Journal of Physics: Conference Series **120** (2008) 072027 doi:10.1088/1742-6596/120/7/072027

Monitoring the Atlas Distributed Data Management System

Ricardo Rocha[1], Miguel Branco[1], David Cameron[2], Benjamin Gaidioz[1], Vincent Garonne[1], Mario Lassnig[1,3], Pedro Salgado[1]

[1] CERN, European Organization for Nuclear Research
[2] University of Oslo, Norway
[3] University of Innsbruck, Austria

E-mail: ricardo.rocha@cern.ch

Abstract. The ATLAS Distributed Data Management (DDM) system is evolving to provide a production-quality service for data distribution and data management support for production and users' analysis.

Monitoring the different components in the system has emerged as one of the key issues to achieve this goal. Its distributed nature over different grid infrastructures (EGEE, OSG and NDGF) with infrastructure-specific data management components makes the task particularly challenging. Providing simple views over the status of the DDM components and data to users and site administrators is essential to effectively operate the system under realistic conditions.

In this paper we present the design of the DDM monitor system, the information flow, data aggregation. We discuss the available usage, the interactive functionality for end-users and the alarm system.

1. Introduction

With the start of the LHC at CERN scheduled for the second quarter of 2008 also the offline software systems have been focusing more on operational issues and spending less time in additional developments or functionalities. Monitoring plays a central role in any operational activity so a number of different systems have been developed during the last two years focusing on several areas: workload management, data management, site/service reliability and many others.

This becomes only more true when the system has to serve a community as large and spread as the one making the ATLAS experiment. A tiered topology has CERN as the starting point for data collection - the Tier0. Around it there are ten big Tier1 centres, located in Europe, Asia and North America, each serving smaller communities usually tied to a specific Tier1 by geographical proximity - Tier2s. Currently there are already more than 100 sites, and this number will certainly increase once small Tier3 centres start participating more actively.

The data management part of the system is of extreme importance, as in ATLAS the matchmaking between tasks and available resources is done based on the location of the data. Having datasets quickly exported to the centres where the processing should occur is vital for

successful operation. In the following sections we go through the details of how the flow of data and the different components of the system are monitored, how operators can access the monitoring data and which mechanisms exist to alert in case of misbehaviors.

2. ATLAS Distributed Data Management

The Atlas DDM system is composed of different components which interact with each other to perform the different bookkeeping and data movement tasks. The catalogs take care of the first part - bookkeeping - while the site services try to answer the different user requests to spread existing data among the different sites - called *subscriptions*.

Central Catalogs Include the *Subscription* catalog, which stores data movement requests from users; the *Location* catalog, storing the sites where replicas of the data can be found and additional information about its completeness; the *Content* catalog, tracking relationships between files and the coarser grained unit of a dataset; and finally the *Repository* catalog, which keeps track of existing datasets and their different versions

Site Services A collection of agents, each performing a distinct task. There are agents fetching new subscriptions from the central catalogs, evaluating appropriate replicas for each file in a dataset subscription, submitting transfers to the File Transfer Service (FTS), registering files in the local storage catalogs once they are successfully transferred into the site or registering new dataset locations in the central catalogs once subscriptions are complete

We have introduced a central concept in the system, the *dataset*. With the expected amount of data being so large[1], and the file sizes varying from the controlled production data to the more chaotic user analysis results, having collections of files greatly reduces the dimension of the bookkeeping task and improves performance as the dataset is also used as the unit of data movement. And being simply an abstraction over the physical data stored in individual files, datasets are also dynamic, as users can add or remove files from the collection, creating new versions of the same dataset.

Subscriptions can then be made on datasets or specific dataset versions, triggering the data flow which is the main focus of the monitoring system.

2.1. Data Flow

From a bookkeeping point of view a dataset can be in state *Open*, where new files can be added or old ones removed; *Closed*, where the latest version of the dataset is finished and changes can only be done by reopening it; or *Frozen*, where no more changes can be done to it.

