Structured storage in ATLAS Distributed Data Management: use cases and experiences

Mario Lassnig\textsuperscript{1}, Vincent Garonne\textsuperscript{1}, Angelos Molfetas\textsuperscript{2}, Thomas Beermann\textsuperscript{1}, Gancho Dimitrov\textsuperscript{1}, Luca Canali\textsuperscript{1}, Donal Zang\textsuperscript{3}, Lisa Azzurra Chinzer\textsuperscript{4}, on behalf of the ATLAS Collaboration

\textsuperscript{1} European Organization for Nuclear Research (CERN), 1211 Geneva, Switzerland
\textsuperscript{2} University of Sydney, New South Wales 2006, Australia
\textsuperscript{3} Chinese Academy of Sciences, Beijing 100049, China
\textsuperscript{4} Università di Napoli “Federico II”, 80138 Napoli, Italy
E-mail: mario.lassnig@cern.ch

Abstract. The distributed data management system of the high-energy physics experiment ATLAS has a critical dependency on the Oracle Relational Database Management System. Recently however, the increased appearance of data warehouse-like workload in the experiment has put considerable and increasing strain on the Oracle database. In particular, the analysis of archived data, and the aggregation of data for summary purposes has been especially demanding. For this reason, structured storage systems were evaluated to offload the Oracle database, and to handle processing of data in a non-transactional way. This includes distributed file systems like HDFS that support parallel execution of computational tasks on distributed data, as well as non-relational databases like HBase, Cassandra, or MongoDB. In this paper, the most important analysis and aggregation use cases of the data management system are presented, and how structured storage systems were established to process them.

1. Introduction

ATLAS is a global collaboration of more than 3000 persons to conduct high-energy physics research. The collaboration uses the ATLAS particle detector, which is installed at the Large Hadron Collider (LHC), to record the results of particle collisions. Both the LHC and ATLAS detector are located in Geneva, Switzerland at the European Organization for Nuclear Research (CERN). The data that is recorded by the detector is then processed by globally distributed computing centres for analysis. Additionally, the computing centres produce simulated particle collisions for comparison with detector data. The aggregate data volume amounts to 20 Petabytes per year, not including temporary and transient data, and is currently at 90 Petabytes. This rate is likely to
grow in correspondence with the energy and luminosity increases of the LHC in the coming years.

ATLAS uses a distributed data management (DDM) system, called Don Quijote 2 (DQ2), to organise experiment data[1]. This includes the registration of files, the aggregation of files into data sets, the transfer of data sets between computing centres, consistency verification and repair, monitoring and reporting, quotas, and permissions. More importantly however, the DQ2 system provides a platform as a service (PaaS) for members of the collaboration to address specific data management needs. For example, if a physics subgroup of the collaboration needs to reorganise some data in a different way for analysis, they can use the platform programming interface in the DQ2 system to achieve this.

Data management operations are issued to the DQ2 system using clients that submit remote procedure calls. Currently, this amounts to a transactional workload of about 25 million daily events, equivalent to an average rate of 300 per second. A historical record of these data management events is stored in a relational database and log files. These events are analysed for trends, data popularity, placement and deletion decisions, accounting reports, and much more. In the current implementation, relational databases play a critical role for LHC computing in general and for ATLAS workloads in particular. Notable examples are their role for online data acquisition at the detectors, for offline data reconstruction and analysis and for managing file catalogues and in support of grid job management.

The DQ2 system has two types of database workload, transactional and analytical processing of historical information to generate summaries. The analytical processing of historical information is very time consuming due to the large number of rows that need to be processed. Additionally, the high dimensionality of the rows increases the number of possible summaries that can be requested. The analytical processing of historical events put a very high load on the database, which degraded the database’s ability to handle the transactional load, thus affecting for DQ2 system as a whole. For this reason, automated procedures were put in place that export historical information daily into various different database backends where the information is processed separately.

