Development of noSQL data storage for the ATLAS PanDA Monitoring System

M.Potekhin
Brookhaven National Laboratory
for the ATLAS Collaboration

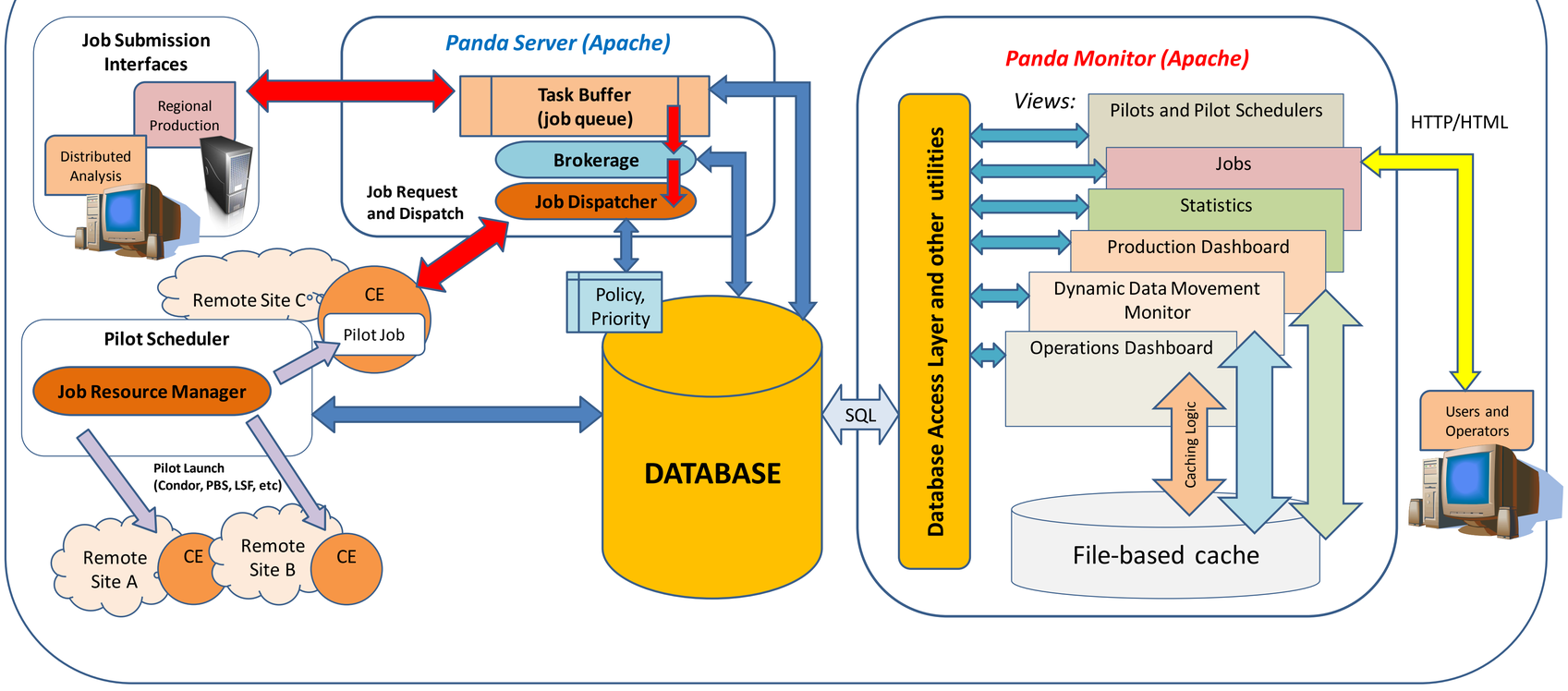
Overview of Panda Workload Management System

Panda is built on the concept of Pilot Job Framework. Workload is assigned to successfully activated and validated Pilot Jobs, which are lightweight processes that probe the environment and act as a 'smart wrapper' for the payload. This 'late binding' of workload jobs to processing slots prevents latencies and failure modes in slot acquisition from impacting the jobs, and maximizes the flexibility of job allocation to globally distributed resources.

