Report from CHEP2012

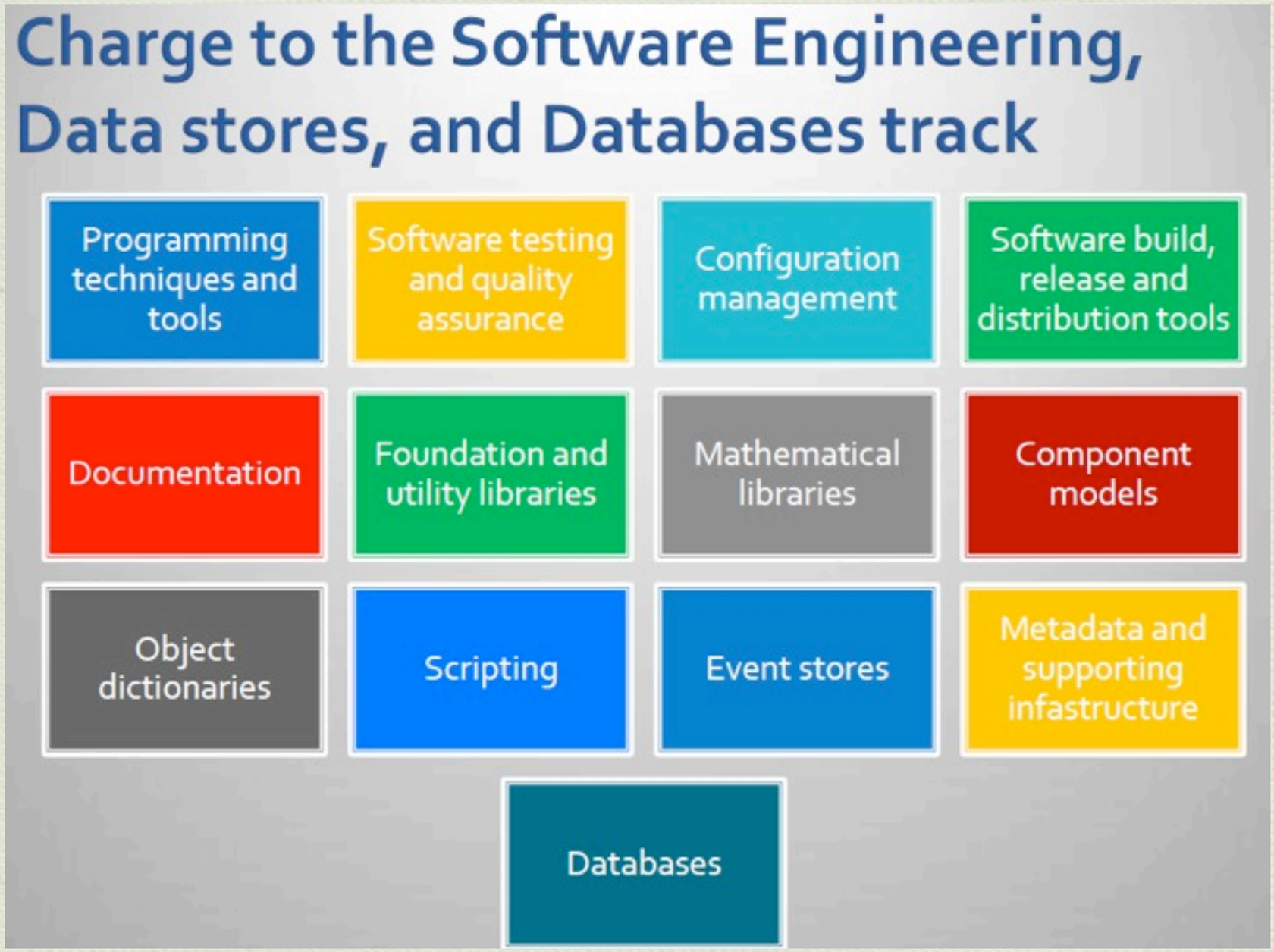
Luca Tomassetti

Summary

- Track on Software Engineering, Data Stores and **Databases**
- Mon. 21/05: 2 sessions
- Tue 22/05: 2 sessions + posters
- Thu 24/05: 2 sessions + posters

