Journal of Physics: Conference Series **513** (2014) 042027 doi:10.1088/1742-6596/513/4/042027

Performance of popular open source databases for HEP related computing problems

D Kovalskyi, I Sfiligoi, F Wuerthwein, A Yagil
University of California, San Diego, USA
E-mail: dmytro@cern.ch

Abstract. Databases are used in many software components of HEP computing, from monitoring and job scheduling to data storage and processing. It is not always clear at the beginning of a project if a problem can be handled by a single server, or if one needs to plan for a multi-server solution. Before a scalable solution is adopted, it helps to know how well it performs in a single server case to avoid situations when a multi-server solution is adopted mostly due to sub-optimal performance per node. This paper presents comparison benchmarks of popular open source database management systems. As a test application we use a user job monitoring system based on the Glidein workflow management system used in the CMS Collaboration.

1. Introduction

Databases have been actively used in HEP computing for many years. In the past a choice of a proper solution was pretty straightforward. Relational database management systems (RDMS) dominated the market with the two main open source projects: MySQL [1] and PostgreSQL [2]. High demand for scalable database solutions driven by a rapid growth of social networks led to a creation of new non-relational database management systems that allow for easy deployment of large databases across multiple servers.

New generation of HEP experiments may have thousands of users with petabyte data volumes to process. In this environment it is not always clear if a single server database management system (DMS) is sufficient. A migration from one DMS to another is always a time consuming non-trivial problem that should be avoided if possible. It is tempting to solve this problem by using a highly scalable non-relational DMS for any application. Unfortunately, if such a DMS performs badly for a single server, we may need to use more hardware than it is really necessary for a given task. In other words we need to make sure that scalability is not achieved at the cost of a poor performance of a single node.

In this paper we report performance tests of a few of the most popular relational and non-relational database management systems for a single server solution. As a test application we use a beta-version of a new CMS user job monitoring system built on top of GlideinWMS (Glidein based Workload Management System) [5]. We have tested the following DMS: MySQL, PostgreSQL, MongoDB [3], Apache Cassandra [4].

2. Test procedure

The test procedure is based on a fully functional user job monitoring system called GlideMon. It is based on a single MySQL server used in parallel with the CMS dashboard [6]. In the system the information content is reduced to a minimum that allows users to understand the status of their jobs. For failed jobs a clear summary of causes is provided as well as log files. The log files are stored outside of the monitoring system.

For the test we took a snap-shot of the GlideMon database. It has about 14 million individual jobs total. Most queries that we use are only concerned with the last two weeks of data processing. The last two weeks contain roughly 2.2 million jobs and 5 thousand tasks for 251 individual users. The total database size including indexes is about 10-20 GB depending on the DMS.

The tests were performed on the following computers:

- **Modern Server** - Intel Xeon X5650 @ 2.67 GHz, 12 physical cores, 48GB RAM, SSD Intel 320 MLC, CentOS (6.4)
  - MySQL - 5.1.69-1.el6_4
  - PostgreSQL - 8.4.13-1.el6_3
  - MongoDB - 2.4.7-mongodb_1
- **Multi-core Server** - Intel Xeon E5-2660 @ 2.20GHz, 16 physical cores, 128GB RAM, RAID on 15K RPM SAS disks, CentOS (6.4)
  - MySQL - 5.1.69-1.el6_4
  - PostgreSQL - 8.4.13-1.el6_3
  - MongoDB - 2.4.7-mongodb_1
- **Workstation** - Intel Core 2 Duo E8500 @ 3.17 GHz, 2 physical cores, 8GB RAM, HDD + SSD Samsung 840 Pro, Fedora 18
  - MySQL - 5.5.32-1.fc18
  - PostgreSQL - 9.2.4-1.fc18
  - MongoDB - 2.2.4-2.fc18
  - Apache Cassandra - 2.0.0

We used two typical queries that put significant load on the production system. The first one is a complex aggregation query that counts the number of analysis jobs in each state for each user within the last N days. It is normally executed in a single thread to create "materialized views" (a set of temporary tables), which are queried directly by a web server in the GlideMon system. The second query is a fairly simple select request to list all active tasks for a given user with a number of jobs summary. It is a dynamic query in the production system that is executed for each user request. The test procedure did not cover simultaneous read/write access issues since it was not relevant in our monitoring system. For the same reason, we also did not push the database size to the limit of the available space.

All relational database implementations use an identical schema with the same list of indexes. In the case of non-relational databases, we stored all information in a single table that effectively represents an SQL join of the relevant tables that are used for the two test queries. In the case of MongoDB a few additional indexes were added for the documents that corresponds to primary keys in the relational database schema.

The difference in the disk space utilization plays a role in the overall performance, since the OS can be more efficient in caching data in the memory for smaller databases. MongoDB takes significantly more disk space (24GB) compared with other DMS: MySQL(4GB), PostgreSQL(5GB) and Cassandra(5GB). The difference is not fully understood. It can be partially explained by the denormalization of data in MongoDB and Cassandra cases and the need for extra indexes in MongoDB (Cassandra was not indexed).

