CERN

Openlab Summer Student Report

Monitoring system of data subscriptions for ATLAS on the Grid

Author:
Pedro André Cunha

Supervisor:
Dr. Simone Campana

September 9, 2009

Contents

Chapter 1

Introduction

The present report is a description of the work done during a Openlab Summer Student Internship. The project is directly related with the IT/GS group and the ATLAS DDM team and it was supervised by Dr. Simone Campana, a member of the same group.
The main objective of this project was to provide a system which could help the DDM team in there tasks of solving problematic data subscriptions.

In this report first of all we are going to explain the problem and why do we implement this system (Sec. 2).
Then we give an overview of the architecture of the system (Sec. 3).
Furthermore a user guide which explains how to use the system is provide (Sec. 4).
For technical details and installation instructions a developer guide had be produced (Sec. 5)
Finally a conclusion about the project and about possible improvements can be found in Sec. 6

Chapter 2

The ATLAS data management system and its operational issues

The ATLAS [ATL] Distributed Data Management (DDM) [DDM] project was established to develop a scalable and reliable system for data organization and placement, on top of the WLCG Grid infrastructure [WLCG]. The DDM software (DQ2) [DDM] was therefore developed to manage the discovery, replication and deletion of ATLAS data across sites, provide bookkeeping of data organization and placement, interoperate with different grid resources, automate enforcement of management policies like the ATLAS computing model and enforce access controls, managing user and group quotas and accounting.

In DDM, data are organized in datasets, where a dataset is a mutable or immutable collection of files. Datasets are the unit of data replication and data organization: aggregating file in datasets allows to scale down the number of entities for data transfer, deletion and discovery, beside offering a simple way to share and aggregate data in a convenient way across the collaboration. The data distribution model relies on the concept of subscription, i.e. the intention to place a dataset at a given site. A complete or incomplete copy of a dataset at a site is called replica. The core of the system has been developed as a set of independent clients and services, which can be divided into three categories: Central Services, Local Services, End User tools. The system activity is monitored via the ATLAS DDM Dashboard [DASH].

ATLAS runs frequent computing exercises to test the readiness and scalability of its computing system and the underlying grid infrastructure.

One of those exercises is STEP, which is a *Scale Testing for the Experiment Programme*, the STEP'09 is the 2009 version which ran from the 1st to the 15th of June. The STEP [STEP] challenge consisted on stressing the

Grid in workload management and data management at the scale expected in data taking in order to analys the scalability of the system for the four experiments of WLCG (ATLAS, ALICE, CMS, LHCb).

The results show that some data subscriptions can hang on the system for several weeks and then being aborted ("broken") by the system, while generally healthy subscriptions should conclude in few hours. There are plenty of reason why this can happen: misconfiguration of the servers, wrong subscription parameters, unreachable data, etc.

In day by day operations, experts must go through a series of steps to cure problematic subscriptions :

- Identify them (there are more than 50'000 subscription running everyday)
- Dig through DQ2 and Dashboard to get additional informations
- Cure the problem

Most of these tasks are performed using either DQ2 CLI and Dashboard Web Interface however the data mining is still very difficult and time consuming due the amount of subscriptions running in the system.

The system proposed tries to support the ATLAS DDM team to detect and categorize 'problematic' subscriptions, providing al the details necessary to take proper actions.

Chapter 3

The ATLAS DDM Subscriptions Monitor

The ATLAS DDM Subscription Monitor is a system which can be splited in two parts.

The first part takes care of gather information from DQ2, Dashboard and Configuration Files of the DDM 'Transfer Agents'. While the second part takes care of visualize the gather information.

The next sections will cover which subscriptions we are looking for and how we gather the information.

3.1 The Subscriptions

A Subscriptions is a request of a Dataset (which is roughly is a set of files) which has information about who request this dataset, to where should it be transfered, etc. There are two states of subscription which the system is looking for, Broken Subscriptions and Queued Subscriptions running for more than 7 days.

3.1.1 Broken Subscriptions

The Broken Subscriptions are subscriptions which the system decided to abort, either because it ran for more than 15 days, or because an error happend (like files that cannot be found to be transfered).

The information displayed about these subscriptions are:

- Creation Date - The first time this dataset has be subscribed.
- Broken Date - The date when the system broke the subscriptions.
- Cloud - The cloud of the destination site. ATLAS sites are grouped in Clouds

- Site - The destination site (where the dataset is going to be transfered).
- Dataset Name - The name of the dataset.
- Dataset State - State of the dataset.
  - Open - The dataset is being modified.
  - Closed - The dataset is not being modified, but can be in the future.
  - Frozen - The dataset is finished, no modification can be made.
  - Deleted - The dataset has been deleted from the system.
- Transfer Status - The counting of the files transfered or failed.
  - Done - The file has been successfully transfered.
  - Staged - The file have been retrieved from a Tape Media.
  - Failed - The file could not be transfered because the transfer failed.
  - No-event - The file has no events.
- Reason - The reason why the system broken the subscription.

3.1.2 Queued Subscriptions

The Queued Subscriptions are running subscriptions, which means that the transfer of the dataset is still in progress. The Subscriptions that the system is looking for are the ones which have been running for more than 7 days, which is a 'first symptom' that something might be wrong.

The information displayed about these subscriptions are:

- Creation Date DQ2 - The last time this subscriptions has been queued.
- Creation Date Dash - The first time this subscriptions has been 'seen' by Dashboard.
- Modification Date Dash - The last time this subscriptions get an update and Dash has noticed it.
- Cloud - The cloud of the destination site.
- Site - The destination site (where the dataset is going to be transfered).
- Dataset Name - The name of the dataset.
- Dataset State - State of the dataset.
  - Open - The dataset is being modified.

  - Closed - The dataset is not being modified, but can be in the future.
  - Frozen - The dataset is finished, no modification can be made.
  - Deleted - The dataset has been deleted from the system.

- Dashboard State - The state of the subscriptions on Dashboard
  - Complete - All the files had been successfully transfered.
  - Staged - All files have been retrieved by a TAPE media
  - Queued - The transfer is still running.
  - Broken- The subscriptions has been broken, i.e. stop running either because it couldn't find some files or because it has been running for more than 15 days.

- Share - For which share this subscriptions is running for.
- Served - Which host is running the agent which serves the site of this subscription.
- Dataset Size - The size of the total of the files in a dataset
- Owner - Who has created the subscription
- Sources - If the subscriptions has defined from where it should get the files

3.2 Data Gathering

As mention above the system is made of two components: the first one gathering information and the second one displaying it. This section will focus on the first.

