The CMS Data Management System

M Giffels[1], Y Guo[2], V Kuznetsov[3], N Magini[1] and T Wildish[4]

[1] CERN, CH-1211 Genève 23, Switzerland
[2] Fermi National Accelerator Laboratory, Batavia, Il, USA
[3] Cornell University, Ithaca, NY, USA
[4] Princeton University, Princeton, NJ, USA

E-mail: Nicolo.Magini@cern.ch

Abstract. The data management elements in CMS are scalable, modular, and designed to work together. The main components are PhEDEx, the data transfer and location system; the Data Booking Service (DBS), a metadata catalog; and the Data Aggregation Service (DAS), designed to aggregate views and provide them to users and services. Tens of thousands of samples have been cataloged and petabytes of data have been moved since the run began. The modular system has allowed the optimal use of appropriate underlying technologies. In this contribution we will discuss the use of both Oracle and NoSQL databases to implement the data management elements as well as the individual architectures chosen. We will discuss how the data management system functioned during the first run, and what improvements are planned in preparation for 2015.

1. Introduction

The CMS [1] experiment at the LHC has recently successfully concluded its first run. Between the fall of 2010 and the spring of 2013, over 10 billion raw data events were recorded, and 15 billion Monte Carlo events produced. The data were analysed using the distributed resources of CMS, managed through the Worldwide LHC Computing Grid (WLCG) [2] and through the CMS dataflow and workflow management tools.

The operational aspects of the CMS computing system are described separately [3]. In this paper we describe the performance of the CMS data management system during the first run of LHC. We describe what worked well and what didn't, and analyse the factors that contributed to the performance. Finally, we describe the improvements planned in the major subsystems for the second run of LHC, starting in 2015.

2. Architecture

2.1. Components and data organisation

The CMS data management system has several important tasks:

- Data bookkeeping catalog - describes the data contents in physics terms
- Data location catalog - maintains knowledge of replicas of the data around CMS
- Data placement and transfer management

The core components which achieve these goals are:

- PhEDEx [4], the data-transfer management system

- DBS [5], the Data Bookeeping Service
- DAS [6], the Data Aggregation Service

There are a number of other, more specialised services, such as the RunRegistry, SiteDB, ConditionsDB and ReqMgr, but the core components are the ones the user mostly interacts with. All the components of the data management system are designed and constructed separately, interacting with each other and external users by means of web services.

CMS data is organised into datasets by physics content. Dataset size varies considerably, usually in the range of 0.1 - 100 TB. Within a dataset, files are organised into *blocks* of 10 - 1000 files, purely as an aid to scalability. The data management tools can manipulate individual blocks of files, instead of entire datasets, which offers a finer granularity of control.

Files are typically about 2.5 GB in size. Small files from various processes are merged to maintain a minimum useful size to help scalability of storage and catalogs.

2.2. Computing infrastructure

The CMS computing infrastructure distinguishes between several tiers of computing centres, with a single Tier-0, several Tier-1s, and a large number of Tier-2s.

The Tier-0 is responsible for custodial storage of a copy of the raw data, for calibration and prompt reconstruction and for distributing the raw and reconstructed data to the collaboration. The Tier-1s between them maintain a second custodial copy of the raw data, host Monte Carlo production and re-reconstruction, and distribute data further to the Tier-2s. The Tier-2 sites are where the physicists perform their analyses. Monte Carlo data produced at the Tier-2s is uploaded to the Tier-1s for custodial storage and distribution to other sites.

The original computing model [7] called for a hierarchical organisation of tiers, with each Tier-2 attached to a single Tier-1 and each Tier-1 serving only a handful of Tier-2s. For our system (one Tier-0, 7 Tier-1s, and about 50 Tier-2s) this would mean less than 100 bidirectional data-transfer links. In practice, we now allow Tier-2s to connect to all Tier-1s and to all other Tier-2s, resulting in a fully-connected mesh of about 2000 data-transfer links.