Summary

- 26 talks
- 71 posters
-

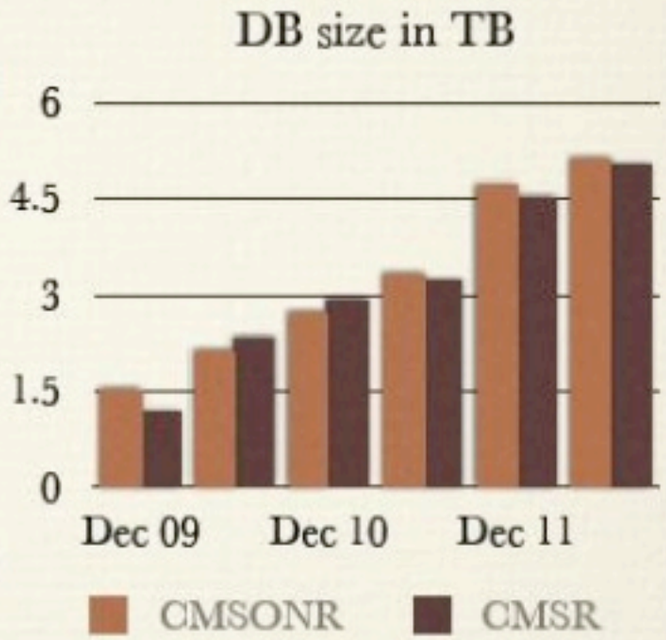


Contributions (talks)

- CMS experience with online and offline Databases
- A Programmatic View of Metadata, Metadata Services, and Metadata Flow in ATLAS
- Comparison of the Frontier Distributed Database Caching System with NoSQL Databases
- ATLAS DDM/DQ2 & NoSQL databases: Use cases and experiences
- LCG Persistency Framework (POOL, CORAL, COOL) - Status and Outlook
- Evolution of grid-wide access to database resident information in ATLAS using Frontier
- Handling of time-critical Conditions Data in the CMS experiment - Experience of the first year of data taking

CMS online & offline dbs

- DB growth about 1.5 TB/yr
  - both online and offline
- Condition data is only a small fraction
  - ~ 300 GB at present
  - growth: + 20 GB/yr
  - about 50 Global Tags created each month



A. Pfeiffer

Smooth operations was a theme of DB session

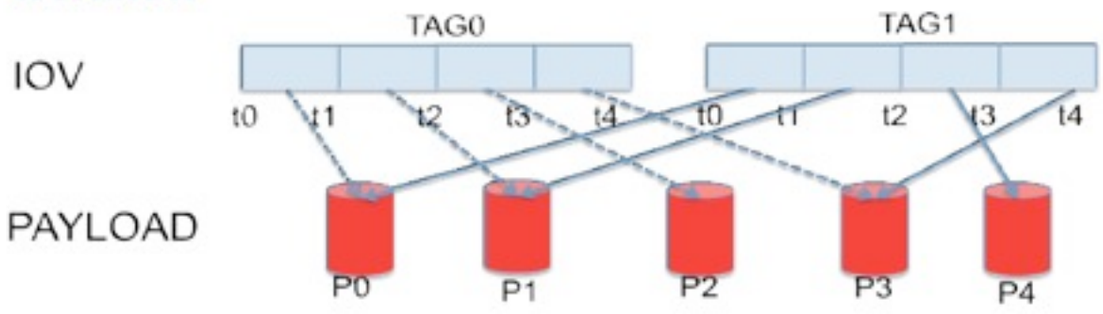
- Very smooth running
  - CMSONR availability: **99.88 %**
    - 10.5 hours downtime overall in 2011
  - CMSR availability: **99.64 %**
    - 30.7 hours downtime overall in 2011
  - SQL query time stable (few msec)

downtime includes all power-cuts, node reboots, hangs, (some) maintenance, ...