A central and crucial component of the Panda architecture is its database, which at any given time reflects the state of both pilot and payload jobs, as well as stores a variety of vital configuration information. It is driven by the *Panda Server*, which is implemented as an Apache-based Python application and performs a variety of brokerage and workload management tasks. Guaranteed consistency of the database is crucial for correct operation of the Panda server.

By accessing this database, another Panda component – the *Monitoring System* – offers to its users and operators a comprehensive and coherent view of the system and job execution, from high level summaries to detailed drill-down job diagnostics.

Main components of Panda Architecture and Current Monitor Implementation



Characteristics of the currently deployed Panda Monitoring System and its database backend

Panda Monitoring System is a Web application that performs complex tasks of database queries and data aggregation, in order to provide the end user with a variety of interconnected views of the state of the system and its configuration. The state of the system includes the status of pilot jobs, payload jobs, data that's being transmitted in support of payload execution, and other parameters. It is reflected in the database, which in its current implementation relies on Oracle RDBM. The data is indexed based on most popular queries. Other optimization tools are employed such as data partitioning, hints, bind variables. Data is largely de-normalized. Also, the data is segregated into a "live" segment (table) which is frequently updated by the server, and the archive table, which remains largely static.

Motivations For System Evolution

With more than 500,000 new entries generated daily, and multi-terabyte data load, the Oracle database used in Panda is approaching the scale of what is called "Big Data". Despite ongoing query optimization effort, there are capability and performance limits imposed by available resources of the current Oracle installation. We observe that for purposes of *monitoring applications* which access archived data we do not use any of RDBMS functionality (such as capability to perform joins, or hard consistency). In situations like this, a variety of ***noSQL*** database solutions are often employed by major Web services that require extreme degrees of availability, resilience and scalability, such as Google, Amazon, Facebook, Twitter, Digg and many others, as well as "brick and mortar" businesses . Our goal is to make our systems future-proof and gain experience with technology we may use going forward.

Role of noSQL DB in Panda Monitoring

We plan to use noSQL back-end for finalized, reference portion of the data where consistency is not essential but sheer performance and scalability are. Access to the data is implemented as a Web service, which furnishes data to clients in JSON format.

