Dynamic and adaptive data-management in ATLAS

Mario Lassnig, Vincent Garonne, Miguel Branco, Angelos Molfetas
CERN PH-ADP/DDM, 1211 Geneva, Switzerland
Faculty of Mathematics, Computer Science and Physics, University of Innsbruck, Austria
E-mail: {`mario.lassnig,vincent.garonne,miguel.branco,angelos.molfetas`}`@cern.ch`

Abstract. Distributed data-management on the grid is subject to huge uncertainties yet static policies govern its usage. Due to the unpredictability of user behaviour, the high-latency and the heterogeneous nature of the environment, distributed data-management on the grid is challenging. In this paper we present the first steps towards a future dynamic data-management system that adapts to the changing conditions and environment. Such a system would eliminate the number of manual interventions and remove unnecessary software layers, thereby providing a higher quality of service to the collaboration.

1. Introduction

The high-energy physics experiment ATLAS [1] presents a data-intensive computing environment on an unprecedented scale. The middleware DQ2 [4] is responsible for the experiment's distributed data-management and already scales to multiple petabytes of managed data. DQ2 is built atop the EGEE gLite [7], NorduGrid ARC [5] and Open Science Grid (OSG) [2] middlewares and combines them into a *data-grid* with some *cloud-computing* [9] properties. DQ2 therefore provides a comprehensive single point of entry to all experiment data using a common interface and quality of service over multiple heterogeneous infrastructures. Figure 1 shows a high-level overview of the DQ2 architecture: the grid-middleware at the bottom, a common modular framework in use by all components, distributed site services to manage the data-flow, central databases for bookkeeping and the client interfaces to connect with externals tools and allow users to interact with the system.

The objective of our work is to make the data-management of ATLAS *robust* (able to operate despite abnormalities in the environment), *reliable* (perform the required functions under stated conditions for a specified period of time) and *high performance* (minimise the overall time to perform the required functions). These three properties place orthogonal requirements on the software architecture and therefore require in-depth studies of the runtime behaviour of the whole distributed data management system. This paper presents the first iteration of our work to address these properties from an architectural side for a future data-management system.

Our methodology to approach this problem is to collect usage *traces* from DQ2 in a production environment. These traces are logged interactions with DQ2. We analyse the instrumented traces for behavioural patterns, and then base our decisions for future developments and directions on these analyses. Due to the heterogeneous nature of the environment it is very difficult to present a global view of the behaviour. We have thus introduced a distributed tracing framework that is able to report monitoring data to a central logging facility. Section 2 gives details of the instrumentation. We continue in section 3 with an analysis of a few sample

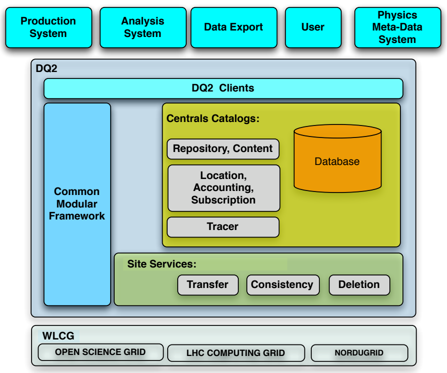

Figure 1. Overview of DQ2 architecture

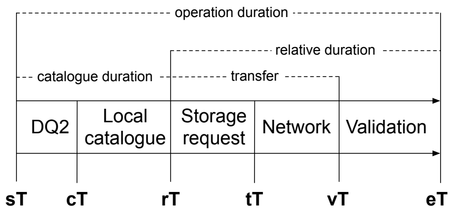

Figure 2. Instrumentation of dq2-get

metrics and show that any attempted prediction of system variables is near futile and therefore other measures must be taken. We conclude in section 4 with the outlook for future changes to DQ2 and a more general view of future improvements in distributed data-management.

2. Instrumentation and model

Accessing files for reading or writing from DQ2 is done through the DQ2 clients, called *dq2-get* and *dq2-put*. dq2-get retrieves files from DQ2 and copies the requested files outside of DQ2 management, similar to an FTP client in a non-grid environment. dq2-put uploads files to mass-storage systems, registers the files in appropriate book-keeping facilities and makes the files thus available through DQ2. Distributed analysis tools, like EGEE GANGA [6] or PanDA pAthena [8], access files through the DQ2 API directly and only in special cases, like constraining data access policies, fall back to calling DQ2 clients. The Tier-0, the experiment's first pass processing facility, also uses the clients to register initial experiment data into DQ2. The sum of all of these interactions forms the overall load and behaviour of the system.