There are three different data type that the system gathers:

- Broken Subscriptions
- Queued Subscriptions
- Statistics - Broken Subscriptions

Since each one has a different architecture we are going through each one

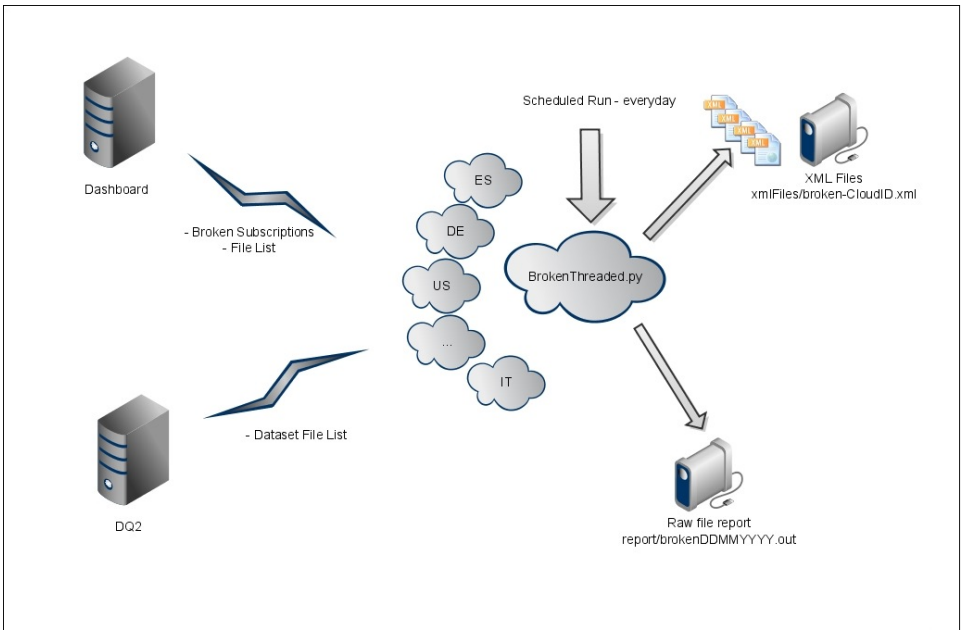

Figure 3.1: Broken Subscriptions Data Gather - Architecture

3.2.1 Broken Subscriptions

The Broken Subscriptions data is gather querying Dashboard and DQ2 and cross-checking the results.
An overview about the way this module is design can be seen in the Fig. 3.1
This module consists of several submodules:

- Get Broken Subscriptions
- Get Files List and State
- Save data into XML Files
- Save data into RAW report Files

Get Broken Subscription This part of the module takes care of querying Dashboard to get all the subscriptions broken on the last 30 days. This process is repeated for each site of each cloud.

Get Files List and State This part queries DQ2 and Dashboard about a subscriptions and datasets. The reason why we need to query both is because some file might never reach dashboard, so having the original file list from DQ2 it is easy to find out which files are missing. From Dashboard files list it's possible to get the transfer status of a file (Done, Staged, Failed), so then we can present a report about how much has been transfered.

Save data into XML Files After gathering all the data this information needs to be store so it can be used by the second part of the system. For this we used XML files, each cloud has its own XML and the information in the XML is from 30 days before the actual day, i.e. there are no long-term history. The DTD of the XML can be obtain in the App. A

Save data into RAW report Files In order to give the possibility to other programs to use the data, everything is dump in a text file. Then it's possible to parse the file and use this information. The structure of this file can be obtained in the App. B

3.2.2 Queued Subscriptions

The Queued Subscriptions data is gather querying Dashboard, DQ2 and the Configuration.
An overview about the way this module is design can be seen in the Fig. 3.2

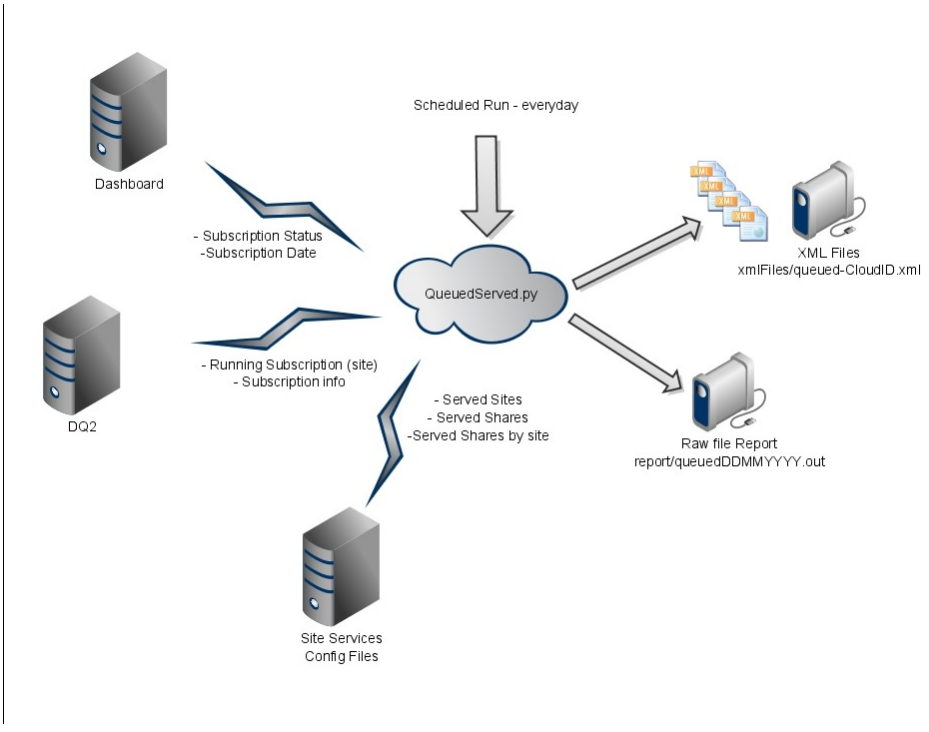

Figure 3.2: Queued Subscriptions Data Gather - Architecture

This module consists of several submodules:

- Get Shares/Sites Servers
- Get Queued Subscriptions

- Get Subscription Info
- Save data into XML Files
- Save data into RAW report Files

Get Shares/Sites Servers This information is need to check which agent host is serving each site, and to know which share is being served. This is also useful to get the blacklisted sites. This data is retrieved by parsing the configuration of the agent host.

Get Queued Subscriptions This part of the module takes care of querying DQ2 to get all the running subscriptions of each site.

Get Subscription Info The module query DQ2 and Dashboard in order to get more information about the subscriptions (like the share, the owner, the size of the dataset, etc.). At this point it's possible to know when it had been queued, so we can keep only the ones running for more than 7 days.

Save data into XML Files After gathering all the data this information needs to be store so it can be used by the second part of the system. For this we used XML files, each cloud has its own XML and the information in the XML is regarding only the current day, i.e. there are no history. The DTD of the XML can be obtain in the App. A

Save data into RAW report Files In order to give the possibility to other programs to use the data, we dump everything in a text file. Then it's possible to parse the file and use this information. The structure of this file can be obtained in the App. B

3.2.3 Statistics

The Statistics data is gather querying only Dashboard. It basically looks for the broken subscriptions of the day and saves the data categorizing it by cloud and state.
The design of the module can be seen in the Fig. 3.3

This module consists of several submodules:

- Get Broken Subscriptions
- Get Dataset State
- Get / Create XML
- Append Information to XML