2.3. Data location and the Trivial File Catalog

CMS does not use a central file catalog to store information about file location at each site. Instead, files are known by their *Logical File Name* (LFN), which maps to a different *Physical File Name* (PFN) at each site. We maintain the knowledge that a file exists at a site by storing the LFN-to-site mapping (in PhEDEx), but the mapping from LFN to PFN is the responsibility of each site.

To achieve this, each site maintains a *Trivial File Catalog* (TFC), which is essentially a regular-expression map from LFNs to PFNs and vice-versa. Anyone wishing to know the PFN for a file at a site simply processes the LFN through the regular expressions in that site's TFC to obtain their answer. This approach has several advantages:

- Batch jobs running at the site do not need to contact central services to perform LFN to PFN conversions, they simply examine the local TFC,
- Sites can change their storage layout on the fly, without synchronising changes in each PFN to a central service. As long as they maintain their TFC in sync with their storage, this is transparent to the rest of the system,
- Sites can perform complex matching for special situations. E.g. by matching the first letter of a GUID portion of an LFN, files can be load-balanced across several backend servers.

3. PhEDEx

The Physics Experiment Data Export (PhEDEx) [4] project was started in 2004 to manage global data transfers for CMS over the grid in a robust, reliable, and scalable way.

PhEDEx is based on a high-availability Oracle database cluster hosted at CERN (Transfer Management Data Base, or TMDB) acting as a "blackboard" for the system state (data replica location and current tasks). Users can request transfers of datasets or blocks of files through the interactive PhEDEx web site [8]; a web data service [9] is also available for integration with other CMS data management components.

PhEDEx software daemon processes or "agents", based on the Perl Object Environment (POE) [10] framework, then connect to the central database to retrieve their work queue, and write back to TMDB the result of their actions. The TMDB has been carefully designed to minimize locking contention between the several agents and cache coherency issues by using a row "ownership" model where only one specialized agent at a time is expected to act on a given set of rows. Each of the agents performs a specific task, progressing the transfer state machine towards the desired final state: creating a new replica of the files at the destination.

Central agents running at CERN perform most of the intelligence of data routing and transfer task creation, expanding the blocks into their current file replicas, and calculating the path of least cost for each file from the available sources to the destination. The download agents running at the sites fetch these tasks from TMDB and initiate the file transfers using specific plugins for different grid middleware. To ensure reliability and robustness, each file transfer is independently verified, and intelligent backoff and retry policies are applied in case of failure, aiming for eventual completion of all transfer subscriptions.

Performance metrics are constantly recorded in TMDB, summarising snapshots of the TMDB state into dedicated status monitoring tables at regular intervals. Historical information is further aggregated from the status tables into time-series bins of data on transfer volume, transfer state counts, and number of failures. Since the website and data service only access these monitoring tables instead of the live tables to provide monitoring information, the performance of the user monitoring and of the transfer system are largely decoupled.

Additional data management actions that may be requested with PhEDEx and are executed by dedicated agents are data deletion and data consistency verification.

3.1. PhEDEx performance

During the first run of LHC, CMS steadily transferred data with PhEDEx with peaks in global speed exceeding 5 GB/s, distributing more than 100 PB of replicas over all sites.

The scalability of PhEDEx as transfer management system was proven multiple times in the past with realistic simulations running in dedicated testbed instances, where the aggregated simulated transfer rates were pushed far beyond the requirements of CMS. The tool used to run the tests was the "LifeCycle agent" [11], designed to exercise all components of the system for an extended period of time. The LifeCycle agent simulates the behavior of CMS data production components and users, regularly injecting new file replicas at various nodes on a clone of the production infrastructure. It then subscribes data to other nodes for transfer, and requests deletion of some of the data. The site download agents in the testbed are set up to execute fake transfers with a configurable bandwidth and failure probability, exercising error handling and retries. During the latest stress test with the LifeCycle agent in 2011, the system continued to work well running simulated transfers at rates at least ten times higher than the scale of production transfers.

