# Comparison of the Frontier Distributed Database Caching System to NoSQL Databases

**Dave Dykstra**
Fermilab, Batavia, IL, USA

**Email:** dwd@fnal.gov

**Abstract.** One of the main attractions of non-relational "NoSQL" databases is their ability to scale to large numbers of readers, including readers spread over a wide area. The Frontier distributed database caching system, used in production by the Large Hadron Collider CMS and ATLAS detector projects for Conditions data, is based on traditional SQL databases but also adds high scalability and the ability to be distributed over a wide-area for an important subset of applications. This paper compares the major characteristics of the two different approaches and identifies the criteria for choosing which approach to prefer over the other. It also compares in some detail the NoSQL databases used by CMS and ATLAS: MongoDB, CouchDB, HBase, and Cassandra.

## 1. Introduction

This paper compares the Frontier Distributed Database Caching System [1] to several "NoSQL" Database Management Systems. The goal of the paper is to increase familiarity with all of them to help the right tool to be chosen for each application.

Experience with using these systems is described from the two largest detector experiments from the Large Hadron Collider (LHC): CMS (Compact Muon Solenoid) and ATLAS (A Toroidal LHC ApparatuS). The applications in both experiments that use the Frontier system load what is called Conditions data, which are primarily detector alignments and calibrations that need to be loaded to every processor that examines particle collision events.

Section 2 describes common characteristics of NoSQL databases. Section 3 introduces a common effect that leads to typical requirements on distributed client-server systems, called the Slashdot Effect, which is background for the remaining sections. Section 4 describes the characteristics of the Frontier Distributed Caching System. Section 5 gives examples of Frontier deployments for the CMS experiment's Conditions data, both Offline and Online. Section 6 compares Frontier to NoSQL in general, and Section 7 compares Frontier to the specific NoSQL database management systems MongoDB, CouchDB, HBase, and Cassandra.

## 2. Common Characteristics of NoSQL Databases

The name "NoSQL" is used to refer to a large variety of Database Management Systems (DBMS) with many different characteristics. This section describes some of the most common characteristics.

### *2.1. Non-relational*

The primary unifying characteristic is something they are not: they are not traditional Relational Database Management Systems (RDBMS). RDBMS store data in a structured tables, where rows each have predefined column names and types. The predefined structures are also called "schemas."

NoSQL systems, on the other hand, generally allow more flexibility in the structure of data. They usually have keys which can return any arbitrary data for different values of a key, including possibly other keys that are nested. As long as the application program knows how to handle the flexible results, the database doesn't care. Many of the systems allow also specifying some structure to the data in order to improve performance with indexing.

As a result of the flexible structure, most of the NoSQL systems then do not support the standard RDBMS Structured Query Language, or SQL. Some of them do support it for compatibility with existing applications, however, since the row/column structure is a subset of the more general key/value structure.

### *2.2. Distributed*

One of the most attractive characteristics of popular NoSQL systems is that they are able to distribute their data across a large number of commodity computers, both at local and remote locations. Through that they achieve both fault tolerance and high scalability. This enables them to reliably scale to large numbers of read operations and, in many cases, also of write operations.

### *2.3. Eventual Consistency*

In order to perform well in a distributed environment, most (but not all) of the NoSQL systems give up atomic operations in favor of eventual consistency. That is, instead of providing ACID (Atomic, Consistent, Isolated and Durable) transactions, they provide BASE (Basic-Availability, Soft-State, and Eventual Consistency). Whether or not this is acceptable depends on the application; many applications do fine with BASE but others need ACID [2].

## 3. The Slashdot Effect

A frequent requirement on client-server applications is the ability to handle reading the same data from a very large number of different clients. The "Slashdot Effect," also known as "slashdotting," occurs when a popular website includes a hyperlink to a much smaller website and many people click on the link at around the same time and overwhelm relatively small web servers. The term was coined in the early days of the technical news aggregator "slashdot.org" after it became popular [3].

Companies typically handle the Slashdot Effect for their web service with a Content Delivery Network (CDN), either by contract or by owning their own. A CDN provides multiple servers around the world that either cache web sites or contain copies of them. That way the client requests get distributed among a large number of servers and no one server becomes overloaded.

Some database applications have a similar requirement to handle many readers of the same data at about the same time.