We do not show any results for Apache Cassandra since we were not able to implement needed functionally. Unfortunately we realized that this DMS does not fit our task only after creating a functional database and starting to query it. The main problem is that Cassandra is not designed for complex filtering or aggregation. The fact that the query language used in Cassandra (CLQ) is effectively SQL adds to the confusion that Cassandra can be used as a general purpose DMS. To illustrate the limitations of the system we can mention that a simple query to count the total number of elements (columns in Cassandra terminology) may take hours on a simple database like the one that we used. In general this DMS is designed for queries that return a very small number of elements and all the complex aggregation and filtering operations should be performed at the time when the data is inserted into the database.

3. Results

3.1. Complex aggregation query tests

The test consists of a set of queries with an increasing range of aggregation. Table 1 shows the total execution time of the test and Figure 1 shows the query execution time as a function of the number of days that we aggregate. The results in the table indicate that MySQL and PostgreSQL showed similar performance on all hardware with slightly better results demonstrated by MySQL (30% faster). MongoDB showed much worse performance on the Workstation with just 12GB of RAM (almost an order of magnitude slower). Using more advanced hardware we managed to reduce the gap between MongoDB and RDMS, but even in this case MongoDB was a factor of 3 slower.

The results presented in Figure 1 effectively show how the query execution time changes with the increase of the number of elements that have to be counted. MongoDB shows a sharp increase in the execution time above a certain threshold. Replacing the hard drive with a solid state disk, we observed a factor of two drop in the execution time for the queries with a large number of elements. The SSD had no impact for queries accessing a small number of elements. Moving to more advanced hardware we observed an overall reduction in the execution time by about a factor of two for all queries. These results indicate that MongoDB can be more demanding to hardware and available memory for complex queries.

3.2. Simple query tests

While a poor performance of a complex query can be easily noticeable, in practice such queries are heavily optimized and if it is necessary a data model used in the database is modified to avoid the problem. For a database that has a sizable number of users, performance of simpler queries can become an issue that is more important and harder to handle.

The simple query test results are shown in Table 2. Comparing relational database management systems we found a significant performance gain for PostgreSQL with respect to MySQL. While the median execution time for a single query is identical for the two system, total execution time of 1000 queries is a factor of two smaller for PostgreSQL. This means that PostgreSQL can perform significantly better than MySQL for the type of query that we used mostly by delivering more consistent query execution time.

MongoDB showed results comparable with MySQL using advanced hardware. With 10 simultaneous queries MongoDB managed to outperform MySQL, but its performance was significantly degraded on a weaker machine. Nevertheless the performance of MongoDB was never worse than a factor of two compared with MySQL.

While our tests were all done in a single server mode, it is interesting to observe how well different DMS results scale with the number of physical cores in a server. Table 3 shows scalability for MySQL, PostgreSQL and MongoDB comparing the total execution time for a test with one thread with respect to the 12 and 16 simultaneous threads that corresponds to the

number of physical cores in the two servers that we tested. We found that MySQL has a major issue with the scalability, while PostgreSQL and MongoDB scaled very efficiently.

3.3. Experience deploying different database management systems

MySQL and PostgreSQL are well established relational DMS. The systems are widely known and there is a lot of good literature available. Database administration tools and programing interfaces are well developed. Despite their popularity, deploying both MySQL and PostgreSQL can be challenging for non-experts mostly due to strict access rules especially in the case of PostgreSQL. In general this is not an issue, just something that needs to be planned for.

MongoDB and Cassandra are very popular solutions in a new generation of non-relational database management systems. They are quite easy to deploy and fairly easy to use. Both systems are under active development and it can be a challenge to find good documentation, but this is a minor issue.

Hardware	MySQL	PostgreSQL	MongoDB
	unbuffered / buffered execution time (secs)		
Modern Server (ssd, 48GB)	35 / 27	38 / 34	114 / 91
Workstation (ssd, 12GB)	37 / 26	41 / 37	186 / 173
Workstation (hdd, 12GB)	85 / 27	61 / 37	277 / 242

Table 1. The complex aggregation query test results.

Hardware	Threads	MySQL	PostgreSQL	MongoDB
		one query **median** / **95%** / **whole test** time (secs)		
Modern Server (ssd, 48GB)	2	0.02 / 0.3 / 33	0.02 / 0.1 / 12	0.02 / 0.4 / 36
Workstation (ssd, 12GB)	2	0.02 / 0.3 / 33	0.02 / 0.1 / 15	0.03 / 0.4 / 43
Workstation (hdd, 12GB)	2	0.02 / 0.3 / 33	0.02 / 0.1 / 15	0.05 / 0.8 / 78
Modern Server (ssd, 48GB)	10	0.03 / 0.5 / 12	0.02 / 0.1 / 3	0.02 / 0.3 / 7
Workstation (ssd, 12GB)	10	0.08 / 1.2 / 26	0.10 / 0.4 / 14	0.12 / 2.0 / 43
Workstation (hdd, 12GB)	10	0.07 / 1.2 / 27	0.10 / 0.4 / 14	0.12 / 2.5 / 59

Table 2. The simple query test results. The total number of queries per test is 1000. Memory based caching of disk I/O is in use.

