Development of a noSQL storage solution for the Panda Monitoring System

“Database Futures” Workshop at CERN
June 7th, 2011

Maxim Potekhin
Brookhaven National Laboratory
Physics Applications Software Group

potekhin@bnl.gov

Brief Intro: Panda Workload Management System (1)

Panda is the principal workload management system used in ATLAS. It is a “pilot-based” framework. Pilot jobs are started remotely on participating sites, and after they are successfully activated and perform validation of the environment on the worker node, the payload is acquired and executed. The job (payload) is matched to a pilot by the central service based on a number of criteria (“brokerage”).

Pilots, jobs and other entities in Panda transition between their various states as this cycle is repeated over and over. The state of object is recorded in RDBMS (Oracle) and guaranteed consistency of these data is crucial for operation of Panda.

Brief Intro: Panda Workload Management System (2)

After a Panda job is finished, the data associated with its attributes is preserved for troubleshooting, accounting, optimization and other purposes. It is accessed by the Panda Monitor web service and a few other systems. We note that archive-type data is largely de-normalized, does not require most of the features of RDBMS and

- There are classes of queries which include time and date ranges where Oracle typically does not perform too well
- An option to unload significant amounts of archived reference data from Oracle to a highly performing, scalable system based on commodity hardware appears attractive

Brief Intro: Panda Workload Management System (3)

David Edward Jaffe	Error details: **trans:** segmentation violation message							
1244908542 David Edward Jaffe	jobsetID=92 fmcp11a_nuwa_trf	running	2011-05-31 15:17	5 days, 8:08:28	13:43:29	06-06 12:47	OSG/OSG.LBNE_DAYA_1, analysis-run	988
1244908541 David Edward Jaffe	jobsetID=92 fmcp11a_nuwa_trf	running	2011-05-31 15:17	5 days, 7:03:52	14:48:05	06-06 12:47	OSG/OSG.LBNE_DAYA_1, analysis-run	989
1244908540 David Edward Jaffe	jobsetID=92 fmcp11a_nuwa_trf	finished	2011-05-31 15:17	2 days, 20:53:04	9:42:30	06-03 21:53	OSG/OSG.LBNE_DAYA_1, analysis-run	989
1244908539 David Edward Jaffe	jobsetID=92 fmcp11a_nuwa_trf	failed	2011-05-31 15:17	2 days, 17:53:35	10:38:28	06-03 19:49	OSG/OSG.LBNE_DAYA_1, analysis-run	989
	Error details: **trans:** segmentation violation message							
1244908538 David Edward Jaffe	jobsetID=92 fmcp11a_nuwa_trf	failed	2011-05-31 15:17	2 days, 17:17:26	18:52:47	06-04 03:27	OSG/OSG.LBNE_DAYA_1, analysis-run	989
	Error details: **trans:** segmentation violation message							
1244908537 David Edward Jaffe	jobsetID=92 fmcp11a_nuwa_trf	failed	2011-05-31 15:17	2 days, 16:00:36	12:30:34	06-03 19:48	OSG/OSG.LBNE_DAYA_1, analysis-run	989
	Error details: **trans:** segmentation violation message							
1244908536 David Edward Jaffe	jobsetID=92 fmcp11a_nuwa_trf	failed	2011-05-31 15:17	2 days, 15:59:35	12:31:37	06-03 19:48	OSG/OSG.LBNE_DAYA_1, analysis-run	990
	Error details: **trans:** segmentation violation message							
1244908535 David Edward Jaffe	jobsetID=92 fmcp11a_nuwa_trf	finished	2011-05-31 15:17	2 days, 12:52:50	9:50:15	06-03 14:00	OSG/OSG.LBNE_DAYA_1, analysis-run	990
1244908534 David Edward Jaffe	jobsetID=92 fmcp11a_nuwa_trf	failed	2011-05-31 15:17	2 days, 12:46:03	22:15:52	06-04 02:19	OSG/OSG.LBNE_DAYA_1, analysis-run	990
	Error details: **trans:** segmentation violation message							
1244908533 David Edward Jaffe	jobsetID=92 fmcp11a_nuwa_trf	failed	2011-05-31 15:17	2 days, 12:43:56	23:25:43	06-04 03:27	OSG/OSG.LBNE_DAYA_1, analysis-run	990
	Error details: **trans:** segmentation violation message							
1244908526 David Edward Jaffe	jobsetID=92 fmcp11a_nuwa_trf	failed	2011-05-31 15:17	4 days, 1:44:54	16:57:08	06-05 09:59	OSG/OSG.LBNE_DAYA_1, analysis-run	992
	Error details: **trans:** segmentation violation message							
1244908525 David Edward Jaffe	jobsetID=92 fmcp11a_nuwa_trf	finished	2011-05-31 15:17	4 days, 1:31:06	8:52:17	06-05 01:40	OSG/OSG.LBNE_DAYA_1, analysis-run	992
1244908523 David Edward Jaffe	jobsetID=92 fmcp11a_nuwa_trf	failed	2011-05-31 15:17	1 day, 13:16:31	1 day, 15:15:22	06-03 19:49	OSG/OSG.LBNE_DAYA_1, analysis-run	992
	Error details: **trans:** segmentation violation message							
1244908518	jobsetID=92 fmcp11a_nuwa_trf	failed	2011-05-31 15:17	3 days, 16:12:39	1 day, 10:27:11	06-05 17:57	OSG/OSG.LBNE_DAYA_1, analysis-run	993