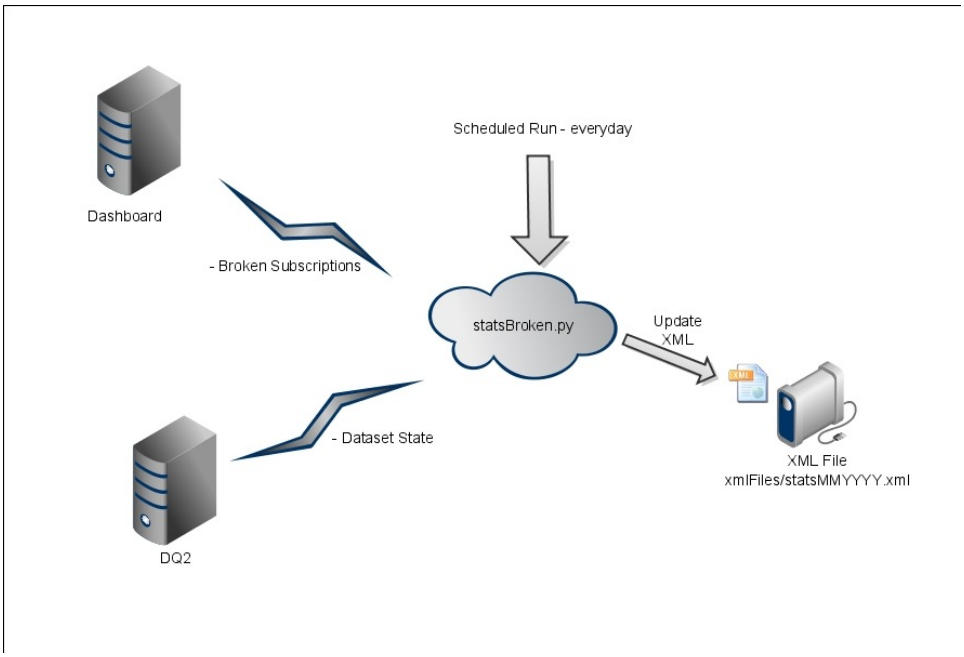

Figure 3.3: Statistics of Broken Subscriptions Data Gather - Architecture

Get Broken Subscriptions This information is obtained by querying Dashboard. It gets the broken subscriptions of the actual day.

Get Dataset State This part of the module takes care of querying DQ2 to get the actual state of the dataset(Open, Closed, Frozen, Deleted).

Get / Create XML After gathering all the data and process it, the module needs to get the xml of the month. Since the statistics are saved in a monthly XML the module need to check if the XML already exist and take care of keep the data on it.

Append Information to XML After getting the existing XML or creating a new one, the processed data need to be append in order to do not erase the previous data.With the technique we can get an history of one month per file. The DTD of the XML can be obtain in the App. A

3.3 Displaying Data

After having all the data processed and stored in XML files, there is a need to display this information and a way to filter and search it.

We decided to develop a website which can display the information and offer some tools to filter and / or search within the data. Each module has

his own webpage. We are going to give an overview of each one. For more details on how to use it, please have a look in Sec. 4

3.3.1 Broken Subscriptions Webpage

This webpage displays the broken subscriptions of the last 30 days. The Fig. 3.4 is an example of what the webpage looks like.

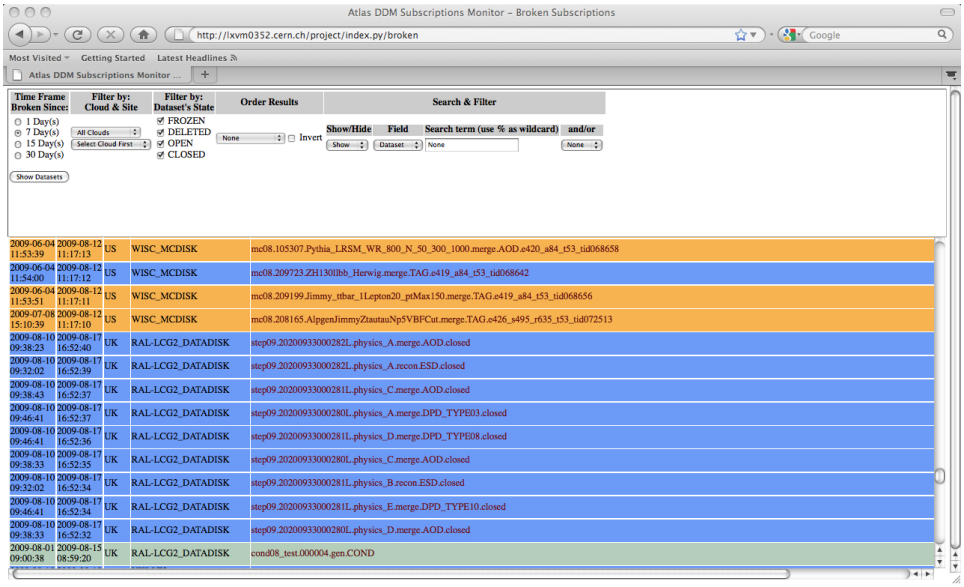

Figure 3.4: Broken Subscriptions - Webpage

There are two parts, the Search&Filter Box (top part) where the user can define his criteria and filter the information (more on that in Sec. 4) and the Results Box (bottom part) where each row represents a broken subscriptions and shows some information. To get more details about a subscription the user can click on the dataset name and get some more details as in the Fig. 3.5 For a descriptions of each field please have a look in Sec. 3.1.1

The website basically loads the XML files and parse them. Then it pick only the subscriptions which match the criteria defined and sort them as the defined order. Finally it displays them in a table.

3.3.2 Queued Subscriptions Webpage

This webpage displays the queued subscriptions running for more than seven days. The Fig. 3.6 is an example of what the webpage looks like.

There are two parts, the Search&Filter Box (top part) where the user can define his criteria and filter the information (more on that in Sec. 4) and the Results Box (bottom part) where each row represents a queued subscriptions and shows some information. To get more details about a subscription the