3.2. PhEDEx improvements

PhEDEx was a mature product during data taking in LHC Run 1, and development was focused on adding support for more flexible workflows, and on providing more tools for the transfer

operators. PhEDEx 4.0 was released in early 2011 and added full support for transfers and deletions of individual blocks, allowing to subscribe only a part of a dataset. PhEDEx 4.1, deployed in 2012, included a new monitoring system for transfer latencies, which was described in [12]. In this release cycle the core agent and namespace libraries were also gradually refactored [13] and can now be used in other projects; the Namespace framework, in particular, was the base for the new storage space monitoring system [14]. Current effort is dedicated to a new framework to handle user and operator requests, able for example to support requests to invalidate data [15].

4. DBS

The Data Bookkeeping Service (DBS) [5] provides a catalog of event metadata for Monte Carlo and recorded data of the CMS experiment. DBS contains record of what data exist, their process-oriented provenance information, including parentage relationships of files and datasets, their configurations of processing steps, as well as associations with run numbers and luminosity sections to find any particular subset of events within a dataset, on a large scale of about $200,000$ datasets and more than 40 million files, which adds up to around 700 GB of metadata.

The current DBS, DBS 2 [16] was designed in 2006-2007, before the LHC started its operation. CMS did not have a standardised service architecture for the implementation, deployment and operation of that kind of web service. Thus DBS 2 was implemented using Java servlets in an Apache Tomcat container and XML RPC has been the first choice for the client-server communication. As persistent storage system of the metadata, an Oracle database backend provided for CMS [17] is utilised. DBS 2 additionally supports a MySQL database backend, however it is currently not used in the production environment.

The DBS is a federated system with multiple instances for different scopes. Driven by the principle in CMS to separate official and user-created data, the Global-DBS contains metadata related to all official CMS data, real or simulated. Metadata related to user-created datasets can be stored in two physics analysis DBS instances. Besides those instances there are also a CAF instance, that records data from the prompt reconstruction stream used for detector calibrations and diagnostics, and a CMS Tier-0 DBS instance, that records information for data from the detector as it is processed by the Tier-0 facility.

Although DBS 2 sustained the load in LHC Run 1, it has in some cases very "thick" client APIs, which led to numerous problems with API versioning and scalability issues. In addition, the CMS data processing model has evolved a lot, in a way that could not be anticipated by the DBS 2 design back in 2006, so many requests were made to store additional data in DBS 2, which were not entirely consistent with its original purpose. A project review in 2009 led to the decision to re-design DBS, to better match the CMS data processing model and to better integrate with the DMWM projects.

The CMS DMWM project has meanwhile developed a standardised architecture based on the Representational State Transfer API (REST) [18] for its web services based on Python, CherryPy and SQLAlchemy. Thus DBS 3 has been re-designed and re-implemented in Python utilizing the CMS DMWM standards for RESTful web services. The client-server communication is stateless and REST also imposes the discipline of thin clients, which enhances the scalability of the service. Javascript Object Notation (JSON) is used as data-format for the client-server communication, being a lightweight replacement for XML RPC. The database schema of DBS 3 also has been revised, based on the experiences with DBS 2. The schema has been denormalised, since the DBS 2 schema was fully normalised, leading to excessive table joins and lock contention. In DBS 3 some tables were removed, as well as some relationships that were better modeled outside of the DBS schema. These changes sped up searches and also improved the insertion of data by removing foreign keys and lock contention. In addition, the DBS 3 particularly benefitted from the narrower and more precise project scope compared to DBS 2. The integration with

other services (PhEDEx and DAS) in the CMS DMWM project made a narrower scope possible, without impairing the query features, so for example the data location is not anymore stored in DBS, since it is naturally available from PhEDEx. DBS 3 is currently deployed in parallel to DBS 2 for validation and integration with other DMWM projects. The final switch to DBS 3 is expected by the end of 2013.