Motivation

We need to scale out as the number of jobs processed daily by Panda is steadily growing with peak capacity of about 840k jobs per day. Using a noSQL solution allows us to solve scalability issues without putting more demands and load on the mission-critical Oracle infrastructure of ATLAS. In addition, this gives us an opportunity to optimally index data, with resulting performance improvements (more details below).

Cassandra is one of many noSQL platforms available today, and we chose it because of existing expertise in ATLAS, and ease of installation and configuration.

Objectives of the project

Ultimately, we aim to create of a noSQL-based data store for archived reference data in Panda. In the validation stage, it would be the data source for Panda Monitor and will continuously function in parallel with the Oracle DB in order to ascertain the performance and characteristics of such system under real life data volume and query loads.

To this end, we are pursuing multiple objectives, in the context of Cassandra DB:

- to create an optimal data design for the data generated by the Panda server as well as indexes
- to perform a performance comparison of the recently built cluster at BNL to the existing Oracle facility at CERN
- To determine the optimal hardware configuration of that system and its tuning parameters

History of the project

- In Jan-Feb 2011, using the 4-VM Cassandra R&D cluster at CERN (created by ATLAS DDM team), gained experience in data design and indexing of Panda data, as well as operational experience with Cassandra
- Demonstrated usefulness of map-reduce approach to generation of date range indexes into Cassandra data, which results in marked performance improvement
- In March 2011: commissioned a 3-node real hardware cluster at BNL and loaded a year worth of real Panda data in a modified format (based on initial CERN experience). Ran a series of performance tests aimed at the “worst-case performance” estimation, to set the baseline for further optimization.
- May 2011: upgraded to SSD, testing in progress

The Cassandra Cluster at BNL

Currently we have 3 high-spec nodes and tested the cluster in 2 data storage configurations.

Each node:

- two Xeon X5650 processors, 6 cores each
- due to hyper-threading – effectively 24 cores
- 48GB RAM on a 1333MHz bus
- Storage Configuration 1: six 500GB, 7200RM SATA RAID0
- Storage Configuration 2: five consumer-grade 240GB SSD in RAID0 configuration (recent)

We also have a separate client machine from which to do queries against the Cassandra DB.

Mapping/indexing of the Data (1)

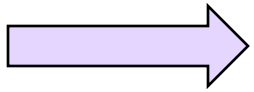
Note on indexing: native ability to index individual columns is a relatively new feature in Cassandra, introduced in mid-2010. Since our data is mostly write-once, read-many, we can do a better job by indexing data in map-reduce fashion, and we don't need to create extraneous columns to implement an equivalent of composite indexes (so we save space).

Typically, the indexing process is run at the same time as the data is loaded into Cassandra, however we can add more indexes later if and when needed.

Proper index design helps to dramatically improve performance. Example: find number of failed jobs for user XYZ in a range of dates, in cloud ABC. Can takes minutes in standard SQL query vs tens of milliseconds on Cassandra with map/reduce.

Mapping/indexing of the Data (2)

We identify popular/high demand queries made against the database, and create MAPS. In the above example,

USER::CLOUD::STATUS::DATE  LIST of IDs

Index thus created takes up approx. ~1% of space compared to the data that's being indexed – affordable.