The choice of technology for analytical processing is important in view of data growth and future introduction of new use cases. In particular, factors like scalability, throughput, cost and ease of use are critical. Historically, CERN has been strongly focused on relational, Oracle based, solutions but these new analytical workloads have shown in some cases to be cumbersome to implement in the Oracle ecosystem without specialist knowledge. Additionally, due to the shared nature of the Oracle instance at CERN, execution time of the analytical workloads has been unsatisfactory, and sometimes even had detrimental effects on the throughput of other, non-analytical, transactions. However Oracle-based solutions have costs associated with procurement of additional hardware (CPU and storage) of the type suitable for Oracle production databases, and in some cases require additional Oracle licensing costs. For this reason, several other technologies were evaluated that could help offload the Oracle database and process the workload. Structured storage systems, colloquially called NoSQL systems, promise to deliver high throughput, high availability, and fault tolerance on relatively cheap commodity hardware. The purpose of this paper is therefore to report on the
The remainder of the paper is organised as follows: first, a description of the different technologies that were used are presented, including Oracle, MongoDB, Cassandra, and Hadoop. Second, six different workloads are described, HTTP sharing, log file centralisation, log file analysis, trace analysis, wildcard search, and accounting. Each use case presents a separate case study, since different technologies and implementations have been used in each case. Finally, a summary is given on the experiences and lessons learnt.

2. Technologies

This section describes the most important features of the technologies that were used: Release 10g and Release 11g by Oracle Corporation with the Partitioning, Real Application Clusters and Real Application Testing options[9], MongoDB 1.8.5 by 10gen, Inc.[8], Cassandra 0.8.4 by the Apache Software Foundation[2], and the Cloudera Distribution for Hadoop cdh3u3 by Cloudera, Inc.[7] and the Apache Software Foundation[3].

2.1. Oracle

The Oracle Database is a commercial object-relational database management system, and is the main database in use in the ATLAS experiment. Oracle is primarily geared towards online transactional workloads (OLTP), and provides an immense feature set for clustering, high-availability, backup, and much more. Oracle is also the primary database for the DDM system, having more than 4 billion rows stored, and an average transaction rate of 300 Hz.

Table 1 gives an overview of the used hardware, both for the Oracle instances, and the nodes used for the structured storage systems. The immediate observation is that the structured storage cluster has many more nodes available than the Oracle instances. But whereas the Oracle instances are connected to a shared storage system, each node

|                  | Oracle Release 10g2 | Oracle Release 11g2 | Structured Storage |
|------------------|---------------------|---------------------|--------------------|
| Nodes            | 3                   | 4                   | 12                 |
| Architecture     | Linux x86_64        | Linux x86_64        | Linux x86_64       |
| CPU Cores        | 16 (L5410)          | 16 (E5630)          | 16 (L5520)         |
| RAM              | 16 GB               | 48 GB               | 24 GB              |
| Storage          | Oracle ASM          | NetApp FAS3240      | –                  |
| Storage Network  | Dual FC 4Gig        | Dual 10 GigE        | –                  |
| Disk             | 140 SATA            | 72 SATA             | 24 SATA            |
| Cache            | 12GB                | 2GB + 512GB SSD     | –                  |
| Network          | 1 GigE              | 10 GigE             | 1 GigE             |

Table 1: Configuration of Oracle Database, and the structured storage cluster nodes.
in the cluster has its own disks in a JBOD configuration. Additionally, Release 11g has a low-latency SSD cache to improve query response time.

An Oracle technology that has recently shown to be of value for the database community at CERN is Oracle 11g Active Data Guard. This technology allows the deployment of read-only replicas of production databases, with minimum latency. That way, analytical workloads could run on a dedicated instance directly on production data. Therefore, offloading analytical workload into specialised Oracle database systems is a possibility that will be investigated further.

2.2. MongoDB

Written in C++, MongoDB claims to be a scalable, high-performance NoSQL database. Its distinguishing features as a structured storage system are that it is document-oriented and supports indexes. Document-oriented storage does not rely on a static schema for its contents, but instead models the schema together with the data in an associative array identified by a unique key. These associative arrays are called collections, and can contain sub-collections. That way, dependencies can be expressed in a natural way, without having to create relational schemas. An example schema can be seen in table 2. The document is specially marked with the _id row key, and then has two additional keys, groups, and selections. Groups stores an ordered array of values natively, whereas selections contains a sub-document with two keys again. This example schema will be used later again to assess the other non-relational databases.

Additionally, MongoDB provides a distributed file system called GridFS to store large files. Each file is split into chunks that are stored in a dedicated binary collection, and file metadata is stored in a metadata collection.

