Procedia Computer Science

Volume 66, 2015, Pages 448–457

YSC 2015. 4th International Young Scientists Conference on Computational Science

PanDA Workload Management System meta-data segmentation

M. Golosova[1], M. Grigorieva[1], A. Klimentov[2], E. Ryabinkin[1]

[1]*National Research Center "Kurchatov Institute", Moscow, Russia*

[2]*Brookhaven National Laboratory, Upton, NY, USA*

marina.golosova @cern.ch, maria.grigorieva@cern.ch, alexei.klimentov@cern.ch, rea@grid.kiae.ru

Abstract

The PanDA (Production and Distributed Analysis) workload management system (WMS) was developed to meet the scale and complexity of LHC distributed computing for the ATLAS experiment. PanDA currently distributes jobs among more than 100,000 cores at well over 120 Grid sites, supercomputing centers, commercial and academic clouds. ATLAS physicists submit more than 1.5M data processing, simulation and analysis PanDA jobs per day, and the system keeps all meta-information about job submissions and execution events in Oracle RDBMS. The above information is used for monitoring and accounting purposes. One of the most challenging monitoring issues is tracking errors that has occurred during the execution of the jobs. Current meta-data storage technology doesn't support inner tools for data aggregation, needed to build error summary tables, charts and graphs. Delegating these tasks to the monitor slows down the execution of requests.

We will describe a project aimed at optimizing interaction between PanDA front-end and back-end, by meta-data storage segmentation into two parts – operational and archived. Active meta-data are remained in Oracle database (operational part), due to the high requirements for data integrity. Historical (read-only) meta-data used for the system analysis and accounting are exported to NoSQL storage (archived part). New data model based on usage of Cassandra as the NoSQL backend has been designed as a set of query-specific data structures. This allowed to remove most of data preparation workload from PanDA Monitor and improve its scalability and performance. Segmentation and synchronization between operational and archived parts of jobs meta-data is provided by a Hybrid Meta-data Storage Framework (HMSF). PanDA monitor was partly adopted to interact with HMSF. The operational data queries are forwarded to the primary SQL-based repository and the analytic data requests are processed by NoSQL database. The results of performance and scalability tests of HMSF-adopted part of PanDA Monitor shows that presented method of optimization, in conjunction with a properly configured NoSQL database and reasonable data model, provides performance improvements and scalability.

Keywords: PanDA, workload management system, meta-data segmentation, hybrid storage

doi:10.1016/j.procs.2015.11.051

1 Introduction

Vast volumes of scientific data are generated by experiments and observations. Historically, distributed processing and data management in large-scale scientific experiments are provided by specialized software and hardware systems, for example Distributed Data Management (DDM) or Workload Management systems (WMS). These systems contain tools for data management in a heterogeneous computing environment, including execution of computing tasks on high-performance clusters and supercomputers, providing access to the data files, distribution of computing resources, queuing tasks, and others.

In order to be interpreted and accessed, experimental results and processing tasks must be accompanied by auxiliary meta-data that describe actions performed on computational jobs, stored data or other entities. Volume of these meta-data takes one to the realms of Big Data on many occasions. These meta-data can be used to obtain information about current state of the system, aggregation of data for summary purposes and for statistical and trend analysis of the processes this system drives. Processing and analysis of huge amounts of auxiliary meta-data, surrounding the life cycles of scientific experiments, is no less challenging task than the management and processing of experimental data files and results.

One of the largest scientific collaborations, conducting research in high-energy physics and nuclear physics at the LHC at CERN is the ATLAS (A Toroidal LHC ApparatuS). It works at petabyte scale production and distributed analysis processing, includes more than 3,000 scientists from 180 institutions and 40 countries. Storage, processing and analysis of experimental data are provided by using a distributed grid infrastructure. As a workload management system for production and distributed analysis processing capable of operating at LHC data processing scale, ATLAS uses PanDA (Production and Distributed Analysis) WMS (Klimentov A., 2015), which currently distributes jobs among more than 100,000 cores at well over 120 Grid sites, supercomputing centers, commercial and academic clouds. ATLAS physicists submit more than 1.5M of data processing, simulation and analysis jobs per day. The system keeps all meta-information about job submissions and execution events in Oracle RDBMS, and the above information is used for monitoring and accounting purposes. But due to the vast volumes of stored meta-data, currently used storage technology doesn't provide required level of data processing rate. This paper describes a method to optimize the interaction between PanDA back-end and front-end applications.