Analysing the behaviour requires traces of these interactions and the DQ2 clients report every file transfer with meta-data information and timing information. Figure 2 shows a sample workflow for dq2-get when getting multiple files of a dataset and when instrumentation is measured. The following timing variables are instrumented:

(i) sT: startTime is the beginning of the operation
(ii) cT: catalogueTime is the beginning of the interaction with catalogues local to a specific grid site site. $cT - sT$ is the time spent querying DQ2 catalogues.
(iii) rT: relativeTime is the time when dq2-get forks into a parallel download of constituent files of a dataset. $rT - sT$ is the overall time spent in catalogues.
(iv) tT: transferTime is the time when the first byte arrives at the destination. $tT - rT$ is the time spent waiting for a (mass-)storage system to *stage* a file to be ready for transfer.
(v) vT: validationTime is the time when checksum calculations and file-size verifications begin.
(vi) eT: endTime is the end of the operation.

The following variables carry additional information about the transfer:

(i) A *UUID* (universally unique ID) for each operation. Every file transfer of a single dq2-get/put will have the same UUID. That way compound file transfers can be identified.
(ii) The *eventType*, if it's either a get or a put.
(iii) The *eventVersion* of the client tool involved.

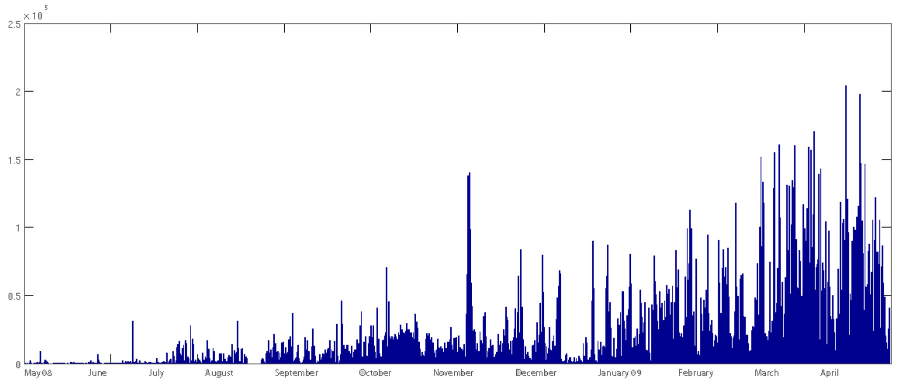

Figure 3. Aggregate volume of events (May 2008 – May 2009)

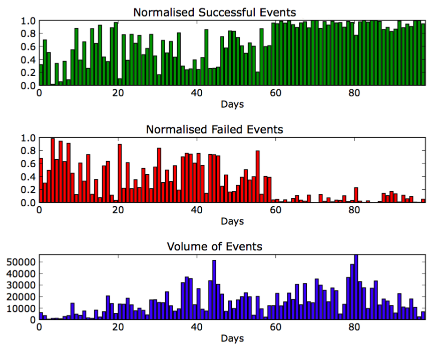

Figure 4. Normalised events/volume (site A)

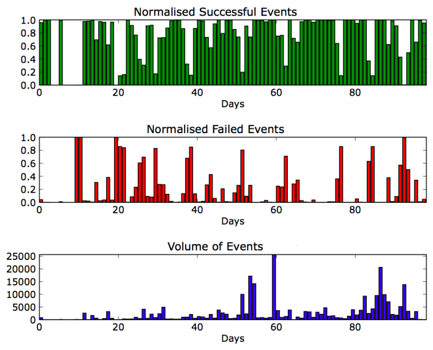

Figure 5. Normalised events/volume (site B)

(iv) The *localSite* and *remoteSite* involved in the transfer.
(v) The *dataset*, *version* and *file* information.
(vi) The number of *bytesTransferred*.
(vii) The *protocol* used in the transfer.
(viii) The client *exitState*.
(ix) An MD5 hashed *usrString* of the executing user's certificate distinguished name.