Unfortunately, after first evaluations MongoDB was found unsuitable for analytical workloads because of its limited data processing capabilities. Only a single data processor can run at one time, which is not sufficient. Additionally, there are global locks on tables, which presents a problem with the heavily concurrent use cases, and the requirement to explicitly partition the data, which makes the data model not automatically scalable. However, MongoDB was found suitable as an easy to deploy backend for smaller application, that is, up to 100 million rows, that need fast look-ups. For the mentioned

```
{
   _id: 'Main Account User', // Row key
   groups: ['group_a', // Key containing native array
            'group_b',
            'group_c'],
   selections: { // Key containing sub document
      'select_a': 123, // Sub key
      'select_b': abc
   }
}
```

Table 2: Example schema in MongoDB.
use cases however, MongoDB was not pursued further.

In the meantime, MongoDB version 2 was released that started to address these concurrency issues. Also the possibility of multiple data processor is being implemented. Therefore, MongoDB will definitely be evaluated again in the future.

2.3. Cassandra

Written in Java, Cassandra claims to be a **highly scalable, eventually consistent, distributed, structured key-value store**. Its distinguishing feature is the high-availability of the system, with no single point of failure, and index support. It uses a column-based data model, in which key-value pairs are stored in hierarchical collections, so called **Column Families**. Each column family corresponds to a separate file, therefore column families should contain key-value pairs that belong together, that is, contain the results of a query. Additionally, each column can be indexed as well. There are no compound indexes however.

Table 3 shows the equivalent example schema as previously used for MongoDB. The most prominent difference to MongoDB is that the actual value of the row identifier becomes implicit, that is, there is no \_id key. Sub-schemas, like **selections** are supported as well. Additionally, keys and values are Java byte streams. This means that every Java object that can be serialised into a bytestream, for example, an array, can be used as both either key or value.

2.4. Hadoop

Written in Java, Hadoop claims to be a framework for **reliable, scalable, distributed computing**. Its distinguishing feature is that it is not only a database, but rather a modular ecosystem with many components. It is modelled after the basic data infrastructure by Google, the **Google File System** and **BigTable**. Hadoop includes a distributed file system, a distributed processing framework, a non-relational database,
Table 4: Example schema in HBase.

and several implicit and explicit data processing programming languages. It’s basic architecture has a master node, containing both the HDFS index, called namenode, and the job scheduler, called jobtracker. The cluster then consists of multiple storage and worker nodes, called datanode and tasktracker respectively. The master node is the single point of failure.

The Hadoop Distributed Filesystem (HDFS) is the most prominent component. At its core, HDFS stores files that can be organised in directories. Each file, if necessary, is split according to a preset chunksize, 64 Megabytes by default, and these chunks are distributed in whole to the nodes in the cluster. A replication factor can increase the number of additional copies per block, thus yielding fault-tolerance and read throughput improvements through repetition. Should a node in the cluster fail, then the lost blocks will be automatically and transparently replicated from the existing copies. Additionally, HDFS can be mounted as a filesystem in user space via FUSE, allowing to be used as a regular POSIX filesystem. However, since HDFS has delayed and reordered writes, the contents of written files are not guaranteed to be available immediately, as required by POSIX. If the file must be used immediately after changing its contents, then a throwaway operation is required, like a single read of the first byte of the file to /dev/null, to force the file to flush to HDFS. This issue only appears in the HDFS via FUSE case, and never when using HDFS directly via Hadoop.

The MapReduce framework in Hadoop can schedule and run tasks on the cluster. It reads HDFS blocks in parallel and organises the data processing in two phases, a selection phase called the mapper, and an aggregation phase called the reducer. The mapper only selects which parts of the input file should be read, and the reducer then works on these selections. This framework is therefore especially useful for embarrassingly parallel workloads. As an example, the mapper might select every line from a CSV file where the first column matches a particular pattern, and the reducer could then sum up the values of the selected lines.

HBase is a database for column-based key-value pairs, and uses HDFS as its storage system. Its architecture supports multiple HBase masters, and therefore does not have a single point of failure. It’s distinguishing features are that it supports billions of rows
times millions of keys, that key-value pairs can be versioned, and that it can serve as a source and sink for all data operations within Hadoop. For example, it is possible to write a MapReduce job that reads data from a flat file and writes the results natively to HBase without conversion, and vice versa. As shown in table 4, the distinction in the schema model is that a particular row key can have many column families, which is exactly the opposite to the Cassandra model. Each table has a separate file, therefore it becomes less important to organise the table for a potential query. This however can potentially result in higher IO, as useless data needs to be written. It is therefore suggested in the HBase documentation to keep the number of column families lower, and instead put the necessary data into the key-value pairs, for which there is no limit.