The above is just one example, in reality we create many maps to reflect the queries made by users.

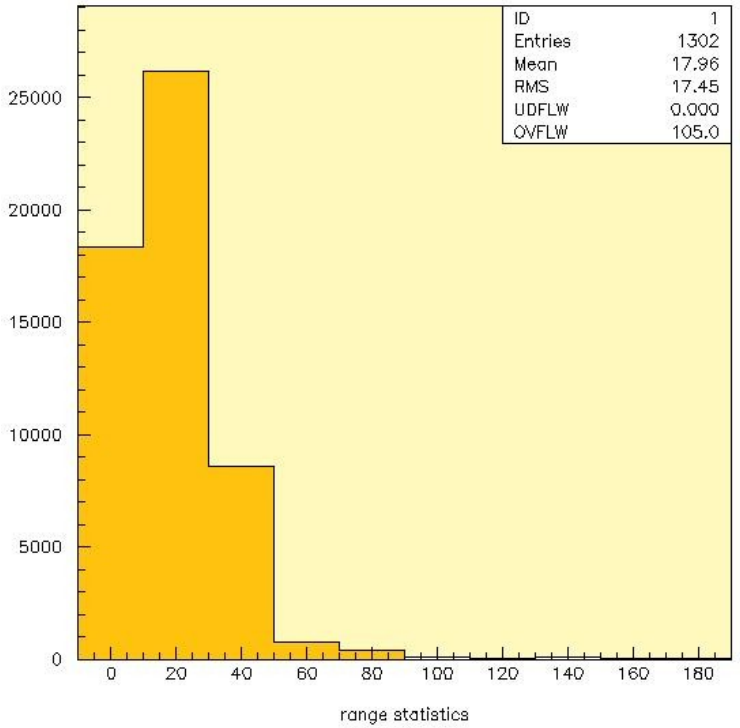
Patterns in the Data

In Panda, jobs are often submitted not individually, but in batches which can measure anywhere from 2 to hundreds of jobs. This is demonstrated by the following plot which shows the distribution of lengths of continuous ID ranges as they are found in the database.

Exploit the pattern:

Use bucketing for optimal performance!

One trip to DB instead of ten or a hundred.