In the same way, a dataset subscription can be in different states, depending on which stage each of the files it contains are at any given time. The full state machine for creating a replica of a dataset or dataset version in a given site is depicted in Figure 1.

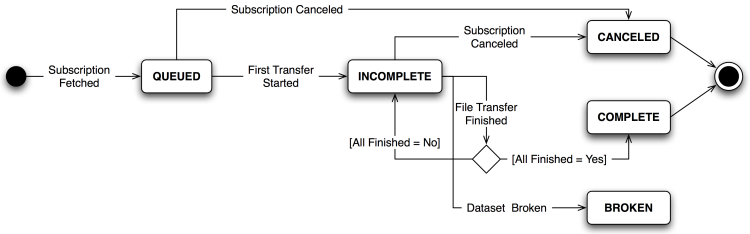

Figure 1. Dataset Subscription State Diagram

Queued The dataset has been taken for processing by the site services, but its contents have not yet been retrieved from the content catalog

Incomplete Dataset contents are known, and files have been queued locally for transfer where appropriate

Canceled The user canceled the subscription. Files already transferred are not removed, but the ones still in the queue will not be replicated

Broken The subscription is invalid and the site services have given up on serving it. This is usually due to the wrong definition of the dataset itself

Complete The subscription has been fully served, and all dataset files are stored and registered in the destination site

Individual file replications follow a much more complex state machine. Figure 2 represents a simplified version, where states resulting from failures are skipped.

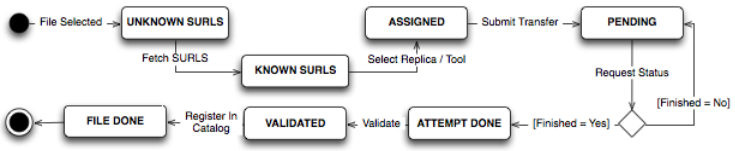

Figure 2. File Movement State Diagram

Unknown Surls No information about potential source replicas is currently known

Known Surls The list of potential source replicas has been retrieved

Assigned A valid source replica has been selected, and the transfer has been placed in the queue

Pending The transfer has been submitted to the File Transfer Service (FTS) and the transfer request status is being periodically checked by the site services

Attempt Done The FTS reported the transfer attempt has been finished. If it was not successful, a new submission will be made, using the same or a different source replica depending on the error returned

File Done The replica has been registered in the site local file catalog, the last step being performed

A file movement can also be put into a *Hold* state, when multiple failures have happened and the system cannot resolve them itself, requiring human intervention. Also, any file replica can be later put into a *File Bad* state, meaning that after being successfully stored and registered in a site a local change occured, and DDM has been unable to use this new replica for other file movements - usually it either became corrupted, impossible to retrieve from storage or it was unregistered from the local storage catalog.

3. DDM Dashboard

The ATLAS DDM dashboard design results from the use cases of two different types of users. Site and service operators start from a system overview, digging deeper into site, dataset or file movement details as required. On the other hand, end users submitting dataset movement requests are less interested in site performance, focusing directly on the current status of their own subscriptions.

The DDM dashboard is composed of two main sets of components: the web application and all the actions available through its HTTP based interface; and the agents, each taking care of a specific task. All of them use a unique backend database, where all data concerning the monitoring of DDM and grid fabric components is stored.

3.1. Web Application - Querying

The web application's most visible use is of course the access to the data using a web browser. *XHTML* is the default output format for any query resulting in response data, though other formats are available where appropriate. The available functionality via the web interface is depicted in Figure 3.