## 4. Frontier Characteristics

The Frontier Distributed Database Caching system (hereafter referred to as Frontier or Frontier/Squid) was designed to handle the Slashdot Effect for applications that use Relational Database Management Systems; that is, those applications that have many readers of the same data, and relatively few writers. The following are some of its characteristics.

- It distributes read-only SQL queries; it is **not** a NoSQL system. That brings the advantages of using long-supported and well-understood RDBMS as the backend database, with all its stability, strong support, and existing software.
- Its protocol follows the REST (REpresentational State Transfer) specification of HTTP, so it is easily cacheable with standard web proxies [4]. The current deployments use the popular open source caching web proxy called Squid [5].

- Squids are deployed within the Local Area Networks of the clients. This makes an ideal Content Delivery Network because most of the traffic goes on the high capacity, low latency LANs. This is an ideal CDN also because web proxy caches are practically maintenance-free; once they are configured and started, they require very little intervention.
- When there are simultaneous requests for the same data using the same web proxy cache, Squid can collapse them into a single request to the upstream server (if the non-default option collapsed_forwarding is turned on). If the requests for the same data come close together but not at the same time, the data is served from the local cache and the upstream server isn't contacted at all.
- When there are more simultaneous different requests than a small configurable number, those requests are queued at the Frontier servers. This slows down the clients that are waiting for the data but avoids overloading the database servers. (The queue sizes are also limited and in CMS the operators are notified when one of them is nearly full [6]. In practice the queueing is only used for short periods of time each day).

## 5. Frontier/Squid Deployment Examples

The current version of the Frontier/Squid system is deployed in production in three High Energy Physics applications at the Large Hadron Collider. They all distribute "conditions" data, which are mainly detector alignments and calibrations. All the worldwide jobs that are processing particle collision events need to read the conditions from the detectors which were valid at the time of the collisions. Since related events tend to be processed close together, usually there are many jobs reading the same conditions data at about the same time. The three deployments are for CMS Offline, CMS Online, and ATLAS Offline. The remainder of this section will take a closer look at the first two and at the limits of the components of the Frontier/Squid system.

### *5.1. Frontier/Squid Deployment Example: CMS Offline Conditions*

Figure 1 shows the architecture of the CMS Offline Conditions deployment of Frontier/Squid.