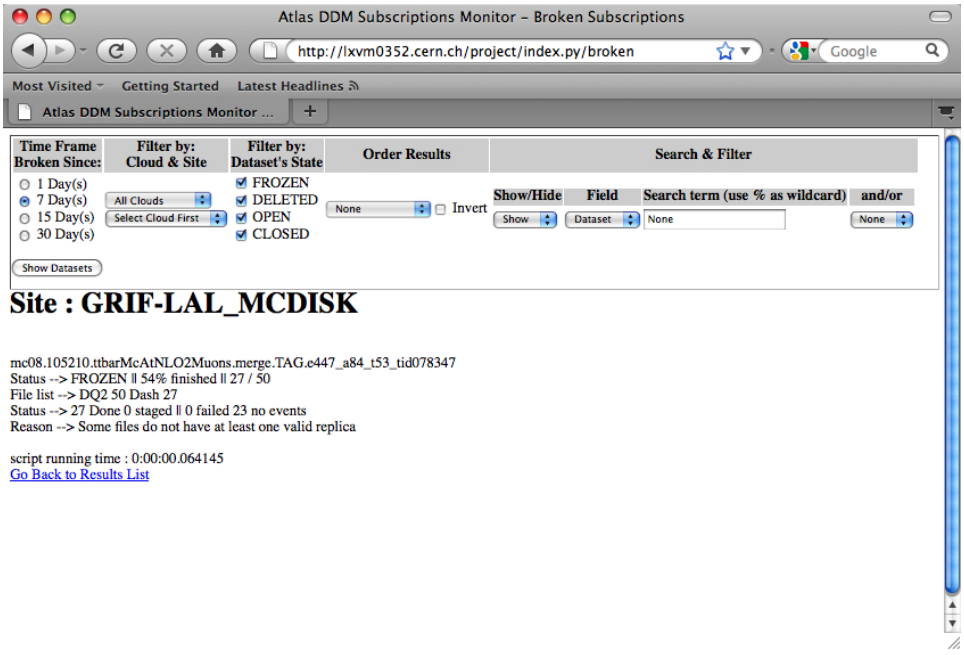

Figure 3.5: Broken Subscriptions Details - Webpage

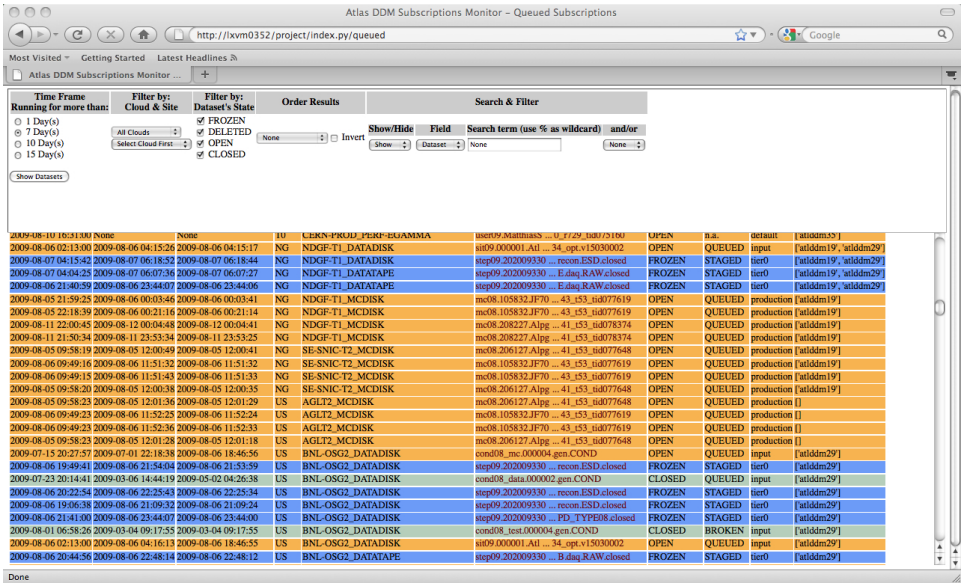

Figure 3.6: Queued Subscriptions - Webpage

user can click on the dataset name and get some more details as in the Fig. 3.7 For a descriptions of each field please have a look in Sec. 3.1.2

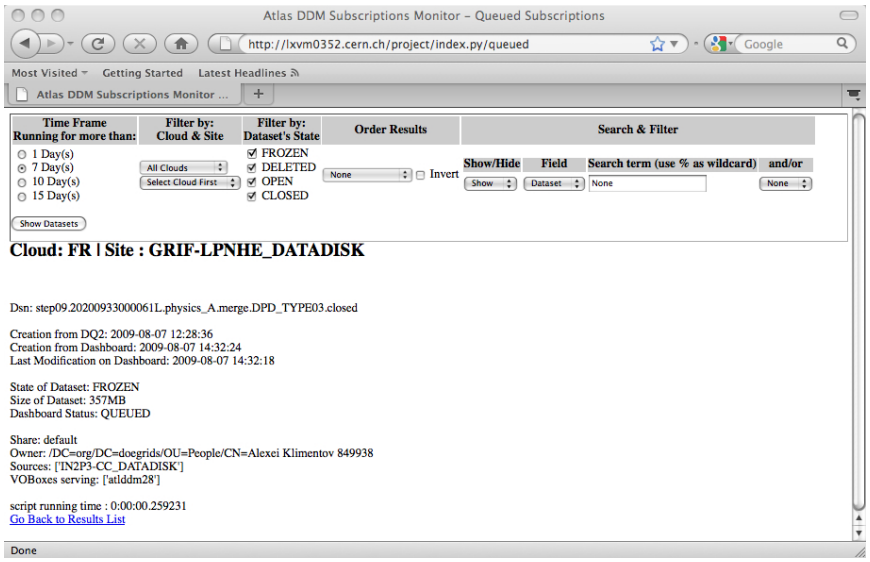

Figure 3.7: Queued Subscriptions Details - Webpage

The website basically loads the XML files and parse them. Then it picks only the subscriptions which match the criteria defined and sort them as the defined order. Finally it displays them in a table.

3.3.3 Statistics Webpage

This webpage displays the statistics of the broken subscriptions. The Fig. 3.8 is an example of what the webpage looks like.

The page has two parts, the selection box where the user can define which month or month range he wants to see, and the charts per say. There are two charts the first one shows the broken subscriptions per Cloud and the second one per dataset state (more on that in Sec. 4) The website basically loads the XML files affected by the criteria and process data in order to request the charts to Google Charts API [GCH].

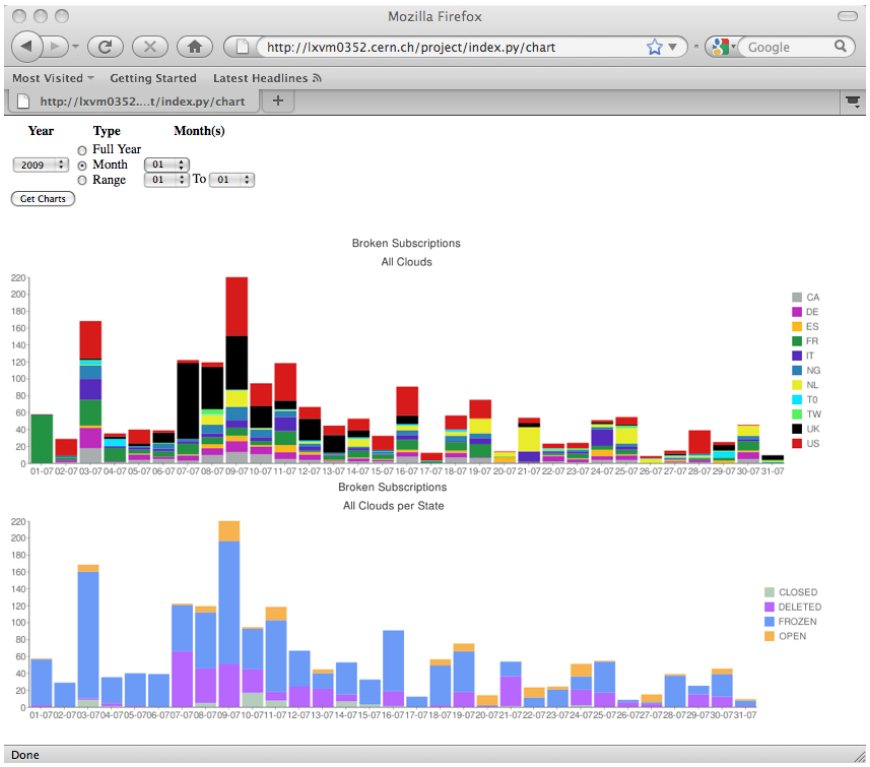

Figure 3.8: Broken Subscriptions Statistics - Webpage

Chapter 4

User Guide

The main page of the graphical web interface contains links to dedicated pages for broken subscriptions, queued subscriptions and statistics.

