NoSQL technologies for the CMS Conditions Database

Roland Sipos (CERN) for the CMS Collaboration
Overview

• Intro
  ○ NoSQL
  ○ Conditions Database and motivations

• Candidates
  ○ Options and choices

• Prototypes
  ○ Deployment aspects
  ○ Empirical evaluation and results

• Outro
  ○ Application layer and integration
  ○ Outlook
Intro - CondDB and NoSQL
NoSQL - General

NoSQL in keywords.

- Only a buzzword
  - Meaning: “One size does not fit all!”
- CAP Theorem
- ACID vs BASE
- Different models
  - Doc. store, Key-Value, Column oriented, BigTable

**NoSQL means: “we have options”!**

Not against relational DBs, but a complement to those!
Conditions Database

Alignment and Calibration constants, that records a given “state” of the CMS Detector.

Essential for the analysis and reconstruction of the recorded data.

Also critical for the dataflow and need to be properly re-synchronized during the data processing.

Poster: The CMS Condition Database system (Contr.ID: 130)
CondDB - Details

Conditions are free from:

- Full table scans
  - Only “by key” access
- Joins
- Complex, nested queries
- Transactions
  - Data is written once, and never deleted, altered
- Absolute consistency
  - Only consistency criteria: newly appended data should be available for reads ASAP!
    (in less than few seconds)
Find alternative data storing technologies for the CMS Conditions data for:

- Storing BLOBs
- And it’s meta data
- In a read-heavy environment

Further requirements:

- Durability
- High availability
- (Optional scalability)

Do we really need relational access for such use-case?
source: Tim Gasper - Big Data Right Now: Five trendy open source technologies

**NoSQL - Options**

### Non-Relational
- **Flat, Hierarchical, Network, etc...**
- **Document**
  - Lotus Notes
- **NoSQL**
  - Couchbase
  - Riak
  - Redis
  - Voldemort
  - BerkleyDB
  - Cassandra
  - Accumulo
- **Column oriented**
  - BigTable
  - HyperTable
  - HBase
- **Graph**
  - Neo4j
- **DaaS**
  - SimpleDB
  - App Engine

### Relational
- **Analytic**
  - Hadoop
  - Cloudera
  - SPARK

### NewSQL
- **Brand new**
  - Clustrix
  - VoltDB
  - SnakeSQL
- **RDBS Add-on**
  - ScaleDB
  - MySQL Cluster
  - GenieDB
  - Tokutek
NoSQL - Candidates

How to chose?

**Empirical evaluation:** Check if a given prototype meets the usability and performance criteria from the original solution.

If more of them passes the criteria, choose the best, based on essential features and performance characteristics.
Prototypes - The candidates
Selection

In multiple phases...

Find:
● Showstopper problems (no-go)
● Barely usable (some issues)
● Promising candidates

Preliminary testing.
Candidates

No-go
- HBase (/w HDFS)
  - BLOB size problem.
- CouchDB
  - Drivers
- Hypertable
  - In development
- etc.: app layer needs, CAP characteristics, durability problems.

Promising
- MongoDB
- Cassandra

So-so
- RIAK
  - Query routing!
- (Couchbase)
CustomSamplers

An extension for JMeter, with CMS specific needs, in order to measure the performance of the different databases.

For each candidate the extension has:

- **Deployers**
  - To build up the data model
- **QueryHandlers**
  - Simulate the CMS workflow
- **ConfigElements**
  - Configure persistency objects
- **Samplers**
  - Report to the testplan listeners
Deployment

Automated virtual environments on OpenStack.

○ Personal tenant - biased by user interactions
○ Thanks to the collaboration with CERN IT, the evaluation was made on dedicated resources
○ Also SSD cached vs. disk comparisons were made

Details:

○ No overcommit
○ Instances are “equally” distributed on the hypervisors. (for 5 node: 2-2-1 on 3 hypervisors)
○ 1 GBit NICs (shared between co-hosted VMs)
Results

Increasing request numbers: 1-9 TPS 
(For both remote and single testplans)

● Exploring limits for saturating factors like:
  ○ Network bandwidth
  ○ Access of persistency objects
  ○ Storage elements (Ephemeral disk/SSD, Ceph)

● Scaling out (different cluster setups):
  ○ Node numbers (5 x m1.large, 4 x m1.medium)
  ○ Routing techniques (Round robin, Token-aware)
  ○ Distributed testing (4 JMeter engine)
Plots

Loadosophia - Roland.Sipos@cern.ch
Composite timeline analysis (request time vs. monitoring)
Remarks

● **MongoDB - 10Gen**
  ○ Scaling ✓
  ○ BLOBs ✓
  ○ API ✓ (however… mongos.)

● **Cassandra - Datastax**
  ○ Scaling ✓
  ○ BLOBs ❌ (splitting of large binaries?)
  ○ API ✓

● **RIAK - Basho**
  ○ Scaling ✓
  ○ BLOBs ✓
  ○ API ❌ (token aware routing? C++ driver?)
Outro - Present and future
Application layer

The current implementation of the session layer is extendable with alternative storage backends.

Steps:

● **Handling persistency objects**
  ○ Extending the software framework with NoSQL support

● **Implement the Session interfaces**
  ○ Implementing the “equivalent” CondDB queries

● **Testing**
Integration

- Release validation
- Find differences between the current solution and the prototypes
  - Using real data
  - Real use-cases - using CMSSW

This will be the final performance comparison between different deployments.
Outlook

- Understand and eliminate issues during the release validation
- Fine-tuning critical performance factors
- Formal evaluation and comparison of the different solutions

Long term project!
Not a “by tomorrow” change, but for LS2.
The end

Thank you for your attention!

Any questions are welcome!

From: http://geek-and-poke.com/