# Life in extra dimensions of database world or penetration of NoSQL in HEP community

# Life in extra dimensions of database world or penetration of NoSQL in HEP community

**V. Kuznetsov**[1], **D. Evans**[2] **and S. Metson**[3]

[1] Cornell University, Ithaca, New York, USA
[2] Fermilab, Batavia, Illinois, USA
[3] Bristol University, Bristol, UK

E-mail: `vkuznet@gmail.com`

**Abstract.** The recent buzzword in IT world is NoSQL. Major players, such as Facebook, Yahoo, Google, etc. are widely adopted different "NoSQL" solutions for their needs. Horizontal scalability, flexible data model and management of big data volumes are only a few advantages of NoSQL. In CMS experiment we use several of them in production environment. Here, we present CMS projects based on NoSQL solutions, their strengths and weaknesses as well as our experience with those tools and their coexistence with standard RDBMS solutions in our applications.

## 1. Introduction

Nowadays the IT infrastructure of any large organization is quite complex by default. It usually consists of heterogeneous networks, it resides in different data centers and heavily utilizes database technology to hold corporate data. Recently a new trend, the so-called NoSQL [1] solutions, joins this conglomerate. More and more companies are adding them to solve raising problems with data scalability. The range of NoSQL solutions varies from key-value stores to graph databases. Even though their advantages are controversial, their penetration into IT infrastructure is obvious. The High Energy Physics community is not an exception. Today CERN attracts most of the attention. It hosts four large experiments at LHC searching for new physics and testing our knowledge of Standard Model. One of them is Compact Muon Solenoid (CMS) [2]. More than three thousand physicists are involved in most sophisticated research program at CMS that yields a few PB of data each year at their disposal. To handle this amount of data CMS experiment relies on hundreds of computing centers around the world interconnected with each other by GRID infrastructure [3]. The CMS software has been around ten years and still is constantly evolving along with experiment requirements. Today it has more than 4M lines of C++ code (CMSSW release), 2M lines of python code (framework configuration, web and data management stack), as well as various applications written in Java and perl (mostly web and data services). Major CMS data-services, such as Tier0 [4], PhEDEx [5], Data Bookkeeping System [6], Run Summary [7] are based on standard 3-tier architecture and rely on ORACLE RAC. Combined, they accumulate ∼O(100GB) of meta-data every month. Although such infrastructure served us well in first couple of years of data taking, the application growth, big data scale and changing requirements enforced CMS developers to bring NoSQL solutions into CMS software stack. Here we discuss reasons behind this choice and our experience with those tools in CMS production environment. We start with motivations in Sect. 2, followed by overview of three CMS projects based on NoSQL in Sect. 3 and summarize our experience with them in Sect. 4.

## 2. Motivations

Efficient data management at PB scale is a very challenging task. Existing CMS data-services rely on ORACLE back-end supported by CERN IT. Although they are holding the load we start to see limitations of the systems where a new set of use cases was hard to accommodate in terms of existing infrastructure.

For example, CMS data management tools required off-site deployment. Some of the CMS sites were unable to deploy ORACLE database due to license issues and therefore swap to MySQL back-end. Although such system worked[1], we faced with a problem of data replication across distributed sites. As a result, the system was re-designed to use document-oriented data store, the CouchDB [8]. It provided replication capabilities, map-reduce operations and eliminated issues with schema evolution.

Another example is a data discovery challenge due to growth of meta-data in different databases. For example, in CMS we have data transfer management system, data bookkeeping system, run summary, conditions databases, etc., which contains valuable information for physicists. But quite often users required to query multiple services in order to find desired information. It became clear that new level of abstraction is required to deal with different interfaces, data formats, naming and schema conflicts. Therefore, CMS built the Data Aggregation System (DAS) on top of MongoDB [9], the open source document-oriented database. The schema-less database with flexible query language, fast read/write operations and horizontal scaling, was very suitable for DAS implementation and allows us to resolve all aforementioned issues.

It is also common to use simple key-value store to increase a throughput of existing database back-end [10]. But in some cases it can represent a valuable alternative to RDBMS when complexity of the server setup and its maintenance can be avoided. One of the examples of such successful substitute will be discussed in Sect. 3.3.

These are only a few examples where usage of NoSQL can be beneficial. From our experience we see that usage of relational databases can be supplemented by NoSQL solutions to fulfill the modern scale of data management. We can summarize our motivations toward usage of NoSQL as following:

- heterogeneous environment leads to a highly complex data operations whose requirements are not known a-priory;
- evolving data model and big data scale often conflict with underlying schema constraints and become a real bottleneck at application layer that is resolved by de-normalization of the schema or separation of the database into multiple services;
- the tradeoff of ACID (atomicity, consistency, isolation, durability) properties in favor of high availability of the data is often a case in dynamic, distributed environment;
- real time features, such as analytics, aggregation, map-reduce, horizontal scalability and distributed operations become de-facto standard in modern world;
- licensing issues, administrative and maintenance cost of RDBMS has a long term commitment and often can be avoided.