This chapter will provide guidelines to the end user for accessing the various pages, refining a search criteria and display the information.

4.1 Broken Subscriptions

As mentioned before, this webpage consist in two parts: a Search&Filter box and a Result box.

4.1.1 Search&Filter Box

This box give the possibility to the user to narrow or wide the results. This box is composed by several sections, each one is independent of each other (except the last one). In Fig. 4.1 each section is marked with a letter, details for each marker are provided below:

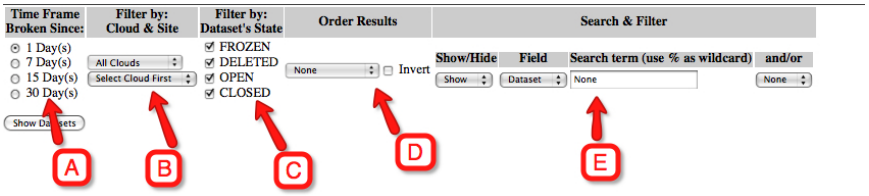

Figure 4.1: Broken Subscriptions Search&Filter Box

Time Frame - A

This section defines the time frame of the results. Here the user can select the subscription which broke on the last X days. For example, selecting the option `7 day(s)` will select all the subscriptions which broke on the last 7

days. It's easy to understand that as bigger is the time frame bigger will be the result list.

Cloud&Site - B

This section narrows the results to a cloud or to a site. Selecting a cloud, the results will be only about this cloud. After selecting a cloud it's possible to select a site from this cloud, then only the results from this site will be displayed. The action of selecting a cloud speeds up the time to display the results since the system will parse just one XML file.

Dataset State- C

In this section the user can choose which dataset states will be displayed on the results. This very simple, just check the states that should be displayed and uncheck the ones that should be hided.

Order - D

This section gives you the possibility to order the results by multi-criteria, i.e. order by different fields. To use it, just select the field you want, then another box will show up and the user can restart the process. The small checkboxes on the side are to invert the results order, for instance if the user choose the field site, the results will be in a ascedent order by the name of the site, on the other side if you tick the invert option, the results will be in a descent order. The explanation of each field can be consulted in Sec. 3.1.1

Search&Filter- E

This section is the more complex, yet the more powerful. In this section the user can construct several logic constraints and combine them in order to hide or show specific results. We are going to drill down each option of this section.

- Show or Hide - This option will display or hide the results which match the contraint.
- Field - This option will define which field needs to be matched with the search term. The available fields are Cloud, Site and Dataset.
- Search Term - In this field the user needs to insert his search term. This search term is the string which will be matched with the defined field over the results. There is the possibility to use an wildcard character (%) which will match any character. An example for the wildcard use can be to select all the MonteCarlo Subscription, the search term would be `mc08%`, so all the dataset with their name starting with `mc08` will be matched.

- And / Or - This field takes care of combining the different constraints. Since it is not possible for the user to define his own structure of the query, the user needs to use the following convention. Let's call each row (show/hide, search field, search term) a clause. When the field and/or is defined it will be the logical operation between the logical result of the previous clauses and the new one. Moreover we present an example which will make it more clear.
  Let's name clauses as C_x where x is the number of the clause and O_x the logical operation where x is the numer of the operation, the results of an logical operation is represented as R_x, where x is the number of the operation. The operation O_i will be the logical operation between R_i and C_{i+1} . So the expressions will be evaluate like that.

$$(C_i O_i C_{i+1}) O_{i+1} C_{i+1}) \cdots O_n C_{n+1}$$

Example Let's imagine that the user wants to hide all the sites from Romania and all the sites from the German cloud but display the mc08 dataset even if they are from Romanian or German sites.

The first clause would be `Hide | Site | RO% | And` - which hides all the site which the name start with 'RO' (basically the Romanians site).

The second clause would be `Hide | Cloud | DE | Or` - which hides all the sites which are in cloud with the name 'DE'.

The third clause would be `Show | Dataset | mc08%` - which shows all the dataset with the name starting with 'mc08'.

This is the resulting logic expression:

([Hide | Site | RO%]∧[Hide | Cloud | DE])∨[Show | Dataset | mc08%]

This expression will hide all the subscriptions from Romanian sites and from German cloud except if it is a dataset with the name starting with mc08.

4.1.2 Results Box

This part displays the subscriptions which fit the criteria defined on the Search&Filter Box. It's a table with some information about the subscriptions, each field is defined in Sec. 3.1.1

The information in the details fields can give the user an overview about how much has been transfer, however if it's not possible to display this it tells why the information is unavailable.
If the problem is caused by DQ2 the text will be `n.a.-DQ2` on the other hand if the problem is caused by Dashboard the text will be `n.a.-Dash`

For each subscription it's possible to get more details about it, clicking on the dataset name, another page will be displayed. Then some details about the status of the transfer and a link for dashboard's subscriptions page are displayed.

4.2 Queued

As mentioned before, this webpage has two parts a Search&Filter box and a Result box, we will focus on each one separately.

4.2.1 Search&Filter Box

This box give the possibility to the user to narrow or wide the results. This box is composed by several sections, each one is independent of each other (except the last one). Since this box is very similar to the one from the broken subscriptions we are going to describe only the different aspects and let the user go through the common aspects in Sec. 4.1 As you can see in Fig. 4.2 each different section has a letter, we are going to give details about each section.

Figure 4.2: Queued Subscriptions Search&Filter Box

Time Frame - F

This section will define the time frame of the results. Here you can select the subscription which are running for more than X days. For example, selecting the option `10 day(s)` will select all the subscriptions which are running for 10 days. The option for 15 might look odd (after 15 days the subscriptions should be broken) but it can be useful to detect problems.

Order - G

This section gives you the possibility to order the results by multi-criteria, i.e. order by different fields. To use it, just select the field you want, then another box will show up and you can restart the process. Those small checkboxes on the side are to invert the results order, for instance if you choose the field site, the results will be in a ascedent order by the name of the site, on the other side if you tick the invert option, the results will be in a descent order. The explanation of each field can be consulted in Sec. 3.1.2

4.2.2 Results Box

This part displays the subscriptions which fit the criteria defined on the Search&Filter Box. It's a table with some information about the subscrip-

tions, each field is defined in Sec. 3.1.2

For each subscription it's possible to get more details about it, clicking on the dataset name, another page will be displayed. Then some details about the owner and the defined sources of the subscriptions are displayed.

4.3 Statistics

This web page is different from the broken and queued subscriptions page, instead of presenting information about subscriptions this page present a 'report' about the broken subscriptions.

There are basically two parts, one where you can select the information to be displayed and another where the charts are presented.We will focus on each one separately.

4.3.1 Selection Box

This section gives the possibility to the user to select which information he wants to see.There are three possibility:

- Full Year - Gives a monthly view over all the months of the selected year.
- Month - Gives a daily view over the selected month.
- Range - Gives a monthly view over the selected months of the selected year.

After selecting the information that the user wants and clicking on 'Get Charts' the charts will be loaded.

4.3.2 Charts

There two charts which are displayed as results of the Selection Box.

The first one is the results of broken subscriptions over the selected time frame, grouped by cloud.

The second one is the results of the broken subscriptions over the selected time frame, grouped by the dataset state.