2 PanDA meta-data management issues of the backend and front-end interaction

Database is the central component of the PanDA architecture. Currently RDBMS (Oracle or MySQL) is used as the storage back-end. When the computational job is completed, its meta-data remains in the database as it is useful for statistical analysis of recent workflows, detection of faulty resources, prediction of future usage patterns, etc. Full archive of the PanDA now hosts information of 900 million of records – all the jobs since the system started in 2006. Figure 1 shows the number of finished jobs per day for the last two months - currently it's up to 1.7 million jobs per day.

2.1 Specificity of the meta-data storage

To provide better performance and scalability, the PanDA jobs meta-data stored in relational storage are partitioned into actual ("live") and archive (historical) parts. Actual part contains the state of live objects (queued/executing/paused jobs). Basically, each job is presented as a single record in relational table “activejobs” (executing jobs) or ”jobsarchived” (finished jobs). Archive contains a much larger set of read-only data about finished jobs. These data are migrated from actual part after a certain period of time and used only for analytical tasks. But, as the archived data volume grows, the underlying software and hardware stack encounters certain limits that negatively affect processing speed and the possibilities of meta-data analysis.

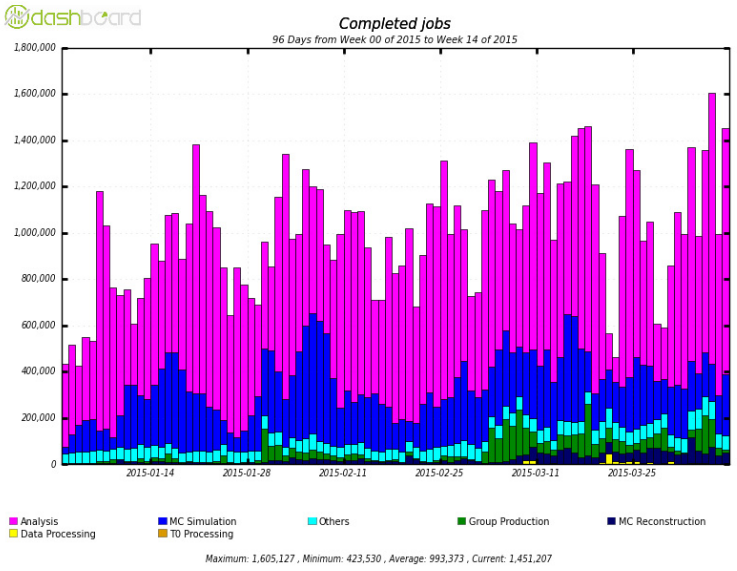

Figure 1: Scale of the challenge: up to 1.7M completed jobs per day

2.2 PanDA Monitor as a front-end application

PanDA front-end application - BigPanDAmon - is responsible for visualizing the state of current and historical jobs, tasks, datasets, and performs run-time and retrospective analysis of failures on all used computing resources (J. Schovancova, 2014).

BigPanDAmon is a modular monitoring package, developed in Python and based on Django Framework (Django Documentation: Django Software Foundation, 2015), which helps to separate data access layer from data visualization. The monitoring is built around common key objects of the PanDA system, such as PanDA job or PanDA resource. Each object is described as a set of Django Views. The View visualizes data in form of tables, plots, and lists. Data access layer provides pre-filtered information describing a set of objects, tailored to the corresponding View. Currently BigPanDAmon provides a realtime and short-term-history monitoring tool for PanDA system.

2.3 Job errors monitoring issue

One of the primary goals of PanDA monitoring is the spotting job failures. It allows users to focus on the most serious errors, collect as much information as possible, and raise the issue with an expert or a responsible person, so that the issue can be resolved. Error messages play a key role in determining the potential problems that might arise and helps in mitigating the resource downtime before it can occur.

To build the errors report, the monitor first executes basic object selection with Django Querysets. In special case of the model of PanDA job the Queryset has been replaced by its generalization, Querychain. DB queries are optimized using the raw SQL code. Then the monitor prepares obtained meta-data for the report using a set of specific data processing modules. Delegating data aggregating tasks to the monitor, instead of using database-specific data processing tools, slows down the execution of user requests. Figure 2 shows that the total page generation time, including database request and aggregating obtained meta-data, dramatically increases with the growth of the number of processing jobs. The nonlinear effect that can be noticed on the upper graph comes from the hardware limits of the monitor server; basically, one could expect near-linear growth of the total page generation time providing the unlimited memory resource.