Finally, there are several data processing languages available, most prominently Hive [5] and Pig [6]. Both adhere to the basic concept that a data source is defined column-wise, like a table in a relational database, on which operations can be executed. These operations are then transformed into appropriate MapReduce operations. The difference with both approaches is that Hive supports SQL statements to query the data, whereas Pig uses its own data pipeline mechanism. The advantage of using SQL is especially important when legacy code needs to be ported. However, it does come with the disadvantage that query plans need to be evaluated, which is dependent on statistics gathered on the data itself. On the other hand, the data pipeline approach of Pig works in an explicit way by binding subsets of the data to variables. Through merging of the algebraic dependencies of the variables the shortest possible access path to the data can be provided, resulting in drastically reduced IO reads.

3. Use cases

In this section, six use cases are presented that showcase how structured storage systems have been integrated into the ATLAS distributed data management system. This includes the sharing of files via HTTP, centralisation and analysis of API logs, the analysis of event traces, wildcard searches on the data management repository catalogues, and accounting of the contents of the data management system.

3.1. Log file centralisation and analysis

The DDM programming interface is load-balanced amongst six web-based frontends, served by Apache httpd. The logs files written by Apache httpd contain a wealth of information on the usage of the system, and can be analysed for usage patterns. Currently, the daily aggregate amount is about 15 GB of logs, with 25 million events per day. The log history dates back to the beginning of November 2011, with an aggregate volume of 2 Terabytes.

The log centralisation and analysis is implemented with Hadoop. HDFS is mounted via FUSE on the frontend nodes, and the Apache configuration writes the log files to the mounted filesystem during log file rotation. The analysis itself is written in a Python mapper and reducer. The actual input and output is done with the Hadoop Streaming API, which means reading from /dev/stdin and writing to /dev/stdout. Running the analysis against the full 2 TB of log file data takes 50 minutes, with an average IO throughput of 700MB/sec.
3.2. HTTP Sharing
The DDM system provides access to distributed storage through grid middleware interfaces, including gsiftp or SRM. However, sometimes it is necessary that clients can access the data without installing the grid clients, which poses a complex problem due to permission, ownership, and transfer protocols.

The HTTP sharing is implemented with Hadoop. HDFS is mounted via FUSE. Files are then selected via a web-based interface, which authenticates users via their x509 certificates. Once authenticated, files are downloaded from the distributed mass storage systems using grid clients directly to HDFS. These files are then served by Apache httpd directly from the mounted filesystem. As this is supposed to be for temporary use only, files older than 3 days are removed from HDFS.

3.3. Trace analysis
DDM clients that enact data moment on the grid send event information back to the central DQ2 components, to be written to the Oracle database. That way the distributed client actions can be traced over time. This information includes, for example, the type of request (get, put), the involved locations (local cluster, remote cluster), the data sets and files, or timestamps (start time, setup time, end time). Currently, about 7 million traces arrive daily at the database with an interaction rate of 300 Hz. The aggregate volume is more than 3 billion events, dating back to May 2008.

There are three implementations of the trace analysis, one with Oracle, one with Cassandra, and one with Hadoop HBase. The Oracle implementation is the primary data store, as this is where the trace events are recorded. The implementations in Cassandra and HBase build upon the Oracle primary data store, and provide additional summarisation and aggregation metrics, for example, the number of successful requests per minute, hour, and day. The idea is to calculate the metrics continuously and incrementally at different granularities, and keep them up to date with incoming traces. This increases the interaction rate on the summarisation backend from the original 300 Hz to multiples of that, depending on the number of metrics. As Cassandra and HBase are both geared towards linear scalability of insertion rates, this is a feasible approach. The current upper limit on the cluster is an interaction rate of 9000 Hz.

Concluding, the Oracle implementation continues as the primary data store and is used as the data source for the a data mining service. Ad-hoc queries however are not used anymore. The Cassandra implementation has been deprecated and ported to HBase. The porting effort was minimal, as both products use a similar data model. Now, ad-hoc queries and summarisation run against HBase.

