CMS Offline experience with NoSQL data stores

Valentin Kuznetsov, Cornell University, on behalf of CMS collaboration

Database Futures Workshop, CERN, June 2011

Outlines

- CMS experiment and its software
- NoSQL technologies used in CMS
  - Data Aggregation System (MongoDB)
  - WMAgent (CouchDB/MySQL)
  - IgProf (KyotoCabinet/SQLite)

CMS software

- CMS software has been around for last 10 years
  - C++ (framework), python (web && data management), Java (web && data services)
- Major data-services, such as PhEDEx, Data Bookkeeping System, Run Summary are based on ORACLE back-end
  - MySQL and SQLite has been used during development phase
- NoSQL solutions have started being used in the last 2 years

Projects based on NoSQL

- Data Aggregation System
  - An intelligent cache in front of CMS data-services; fetch and aggregate data on demand upon user queries; next generation of data discovery in experiment
- WMAgent
  - Data and Workflow management tool for job submission and execution engine; dispatch and manage jobs
- IgProf
  - Main tool for performance tuning of CMS software (core framework)

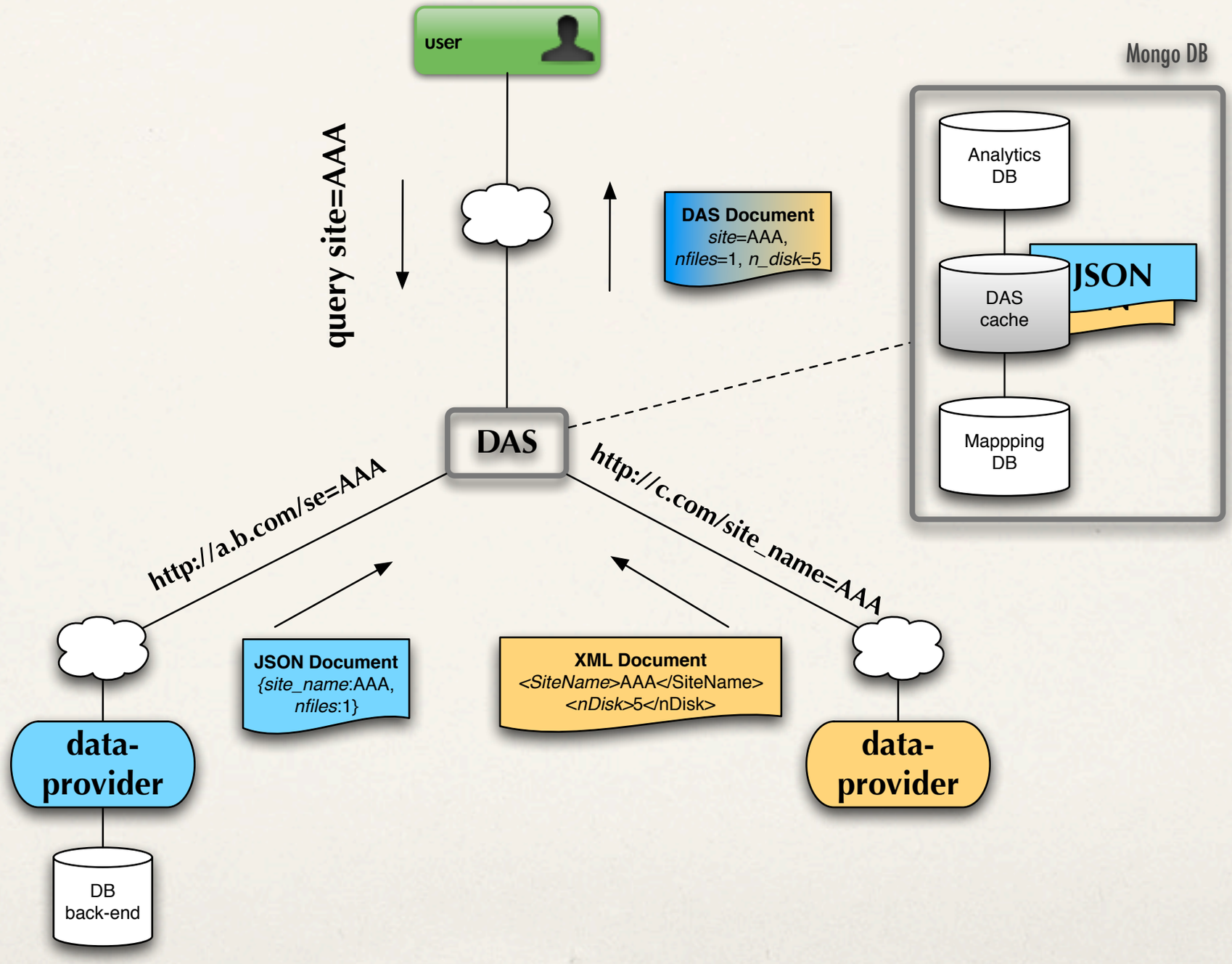
DAS & MongoDB

- **Requirements**
  - Fetch meta-data from distributed data-services and allow precise queries to discover CMS data
  - Fast read/write; store/aggregate unstructured documents; Query Language; simple to scale horizontally
  - Be data agnostic
- **Implementation**
  - Python web framework (CherryPy) and MongoDB back-end

Why MongoDB?

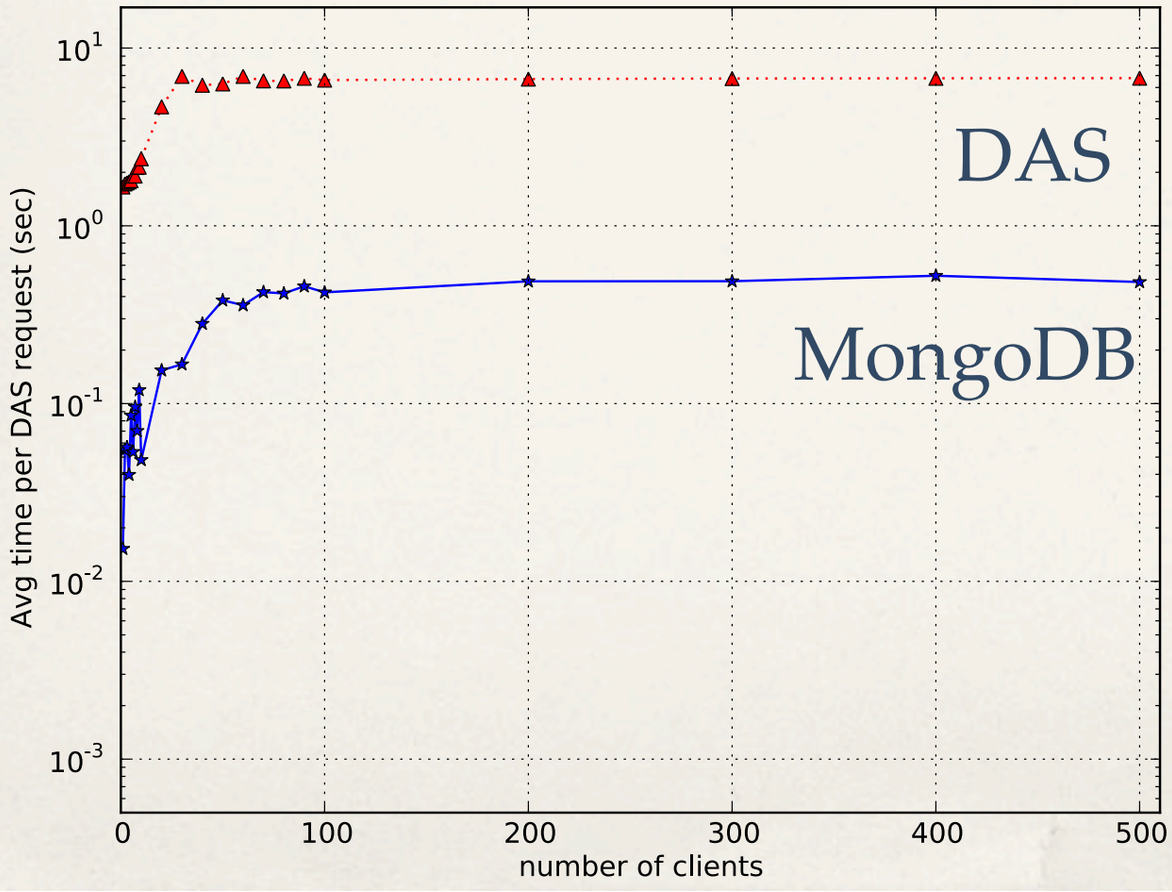
- Fast database with plenty of native language drivers
- Storage of JSON documents via BSON (binary JSON)
- Simple querying via flexible Query Language (on par w/ SQL)
- Support multiple indexing
- Data organized in collections (a la databases)
- Replication and sharding
- Open source, commercial support via http://www.10gen.com/support