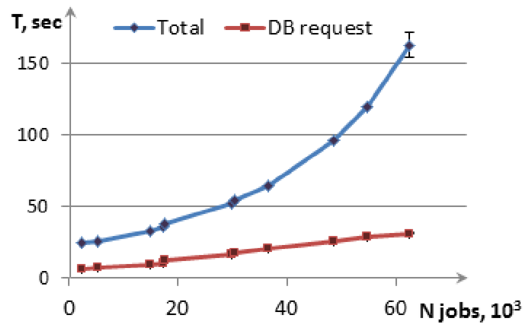

Figure 2: Page generation time

Since we have already collected meta-data for almost 10 years and the amount of accumulated meta-data is constantly increasing, there is a need of long-term failures forecasting and analysis of system behavior under various conditions. Building errors report, based on historical data for a long time interval (months, years) may take considerable time, exceeding the reasonable time of web page generation.

To optimize the interaction between PanDA back-end and front-end applications proposed a technique of meta-data storage segmentation into two parts – SQL for operational meta-data and NoSQL for archived records. Active meta-data are stored in Oracle database, due to the high requirements for data integrity. Historical (read-only) meta-data, used for the system analysis and accounting, are exported to NoSQL storage in request-specific data structures with pre-filtered data, describing errors. It allows to remove most of data preparations workload from BigPanDAmon data access layer and improve its scalability and performance.

3 PanDA meta-data segmentation

3.1 Hybrid Meta-data Storage Framework architecture

Segmentation and synchronization between operational and archived parts of job meta-data are provided by a Hybrid Meta-data Storage Framework (HMSF), which operates on both SQL and NoSQL back-end storages. The basic modules of the HMSF are the Storage (soft wrapper around the databases) and the Storage API (interface that interprets requests from external applications).

The structure of Storage consists of three sub-modules:

- SQL (transparent for external calls wrapper around RDBMS).
- NoSQL (transparent for external calls wrapper around NoSQL-databases).
- Internal logic module, which implements the interaction between SQL and NoSQL parts of the storage and the internal logic of the data distribution between them.

The internal logic of hybrid storage data management includes following tasks:

- Provide consistence (synchronization) of SQL and NoSQL parts.
- Provide inner consistency within NoSQL (this implies detection of unsynchronized data, data synchronization between main tables and dependent/service tables, and does not affect the whole system performance).

- Clean SQL database from archived records.

The structure Storage API also includes three submodules:

- SQL (implementation of requests to the RDBMS),
- NoSQL (implementation of requests to the NoSQL-databases),
- External logic module (which implements requests routing to SQL and NoSQL modules, depending on the type of data, which were requested, and where they should be stored in accordance with the logic of the data distribution).

Currently the Storage API part is under development, and in the future HMSF will use it to communicate with the external applications. But as the realisation will take time, while it is not that essential for the principal schema testing, for now it was decided to put this part of the work aside.

In terms of PanDA environment, HMSF operates as a set of background processes: migration of data from central database (Oracle) to archive (NoSQL), execution of precalculations and query-specific aggregations on job meta-data, clear central storage from unused data, inspection and maintenance of data integrity in NoSQL part of storage.

3.2 NoSQL Data Model for Errors Report

As a NoSQL back-end we have chosen Apache Cassandra database - an open source distributed database management system designed to handle large amounts of data across many servers, providing high availability with no single point of failure. Cassandra is built on Amazon's Dynamo and Google's BigTable (Fay Chang, 2006).

Creating a thoughtful and conscious data model for Cassandra in advance is very important, because it allows using benefits of the specific NoSQL features, providing evenly distribution of data across all cluster nodes. A common data modelling strategy for NoSQL database systems is to store data in query-specific tables. In Cassandra, in contradistinction to RDBMS, we define keyspaces as database schema, and column families (CFs) as tables. Cassandra supports two different models for storing data: single primary key and a composite one. In case of the single Primary Key, the key itself defines the physical location of the data in the cluster: the row of data is sent to nodes by the value of the key hash. Composite key made up of multiple fields of the table and is divided in two parts. The first part is called Partition Key and is responsible for the physical distribution of the data across the storage cluster. The second part is a set of Clustering Keys. They are responsible for data ordering/grouping (clusterization) within a partition.