It is worth to mention that expertise with new products does not come for free, but without trying new technologies we will probably never reach our goals.

[1] Obviously developers were facing schema compatibility issues between different back-ends.

## 3. NoSQL in CMS software infrastructure

Currently there are several CMS projects that are based on NoSQL solutions. Here, we will discuss only three of them:

- the Data Aggregation System (DAS) [11] is an intelligent cache in front of CMS data-services; it fetches and aggregates data on demand upon user queries and represents next generation of data discovery service for CMS experiment;
- the Data and Workflow management (WMAgent) tool [12] is a job submission and execution engine for CMS production machinery; it dispatches and manages jobs across CMS tier centers;
- the Ignominious Profiler (IgProf) [14] is a primary tool for performance studies of CMS software.

The usage of NoSQL varies among these projects, but as you will see shortly it plays an important role in their success.

### *3.1. DAS and MongoDB*

The idea behind DAS [11] was very simple, the system should provide a search interface across distributed relational and non-relational data-services and aggregate information from them. It must operate as a dynamic cache in front of existing infrastructure and hides complexity of their data access. The set of requirements imposed on developers to build such system were the following: DAS should provide easy to use and yet powerful text-based query language; it must accommodate different data formats, security measurements as well as naming and schema differences of underlying data-services; it should be transparent to their API and architectural changes; it must have fast read/write I/O and capable to store and aggregate data from them in a real time. Initially many options were considered, including usage of RDBMS back-end (ORACLE/MySQL), federated database (ORACLE), key-value store (Berkley DB, memcached) and document-oriented databases (CouchDB and MongoDB). Among them the federated and document-oriented database approaches were most appealing. They can be used to abstract data-services, e.g. resolve naming conflicts, they provide search interfaces and they are suitable in distributed environment. The key difference between two approaches was a requirement of data model. As such, the federated database still required a-priory knowledge of data and should be abstracted at user interface level to hide its schema. While the NoSQL schema-less document-oriented databases do not impose such constraints, the data storage and retrieval can be done as is. The aggregation of the records would be done at proper matching of key value pairs and underlying data format can be easily manipulated at presentation layer via JavaScripts. Moreover, the ACID properties of the back-end can be replaced with BASE (basically available, soft state, eventually consistent) alternative, which is more suitable for this case. Even though we had prior experience with usability studies for data discovery service [15, 16], we found that experiment requirements are constantly changing. Such changes were often required schema modifications as well as adaptation of SQL queries. Therefore, a static approach of federated database via development and maintenance of new schema was ruled out in favor of NoSQL approach, where MongoDB represented several advantages over CouchDB for DAS use case. It stores data in binary JSON data format, it provides flexible query language and full index support on any query-able attribute as well as map-reduce operations. In addition, it provided replica sets and sharding for horizontal scaling. Based on published benchmarks and our own measurements we were satisfied with high throughput of MongoDB read/write operations and built DAS around it. Currently, it uses dozens of MongoDB collections (a la databases), including raw and merge caches, logging and analytics collections. MongoDB is used to store data records along with user queries, data-services mapping and presentation templates for different data-services. Due to dynamic nature of the cache the total size of MongoDB constantly varies.

We observed that few thousand user queries yield a million data records on average, such that MongoDB size grows ∼30GB in a month. Our measurements[2] demonstrated that MongoDB can sustain the load of 20k and 7k docs per second in read and write operations, respectively. At this rate DAS performs well, although in coming years we expect the load double in size. Up to now we did not experience any issues with MongoDB, including its well criticized global write-lock, but we pay close attention to deployment stories based on this product.

### *3.2. WMAgent and CouchDB*

The workflow and management system, WMAgent [12], is a set of python modules written specifically to handle Tier-1, integration and MC production jobs for various CMS workflows. WMAgent was designed to support high load job submission and monitoring at a rate of 3K jobs at Tier-0, 10K jobs at Tier-1s, 15K simulation jobs at Tier-2s and 15K analysis jobs at Tier-2s. Jobs submission can be done by different teams at different geographical locations, each handling ∼50 jobs/second at most. At the same time, it was required that job submission should be supported in distributed environment and provides durability with respect to the system failures.