3. Analysis

Every single file-transfer, successful or not, will send one of these trace reports to a central server. At the time of writing there were already 24.558.068 traced file transfers. Figure 3 shows the increasing volume of transfers since May 2008, driven by the adoption and improvement of the DQ2 clients and API. One can clearly see a steady increase of usage during 2008 with an additional boost in 2009, when support for distributed analysis tools was implemented. It is also noteworthy that the volume of events is subject to burst behaviour and not constant.

The high number of transfers is a direct indicator of the performance of all involved components. Since it is very difficult to get all required information from a production grid infrastructure to perform studies such a statistical approach, exploiting the law of large numbers, to a production instance is novel and useful.

Figure 4 and figure 5 show two distinct sites A and B over a period of three months. The top green chart represents the normalised amount of successful events, the red middle chart the number of failed events and the bottom chart the volume of incoming events. When we now speak of a site's *performance* we mean the ability to successfully serve files. It is clear that the number of incoming events has no direct influence on the performance of a site. In fact, site A exhibits an enormous boost in performance at around day 60. Incidentally, this corresponded to the re-configuration of a new router at that site. Site B on the other hand has a high variance in its performance, again with no direct influence from the number of incoming events. What was observed at both sites, however, was a dependence on the number of concurrent requests. As long as transfers were throttled by the DQ2 infrastructure, sites performed acceptably. If the number of concurrent requests, e.g. by dq2-get, increased then site performance dropped remarkably; even as low as 10 concurrent requests could force a site into an unusable state.

These observations yield two important conclusions: first, any scheduling of data to or from a specific site is completely arbitrary, because there is no simple way of predicting future performance from history. This makes any a priori decision on pre-caching of data for ad hoc access impractical. Second, the number of scheduled requests to a site does not make an impact on the success-rate. Consequently, we only need to worry about synchronous, direct requests.

4. Conclusion and future work

Based on our current observations, we argue that a priori caching and movement of data is disadvantageous to overall system performance. The infrastructure should keep a dynamic minimum amount of data replicated automatically and respond automatically to a high variance in observed system behaviour from historical data. For example, data can be annotated to specify the minimum number of replicas for redundancy and access optimisation. Access patterns traced from the DQ2 clients can then increase this number automatically over time if data is requested frequently. This mode of operation could even be accomplished with DQ2 as it is deployed right now. That way, data would be accessible even if potentially well-performing site experience unexpected problems. Furthermore, bandwidth experiments [3] have already shown that less than half of the available network capacity is used between sites, so only access to mass-storage systems is shown to be the bottleneck. By reducing the number of a priori transfers this burden on the mass-storage systems can be eliminated and consequently, the reduced load will free up system resources that could be used for ad hoc dynamic access to data. That way DQ2 could respond to tracer data to automatically and adaptively manage stored data.

Acknowledgments

We are grateful to Ricardo Rocha, Benjamin Gaidioz, Simone Campana, Pedro Salgado, Johannes Elmsheuser und Paul Nilsson and Torre Wenaus for their contributions to DQ2.

References

[1] ATLAS Collaboration, "ATLAS Technical Proposal", *CERN*, 1994

[2] P. Avery et al., "Open Science Grid: Building and Sustaining General Cyberinfrastructure Using a Collaborative Approach", *First Monday*, Vol. 12(6), First Monday, 2007

[3] I. Bird et al., "The organization and management of grid infrastructures", *Computer*, Vol. 42(1), IEEE Computer Society, 2009

[4] M. Branco et al., "Managing very-large distributed datasets", *Lecture Notes In Computer Science*, Vol. 5331, Springer, 2008

[5] M. Ellert et al., "Advanced Resource Connector middleware for lightweight computational Grids", *Future Generation Computer Systems*, Vol. 23, 2007

[6] J. Elmsheuser et al., "Distributed analysis using GANGA on the EGEE/LCG infrastructure", *Journal of Physics: Conference Series*, Vol. 119(072014), IOP, 2008

[7] E. Laure et al., "Middleware for the next generation Grid infrastructure", Proc. of *Computing in High Energy Physics and Nuclear Physics*, Interlaken, Switzerland, 2004, pp. 826

[8] T. Maeno, "Distributed Analysis on Panda", https://twiki.cern.ch/twiki/bin/view/Atlas/DAonPanda, BNL/CERN, 7. May 2009

[9] A. Weiss, "Computing in the Clouds", *netWorker*, Vol. 11(4):16-25, 2007