Cassandra performs best if all the data required for a given query is located in the same CF. Given that Cassandra has no support of foreign key relationships and we cannot JOIN multiple CFs to satisfy a query, we have to denormalize the data model so that a user/application request can be served by the results of one query. Instead of doing multiple reads from multiple tables and partitions to gather all the required data for a response, we should modify the data storage logic and insert the required data multiple times into every row that might be needed by the future requests. This way, all the required data can be available in just one read which prevents multiple lookups.

To improve BigPanDAmon performance, job meta-data aggregation logic was added to the HMSF. Instead of fetching all records for failed jobs and then creating error summaries in Web application, we precalculate summaries for defined time intervals and then put the summarized data into the new (query-specific) NoSQL Cassandra CFs.

Current Cassandra data model includes main table "Jobs", cloned from Oracle database, and a set of auxiliary query-specific tables with precalculated and aggregated data (Figure 4). In auxiliary CFs error meta-data are spread evenly across Cassandra nodes according to the time interval - 24 hours. It fits the limitation on the partitions size (no partition can store more than 2 billion cells) for Cassandra and it's a safe approach as long as PanDA handles less than 1 billion jobs per day (even for that unfortunate day when all jobs are failed).

Cassandra imposes restrictions on the requests, depending on partition and clustering keys. Auxiliary tables are constructed in accordance with these restrictions (Datastax documentation: CQL for Cassandra 2.1. Restrictions on the use of conditions). Table 1 presents error summary issues in terms of database requests with partition and clustering keys.

Request	Partition key	Clustering key
get all meta-data of job by identifier (pandaID)	pandaID	-
get all job's errors by given day	date (partition by 1 day)	-
get all job's errors for site by given day		Computing Site
get errors with certain error code for site by given day		Computing Site, Error Code
get number of errors by given day within every 30 minutes interval		Base_mtime (30 minutes intervals)
get number of errors for site by given day within every 30 minutes interval		Base_mtime (30 minutes intervals), Computing Site
number of jobs with errors within 30 min. interval or within every interval of a given day		Base_mtime (30 minutes intervals)

Table 1: Error summary issues

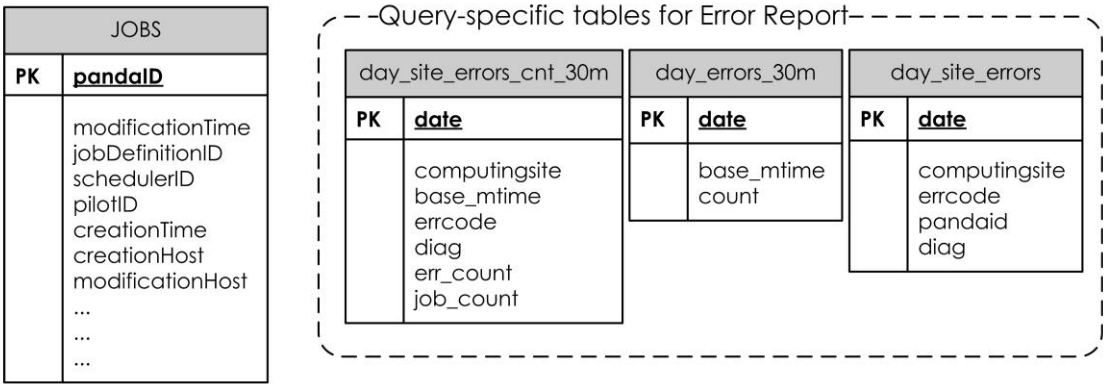

Figure 3: Cassandra data model

The flow diagram for synchronization and precalculation (synchronization of the tables within the NoSQL database) of job meta-data within HMSF is shown on a Figure 4. Coordinator node starts the process, sending the data request to the storage and stores results in the internal database-neutral format (1). Then it transforms received set of data to Cassandra-specific format and/or make data precalculation and aggregation (2), and then sends it to the various NoSQL tables (3). This process is repeated until there's no more data in the source table matching the initial synchronization parameters (e.g. time interval).