CMS conditions

Data model

- Payload : data structure designed according to the detector/task needs.
  - Typically: header + param container(s)
- IOV (Interval Of Validity): array of intervals (time or run number) with their associated Payload references
- Tag: label identifying/categorizing a specific IOV.
- Global Tag: A consistent set of Tags required for a given workflow



Applications: strategy

- Enforce the DB access via a common software
  - Public interface with a single, concrete implementation
- Support of a well defined set of use cases
  - Allow to control the volumes and access patterns
  - Queries are predefined and can be tuned a priori
  - No support for arbitrary query on the IOV or Payloads
- Focus on data integrity
  - IOV updated appending new intervals
  - IOV never deleted
  - Limited manipulation of tags

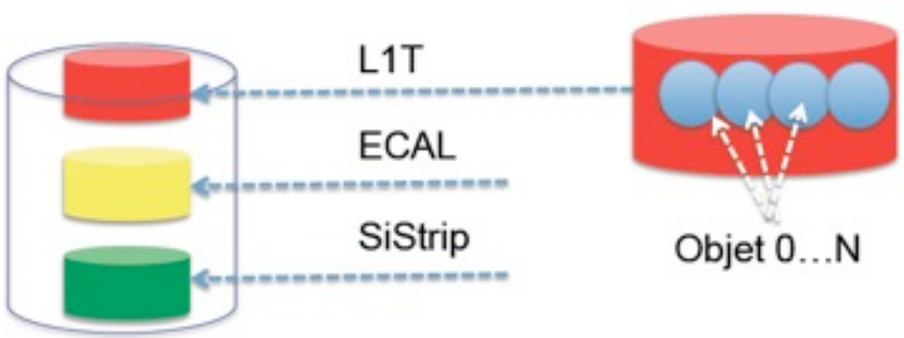
CMS conditions

Storage model

- The storage is based on ORACLE

more details: [163 -

- Data grouped by source (*Detector* or *Task*)
  - Individual schema (ORACLE *account*)
- Object-Relational approach
  - reduces the 'relational' complexity of the schema
  - object instances are mapped to records in their tables
  - blob streaming adopted
    - for large arrays (>200 elements)
    - for multi-parameters or complex structure
    - for files



An useful data set is identified by few queries

all involving 1 table, 1PK

- Resolve the IOV OID from the Tag
- Load the IOV identified by the previous ID
- Load the Payload corresponding to the target time window

Two more queries are required to resolve the mapping Object/ Relational at run time

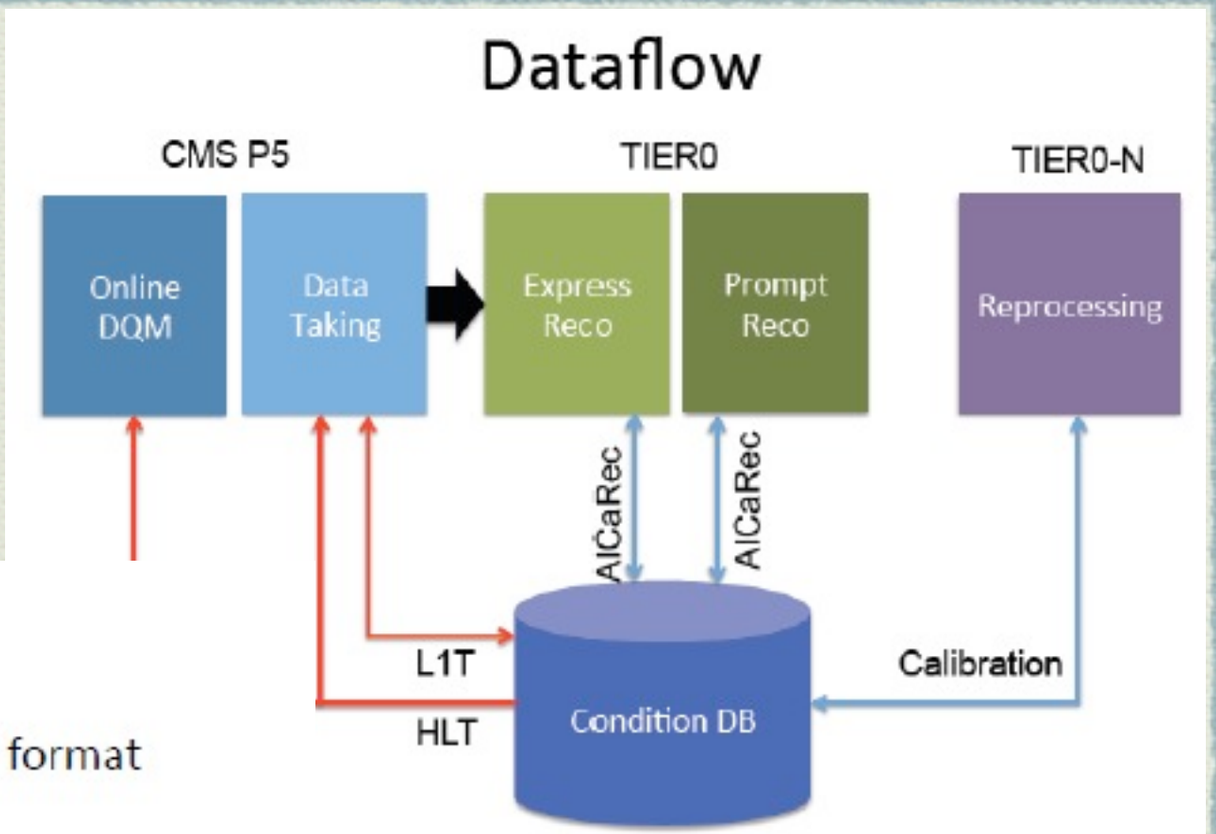
- Queries are simple and well established
- Cursors contain most of the time one row only!