In both of the them the X axis represents the month or the day, depending on the time frame. The Y axis represents the number of Broken subscriptions of this day or month.

Chapter 5

Developer Guide

This chapter will describe the structure of the python modules, the way the XMLs files are managed, the way the Report files are managed, how to check the log files and finally an installation guide.

5.1 The Structure

As mentioned before (Sec. 3) the system consists of several submodules, the part which gathers information and the part which takes care of the information's visualization. The data gather are python scripts which run every night as a cron jobs [CRON] and saves the information in XML files. On the other side for the data visualisation an apache server with mod_python takes care of the job.

5.1.1 Data Gather Scripts

There are three scripts which gather information:

BrokenThreaded.py

This script will query dashboard and DQ2 in order to get broken subscriptions and information about them and its dataset.

This script start a thread per cloud and when a dataset is big is starts another thread to take care of it(however there is a limit for the amounts of threads per cloud, the default is one extra thread for processing a dataset).

Each thread goes through all the sites and get the broken subscriptions of this site from Dashboard. Then it queries DQ2 to get the file list and the dataset state. After all this data it cross-checks the information from DQ2 with Dashboard and get the missing files in the destination.

Having all that, the script will save the information into the cloud XML file, with the name `broken-ID.xml` where ID is the Cloud ID.

When all cloud threads finish the script sorts all the subscriptions by cloud/site/creation date and dumps everything into the report file, which has the name `brokenDDMMYYYY.out` where DD is the day, MM the month and YYYY the year.

QueuedServed.py

This script will query Dashboard, DQ2 and Configuration Files in order to get queued subscriptions and information about them and its dataset.

The first job of this script is to check the configuration Files from the Agent Hostes in order to get the served sites, the served shares and the blacklisted sites. For that the script parse a 'report' of the configurations, located in AFS

This script is single thread so it picks a cloud and starts going through all the sites and geting all the queued subscriptions of this site from DQ2. Then it queries again DQ2 to get the creation date (very important to get the running time) and specific information about this subscriptions (like the share, the owner, etc.). It queries also Dashboard to get the information that Dashboard knows about this dataset.

Having all that, the script will save the information into the cloud XML file, with the name `queued-ID.xml` where ID is the Cloud ID.

When all clouds are processed the script sorts all the subscriptions by cloud/site/creation date from Dashboard and dumps everything into the report file, which has the name `queuedDDMMYYYY.out` where DD is the day, MM the month and YYYY the year.

statsBroken.py

This script will query Dashboard and DQ2 to get the broken subscriptions of the day before.

It starts by querying Dashboard to get all the broken subscriptions from the previous day about a site, then it queries DQ2 to get the state of the dataset. This process is repeated for every site, of each Cloud.

Having all the information the script will save the information into the month XML file, with the name `statsMMYYYY.xml` where MM is the month and YYYY the year.

5.1.2 Website Modules

The website module is composed by four different package:

- searchBox
- brokenData
- queuedData

- stats

The entry point of the website is the file index.py which is the script which answer all the requests from mod_python(Apache python module).It basically gets the request, parses and escapes the parameters (from POST method). After having the parameters it basically asks the right module to give the HTML code, then it constructs the output and returns it to mod_python, to be displayed in the user Web Browser.

An overview of content of each package and some details about it can be read just below. However for getting implementation details it's strongly recommend to have a look into the comments included in the source code.

searchBox

This package takes care of creating the HTML code for the Search&Filter box
It is composed by a single script, `searchHTML.py` , which dynamically constructs each component of the search box. Some components for Queued Subscriptions are different from Broken subscriptions, in this case the scripts handles them separately.

brokenData

This package handles the requests about the Broken Subscriptions. It has two scripts, brokenSubs.py and brokenSubInfo.py

brokenSubs.py This script takes care of parsing the XML files with broken subscriptions and to filter them depending on the criteria defined on the Search&Filter Box.If the criteria narrows the results to only one cloud, the script will 'take a shortcut' and only parse the xml file about this cloud.
Finally the script will transform the information about the subscriptions into HTML code and at the end it will return it to index.py.
As a rough idea, this script generates the result box of broken subscriptions

brokenSubInfo.py This scripts takes care of generating the detail page of a broken subscription. It gets the information about which subscription it should work(by GET Method) and then it parses the corresponding XML File to extract the info that he needs.
Finally it returns the HTML code about the information to index.py.

queuedData

This package handles the requests about the Queued Subscriptions. It has two scripts, queuedSubs.py and queuedSubInfo.py

queuedSubs.py This script takes care of parsing the XML files with queued subscriptions and to filter them depending on the criteria defined on the Search&Filter Box.If the criteria narrows the results to only one cloud, the script will 'take a shortcut' and only parse the xml file about this cloud.
Finally the script will transform the information about the subscriptions into HTML code and at the end it will return it to index.py.
As a rough idea, this script generates the result box of queued subscriptions

queuedSubInfo.py This scripts takes care of generating the detail page of a queued subscription. It gets the information about which subscription it should work(by GET Method) and then it parses the corresponding XML File to extract the info that he needs.
Finally it returns the HTML code about the information to index.py.

stats

This package takes care of generating the statistics web page (Sec. 4.3).

The package is composed by a single script, `charts.py` which takes care of generating the selection box and the Google Charts URLs. Depending on the criteria from the selection box (POST Method) the script will prepare a monthly view or a daily view (if the request is for only a month it shows a daily view, otherwise it shows a monthly view).

In case of a daily view the script will parse only the XML of the corresponding month. Otherwise it will parse the requested months. After parsing them, in the monthly view it needs to digest the information in order to sum the broken subscriptions of each day.

Finally it scales and encodes the data (needed for Google Charts) for the URLs, in order to return the HTML code of the selection box and the charts.

5.1.3 Common Modules

The common modules are aggregated into the package `common`. This package contains some scripts which offers values or functions transversal to several scripts. It is composed by three distincts modules:

- values.py
- funcs.py
- log.py

A brief overview of each module will be given, but for getting implementation details it's strongly recommend to have a look into the comments included in the source code.

values.py

This module provides default values which are used by all the scripts, like root path for the project, or used by more specific scripts, like the color table of the results box or the mapping of the order tokens.

funcs.py

This module provides functions which are used by all the other scripts, like getting all the sites from a cloud or the muti-criteria sorting function, or more specific ones, like encoding the data for Google Charts or parsing the values from the Search&Filter or Selection Box.

log.py

This module takes care of giving a logger to any script related to the website. It only takes care of the website log, since the other logs are related with the data gathers. It basically gives a logger with the same configuration changing only the name.

5.2 XML Files

The XML files are the tool which pass information from the data gather and the website. The system uses three different types of XMLs for three diffrent type of date.