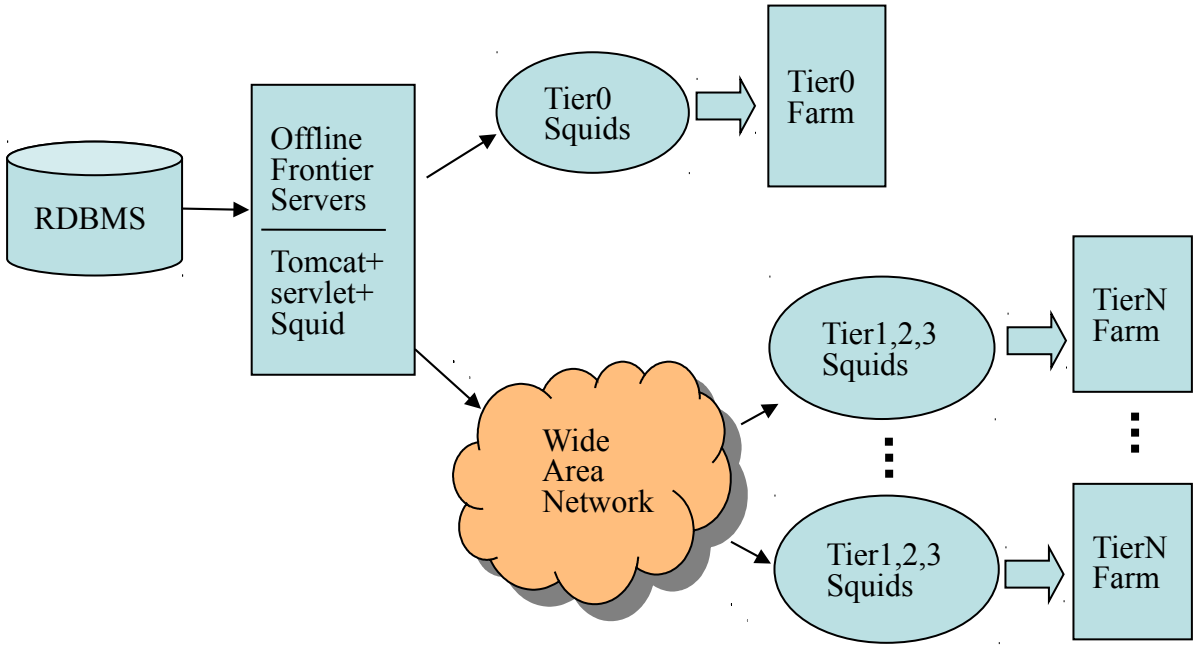


**Figure 1:** CMS Offline Conditions Frontier/Squid deployment architecture

In this deployment, a highly-available Oracle RDBMS and 3 Frontier servers are at CERN. Each Frontier server runs an open source java Tomcat [7] process that contains Frontier servlets. Each servlet reads from the RDBMS using SQL and converts the responses into HTTP/REST cacheable responses. On the same server machine there is a reverse-proxy Squid which caches the responses. Then at each site, both locally at the CERN Tier 0 site and at each of the approximately 100 Tier 1, Tier 2, and Tier 3 sites there are more Squids to cache the responses locally. CMS software running on each of the worker nodes in the compute farms, which include a frontier client library, converts the responses back from HTTP to SQL responses. Note that the only custom software in the Frontier/Squid system is the servlet in Tomcat and the frontier client library. Currently the only CMS Offline Frontier servers are at CERN, but CMS is working on replicating the database and servers at another site for increased availability.

This system handles an average of about 500,000 total requests per minute total worldwide by all of the approximately 100 distributed Squids, and an average of 500 Megabytes per second. That may not seem like very much, but because conditions are mostly loaded near the beginning of jobs, and related jobs tend to start together at sites, peaks on individual squids are often significantly higher than their average, by a factor of 5 or more. (Also, these numbers don't include Tier 3 sites because statistics from them are not collected together, but they tend to be relatively small anyway.)

By contrast, the 3 central Frontier servers at CERN see a total average of 4,000 requests per minute and deliver an average of 0.5 Megabytes per second. That is a factor of 125 improvement on requests and a factor of 1000 improvement on bandwidth (it would be more if Tier 3 were included). The difference in improvement is primarily because of the If-Modified-Since caching policy that is used, where a majority of the time only small timestamp checks need to be done and then cached items can be reused if nothing has changed. A majority of those requests are satisfied by the Frontier servlet without having to contact the database [8].

*5.2. Frontier/Squid Deployment Example: CMS Online Conditions*

Figure 2 shows the architecture of the CMS Online Conditions deployment of Frontier/Squid.

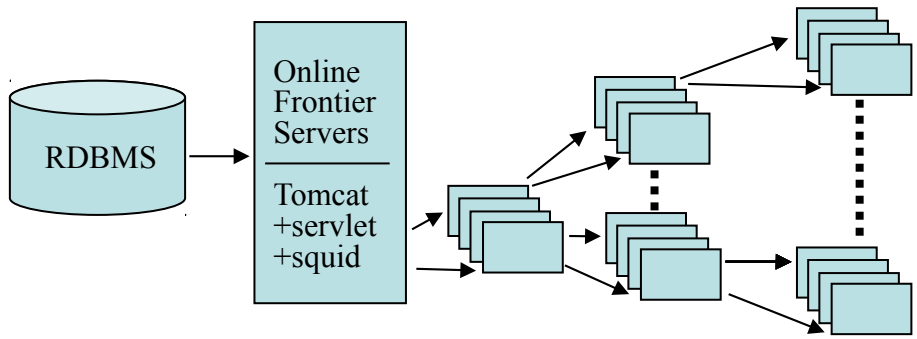


**Figure 2:** CMS Online Conditions Frontier/Squid deployment architecture

The configuration of the Squids in the CMS Online deployment is very different because the requirements are very different. In this each worker node each has its own Squid, arranged in a hierarchy where each Squid feeds up to 4 others. This is needed because all of the 1400 nodes have to be loaded at the same time with about 100MB of data as quickly as possible, and that would overwhelm a small number of servers; it would be an extreme Slashdot Effect. This deployment demonstrates the flexible power of having a protocol that can make use of proxies; as many as are needed can be easily inserted for more bandwidth.

*5.3. Limits of Frontier Tomcat servlets and Squids*
In general the limits on Frontier Tomcat servlets and Squids, at least with applications similar to CMS Conditions, are the network capacities of their host machines, not the CPU or disk.