CMS conditions

Reading/Distribution

- Load from the reco jobs at Tier0/1s is potentially high
  - >200 condition objects to read with 3-5 tables => ~800 queries
- Data is read-only for a time scale of 10 mins
- FronTier caches minimize the direct access to ORACLE servers
  - At the price of the latency implied by the refreshing policy
  - 2 Frontier services (P5 and Tier0/1) *more details:[220,D.Dykstra]*
- Snapshot from Oracle DB ensure reproducibility
  - Data set exported in a dedicated server
- SQLite files for simple shipping data through the network
  - Used by the Offline DropBox to export calibration data into ORCON

CMS conditions



Most of the work is currently spent in operation

- Follow-up of data taking and processing needs
- Migration of existing data sets to a new CMS proprietary format

Only little development are still ongoing

Focus of the current phase is consolidation of the (still) critical areas

- Bookkeeping system for the DropBox
- Security for DB access (authentication and authorization)
- Improvements for Monitoring System
- Handling of schema evolution for Blob-based storage

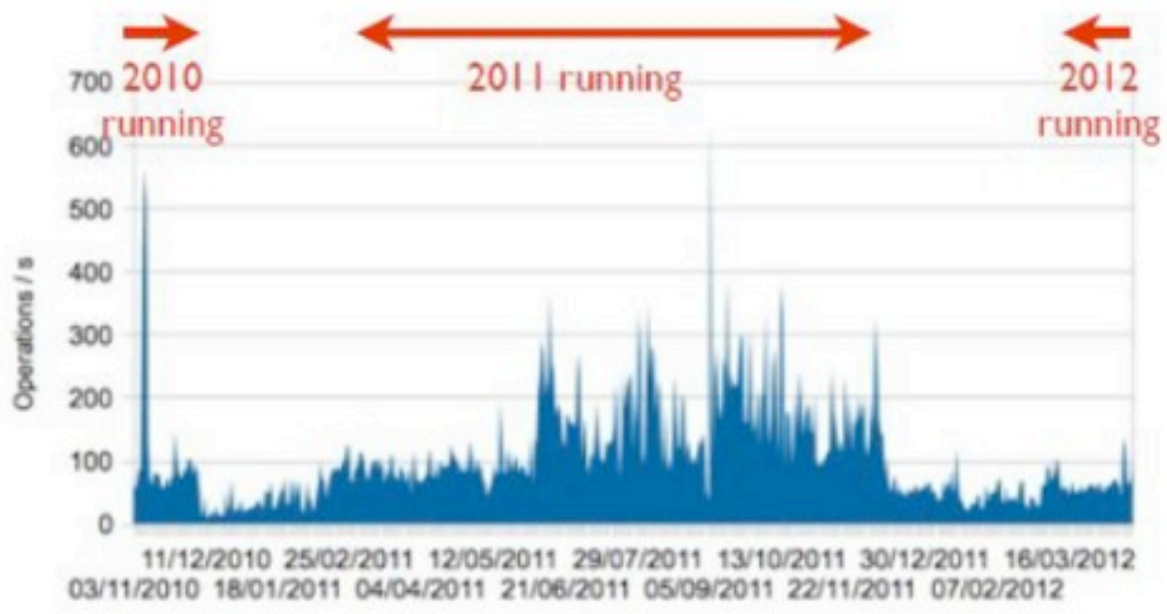
No major changes are foreseen in the system for 2 years