Broken For the broken subscriptions, the files used are the ones with the name `broken-ID.xml` where ID is the cloud ID, each cloud has its own XML file. The XML files contains the broken subscriptions of the last 30 days, these files are re-created every day by the script `BrokenThreaded.py`. About the structure of the XML further information is provided in App. A

Queued For the queued subscriptions, the files used are the ones with the name `queued-ID.xml` where ID is the cloud ID, each cloud has its own XML file. The XML files contains the queued subscriptions running for more than seven days, at the present day, these files are re-created every day by the script `QueuedServed.py`. About the structure of the XML further information is provided in App. A

Statistics For the Statistics, the files used are the ones with the name `statsMMYYYY.xml` where MM is the month and YYYY the year. There is an XML file per month, inside each there is info about each day of the month. These files are updated every day, adding the information about the day to

the other content. About the structure of the XML further information is provided in App. A

5.3 Report Files

The Report files are text files where all information about the subscriptions is dumped in them. These files are created to give the possibility to other applications to parse and use the information gather by the scripts. There are two different type of Report Files: The broken subscriptions file and the queued subscriptions file. These files are created everyday and their name is `brokenDDMMYYYY.out` or `queuedDDMMYYYY.out` where DD is the day, MM the month and YYYY the year. A soft link is created for the newest file, it can be accessed by `broken.out` or `queued.out`. The strucure of the files can be consulted in App. B

5.4 Logs

The log files are used to report problems during the execution of the scripts. The system uses several ones:

- broken.log - This log report the problems faced during the execution of the Broken subscription data gather. It notice the subscriptions that Dashboard or DQ2 cannot find or when they cannot respond a request
- queued.log - This log report the problems faced during the execution of the Queued subscription data gather. It notice the subscriptions that Dashboard or DQ2 cannot find or when they cannot respond a request.
- stats.log - This log report the problems faced during the execution of the broken subscriptions statistics data gather. It notice the subscriptions that Dashboard or DQ2 cannot find or when they cannot respond a request
- web.log - This log reports the problems faced when a website request is performed. It notice when the XML files cannot be loaded or other possible problems.

All of them are Rotating Files, meaning that when the log file reach 10 Mb it will be renamed to NAME.log.1 and a new NAME.log will be created. The maximum number of files for a log is 6.

5.5 Installation

This section will give a tutorial about how to install the system in a new server.

5.5.1 Requirments

In order to be able to run the system in a server, the machine needs:

- SLC4
- CERN AFS access - in order to load DQ2 and Dashboard modules
- Apache
- Mod_Python
- Pyhton2.3
- XML libraries for Python

5.5.2 Apache Configuration

After installing Mod_python, there is an handler that needs to be added to the python configuration from Apache. The file is `conf.d/python.conf`, and the handler is :

```
<Directory PATH/TO/THE/PROJECT>
   AddHandler mod_python .py
   PythonHandler mod_python.publisher
   PythonDebug Off
</Directory>
```

5.5.3 Cron Jobs

In order to run the data gather scripts each day, shell scripts were created to be able to run the python scripts as Cron jobs. There are three shell scripts:

- brokenScript.sh - This script takes care of loading the DQ2 / Dashboard module, running the BrokenThreaded.py and to create the soft link for the newest report file.
- queuedScript.sh - This script takes care of loading the DQ2 / Dashboard module, running the QueuedServed.py and to create the soft link for the newest report file.
- statsScript.sh - This script takes care of loading the DQ2 / Dashboard module and running the statsBroken.py

The scripts should be schedule in different times with no overlaps, in order to don't stress the DQ2 and Dashboard servers. This is an example of the schedule(the tab should be considered as the same line):

```
30 00 * * * /var/www/html/project/brokenScript.sh
          > /var/www/html/project/broken.out 2>&1
30 02 * * * /var/www/html/project/queuedScript.sh
          > /var/www/html/project/queued.out 2>&1
00 02 * * * /var/www/html/project/statsScript.sh
          > /var/www/html/project/stats.out 2>&1
```

Please be aware that all the shell scripts have the path written inside them, so please change them when installing in a different path.

5.5.4 Running

After configuring the apache server and scheduling the jobs, the last task is to start the apache server and everything should run as expected.

Chapter 6

Conclusion

The proposed system was fully implemented and it is currently in production. The ATLAS DDM team is using it as a daily tool to perform there tasks for detecting and fixing problematic subscriptions. Having said that, it can be conclude that the internship was successful and productive.

However, there still room for possible improvements:

- Visual polishing - make the website more 'attractive'.
- 'Real-time' information - update the information more often than once per day.
- Offer more details on the statistics.

6.1 Author's notes

This project gave me the possibility to work in a real issue. I could design and implement a system which is now use as a tool in a daily basis. During the internship I had contact with some new technologies and new concepts.

For this project, a underlying knowledge about the Grid was needed. Despite the short duration I was able to consolidate some concept about it, with a big help from my supervisor. He managed to slice down the information that I did need to know, in order to do not waste my short time with less important topics.

These two months were really interesting, starting from the project to the team I was involved. I would like to thanks, for the support and the good environment, the ATLAS DDM team, especially the supervisor of this project, Dr. Simone Campana.

Bibliography

[ATL] *ATLAS*, ATLAS Collaboration, *ATLAS Technical Proposal*, CERN/LHCC/94-43, 1994

[DDM] *DDM*, M. Branco, D. Cameron, T. Wenaus, *A Scalable Distributed Data Management System for ATLAS*, Conference on Computing in High Energy and Nuclear Physics (CHEP06), February 2006, Mumbai (India)

[WLCG] *WLCG*, The LCG Editorial Board, *LHC Computing Grid Technical Design Report*, LCG-TDR-001, CERN-LHCC-2005-024, June 2005

[DASH] *Dashboard*, J. Andreeva, B. Gaidioz, J. Herrala, G. Maier, R. Rocha, P. Saiz, *Experiment Dashboard - The Monitoring System For The LHC Experiments*,In Proceedings of the 2007 HPDC workshop on Grid monitoring, Monterey,California, USA, June 25, 2007

[STEP] , *STEP 09 -Scale Testing for the Experiment Programme 2009* `https://twiki.cern.ch/twiki/bin/view/LCG/WLCGStep09`

[GCH] *Google Charts API*, `http://code.google.com/apis/chart/`

[CRON] *crontab*, `http://en.wikipedia.org/wiki/Cron`

Prepared in LaTeX 2_ε by Pedro André Cunha

Appendix A

XML Files

There are three different types of XML files in use in this system.

- Broken Subscriptions
- Queued Subscriptions
- Statistics

An explanation of each one can be found below.