4 PanDA front-end adaptation to interact with HMSF

HMSF interacts with SQL as well as with NoSQL databases, so on the side of front-end we need a powerful abstraction that automates all the manual indexing work and provides an advanced query API which simplifies common nonrelational patterns. The Django ORM framework, used in BigPanDAmon to interact with databases, is such an abstraction and it fits our goals pretty well.

Current Django ORM officially supports a set of SQL backends – MySQL, Oracle, PostgreSQL and Sqlite, and provides automatic database routing. But unfortunately, NoSQL databases are not officially supported by Django itself. There are projects like Django-nonrel (https://github.com/django-nonrel), an independent branch of Django that tries to fill this gap and add support for MongoDB or Google App Engine. Backends for ElasticSearch and Cassandra are also in the development. The long-term goal is to add NoSQL support to the official Django release.

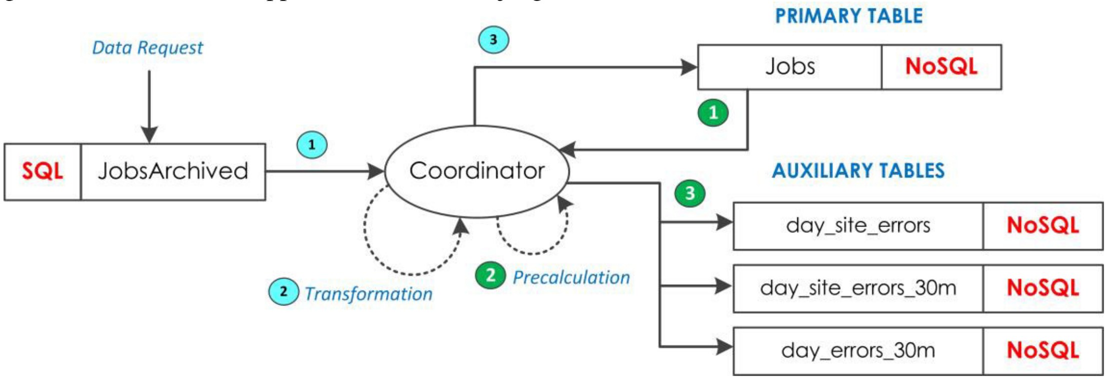

Figure 4: Synchronization and precalculation process

For the adaptation of BigPanDAmon to interact with hybrid SQL/NoSQL meta-data storage back-end, it was moved to the Django-nonrel. Its interaction with Cassandra is based on custom Cassandra database wrapper - django-cassandra-engine. It uses cqlengine which is currently the best Cassandra Object Mapper for Python. The basic features of Cassandra-engine are: working syncdb, migrate commands, automatic connection and disconnection handling, accept all cqlengine and Cassandra driver connection options, support for multiple databases, including relational. In fact, Cassandra-engine with cqlengine is the substitution of core Django ORM objects, like models, QuerySets, managers and compilers.

Query routing logic is concentrated in Django Views of the Monitor: the operational data queries are forwarded to the primary SQL-based repository and the analytic data requests are processed by NoSQL database. The schema of the interaction between BigPanDAmon and hybrid meta-data back-end is presented on Figure 5.

5 Adding Performance and scalability test of HMSF-adopted PanDA monitor application

Performance of the "archived" data storage in the analytical tasks is shown in the quantitative scalability and performance test results, including testing NoSQL storage against Oracle RDBMS. For the testing we have deployed test installation between CERN and Kurchatov Institute (NRC KI): at CERN we have PanDA monitor instance with Oracle storage backend, and at NRC KI - PanDA monitor instance adapted for the interaction with NoSQL database (storage back-end: Oracle database in CERN and Cassandra cluster in NRC KI).

5.1 Cassandra scalability test results

Testing was conducted on the 2 month slice of archived jobs meta-data (~60M jobs, ~100 GB per replica). We had created two test sets: 7.8 M and 13.8 M rows in request-specific CFs (from 30 M and

62 M rows the full archive CFs). The same requests were executed to both of those datasets. Tests results showed that query execution is independent of the total number of records in the requested table - at least, in the given range (Figure 6), and the page generation time is almost a linear function of the number of jobs matching the request.