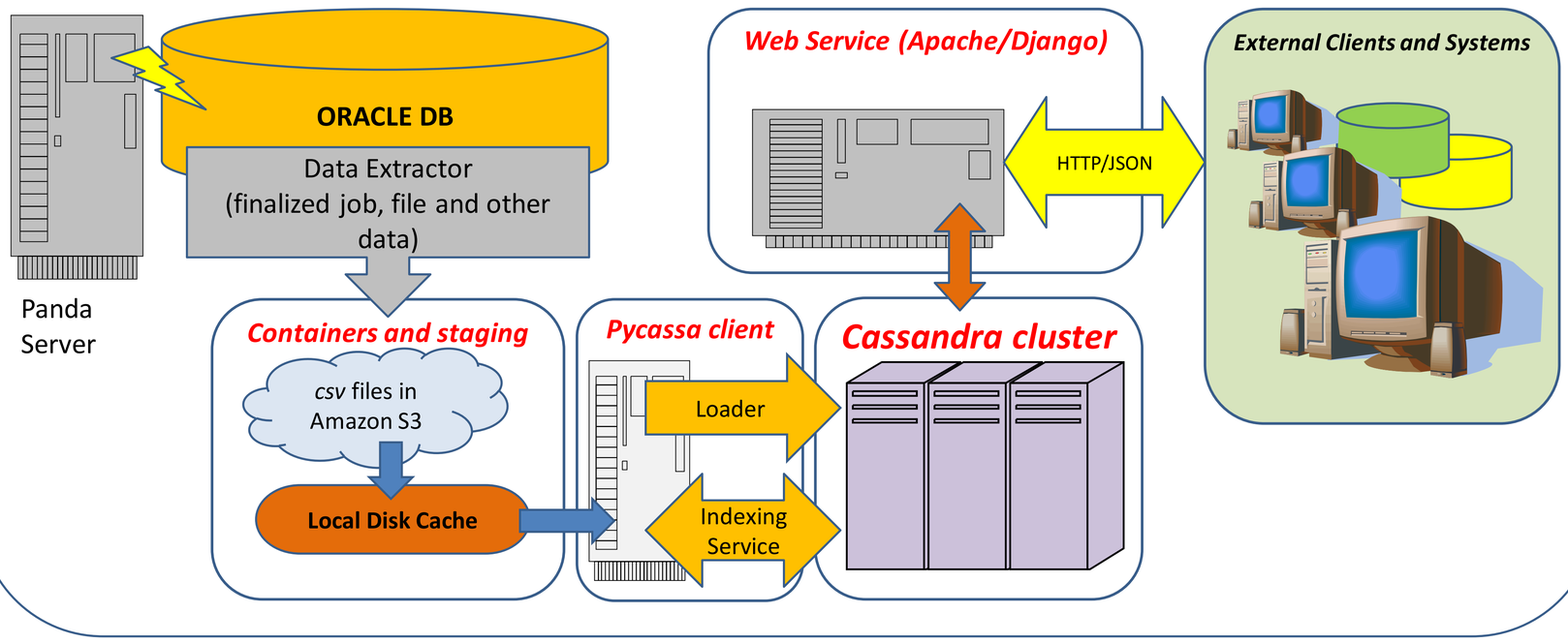
Choice of the noSQL platform

Based on current usage pattern, the criteria for choosing a noSQL platform include:

- Horizontally scalable read performance
- High write performance in order to efficiently absorb data coming from Oracle
- Indexing capability (either built-in or not difficult to implement by hand) roughly matching that in Oracle

There are a number of noSQL solutions that we could use. Based on experience already existing in ATLAS, we chose Cassandra as our platform for monitoring data migration, as it satisfies all of the above requirements.

Architecture of the ATLAS PanDA Monitoring System using noSQL



Features of Cassandra

- Schema-less ("column families" are collections of dictionaries, searchable)
- Ring-like architecture of the cluster; key space divided according to md5 hash; transparent load balancing
- Incrementally and linearly scalable – capacity can be added with no downtime
- Fault-tolerant, with data replication within the cluster as well as cross-data center
- Durability of write operation (once completed, data will survive hardware failure)
- Cache tuning options

Hardware considerations

- Write operation is CPU bound
- Read operation is disk bound
- RAM will help efficient caching of keys and data
- Network speed is important
- Multiple data centers can be used for increased redundancy and fault tolerance
- VM is not optimal for that application since Cassandra is resource-hungry

Implementation of indexes

Two types of indexing solution were tested:

- custom indexes implemented in the application, as distinct column families,(trade ease of maintenance for disk space and performance)
- "native" indexing available in Cassandra since version 0.7 in 2010 (trade disk space for ease of maintenance)

Based on experience, we chose native indexing scheme. To make it work, we augment the data with composite elements, i.e. add columns like PRODUSERNAME+JOBSETID, COMPUTINGSITE+PRODSOURCELABEL+DATE etc

Hardware configuration

Three different configurations were tested in progression:

1. Cluster of 4 VMs at CERN(each with 2 cores, 4GB RAM)
2. Dedicated cluster of 3 servers at BNL (each with 24 cores, 48GB RAM, 6 spindles in RAID0)
3. Dedicated cluster of 3 servers at BNL (each with 24 cores, 48GB RAM, SSD)

Data was loaded either from a disk cache residing on a separate machine, or via an application accessing our repository in Amazon S3

Scalability testing and comparison to Oracle

We have analyzed the pattern of queries done against the Oracle database and used most popular queries for our scalability testing and comparison of our Cassandra cluster to the existing Oracle installation. We find that with configuration utilizing SSD storage, the Cassandra cluster was capable of scaling to at least 3 times the load typically experienced by the Oracle installation. The load was controlled by number of clients hitting the database in parallel and the test was bound by the number of client threads available.

In primary key queries, in terms of plain speed Cassandra consistently outperformed Oracle with 3 to 5ms per query vs 50ms with Oracle. For queries relying in indexes, the Cassandra cluster had a response time of about 4ms vs a spread of 0.5 to 15 ms in comparable Oracle queries.

Conclusions

- Current setup of the BNL Cassandra cluster provides acceptable performance and scaling characteristics for use in monitoring
- We demonstrated flexibility and functionality of composite indexes in Cassandra
- Focus will now shift to building a Web service that would serve JSON data to clients