4. Conclusions

For the type of queries that we tested, relational database management systems showed the best performance in a single server mode. Out of the two RDMS, PostgreSQL showed better consistency of the query execution time and a better scalability with a number of physical cores in a server.

Non-relational systems showed mixed results. Apache Cassandra turned out to be not suitable for our application, while MongoDB showed comparable with RDMS performance on advanced hardware for simple queries and weak performance for complex aggregation queries. MongoDB and PostgreSQL showed close to 100% scalability with the increased number of physical cores.

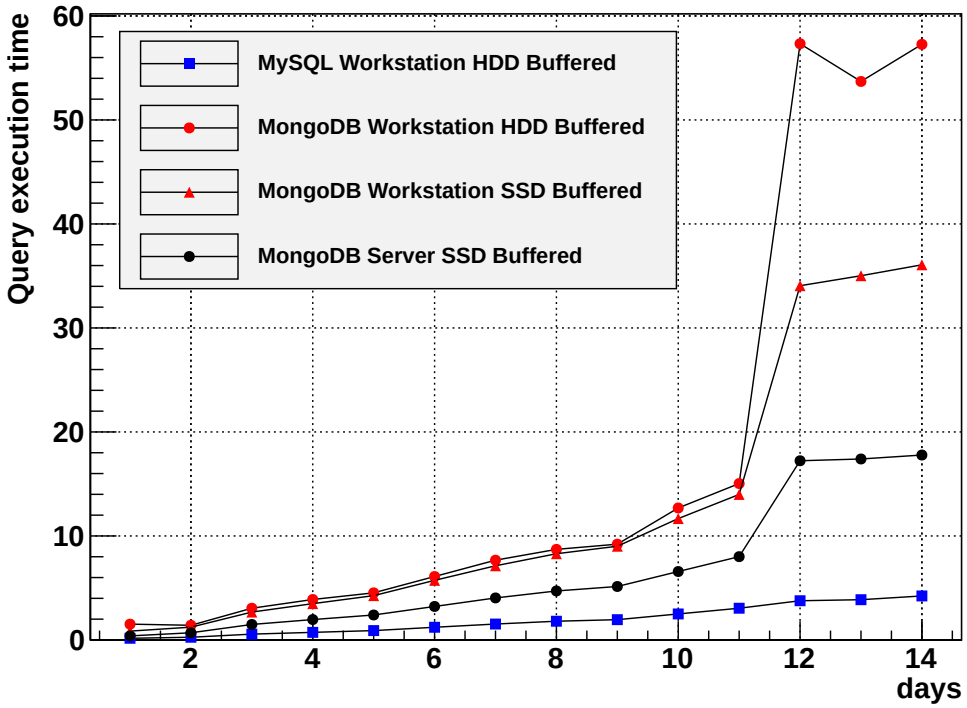

Figure 1. The complex aggregation query results for different reporting periods. Number of jobs considered increases from about 70 thousand jobs for 1 day to about 2 million jobs for 14 days.

Database Management System	Scalability (12-cores)	Scalability (16-cores)
MongoDB	83%	96%
PostgreSQL	78%	100%
MySQL	32%	13%

Table 3. Scalability of various database management systems for the simple query tests. The total number of queries per test is 1000. Scalability is defined as $\frac{1}{N}\frac{t_1}{t_N}$, where N is number of simultanous threads, t_1 is time to finish the test using one thread and t_N is time to finish the test using N threads. Two servers where used - one with 12 physical cores and the other with 16 cores. Memory based caching of disk I/O is in use.

Given that a single core performance of modern CPUs is not likely to increase, this property of MongoDB can be very useful even in a single server mode.

If it is likely that a database management system may outgrow a single server, using MongoDB from the start can be a wise choice, but it is also likely that it will use more resources (both CPU and disk space) than would be needed with either MySQL or PostgreSQL.

5. Acknowledgments

We would like to thank Nikolay Savvinov and Jacob Ribnik for all the fruitful discussions.

References

[1] http://dev.mysql.com/doc/
[2] http://www.postgresql.org
[3] http://www.mongodb.org/
[4] http://cassandra.apache.org/
[5] I Sfiligoi, "glideinWMSa generic pilot-based workload management system", 2008, J. Phys.: Conf. Ser. **119**, 062044, doi:10.1088/1742-6596/119/6/062044
[6] E Karavakis et al, "CMS Dashboard Task Monitoring: A user-centric monitoring view", 2010, J. Phys.: Conf. Ser. **219** 072038, doi:10.1088/1742-6596/219/7/072038