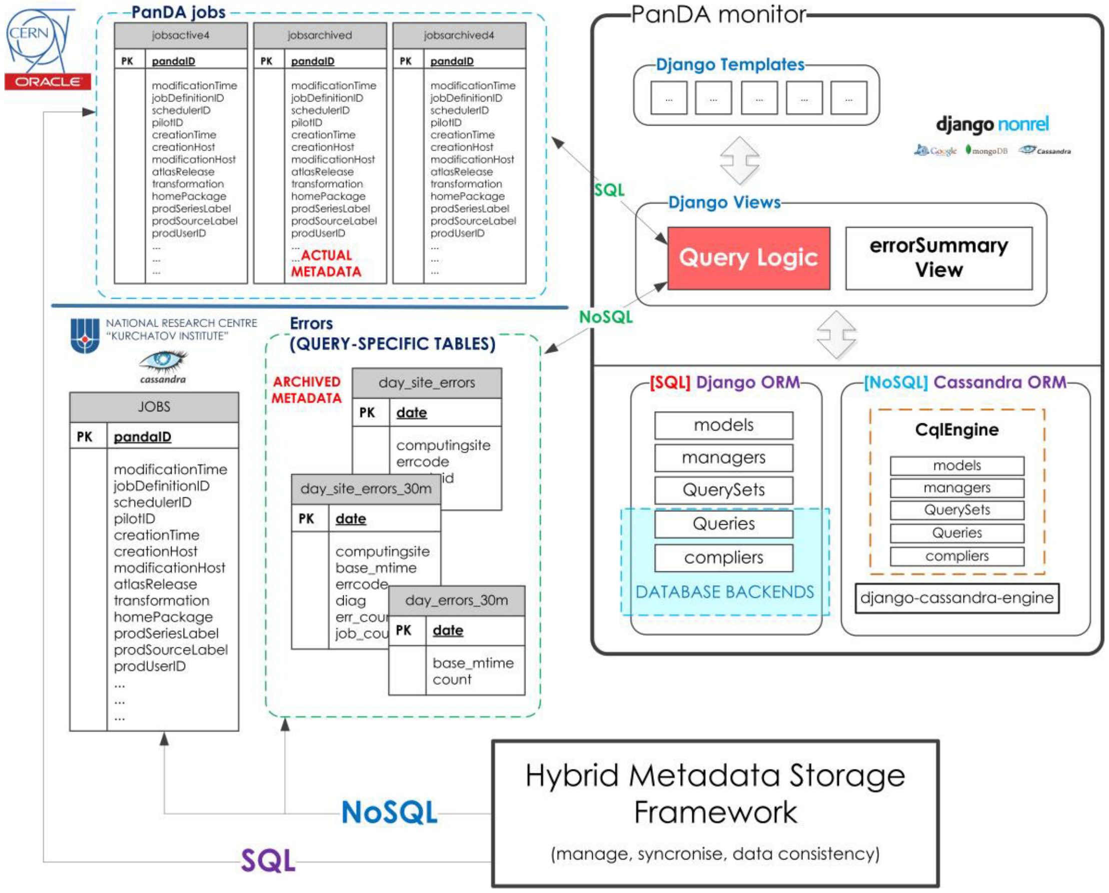

Figure 5: Schema of the interaction between BigPanDAmon application and hybrid meta-data back-end

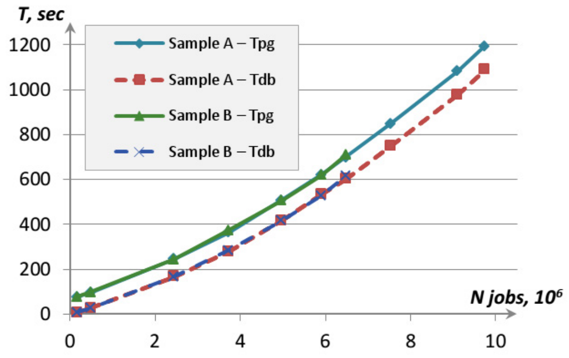

Figure 6: Total (page generation, Tpg) and database (Tdb) request time vs. number of jobs matching the request

5.2 Aggregation effect test results

Cassandra data model has two test tables, optimized for error page: with data prepared for the request ("day_site_errors"), and with the same data aggregated for every 30 minutes ("day_site_errors_cnt_30m"). In aggregated table one database record contains the number of all errors (with the same error code) within given 30 minutes. The tests showed that it took almost 4 times less time to generate same page using aggregated data (Figure 7). And for requests for a longer periods (months, years), it is possible to use daily or weekly summaries to make things even faster. The speed-up was mostly achieved by reducing the number of records stored in database (and, thus, read by application): non-aggregated table had 7.8 M rows, and aggregated held only 1.9 M.