Atlas conditions switch to Frontier successful

A. Dewhurst

- Tier 0 access switched to Frontier from direct access for the start of 2012 data taking

Average number of operations /s per day for the RAL database



ATLAS metadata

Metadata access even before a simple task is run …

- Input dataset selection may have been based upon physics metadata
  - principal ATLAS repository is AMI
- Decisions about what to process and which auxiliary data to use to process it may have been made by consulting COMA
  - e.g., for trigger configurations and conditions data tags
- Data quality decisions about which subsets of which runs are appropriate to use for physics purposes will have been made and reflected in so-called good run lists

Pre-execution metadata access is not just about input datasets and auxiliary data

- Which software versions? What is an appropriate job configuration?
- For production, encoded in a configuration tag that is recorded both in AMI and in the output event data products themselves
- Example of a configuration tag: r2713_p705, which encodes information about

ATLAS metadata

Next? Job initialization is metadata intensive

- Algorithms and auxiliary services require access to non-event data in order to initialize themselves to process event data appropriately
- Some of this information comes from the configuration provided by the job transform itself or specified by the accompanying configuration tag
- BUT … some will depend specifically upon the input dataset.
- Such information may in principle be dataset-level metadata that could, for example, be extracted exactly once from AMI at task definition time and propagated to all jobs.
- Typical current practice in ATLAS, though, is to "peek" into metadata contained in input files
  - Possibly multiple times(!)
- First for the purpose of job options configuration (before a reconstruction step begins, for example)
- Second for service and algorithm initialization within the reconstruction proper

ATLAS metadata

Metadata access during job execution

- Metadata such as detector conditions and calibrations that may vary over the course of a run may be accessed dynamically during event processing
  - either directly from databases or from caches derived and refreshed from such databases.
- There are input metadata transitions as well, arising, for example, when a job processes a sequence of input files.
- In this case the event loop and physics code do not care that a new file was opened, but certain auxiliary metadata may need to be updated or refreshed or flushed or accumulated or merged

Metadata propagation from input to output

- Certain metadata may be propagated from input to output, to be stored either in output data files or in metadata repositories such as AMI.
- Metadata may be created as well, and must be propagated in similar ways.
- Event counts, particularly when filtering is done, are a fundamental but important special case—more on this later
- In any case, machinery must be provided to emit metadata and to transport them from the producing job through the production system infrastructure to appropriate repositories

ATLAS metadata

Conclusions and future directions

- Metadata are integral to every aspect of ATLAS computing
- The intent of this presentation has been to provide an illustrative view of ATLAS metadata, principally from the point of view of the infrastructure and services needed for metadata flow in the context of a single task
- While metadata components and infrastructure have grown organically as the experiment has matured, a number of principles described herein have informed their design and connectivity
- The infrastructure continues to evolve in a variety of ways, with improvements planned
  - to how dataset-level metadata may be used to reduce the need for peeking into input files,
  - to how metadata are emitted and transported from executing jobs to the collaboration's metadata repositories,
  - to machinery for robust accounting of low-rate error conditions in physics data bookkeeping

ATLAS & NoSQL

- So what is NoSQL, pardon, structured storage about?
  - 1. ***Non-relational modelling and storage of data***
    - Use the native data layout of an application
  - 2. ***Linear scalability of data processing***
    - Scalability ≠ Performance

Use cases considered:

1. Log file aggregation
2. Trace mining
3. HTTP cache for dataset downloads

- Structured storage systems are too useful to be ignored
- Hadoop proved to be the correct choice and an excellent platform fc analytical workloads
  - Stable – reliable – fast – easy to work with
  - Survived disastrous hardware failures
- DDM use cases well covered
  - Storage facility (*log aggregation, traces, web sharing*)
  - Data processing (*trace mining, accounting, searching*)
- Miscellaneous
  - All three evaluated products provide full durability, and transactions were
  - We see Hadoop complementary to RDBMS, not as a replacement

- Within one year we had
  - 5 disk failures
    - 20% failure rate!
    - Out of which 3 happened at the same time
  - 1 Mainboard failure
    - Together with the disk failure, but another node
- Worst case scenario experienced up to now
  - 4 nodes out of 12 dead within a few minutes
  - Hadoop
    - Reported erroneous nodes
    - Blacklisted them
    - And resynced the remaining ones
  - No manual intervention necessary
  - Nothing was lost

Frontier vs. NoSQL

	Frontier	NoSQL in general
DB structure	Row/column	Nested key/value
Consistency	ACID DB, eventual reads	Eventual
Write model	Central writing	Distributed writing
Read model	Many readers same data	Read many different data
Data model	Central data, cache on demand	Distributed data, copies
Distributed elements	General purpose	Special purpose

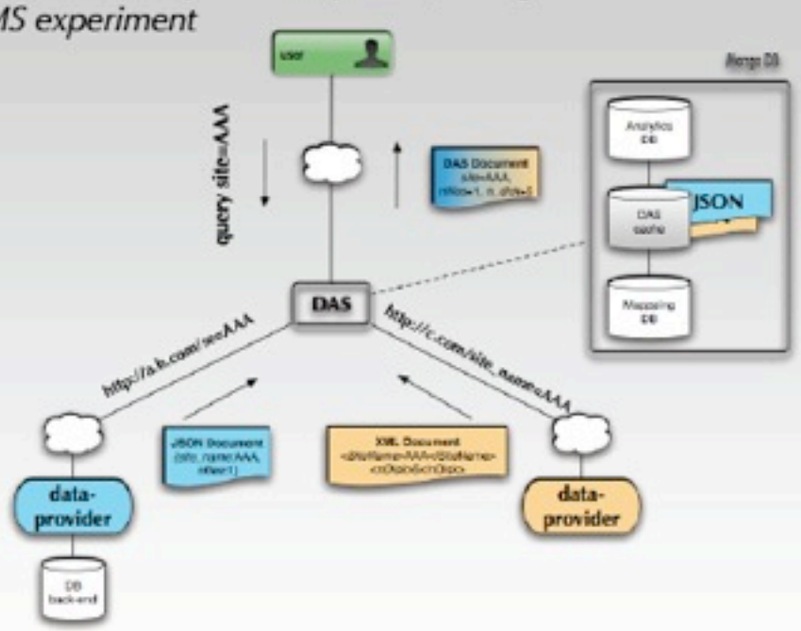
	MongoDB	CouchDB	HBase	Cassandra	Frontier
Stored data format	JSON	JSON	Arbitrary	Arbitrary	SQL types
Flexible queries	Yes	No	No	No	Yes
Distributed write	No	Yes	No	Yes	No
Handles Slashdot Effect	No	Yes, best w/squid	If scaled sufficiently	If scaled sufficiently	Yes
Does well with many reads of different data	Yes	Yes	Yes	Yes	No
RESTful interface	No	Yes	Add-on	No	Yes
Consistency	Eventual	ACID DB, eventual read	Mixed	Tunable	ACID DB, eventual read
Distributed MapReduce	No	No	Yes	Add-on	No
Replication	Few copies	Everything	Tunable	Tunable	Caching

- NoSQL databases have a wide variety of characteristics, including scalability
- Frontier+Squid easily & efficiently adds some of the same scalability to relational databases when there are many readers of the same data
  - Also enables clients to be geographically distant
- CouchDB with REST can have same scalability
- Hadoop HBase has most potential for big apps
- There are good applications in HEP for many different Database Management Systems

Use cases NoSQL in CMS

DAS w/ MongoDB

An intelligent cache in front of CMS data-services; fetch and aggregate data on demand upon user queries; next generation of data discovery in CMS experiment