Each WMAgent is based on WMCore framework [13] and consists of several components: the WMBS (Workflow Management Bookkeeping System) for handling job definitions and dependencies; JobStateMachine to keep jobs progress; JobDump and WorloadSummary subsystems for collecting job output reports and summaries, respectively. Its initial implementation was based on MySQL back-end. Very soon it was realized that synchronization between distributed MySQL servers represents a real challenge for system developers and did not scale. To resolve this issue different parts of the system (WorkQueue, JobStateMachine, JobDump and WorkloadSummary) were migrated to CouchDB document-oriented database. It provided several advantages over MySQL. The CouchDB is built on top of the Erlang OTP platform [17], a functional, concurrent programming language and development platform, and therefore has excellent support for developing highly concurrent applications. It is a peer-based distributed database system that supports bi-directional replication between distributed databases in reliable, incremental form. In addition, the CouchDB guarantees ACID properties at file level and never overwrites committed data. Such database features provide high availability of our data together with bookkeeping mechanism for WMAgent workflows. The map-reduce operations were sufficient for generating summaries, job progress and output reports. Finally, the back-up of CouchDB was trivial operation due to append only file format suitable for database replication to another node or writing DB file to CASTOR tape system.

Currently we have several CouchDB databases deployed at CERN and FNAL with replication rate of a few documents per second between different WMAgents. After six months of running, the busiest CouchDB contains more than six million documents (300GB in size) and constantly growing. Even though we experienced a couple of issues with CouchDB, see next chapter, the system works very reliably.

### *3.3. IgProf and KyotoCabinet*

The Ignominious Profiler (IgProf) tool [14] is not as big as other two systems discussed here, but it is very well crafted and critical for debugging, analyzing and fine-tuning of CMS software. Originally it was written as complementary tool to standard gprof, jprof, valgrind and other profiling and instrumentation tools, but it grown beyond them in its capacity and widely used by CMS developers to carry various tasks. For example, it is used by build system to generate a few hundred profiles per integration build. Its output consists of SQLite database files (each of the order of few MB) that are stored on AFS area, where they can be used by igprof-navigator

[2] Benchmarks are based on usage of MongoDB python driver

tool. The toolkit should be light-weight and portable to allow users quickly plug it into various tasks. While SQLite files were sufficient for navigating a single profile at a time, the comparison of multiple profiles leads to a large overhead with handling SQLite databases, especially if someone wants to serve IgProf data into web site via CGI script. Turns out that static nature of stored profile symbols among analyzed runs allows their compression at a very good level. As a result IgProf developers switched to Kyoto Cabinet [18] key-value file-based store for igprof-book application. It is capable of storing and operating millions of keys in fraction of a second and provides superior database compressions. For example, the igprof-book database of 16K+ profiles is only ∼5 GB in size versus what would be ∼100 GB in case of SQLite equivalent. Therefore, such choice was excellent substitution of more complex web-service infrastrcuture.

## 4. Production experience

Despite the novelty of discussed tools their maintenance was minimal and did not involve IT/DBA support. The learning curve was short, due to simplicity of their APIs and underlying data format, but the efficiency of those tools came with a gained knowledge along their usage.

In a case of MongoDB the high throughput was based on maintaining its indexes and accessed data in RAM. Although the initial setup was easy, the growth of the data forces us to revisit cache internals several times. For example, on average we observed 10K requests per day that yields 1M documents going in and out of MongoDB. Such dynamics was determined by lifetime of the records associated with user queries. The expiration timestamps were set by participating data-services providers. The size of the stored objects was <1KB. Initially we re-index database several times to reduce its index size, while after code re-factoring and re-arragenment of query plans it was no longer required.

The DAS and MongoDB were deployed on a single node with 8 CPU cores, 24 GB of RAM and 1 TB of disk space whose resources were shared with other CMS data-services. Our tests shown that MongoDB can easily maintain 100M records and provide 20k docs/per second read throughput for 500 parallel clients. Currently, the load on a system is not that high and our plans for horizontal scaling are postponed. There were a few minor issues with MongoDB, e.g. its non-standard build tool, some limitations on stored keys as well as primitive support of aggregated functions, but we found excellent support from MongoDB team and were able to resolve all outstanding issues.

The setup and usage of WMAgent/CouchDB was totally different. The WMAgent deployment was done at CERN and FNAL nodes. In total, we had 8 CouchDB databases serving Tier1, MC production and integration WMAgent's instances. Each CouchDB node had 8 CPUs, 48 GB of RAM and suitable disk space. As we stressed earlier, the WMAgent did not require dynamic queries and data-ops were mostly interested in summary reports for production workflows. For that purpose simple map-reduce operations were written in JavaScripts and deployed into CouchDB views. Most of the time the views took more space than data themselves. It is worth to mention that index maintenance in CouchDB should be done with care. For example, re-building the index over the large collection can be very time-consuming operation (it can literally take hours to complete), while accumulation of existing indexes is usually transparent operation.