Frontier Tomcat servlets were recently measured on a 3-year old 8-core machine (with Xeon L5420 cpus at 2.5Ghz) to be able to easily saturate a 1 Gigabit network out when reading from an Oracle database without compressing the result. With gzip compression of the output, however, the maximum output rate drops to 25 Megabytes per second while fully utilizing all of the 8 cpu cores. On the other hand, it saves much bandwidth later in the caches. In production, demand on the CMS Offline Frontier servers is so light that even with gzip compression the servers run on 2-core Virtual Machines and are still always lightly loaded.

Squid was measured two years ago on a machine that was new at the time (with Xeon E5430 cpus at 2.66Ghz) to saturate two bonded 1 Gigabit network connections with one single-thread Squid. It was also measured recently on a modern machine (with Opteron 6140 cpus) with a 10 Gigabit network connection to send up to 7 Gigabits with a single-thread Squid. We are still using a single-threaded Squid2 version because the multi-threaded Squid3 does not yet properly implement If-Modified-Since (Squid3 was a total rewrite in a different programming language). If that does not get fixed soon, we can instead run two Squid2s listening on the same port and in that way saturate a 10 Gigabit network, at a cost of being unable to share the disk cache. Since Squid disk caches for this application do very well with 100GB of space, doubling that space is not difficult.

**6. Comparison between Frontier and NoSQL in general**
Now that Frontier has been examined in detail and NoSQL in general has been introduced, the two systems can be compared.

**Table 1**: Comparisons between Frontier and NoSQL in general

| | Frontier | NoSQL in general |
|---|---|---|
| DB structure | Row/column | Nested key/value |
| Consistency | ACID DB, eventual read | Eventual |
| Write model | Central writing | Distributed writing |
| Read model | Many readers same data | Read many different data |
| Data model | Central data, cache on demand | Distributed data, copies |
| Distributed elements | General purpose | Special purpose |

Below is a brief description of each line in the table.

*6.1. Database structure*
Frontier uses the Relational DBMS row/column structure which has the advantages of being able to use a database with long experience and support. NoSQL databases use nested key/value structure which gives flexibility of data layout.

*6.2. Consistency*
Both Frontier and NoSQL databases provide eventual consistency in the data. In Frontier's case writers always see a consistent database, but readers don't always see it because of varying cache delays. Applications have to be tolerant of delays for both Frontier and NoSQL.

*6.3. Write model*
Frontier has only central writing into the Relational database but NoSQL databases generally support distributed writing. Depending on the application that may be important.

*6.4. Read model*
Frontier supports best many readers of the same data items. NoSQL databases in general support the simultaneous reading of many different data items better than Frontier does, because they can have the data replicated at more servers.

*6.5. Data model*
In the data distribution model of Frontier, the data is stored centrally and then cached on demand in the distributed elements. In NoSQL databases the data is generally sent ahead of time to the distributed elements, with copies of the data stored there permanently. The Frontier system can have a small number of replicas of all the data distributed for reliability as well.

*6.6. Distributed elements*
The distributed elements in the Frontier system are general purpose web proxy caches that can also be used for other applications. With NoSQL databases the distributed elements are special purpose for only that database.

**7. Comparisons With Specific NoSQL DBMS**
In this section specific NoSQL systems will be reviewed and compared. The chosen systems are all currently used in production in some capacity for either the CMS or ATLAS experiments. They are MongoDB, CouchDB, Hadoop HBase, and Cassandra.

*7.1. MongoDB*


The name for MongoDB came from "mongo" for "humongous" because it was intended to support big databases cheaply. MongoDB stores binary JavaScript Object Notation (JSON). JSON is a very common, compact method of storing and exchanging arbitrarily structured data.

MongoDB is more like a standard RDBMS than the other NoSQL systems considered in this paper. It allows any field to be of predefined type and memory-indexed for performance. It also has very flexible queries similar to SQL: queries by fields, ranges, and regular expressions. Only one server is allowed to write any particular data item; a few read-only copies on other servers can be stored and any one of them can take over as the master, if the master goes down. Scaling is then done by sharding, where different data items are distributed among multiple servers. Note that this does not do well with the Slashdot Effect because each data item is never on more than a small number of servers.

CMS uses MongoDB in production for its Data Aggregation Service [9]. They needed the dynamic structure and liked MongoDB's other features. It is a very small installation, however, on only one server.

MongoDB supports MapReduce, where user-defined processing can be distributed across the servers that have replicas of the data. An ATLAS evaluation, however, found that it didn't work very well; that feature is reported to work better in the current version 2 of MongoDB [10].