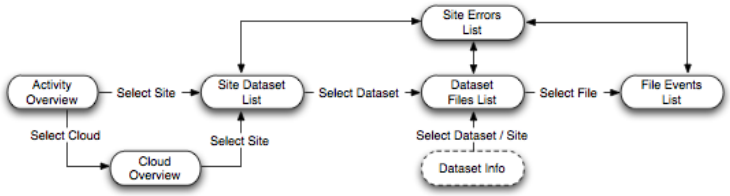

Figure 3. Web Application Navigation

This extra functionality concerning multiple output formats is a direct benefit of having the system built on top of the ARDA Dashboard framework[5]. It offers native translation of the response objects into *XML* or *CSV* formats, as well as the representation of data in a graphic format (only available when the output data is adequate). The mechanism of content type negotiation strickly follows what is defined in RFC 2616[2].

Building on top of the *XML* output format there is also a set of command line interface tools and a python API available as alternatives to the more rigid web browser interface. Both should ease the task of integrating dashboard data into external tools.

3.2. Web Application - Collection

The second use of the web application is for information collection. It turns the same endpoint used for data query into a messaging system, and is used to gather the different messages coming from the multiple site services instances.

The main advantages of using HTTP for delivering these messages are the external dependencies. At the collector side there is already an HTTP server running for the querying

interface, so nothing needs to be added. And at the producer side, as the site services run in standard linux distributions, tools like *curl* are installed by default and the required outbound ports 80 and 443 open for everyone to use.

The main and probably only disadvantage of this option is the lack of reliability in the message delivery. This is a feature that does not exist natively in the HTTP protocol, so a client side layer with persistent storage had to be developed for cases where the endpoint is down or unreachable. Several modern messaging systems provide this functionality both at the client and broker levels.

3.3. DDM Dashboard Agents

Other than the collection of dataset and file transfer information from the site services, there are other individual agents performing different tasks in the DDM dashboard.

3.3.1. SAM Collector The LCG Service Availability and Monitoring (SAM)[3] service gathers information concerning service and site availability by running a set of tests against the different instances running at the sites. It covers all types of services, but for the case of the DDM dashboard data management services are of particular interest, namely the Local File Catalogs (LFC) and File Transfer Services (FTS) running at the Tier1 centres, and the Storage Resource Managers (SRM) and Storage Elements (SEs) running at each site.

For collecting this information an agent uses the SAM XSQL interface, which allows controlled queries to the backend database via HTTP requests.

3.3.2. Storage Space Monitoring Running out of storage space at a site is obviously a show stopper and should be prevented as much as possible. Currently the way available storage space is collected is by querying the BDII service. This is not always reliable, as there is no guarantee the information being published is VO specific. This agent has been developed keeping in mind that better data will be available in the future, so that the same kind of information can be also collected from different sources.

3.3.3. Statistics Collection This agent is responsible of generating cloud and site statistics concerning data movement - throughput, number of files and datasets moved, average dataset and file completion time, dataset and file time in queue, ... - and errors, both during transfer and registration. It generates these values every 10 minutes, giving a close to real time view of the system behavior.

3.3.4. Snapshot Collection Snapshots are usually gathered once per hour, and include detailed site information - size of the file and dataset queue, number of files in each transfer state, status of the underlying grid fabric services, etc.

3.3.5. Statistics Plotting As statistics are being generated, also plots are created showing the different metrics being gathered on a cloud and site basis.

3.3.6. Snapshots Plotting As for statistics plots, snapshot plots follow the snapshot collection, being updated every hour.

3.3.7. Site Information Collection ATLAS provides its own information system where the site topology is defined, called TiersOfAtlas [4]. The dashboard uses this as the main site information provider, complementing it with some information coming from the BDII service.

3.3.8. Alarms and Notifications A single agent is responsible of sending notifications and alarms to the system operators. It periodically checks the different metrics about the DDM site services and the corresponding grid fabric services - LFC, FTS, SRM, SE - and alerts when performance goes below a certain threshold, or one of the services becomes unavailable.

3.4. Performance

There are two main concerns when it comes to performance: the backend database, due to the amount of data being collected; and the web application endpoint, not really due to heavy data querying but due to it also being used to collect the different messages coming from the DDM site services.

Each of these points is described in more detail below, along with the solutions put in place to have a satisfactory quality of service.