3.4. Wildcard search
One especially demanding use case is the search for data sets, replicas, or files using incomplete information. Data sets have names that follow a given convention, for example, a dot-separated convention could be project.energy.streamname.datatype.comment, and a sample data set that would follow this convention could be named, for example, data11.7TeV.merge.AOD.user01. A wildcard search would then ask, for example, for all data11*aod*. The potential result sets can range between zero and multiple millions, and retrieving such results can incur long time delays. This is especially important in
the case of user mistakes, when wildcards are put at the beginning of the search string, or when only later components in the convention are required, for example, a search for datatype only. For that reason, full replica search and file search are not exposed to users of the DDM system in the Oracle implementation. Even though the Oracle implementation is not exposed, it is still possibly to query the requested information via SQL. The underlying tables are changing constantly, and therefore requests are always transactional.

This requirement is relaxed in the Hadoop implementation, which uses a snapshot approach. At a point in time, a snapshot of the contents of the tables in Oracle is dumped to HDFS as a text file. Then, the wildcard query is formulated with Hadoop Pig, which creates a MapReduce job that finds the selection. As there is no reduce step, the search job finishes after the mapper. Without cache warm-up, the execution time in Oracle is 29 seconds, and in Hadoop 14 seconds. Oracle can benefit of cache warm-up here, and bring down repeated executions of the query to 1 second. Nevertheless, since Hadoop uses parallel execution and Oracle serial execution, these values cannot really be compared directly.

3.5. Accounting

The DDM system has an operational requirement for accounting of ATLAS data. This includes information about the logical and physical number of bytes, files, and data sets at particular computing centres or globally, which are then grouped in different dimensions. There are currently 25 dimensions, such as sites, projects, data types, user groups, or other ATLAS DDM-specific attributes. There are three implementations of the accounting, one partial implementation in Oracle, one full implementation in Oracle, and one full implementation in Hadoop. The partial implementation only summarises 3 dimensions, site, project, and data type. The full implementation in Oracle allows for ad-hoc querying of any attribute combination, and the full implementation in Hadoop, with Pig and HBase, as well. Additionally, the Hadoop implementation keeps a snapshot history per day, such that historical accounting can be recreated.

The main concern with the full implementation in Oracle is that it uses a convoluted implementation of triggers and tables that needed physical layout optimisation to cope with the update rate. Since the triggers directly use the production data, a failure in the accounting system could potentially slow down or even disable the production system temporarily which is not acceptable. For this reason, this solution was abandoned and alternatives were sought. The most promising alternative is implemented in Hadoop by periodically dumping the contents of the Oracle tables into flat files on HDFS, and running MapReduce jobs created by the Pig data pipeline on them. This produces the accounting results and stores them directly into flat files and HBase for later consumption. Recently, with the addition of Oracle 11g Active Data Guard, a similar solution might be applicable in the Oracle ecosystem as well, and will be evaluated in the future.

4. Operational experiences

All cluster nodes are managed by Puppet, a data centre automation tool[10]. The configuration is done as a Puppet module, with each component as a possible deployment
These targets are then distributed to particular nodes. That way, nodes can be assigned to host the HDFS master, HDFS slaves, HBase master, and so on. The actual module uses the Cloudera distribution to install necessary RPMs, and configures the nodes automatically. An installation-and-startup cycle takes about 2 minutes per node.

Hourly encrypted backups of the index of the HDFS master are pushed to Dropbox offsite. The current backup size of the index is about 5 Megabytes. As long as not all nodes have a hardware failure at the same time, the cluster is recoverable from this index backup, even when installed to another node. Nevertheless, replicating systems are never secure from cascading problems. In that case, a quick shutdown of a single problematic node can recover the whole cluster again. If the HDFS master has to be reinstalled from the backup, the downtime is at 5 minutes, including downloading and decrypting the backup, and reinstallation of the node with Puppet.

