.defineParser(["-d", "00018400_0000_2012-06-
02.tgz", "mydb.db"])[1], 10)
        print "Extraction and debug mode... OK"

    def testDebugAndVerbose(self):
        '''Test if the verbose mode and the debug goes well as the same time '''

        pass
        print "\n***************Test debug and verbose mode in the same 
time********************\n"
        print "#1 CASE (-d -v) :\n"
        self.assertEqual(functions.defineParser(["-d","-v","00018400_0000_2012-06-
02.tgz", "mydb.db"])[1], 20)
        print "\n#2 CASE (-v -d) :\n"
        self.assertEqual(functions.defineParser(["-v","-d","00018400_0000_2012-06-
02.tgz", "mydb.db"])[1], 10)
        print "Extraction and debug mode... OK"

    def testHelpMode(self):
        '''Test if the help mode goes well in the main module'''

        pass
        print "\n***************Test help option********************\n"
        try:
            extraction.main(["-h", "00018400_0000_2012-06-02.tgz"])
        except SystemExit, e:
                self.assertEquals(type(e), type(SystemExit()))
                self.assertEquals(e.code, 0)
                print "\nHelp mode... OK"

    def testWrongMode(self):
        '''Test if the wrong option case is handle in the main module'''

        pass
        print "\n***************Test wrong option********************\n"
        try:
            extraction.main(["-a", "00018400_0000_2012-06-02.tgz"])
        except SystemExit, e:
                self.assertEquals(type(e), type(SystemExit()))
                self.assertEquals(e.code, 2)
                print "\nWrong option... OK"

    def testNoNameForDatabase(self):
        '''Test if the wrong option case is handle in the main module'''

        pass
        print "\n***************Test no name for database********************\n"
        try:
            extraction.main(["00018400_0000_2012-06-02.tgz"])
        except SystemExit, e:
                self.assertEquals(type(e), type(SystemExit()))
                self.assertEquals(e.code, 2)
                print "\nTest no name for database... OK"

    def testNoFileName(self):
        '''Test if the no file name case is handle in the main module'''

        pass
        print "\n***************Test no file name********************\n"
        try:
            extraction.main(["-v"])
        except SystemExit, e:
                self.assertEquals(type(e), type(SystemExit()))
                self.assertEquals(e.code, 2)
                print "\nNo file name... OK"

    def testTooManyArguments(self):
        '''Test if the too many arguments case is handle in the main module'''

        pass
        print "\n***************Test too many arguments********************\n"
        try:
            extraction.main(["-v","toto", "00018400_0000_2012-06-02.tgz"])
        except SystemExit, e:
                self.assertEquals(type(e), type(SystemExit()))
                self.assertEquals(e.code, 2)
                print "\nToo many arguments... OK"

    def testisThisTypeOfFile(self):
        '''Test the method IsTheTarfile from the functions module'''
```

```
        pass
        print "\n****************Test if the file is a tar 
file*********************\n"
        self.assertFalse(functions.isThisTypeOfFile("test.taz",".tgz"))
        self.assertTrue(functions.isThisTypeOfFile("Brunel_25_222_1.log","log"))
        print 'Test of the function... OK\n'

    def testmycondition_2(self):
        '''Test the method IsTheTarfile from the functions module'''

        pass
        print "\n****************Test if the file is a tar 
file*********************\n"
        self.assertEqual(extraction.myCondition_2("test.taz"),0)
        self.assertEqual(extraction.myCondition_2("job.info"),1)
        self.assertEqual(extraction.myCondition_2("Brunel_25_2_1.xml"),2)
        self.assertEqual(extraction.myCondition_2("Brunel_25_2_1.log"),3)
        print 'Test of the function... OK\n'

    def testmycondition_1(self):
        '''Test the method IsTheTarfile from the functions module'''

        pass
        print "\n****************Test if the file is a tar 
file*********************\n"
        self.assertFalse(extraction.myCondition_1("test.taz"))
        self.assertTrue(extraction.myCondition_1("jtest.tgz"))
        print 'Test of the function... OK\n'

if __name__ == "__main__":
    unittest.main()
```