A few issues with CouchDB were discovered upon running production version of WMAgents. Even though the CouchDB works very reliably, we experienced a few database issues, e.g. the compaction of very large database, with the size above 300GB. But all of them have been recently addressed by CouchDB team and patches were applied to our production code base.

It is worth to mention that both MongoDB and CouchDB databases were put on a private network due to simple security measurements they provide. We relied on apache front-end authentication and group role authorization provided by separate security module. It required all users to supply valid GRID certificate whose DN was verified in internal database and user roles were determined.

On a contrary the igprof components, SQLite and KyotoCabinet, did not require a dedicated server and their setup was trivial.

All of the NoSQL solutions discussed here were build and packaged in custom RPMs. Their deployment was done via standard CMS procedure along with the rest of CMS software stack. The CERN VMs were used for testing and integration phases.

Even though our production experience with discussed NoSQL tools were limited, we are satisfied with their performance and our confidence is partially confirmed by successful stories of their usage in a business world[3].

## 5. Conclusion

The LHC CERN experiments setup a new scale of data management. As we gradually move into PB area a new set of challenges appeared. They include, but not limited to, high-availability of the data, crossing boundaries of distributed databases and efficient management of big data volumes. The software engineers start adopting new tools to accomplish these tasks. We see a new trend of wrapping up the standard 3-tier architecture based on traditional RDBMS back-end with another software layer that provides additional functionality to data management tools.

Here we discussed three NoSQL solutions: MongoDB, CouchDB and KyotoCabinet that are successfully deployed into production environment of CMS experiment. They found a niche in CMS software stack and nicely co-exist with traditional databases. The schema-less feature of MongoDB and CouchDB allows to abstract existing data-services and aggregate information from them. The map-reduce operations provide a new view over the data and used for summary reports, while key-value store (KyotoCabinet) can efficiently substitute static database where data compression is valuable. The choice of NoSQL was justified for each CMS project we discussed here and our production experience with those tools was quite pleasant. They required minimal administrative and maintenance cost, they have been deployed via standard procedure and served our application growth quite well. Even though the size of the data handled by NoSQL is just order of magnitude less of equivalent usage of RDBMS in CMS, their usage is growing quickly.

## Acknowledgments

This work was supported by the National Science Foundation, contract No. NSF-PHY-0970024, and the Department of Energy of the United States of America.

## References

[1] http://en.wikipedia.org/wiki/Nosql
[2] http://cms.web.cern.ch/
[3] http://cdsweb.cern.ch/record/838359
[4] D. Hufnagel, The architecture and operation of the CMS Tier-0, *J. Phys.: Conf. Ser.* **331** 032017, doi:10.1088/1742-6596/331/3/032017
[5] J. Rehn J, et. al., PhEDEx high-throughput data transfer management system, *Proceedings of CHEP06*, Mumbai, India 2006
[6] A. Afaq, et. al., The CMS Dataset Bookkeeping Service, *J. Phys. Conf. Ser.* Vol. **119**, 072001, 2008
[7] W. Badgett, et. al., CMS Web-Based Monitoring, doi: 10.1109/RTC.2010.5750464
[8] Apache CouchDB *http://couchdb.apache.org/*
[9] MongoDB *http://www.mongodb.org/*
[10] J. Petrovic, Using Memcached for data distribution in industrial environment, *Proceedings of the 3rd International Conference on Systems*, ICONS 2008, pp. 368372.
[11] V. Kuznetsov, D. Evans and S. Metson, The CMS data aggregation system *Procedia Comp. Sci.* **1** 1529-37, doi:10.1016/j.procs.2010.04.172

[3] Curious readers can easily find comprehensive list of vendors and commercial companies who are successfully deployed discussed NoSQL solutions in their production environment

[12] S. Wakefield, et. al., The WorkQueue project - a task queue for the CMS workload management system, *Proceedings of CHEP2012*, New York, USA 2012

[13] S. Wakefield et al., Large Scale Job Management and Experience in Recent Data Challenges within the LHC CMS experiment, *in Proceedings of XII Advanced Computing and Analysis Techniques in Physics Research*, Erice, Italy. Nov., 2008, PoS(ACAT08)032.

[14] http://igprof.sourceforge.net/

[15] A. Afaq, et. al., The CMS DBS query language, *J. Phys.: Conf. Ser.* **219** 042043 doi: 10.1088/1742-6596/219/4/042043

[16] A. Dolgert, et. al., A multi-dimensional view on information retrieval of CMS data *J. Phys.: Conf. Ser.* Vol. **119**, 072013, 2008

[17] http://www.erlang.org

[18] http://fallabs.com/kyotocabinet/