A.1 Broken Subscriptions

```
<!-- broken is the root element -->
<!ELEMENT BROKENS ( CLOUD, RUNNING_TIME, DATE ) >

<!-- cloud is the element which represents a cloud -->
<!-- is composed the name, an site element per site-->
<!ELEMENT CLOUD ( NAME, SITE+ ) >

<!-- site is the element which represents the queued
subscriptions of a site-->
<!-- it contains the name of the site and the broken
subscriptions -->
<!ELEMENT SITE ( NAME, BROKEN_SUBSCRIPTIONS ) >

<!-- broken_subscriptions is the element which store
the broken subscriptions of a site-->
<!ELEMENT BROKEN_SUBSCRIPTIONS ( DATASET* ) >

<!-- dataset is the element which contains the information
about a broken subscription-->
```

```
<!-- it contains the name of the dataset, the creation time
 of the subscription on dashboard, the last modification on
 dashboard, the reason for aborting the subscription the
file list count and the status of the files-->
<!ELEMENT DATASET ( NAME, CREATION_TIME, MODIFIED_TIME,
REASON, FILE_LIST?, STATUS? ) >

<!-- dataset state is the attribute which indicates the
state of the dataset-->
<!ATTLIST DATASET state ( CLOSED | DELETED | FROZEN |
OPEN ) #REQUIRED >

<!-- file_list is the element which represents the number
 of files in DQ2 (the 'official' counting) and in Dashboard
 (where some files might be missing)-->
<!ELEMENT FILE_LIST ( DQ2, DASH ) >

<!-- status is the element which contains a counting of the
status of the files transfers.-->
<!ELEMENT STATUS ( DONE, STAGED, FAILED_TRANSFER, NO_EVENT ) >

<!ELEMENT CREATION_TIME ( #PCDATA ) >

<!ELEMENT DASH ( #PCDATA ) >

<!ELEMENT DATE ( #PCDATA ) >

<!ELEMENT DONE ( #PCDATA ) >

<!ELEMENT DQ2 ( #PCDATA ) >

<!ELEMENT FAILED_TRANSFER ( #PCDATA ) >

<!ELEMENT MODIFIED_TIME ( #PCDATA ) >

<!ELEMENT NAME ( #PCDATA ) >

<!ELEMENT NO_EVENT ( #PCDATA ) >

<!ELEMENT REASON ( #PCDATA ) >

<!ELEMENT RUNNING_TIME ( #PCDATA ) >

<!ELEMENT STAGED ( #PCDATA ) >
```

A.2 Queued Subscriptions

```
<!-- queued is the root element -->

<!ELEMENT QUEUED ( CLOUD ) >

<!-- cloud is the element which represents a cloud -->
<!-- is composed the name, an site element per site ,
the script running time and the date when it ran -->

<!ELEMENT CLOUD ( NAME, SITE+, RUNNING_TIME, DATE ) >

<!-- site is the element which represents the queued
subscriptions of a site-->
<!-- it contains the name of the site, which shares are
served and the queued subscriptions -->
<!ELEMENT SITE ( NAME, SHARES, SUBSCRIPTIONS ) >

<!-- subscriptions is the element which store the individual
 subscriptions -->
<!ELEMENT SUBSCRIPTIONS ( SUBSCRIPTION* ) >

<!-- share is the element which store the served shares -->
<!ELEMENT SHARES ( SHARE* ) >

<!-- subscription is the element which contains all the
information about a queued subscription -->
<!-- it is composed by the dataset name, the creation date
on Dashboard, the creation date on DQ2, the date of the last
modification on Dashboard, the state of the dataset, the size
of the dataset in bytes, the state on Dashboard, the owner of
the subscriptions, the share of the subscriptions, the list of
the specified sources and the the list of the voboxes which are
serving this subscription -->

<!ELEMENT SUBSCRIPTION ( NAME, CREATION_DASH, CREATION_DQ2,
MODIFIED_DATE, DATASET_STATE, FILES_SIZE, DASH_STATE, OWNER,
SHARE, SOURCES, VOBOXES ) >

<!-- this is a list of the specified sources for a subscriptions,
```

```
 if no source is specified this list is empty -->
<!ELEMENT SOURCES ( SOURCE? ) >

<!-- this is a list of the vobox which are serving this
subscriptions, if the site of the subscription is blacklisted
the 'blacklist' word will appear here. -->
<!ELEMENT VOBOXES ( VOBOX* ) >

<!ELEMENT CREATION_DASH ( #PCDATA ) >

<!ELEMENT CREATION_DQ2 ( #PCDATA ) >

<!ELEMENT DASH_STATE ( #PCDATA ) >

<!ELEMENT DATASET_STATE ( #PCDATA ) >

<!ELEMENT DATE ( #PCDATA ) >

<!ELEMENT FILES_SIZE ( #PCDATA ) >

<!ELEMENT MODIFIED_DATE ( #PCDATA ) >

<!ELEMENT NAME ( #PCDATA ) >

<!ELEMENT OWNER ( #PCDATA ) >

<!ELEMENT RUNNING_TIME ( #PCDATA ) >

<!ELEMENT SHARE ( #PCDATA ) >

<!ELEMENT SOURCE ( #PCDATA ) >

<!ELEMENT VOBOX ( #PCDATA ) >
```

A.3 Statistics

```
<!-- stats is the root element -->
<!ELEMENT STATS ( BROKEN+ ) >

<!-- broken is the element which represents the statistics
 of the broken subscriptions -->
```

```
<!-- it is composed by the date of the information, a cloud
 element per cloud and the total of broken subscription at this day-->
<!ELEMENT BROKEN ( DATE, CLOUD+, TOTAL_DAY ) >

<!-- cloud is the element which represents the statistics for a cloud-->
<!-- it contains the name of the cloud, total of broken
subscriptions of this cloud, and a break down by state.-->
<!ELEMENT CLOUD ( NAME, TOTAL_CLOUD, OPEN, CLOSED, FROZEN, DELETED ) >

<!ELEMENT DATE ( #PCDATA ) >

<!ELEMENT TOTAL_DAY ( #PCDATA ) >

<!ELEMENT NAME ( #PCDATA ) >

<!ELEMENT DELETED ( #PCDATA ) >

<!ELEMENT FROZEN ( #PCDATA ) >

<!ELEMENT OPEN ( #PCDATA ) >

<!ELEMENT CLOSED ( #PCDATA ) >

<!ELEMENT TOTAL_CLOUD ( #PCDATA ) >
```

Appendix B

Report RAW Files

The Report files are files which contains a report of the information in a text file. They are created in order to allow other programs to parse them and use their information. In this system there are two different types of report files:

- Broken Subscriptions
- Queued Subscriptions

B.1 Broken Subscriptions

The file contain the same information as the XML file, the difference is that in the file the subscription are sorted. The order is cloud / site / creation time. The format of the information is on the first line of the file. This is an example:

```
Creation || Modified || Cloud || Site || Dataset || State
```

Creation This field represents the creation date on dashboard of this subscription.

Modified This field represents the date of the last modification on dashboard of this subscription.

Cloud This field represents the cloud of the subscription destination.

Site This field represents the site of the subscription destination.

Dataset This field represents the name of the subscription's dataset.

State This field represents the state of the subscription's dataset.

B.2 Queued Subscriptions

The file contain the same information as the XML file, the difference is that in the file the subscription are sorted. The order is cloud / site / creation time on DQ2. The format of the information is on the first line of the file. This is an example:

```
Creation DQ2 || Creation Dash || Modified Dash || Cloud
|| Site || Dataset || Dataset state || || Dataset Size || Dash state
|| share || voboxes || owner || sources
```

Creation DQ2 This field represents the creation date on DQ2 of this subscription

Creation Dash This field represents the creation date on Dashboard of this subscription

Modified Dash This field represents the date of the last modification on dashboard of this subscription

Cloud This field represents the cloud of the subscription destination.

Site This field represents the site of the subscription destination.

Dataset This field represents the name of the subscription's dataset.

Dataset State This field represents the state of the subscription's dataset.

Dataset Size This field represents the size in bytes of the files list in the dataset.

Dash State This field represents the state of the subscription on dashboard.

Share This field represents the share of the subscription.

Voboxes This field represents a list of Voboxes which are serving this subscription.

Owner This field represents the owner of the subscription.

Sources This field represents a list of specified sources for the subscription.