*7.2. CouchDB*

CouchDB, like MongoDB, stores JSON objects. It has the very interesting characteristic of using a REST-compliant interface for reading and writing the database. This means that, like Frontier, it can be deployed with web proxies wherever the application requires it, thus handling the Slashdot Effect well. The RESTful interface also makes it easy to insert standard proxies for other purposes including supporting a large variety of authentication methods.

Once it is configured, CouchDB automatically replicates all the data to all servers. This can be very useful when the amount of data is relatively small but is impractical for very large databases and a large number of replicas. In addition to making the data available to read on all servers, CouchDB supports simultaneous writing on all the replicas. It also ensures that all writes are atomic, that all readers see consistent views, and that writing doesn't block reading; that is, it supports ACID transactions. That is accomplished by using Multi-Version Concurrency Control (MVCC) which is a common feature of relational databases but not very common in NoSQL databases. Write conflicts have to be resolved by the application, however. Note that even though all readers always see internally consistent views, that doesn't necessarily mean that all readers will see the exact same view at the same time, because it can take some time for a write transaction to propagate to all the replicas.

Queries are done very differently in CouchDB than other DBMS: the user defines "views" using JavaScript functions that create additional URLs to read later. The programming paradigm for those functions is MapReduce, but the processing of the functions is not distributed to multiple servers so it doesn't get the performance boost of other systems with MapReduce (in particular see Hadoop HBase in the next section).

CMS uses CouchDB in production for several functions in its Workload Management systems [9]. The installation is larger than the MongoDB installation, but still not very large: it has 3 replicas of a CouchDB database at CERN and 4 replicas of the same database at Fermilab.

*7.3. Hadoop HBase*


Hadoop HBase is a database implementation that is built on the Hadoop Distributed FileSystem (HDFS). HDFS is designed for large clusters of commodity computers and automatically distributes file blocks and replicates them across the cluster. If any replica is lost, HDFS automatically replaces it from other replicas. So it is very reliable and works well with large amounts of data. On the other hand, it doesn't scale down very well to small installations. Quite a few of the distributed sites in the Worldwide LHC Computing Grid (WLCG) use HDFS to store data with good results. HDFS has a tunable replication level to control the number of copies that are kept for each data block.

HBase is modeled after Google's BigTable, which is designed to handle data structured with billions of rows and millions of columns. It is especially good for search engine-like applications. HBase is very good at distributed MapReduce, where processing is split up and mapped to run in parallel on the computers that contain the data, and then the results are reduced into a combined answer. It does not supply ACID guarantees for every kind of database interaction, however, just some of them.

HBase also has an SQL compatibility interface via an add-on called Hive. So this "NoSQL" database does support SQL. It also has a RESTful interface add-on called Stargate; the native interface is Java. So if the RESTful interface is used along with distributed proxy caches it could also do well with the Slashdot Effect. (Frontier could also be viewed as a RDBMS add-on).

HBase is used in production by ATLAS in its Distributed Data Manager called DQ2 (Don Quixote 2), for both log analysis and accounting on a 12-node cluster [11]. When ATLAS first tried it for

doing their accounting summary, they found it was 8 to 20 times faster than on the shared Oracle system they had, depending on the HDFS replication level. They have since improved the accounting summary mechanism for both systems and found that for small examples the performance is similar on similar hardware. They believe the HBase system, however, will be able to scale much better.

HBase has been recognized by the WLCG Database Technical Evolution Group as having the greatest potential impact on the LHC experiments out of all NoSQL technologies. The CERN IT organization is setting up a cluster to try it.

*7.4. Cassandra*

Like HBase, Cassandra is also modeled after Google BigTable. It is especially good at distribution over widely separated locations. All nodes in the system are masters, and control is decentralized for good fault tolerance. The system dynamically reconfigures itself as servers are added or removed, with no downtime overall.

The keys and values in Cassandra can be any arbitrary data. It has a concept of "column families" which are used like indexes in relational databases. It has a tunable replication level like HBase. It has tunable in-memory caching of recently read data, on the nodes to which the data has been replicated. It also has tunable consistency, from always consistent to eventually consistent. It supports MapReduce through Hadoop components.

Cassandra was originally written by Facebook for use with their Inbox search feature. They abandoned it in late 2010, however, and now use HBase instead.