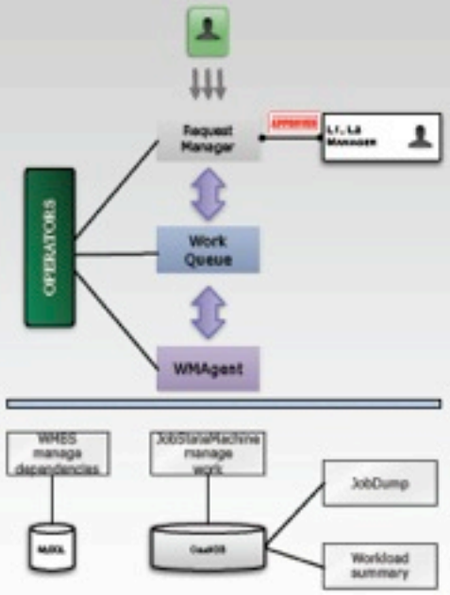
- MongoDB is schema-less document oriented database
  - documents stored as binary JSON; read/write operations are very fast due to memory-mapped files
  - it supports native drivers; multiple indexing; data collections; in-place updates
- Document based queries (on par w/ SQL)
  - Flexible query language; map-reduce; aggregation
- Horizontal scaling via replication and sharding
  - mirrors via LANs and WANs
- Open source; native support for different OSes
- DAS uses MongoDB as cache storage
  - documents from different data-services can be stored and queried together (similar to federated database but does not require a schema)
  - we achieved 20k/7k docs per second for read/write I/O

The DAS system should provide a search interface across distributed relational and non-relational data-services and aggregate information from them. It must operate as a dynamic cache in front of existing infrastructure and hides complexity of their data access.

Use cases NoSQL in CMS

WMAgent w/ CouchDB

Data and Workflow management tool for job submission and execution engine; dispatch and manage CMS jobs



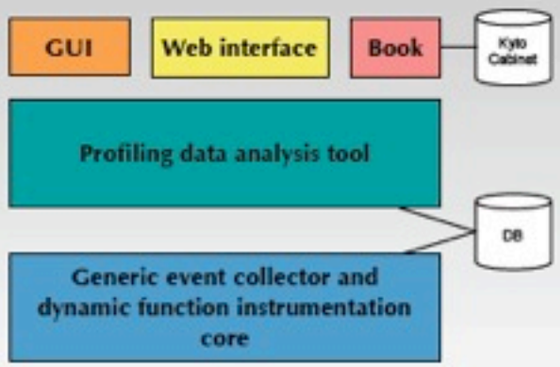
- Effective key-value store; data in JSON
- MySQL stores job definitions and dependencies, while CouchDB handles job progress, job summaries and output reports
- RESTful HTTP API - in common w/ service we write, no need to maintain DAO's
- Limited relationships between data, map-reduce data look-up is sufficient
  - incremental index building maintains performance
- Replication is built in and very simple
- Back-up in CouchDB is simple due to append only file format
  - can either replicate DB to another node or write DB file to CASTOR
- CouchDB is written in Erlang: high concurrency is natively supported
  - open source; clustering solution exists; commercial support is available

Metadata of MonteCarlo job management. A use case very similar to our bookkeeping db

Use cases NoSQL in CMS

IgProf w/ KyotoCabinet

Main tool for performance tuning of CMS software (core framework)



- O(100) profiles/build, O(100M) of keys
- IgProf uses SQLite and KyotoCabinet
  - SQLite to store build profiles
  - Kyoto to analyze profile results (compare multiple one)
- Kyoto is a library of routines for managing a database
  - choose your DB type based on you app
- It is key-value store
  - very fast, elapsed time to store/search 1M records ~1 sec
  - multi-thread safe, supports transactions and ACID properties
  - written in C++; provides APIs for different languages

Conclusions

- Oracle + Frontier is a working solution for LHC experiment, proved to be reliable
- Usage of NoSQL databases limited to special tasks, mainly aggregation, analytics and accounting
- R&D activity on data models should be worthwhile to explore new solutions for conditions db