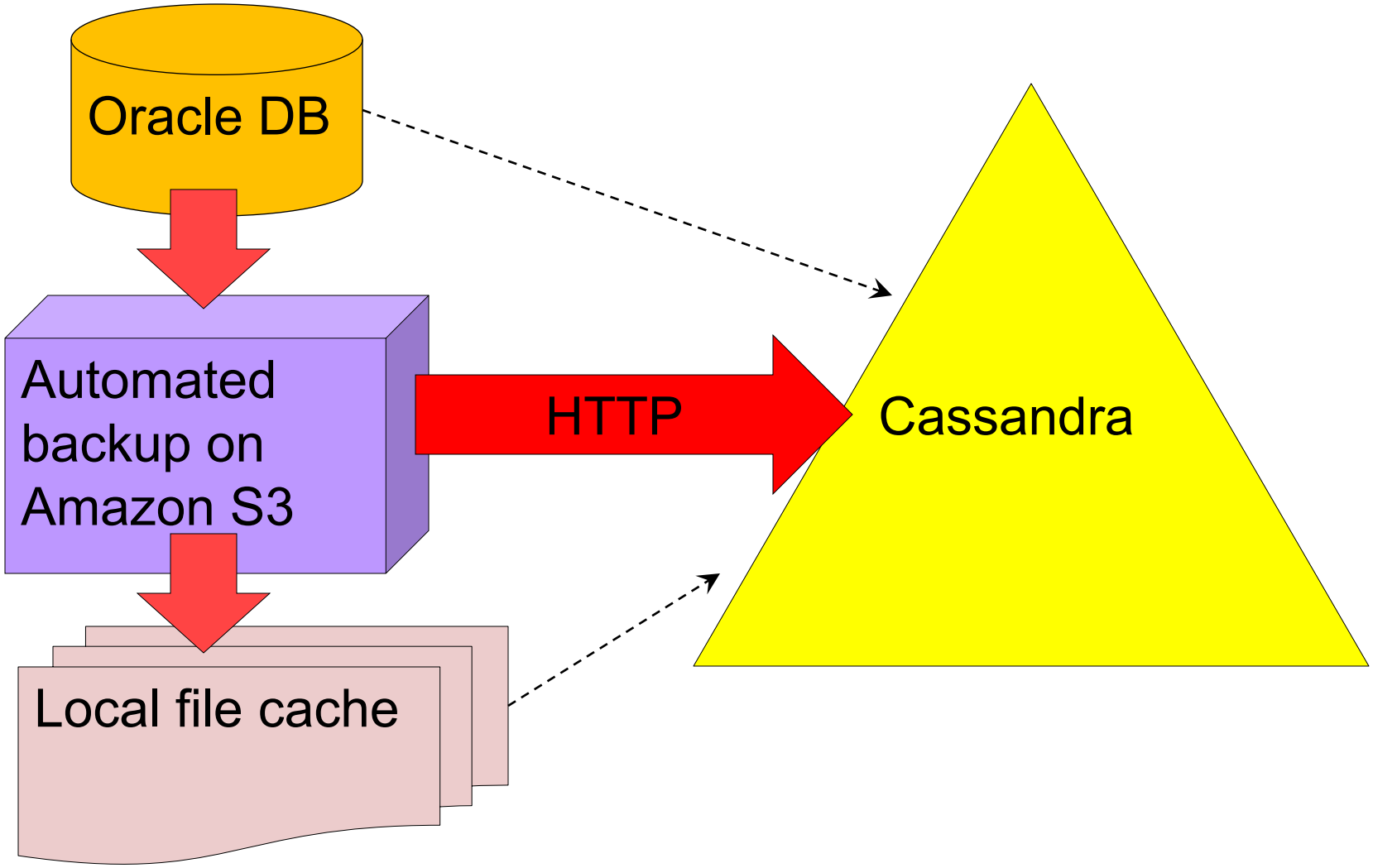
Packaging of the data

Cassandra provides a lot of flexibility in how the data may be formatted and stored. In the application being considered, we could use any of the following

- Simple dictionary of key-value pairs for each attribute (storage penalty as key names are stored for each object)
- JSON strings (same issue)
- CSV (low storage overhead, but necessitates parsing in the client)

We have opted to use the latter option in most recent configuration.

Data sources used to load the Cassandra instance



Testing/Benchmarking Comments

- Using a multithreaded client for data load, with Amazon S3 as current primary source
- Write performance: with proper separation of the commit log and primary data store, in all of our testing, Cassandra was able to handle all data that we could provide from our network sources – hard or impossible to saturate at this point.
- Indexed queries perform on par with Oracle or better, under moderate load
- Our focus now is to consider the worst case scenario with simple primary key queries under moderate to high load (~1000 queries per second), because this will determine the behavior of most applications.
- The Oracle cluster that we use in our benchmarks has >100 spindles of the Raptor class. In Phase 1 of testing at BNL, we only had 18.

Testing with BNL cluster: phase 1, rotational media

Case #1: Large query load, primary key (ID) queries.
Data load: 1 year worth of job data.
Looking at throughput in a random query of 10k items over the past year, vs the number of concurrent clients.
Numbers are seconds to query completion.
Using iostat and Ganglia, we observe disk bound I/O behavior and corresponding scaling in Cassandra. Disk is close to saturation at about 10 concurrent clients, and client network bandwidth also becomes a limiting factor.

	1	10	30
Cassandra	200	300	600
Oracle	500	500	500

Testing with BNL cluster: phase 2, SSD

In the table below, we list time to query completion and disk utilization percentages as reported by iostat. As in the previous case, each client runs 10k random job queries against a year worth of data.

We weren't able stress the disk I/O further because of the network constraints (work in progress).

	1	10	30
Cassandra	135s/2%	150s/15%	220s /27%
Oracle	500s	500s	500s

Conclusions and plans

- With hi-spec machines and SSD we aren't quite scaling horizontally, rather diagonally – however cost/benefit of either approach is still to be evaluated – maintenance and administration of a few dozen low spec machines doesn't come for free either.
- We observe that we have plenty of headroom with the current setup while handling the load of the order of 1500 queries per second that is higher than the current load on the ATLAS Oracle server used for similar purposes.
- At this point, we are satisfied with characteristics of the Cassandra cluster built for the purpose of serving Panda job archive data, and our focus will be on building a functional high performance data store for Panda Monitor and other related systems.
- Will serve JSON to clients, most likely use Django.