3.4.1. Backend Database Rough figures used when designing the system indicated half a million datasets being created per year, each having 10 to 10000 files. With around 80 storage areas at the time (both disk and tape) it was easy to calculate a worst case scenario and from these numbers it was decided to rely on the powerful Oracle service provided by CERN IT.

The most impressive number comes not from the datasets or files, but from the individual file transfer state reports, as the transfers evolve in the state machine introduced in a previous section. This is of course temporary data, useful for real time debugging but less useful once information becomes old and statistics have been gathered. Even so, the period of time where this information is kept should be made as big as possible.

Different Oracle features were used to tackle the problem. *Connection Pooling* is used to reduce the overhead of establishing new database sessions, and this is true for the web application and all agents. *Table Partitioning* is especially important for the file state history table, but is also used in both datasets and files tables. In all cases the record creation time field is used for defining the different partitions, which prevents row movement from the start. And this same field is used in as many queries as possible, to reduce the amount of data being accessed by the different database sessions. The other important feature is related to *Bulk Insertions*. This is one of the most important performance enhancements for any database with high insertion rates, and it matches the need of the site services to also deliver messages in bulk due to the messaging system implementation. There is no mechanism implemented in the system to have partial success of a transaction doing bulk insertions, which means in the case of errors there is an inherent overhead introduced when performing the rollback.

3.4.2. HTTP endpoint Choosing HTTP as the messaging protocol means that the server receiving messages from the different site services instances is hit very hard and should be highly optimized. Monitoring the system has shown peaks of up to 60 requests per second, with 5 requests per second in normal operation. Considering that most of these (90%) are bulk requests, the load on the service is very high.

The option was to use the Apache Web Server, which has proved to be one of, if not the most performant implementation of an HTTP server. Still, some tuning had to be done to accomplish the necessary performance.

The first step was to use Apache in *worker MPM* mode, where requests are served by multiple threads in a single process (or multiple processes with multiple threads). This is the only way to have proper connection pooling in the web application, as the pool implementation used is limited to a single process. At the same time, having less processes and relying on threads to handle requests greatly reduces the memory footprint of the service. A second important feature is the use of the *KeepAlive* directive, which makes the server keep the TCP connection to a specific client open for a certain period of time after the request has been processed, in

the hope that it will issue a new request soon reducing the overhead of establishing new TCP connections.

Finally, the server is kept as minimalistic as possible, so most of the modules being loaded by a default Apache installation are removed and only the absolutely necessary is kept.

4. Conclusions and Future Work

The distributed nature of the ATLAS Data Management system increases the difficulty of monitoring the behavior of the overall system. Having a centralized point where monitoring data is collected, covering both ATLAS specific and grid fabric services is an ideal solution for site and service operators.

Using systems with recognized performance by both industry and research communities, and after several iterations and trials with multiple setups we have managed to provide a working system being used on a daily basis in a production environment.

Future work will certainly involve adding new functionalities, most likely improving the existing alarm system which is still rather limited. The messaging system between the site services and the monitoring application is another area of improvement, possibly replacing it by an equivalent one deployed and managed by the grid fabric infrastructure.

Acknowledgments

This work was funded by EGEE. EGEE is a project funded by the European Union under contract INFSO-RI-031688

References

[1] R. Jones et al, ATLAS Computing Model, January 2005
[2] R. Fielding, J Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, RFC 2616, June 1999
[3] LCG Service Availability and Monitoring (SAM), `https://twiki.cern.ch/twiki/bin/view/LCG/SAMOverview`
[4] Tiers of Atlas, `http://atlas.web.cern.ch/Atlas/GROUPS/DATABASE/project/ddm/releases/TiersOfATLASCache.py`
[5] Andreeva J, Gaidioz Benjamin, Herrala Juha, Maier Gerhild, Rocha Ricardo, Saiz Pablo, Experiment Dashboard: the monitoring system for the LHC experiments, Proceedings of the 2007 workshop on Grid monitoring