As a deployment stress test, the whole cluster was upgraded incrementally from Scientific Linux 5 (SLC5) to Scientific Linux 6 (SLC6) in production. The procedure was to drain a single node, wipe and reinstall the operating system, start Puppet, and have the node automatically synchronise back to the cluster. This was done for all nodes except the HDFS master, where an additional backup was taken just before the wipe. The effective downtime of the cluster was thus only the few minutes it took to reinstall the operating system and start it up again. Temporarily migrating the HDFS master to another node, to minimise the downtime, was not tested however. The upgrade to SLC6 improved the IO throughput of the whole cluster by a factor of 3. As there has been no other changes to the node, we attribute this to the inclusion of the asynchronous IO module (epoll), in the newer Linux kernel. This attribution stems from the fact that the IO waiting times have dropped by a factor of 3 as well.

Apart from deployment, other important operational areas are currently under evaluation. The most important one is running Hadoop as a service. This includes service level agreements, rotations, hot deployments and more, and will most likely be implemented through the Puppet infrastructure. Another area is data synchronisation. It will be important to move data from Oracle into Hadoop, at particular granularities. This can include row-level updates up to full table dumps. There are several products on the market for full dump migration (for example SQOOP) but until now none have yet successfully managed to fulfil all our present data synchronisation needs. Another use case will be data liberation, that is, how to structure the results produced in Hadoop and present them for public consumption. The non-relational content of the data makes it implicitly difficult to work with the data, as opposed to fixed schemas. The main contender right now seems to be a lightweight RESTful web service, in a shared nothing architecture for scale-out scenarios.

5. Conclusion
The increasing demands of analytical workloads have become disruptive for the Oracle database at CERN. In an effort to offload these particular workloads from the Oracle database, structured storage systems were evaluated as potential alternatives. The most promising ones were put in operation to process these workloads, and thus reduce the strain on the Oracle database. This improved the throughput, latency, and makespan for both the analytical workloads, and the transactional operations on the Oracle.
Finally, after evaluation of multiple products, the decision was made to focus on Hadoop. This has multiple reasons: first, the Hadoop ecosystem, especially HDFS and MapReduce, provides all features and scalability that are necessary to fulfill the use cases and beyond. The exception here is high availability. This exception however has been remedied in the current Hadoop release with multiple fault-tolerant masters, though it wasn’t tested yet. Second, HBase encompasses the most important features of the other non-relational database products. The striking omission of indexes has not been a problem in our usage though, as our data models all benefit from the inherently sorted data in HBase. Third, Hadoop is actively developed and supported by a number of large companies with commercial interests, as well as a thriving free and open source community. Fourth, Hadoop is also being used within the High Energy Physics community as a storage system by itself, which gives further incentives for commonalities in the domain.

One important point to note is the ease of use of non-relational databases. Even though the developers were coming from an SQL background, it was easy to adapt to the non-relational way of thinking, because the tools provided similar interfaces. In some cases, the interfaces even mimicked SQL to the syntax, and in some cases, used similar concepts to express data models. However, this new way of thinking has also shown that straightforward application of non-relational methods and models is not something that is sacrosanct, as most of these methods and models can be used in the relational world as well. A good example is the accounting use case, where daily dumps are processed on Hadoop, when something similar could be achieved within Oracle 11g Active Data Guard. Nevertheless, the promise of scalability of non-relational databases has proven to be worthwhile. One has to take care though and not mistake this for immediate performance, as the objective is to make sure that future increased workload can still be processed in linear time.

Additionally, important savings came from development time. One important factor for this was that the data models of the applications could be used in their native way, instead of having to be remodelled relationally. Another important factor for this was that the deployment and operation did not require significant expenses and has been completely automated. This eliminated the need for online administration of the cluster. In the future, the use of structured storage systems in ATLAS DDM is therefore likely to increase, as the versatility is high and the operational overhead is low.

Bibliography
[1] Lassnig M et al., Managing ATLAS data on a petabyte-scale with DQ2, 2008 J. Phys.: Conf. Ser. 119
[2] Apache Cassandra, 2012, http://cassandra.apache.org/
[3] Apache Hadoop, 2012, http://hadoop.apache.org/
[4] Apache HBase, 2012, http://hbase.apache.org/
[5] Apache Hive, 2012, http://hive.apache.org/
[6] Apache Pig, 2012, http://pig.apache.org/
[7] Cloudera, 2012, http://www.cloudera.com/
[8] MongoDB, 2012, http://www.mongodb.org/
[9] Oracle, 2012, http://www.oracle.com/
[10] Puppet, 2012, http://puppetlabs.com/