Cassandra is used in production by ATLAS PanDa monitoring [12]. They chose to host it at BNL on only 3 nodes that were quite high-powered: each node has 24 cores and 1 Terabyte of RAID0 Solid-State Disks (SSDs). They could perhaps have achieved similar performance out of Oracle on similar hardware, but since their Oracle installation supports a much larger application base it wouldn't have been economical to upgrade it all.

7.5. *Comparison summary*

**Table 2:** Summary comparison of specific NoSQL DBs and Frontier characteristics

| | MongoDB | CouchDB | HBase | Cassandra | Frontier |
|---|---|---|---|---|---|
| Stored data format | JSON | JSON | Arbitrary | Arbitrary | SQL types |
| Flexible queries | Yes | No | No | No | Yes |
| Distributed write | No | Yes | No | Yes | No |
| Handles Slashdot Effect well | No | Yes, best w/squid | If scaled sufficiently | If scaled sufficiently | Yes |
| Does well with many reads of different data | Yes | Yes | Yes | Yes | No |
| RESTful interface | No | Yes | Add-on | No | Yes |
| Consistency | Eventual | ACID DB, eventual read | Mixed | Tunable | ACID DB, eventual read |
| Distributed MapReduce | No | No | Yes | Add-on | No |
| Replication | Few copies | Everything | Tunable | Tunable | Caching |

Table 2 shows a summary comparison of major characteristics between the 4 specific NoSQL database systems and Frontier. The entries in the table summarize the points discussed in the sections above.

## 8. Conclusions

NoSQL Database Management Systems have a wide variety of characteristics. Most of them are highly scalable, which is one of their major attractions.

Frontier with distributed Squid servers easily and efficiently add some of the same scalability to relational databases for applications that have a very large number of readers of the same data. It also enables the clients to be geographically distant and still perform well. On the other hand, it requires the application to be able to tolerate eventual consistency.

Of the 4 NoSQL systems considered, CouchDB is the one that can scale the easiest for the Slashdot Effect because its native REST-compliant interface enables it to be cached by HTTP proxies.

Of all current popular NoSQL systems, Hadoop HBase appears to have the most potential for scaling up to handle very large applications.

There are applications in High Energy Physics that make good use of the strengths of many different Database Management Systems.

## 9. Acknowledgements

Fermilab is operated by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the United States Department of Energy. Thank you to Steve Foulkes, Gabriele Garzoglio, Valentin Kuznetsov, Mario Lassnig, Maxim Potekhin, and Eric Vaandering for supplying comments on a draft of this paper and/or for providing information on existing use of NoSQL databases in the LHC experiments.

## References

[1] Dykstra D and Lueking L 2010 Greatly improved cache update times for conditions data with Frontier/Squid *J. Phys.: Conf. Ser.* **219** 072034

[2] Pritchett D 2008 BASE: An Acid Alternative *ACM Queue Magazine* vol **6** May/June issue 3

[3] Adler S 1999 "The Slashdot Effect: An Analysis of Three Internet Publications" *Linux Gazette* March issue 38

[4] Dykstra D 2011 Scaling HEP to Web Size with RESTful Protocols: The Frontier Example *J. Phys.: Conf. Ser.* **331** 042008

[5] Squid: http://www.squid-cache.org *Last accessed on 21 May 2012*

[6] Blumenfeld B, Dykstra D, Kreuzer P, Du R, and Wang W 2012 Operational Experience with the Frontier System in CMS *CHEP 2012, New York, NY, May 2012*

[7] Apache Tomcat: http://tomcat.apache.org *Last accessed on 21 May 2012*

[8] Dykstra D and Lueking L 2010 Greatly improved cache update times for conditions data with Frontier/Squid *J. Phys.: Conf. Ser.* **219** 072034

[9] Kuznetzov V, Evans D, and Metson S 2012 Life in extra dimensions of database world or penetration of NoSQL in HEP community *CHEP 2012, New York, NY, May 2012*

[10] Lassnig M et. al. 2012 Structured storage in ATLAS Distributed Data Management: use cases and experiences *CHEP 2012, New York, NY, May 2012*

[11] Lassnig M, Garonne V, Dimitrov G, and Canali L 2012 ATLAS Data Management Accounting with Hadoop Pig and HBase *CHEP 2012, New York, NY, May 2012*

[12] Ito H, Potekhin M, and Wenaus T 2012 Development of noSQL data storage for the ATLAS PanDA Monitoring System *CHEP 2012, New York, NY, May 2012*