5. DAS

The Data Aggregation Service (DAS) [6] was designed to provide a uniform access to distributed CMS data-services regardless of their security policies, implementation details and data storage solution. DAS provides users with the ability to query underlying data-services via the DAS Query Language (QL). It also aggregates metadata information from various data-services into common records suitable for end users. Here we discuss the details of DAS architecture and the current status of the system.

DAS was built on top of a NoSQL document-oriented database (MongoDB [19]). It provides several benefits for the DAS use case: schema-less storage, embedded query language and very fast read/write operations.[1] DAS has a modular design based on the MVC architecture [20] and performs the following workflow upon user query:

- fetch data from underlying data-services via existing APIs
- store unprocessed data into the cache database
- process data and perform its transformation to a common data representation (we unify differences in naming conventions, units and data-formats, etc.)
- aggregate data on a requested key, e.g. dataset, block, etc.
- store results into the merge database
- present results to end-users and provide a set of filters as well as aggregation functions to perform basic operations for slicing and representing the data in a form suitable for user tasks

This design allowed to develop DAS in a data-agnostic manner. For instance, data can be fetched from underlying data-services regardless of their data format, e.g. JSON, XML, CSV, etc. The data transformation was done via an external set of mappings which was maintaned separately from DAS code development. It also gave us a few benefits which were not foreseen from the design cycle. For example, data aggregation on a common entity, e.g. data name, may show any existing discrepancy in underlying services.

DAS uses an in-house Query Language which was based on entity relationships used by physicists, see [6]. It consists of the following structure:

```
<selection keys> <set of conditions> | <filters> | <aggregators>
```

The selection keys were based on well known entities such as dataset, block, file, run. The conditions were formed via key=value pairs. DAS-QL provides a limited set of filters such as grep, unique, as well as set of common aggregation functions such as sum, min, max. The former support conditional operators and grouping, while the latter can be extended via custom map-reduce functions to support more complex use cases. Therefore to express a question *Find me all datasets at a given site and show only those which have size greater than 50* someone will need to write a DAS query in the following way:

```
dataset site=XYZ | grep dataset.name, dataset.size>50
```

[1] Our benchmark showed that MongoDB can sustain 20k doc/s for reading and 7k doc/s for writing operations, respectively.

It turns out that such flexibility was not always clear to some users, mostly those who were unfamiliar with the DAS-QL syntax. Therefore a further attempt was made to build native support for keyword queries on top of DAS-QL, see [21].

DAS operates as an intelligent cache in front of CMS data-services. It stores results into two caches upon a provided query. The raw-cache is used to store results from data-services *as is*, while the merge-cache stores aggregated records. The lifetime of the records is based on information provided by data providers via HTTP headers. The record maintenance is done in a lazy fashion, i.e. upon a new user query expired records are wiped out from the cache, while new ones come in. DAS server performs many operations in parallel, e.g. it sends concurrent HTTP requests to underlying data-services, processes and stores data using multiple threads, and runs multiple monitoring and pre-fetching daemons. The server runs on a single 8-core node with 24 GB of memory required for efficient MongoDB operations[2].

The discussed modular design, flexible QL and NoSQL storage allows DAS to aggregate information from distributed data-services without imposing any requirements on them. DAS is able to deal with different security models, various APIs, data-formats and naming conventions. Right now it uses dozens of CMS data-services. Data are aggregated into JSON records based on common entity keys so that it is possible to see information from multiple data-services in a single record, e.g. run information comes from DBS, Condition DB and RunRegistry and is represented as a single JSON document listing information from three data-services. Daily load on the DAS server is constanly growing and has about 10k queries/day with $\sim O(10M)$ records going in and out of the DAS cache.

6. Conclusions

The decision to base Data Management on independent core components, with a common user interface provided by the Data Aggregation Service, has brought CMS several advantages. Each of the underlying services is based on the most appropriate technology and can be optimised independently for scalability, evolving without disrupting the overall system.