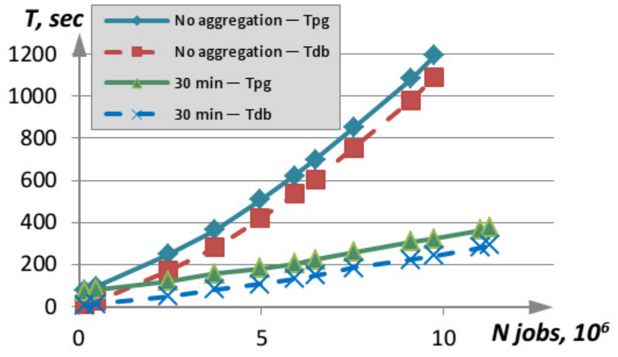

Figure 7: Page generation (Tpg) and DB request (Tdb) time for plain (not aggregated) and aggregated data

5.3 NoSQL Archive performance test results

Finally, we have compared performance of the two PanDA Monitor instances. On the Figure 8 we can see that request and page generation times for NoSQL archive are almost constant (for a given range of data), while the page of original PanDA Monitor slows down significantly. Relying on the results of the NoSQL scalability test we can say that NoSQL part will not stay constant for a wider data range, but still results look promising.

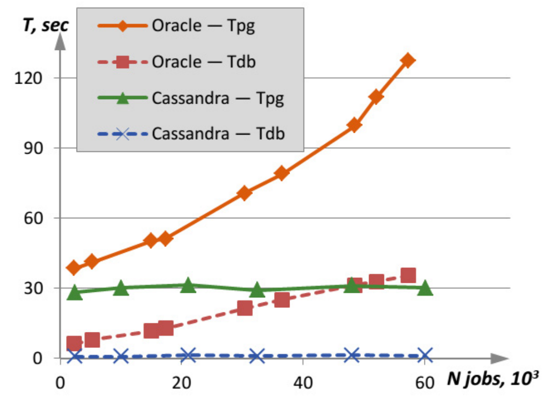

Figure 8: NoSQL Archive performance

6 Conclusion

When working with Big Data, it is hardly possible to perform long-term meta-data analysis without any precalculations. Replacing RDBMS with NoSQL in archive part of meta-data storage can be used to improve availability of historical data. Prototype of NoSQL Cassandra archive was created and tested on a 2-month slice of meta-data from ATLAS PanDA archive. The results of performance and scalability tests of BigPanDAmon, adapted to hybrid SQL/NoSQL storage, show that presented method of optimization, in conjunction with a properly configured NoSQL database and reasonable data model, provides performance improvements and scalability. Cassandra shows a good scalability, and carefully organized data and other elaborations, such as advance data aggregation and precalculation, provide significant performance improvement without adding much complexity to the resulting system.

These results look promising. NoSQL archive will be extended to confirm this belief in future tests. Adaptation of PanDA Monitor to work with the hybrid storage back-end will be continued.

7 Acknowledgements

We wish to thank all our colleagues from ATLAS experiment, PanDA team and CERN IT. This work was funded in part by the Russian Ministry of Science and Education under contract № 14.Z50.31.0024. Testbed resource at NRC KI are supported as a part of the centre for collective usage operated under agreement 14.621.21.0006 with Ministry of Science and Education of Russia.

References

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach

Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber (2006). Bigtable: A Distributed Storage System for Structured Data, http://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf

Datastax documentation: CQL for Cassandra 2.1. Restrictions on the use of conditions, http://docs.datastax.com/en/cql/3.1/cql/cql_reference/select_r.html

Django Documentation: Django Software Foundation (2015), https://media.readthedocs.org/pdf/django/latest/django.pdf

A Klimentov, P Buncic, K De, S Jha, T Maeno, R Mount, P Nilsson, D Oleynik, S Panitkin, A Petrosyan, R J Porter, K F Read, A Vaniachine, J C Wells and T Wenaus (2015). Next Generation Workload Management System For Big Data on Heterogeneous Distributed Computing // Journal of Physics: Conference Series, Volume 608, conference 1

J. Schovancova, K. De, A. Klimentov, P. Love, M. Potekhin, T. Wenaus (2014). The next generation of ATLAS PanDA Monitoring // Proceedings of Science, ISGC 2014, 035