The components are interfaced to each other through a common CMSWEB web service framework, which simplifies integration and regression testing when rolling out new service versions. Exposing the Data Management components through web service interfaces also allows to easily build external services that can integrate their information in a clean manner.

For example, the Victor [22] data cleaning service was developed in 2011 to identify data replicas that are no longer accessed, and can be deleted without disrupting user analysis. Victor is interfaced to PhEDEx through the data service to discover dataset replicas at each site and overall space usage at the sites, and queries a dataset popularity service for the access frequency of file replicas. Combining these data, Victor can then provide lists of the least accessed dataset replicas to be deleted freeing up space at full sites. Looking ahead, the system could be further extended with an external dynamic data placement service, interfaced to PhEDEx to place requests for new replicas of the datasets that are most heavily accessed according to the popularity service.

In conclusion, CMS has developed a Data Management system that performed successfully during the first LHC run, can be flexibly extended and is ready to manage the increased scale of data production during the second run of LHC.

References

[1] The CMS Collaboration 2008 The CMS experiment at the CERN LHC *JINST* **3** *S08004*
[2] Knoblock J *et al.* 2005 LHC Computing Grid Technical Design Report *CERN-LHCC-2005-024*
[3] Gutsche O *et.al.* CMS Computing Operations during Run 1, *submitted to CHEP 2013*

[2] MongoDB relies on indexes fitted in RAM to provide its superior speed

[4] Egeland R, Wildish T and Metson S 2008 Data transfer infrastructure for CMS data taking, *XII Advanced Computing and Analysis Techniques in Physics Research (Erice, Italy: Proceedings of Science)*
[5] Giffels M, Gui Y and Riley D, Data Bookkeeping Service 3 - Providing event metadata in CMS, *submitted to CHEP 2013*
[6] Kuznetsov V, Evans D and Metson S The CMS Data Aggregation System *doi:10.1016/j.procs.2010.04.172*
[7] The CMS Collaboration 2005 The Computing Project Technical Design Report CERN-LHCC-2005-023
[8] R Egeland *et.al.* 2012 The PhEDEx next-gen website *J. Phys. Conf. Ser.* **396** 032117
[9] R Egeland *et al.* 2010 PhEDEx Data Service *J. Phys.: Conf. Ser.* **219** 062010
[10] The Perl Object Environment (POE) *http://poe.perl.org/*
[11] T Wildish *et al.* Integration and validation testing for PhEDEx, DBS and DAS with the PhEDEx LifeCycle agent, *submitted to CHEP 2013*
[12] T Chwalek *et al.* 2012 No file left behind - monitoring transfer latencies in PhEDEx *J. Phys.: Conf. Ser.* **396** 032089
[13] A Sanchez-Hernandez *et al.* 2012 From toolkit to framework - the past and future evolution of PhEDEx *J. Phys.: Conf. Ser.* **396** 032118
[14] C-H Huang *et al.* 2012 Data Storage Accounting and Verification at LHC experiments *J. Phys.: Conf. Ser.* **396** 032090
[15] C-H Huang Request for All - Generalized Request Framework for PhEDEx, *submitted to CHEP 2013*
[16] Afaq A *et al.* 2008 The CMS Dataset Bookkeeping Service *J. Phys. Conf. Ser.* **119** 072001
[17] Pfeiffer A *et al.* 2012 CMS experience with online and offline databases *J.Phys.Conf.Ser.* **396**
[18] Fielding R T 2000 Architectural Styles and the Design of Network-based Software Architectures, Dissertation, *University of California, Irvine* ISBN: 0-599-87118-0
[19] http://www.mongodb.org/
[20] Model-view-controller architecture *http://en.wikipedia.org/wiki/Model-view-controller*
[21] Zemleris V and Kuznetsov V Keyword Search over Data Service Integration for Accurate Results, *submitted to CHEP 2013*
[22] F H Barreiro Megino *et al.* 2012 Implementing data placement strategies for the CMS experiment based on a popularity model *J. Phys.: Conf. Ser.* **396** 032047