DAS & MongoDB, cont'd



DAS & MongoDB, cont'd

- Currently DAS runs on a single node (8 CPU, 16GB RAM) together with other services
  - 6K docs/sec for raw cache population; 7.5K docs/sec for reading/writing records to disk; 20K docs/sec read access rate



Read performance using 500 clients to look-up random records in DAS populated with 50M documents from PhEDEx/DBS data-services (100x of current statistics).

DAS & MongoDB, cont'd

- DAS holds no unique data, can be repopulated from scratch at any time (MongoDB uses memory mapped files).
- Its performance based on fitting DB index in RAM
  - Excellent read and moderate write performance
- MongoDB supports sharding and replica sets
  - Replication is designed for redundancy and easy to setup
  - DAS development version can rely on it; data spread across shards allowing horizontal scale (not yet in production); tested with 6 CERN VM and 5M documents

MongoDB experience

- We are in commissioning phase and our experience is limited
- MongoDB serves the job and fits nicely into DAS architecture
- DAS is capable to aggregate information from dozen of CMS data-services, whose size above 100GB
  - Current cache size around 40GB
  - Users slowly migrated, expecting to finish by end summer
- No operational issues (yet)
- Data sharding and horizontal scaling our next target

WMAgent & CouchDB

- **Requirements**
  - Jobs submission to/across all CMS tier centers, monitoring, management. Low insert/read rate; job look-up via job ID. Job submission should be supported in distributed environment and provides durability with respect to system failures.
- **Implementation**
  - Python, MySQL and CouchDB

Why CouchDB?

- Effective key-value store; data in JSON
- RESTful HTTP API - in common w/ service we write, no needs to maintain DAO's
- Limited relationships between data, map/reduce data look-up is sufficient our uses
  - Incremental index building maintains performance
- Replication is built in and very simple
- Back-up is simple due to append only file format
  - Can either replicate DB to another node or write DB file to CASTOR

Why CouchDB, cont'd

- CouchDB is written in Erlang: high concurrency natively supported
- Apache open source project
- Clustering solution exists (BigCouch), but not yet used in our environment
- Commercial support is available via http://www.cloudant.com and http://www.couchbase.com

WMAgent & CouchDB, cont'd

- WMAgent is designed to support high load job submission and monitoring:

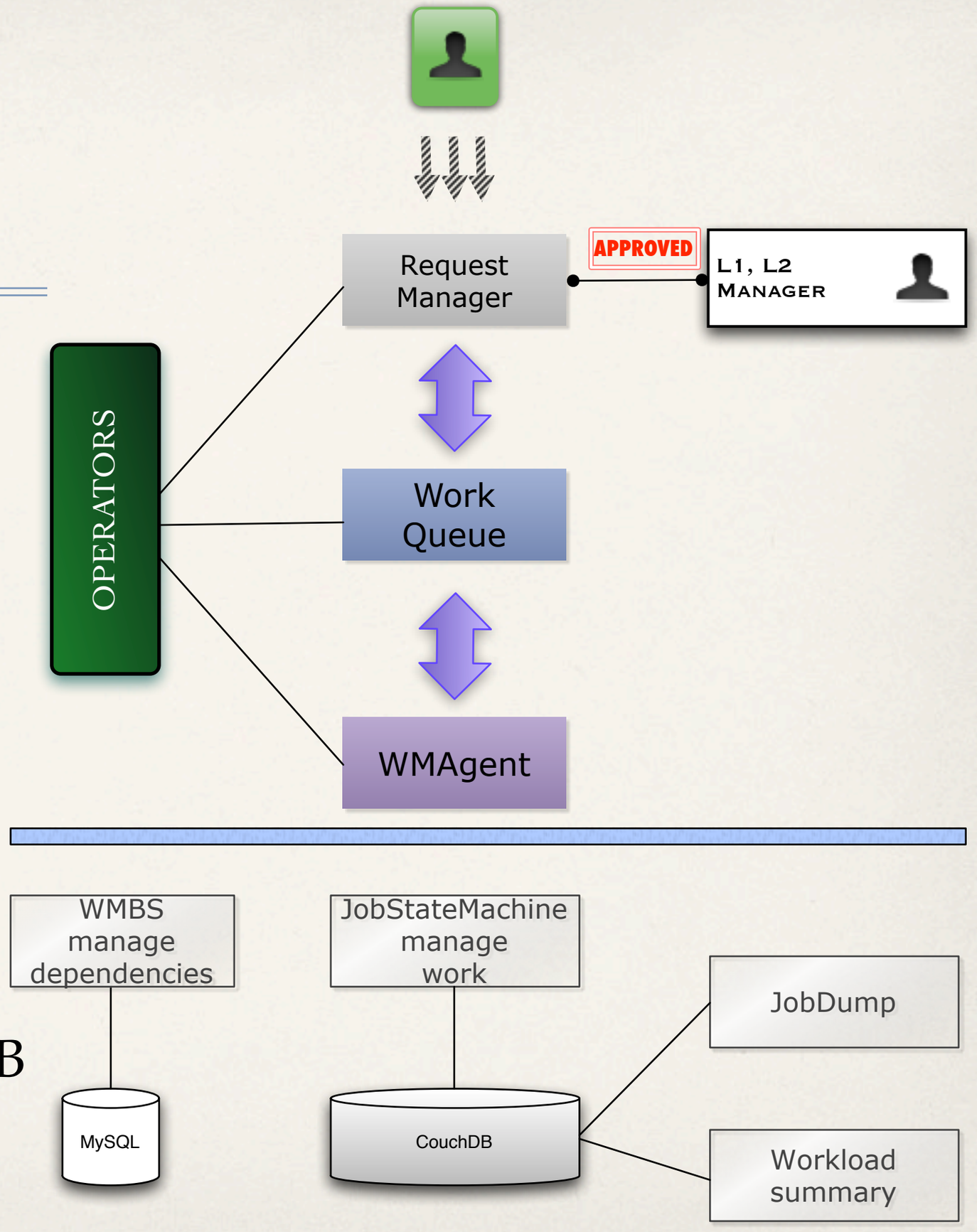
- 3K jobs @ Tier-0, 10K jobs @ Tier-1's, 15K simulation jobs @ Tier-2's and 15K analysis jobs @ Tier-2's

- Jobs submission can be done by different teams at different geo locations

- Expect to have several WMAgent instances running, each handling ~50 jobs/second (limited by grid middleware and data access obstacles)

WMAgent

- WMBS/MySQL job definitions and dependencies
- JobStateMachine/CouchDB keeps jobs progress
- JobDump/CouchDB job output reports
- WorkloadSummary/CouchDB job summaries



MySQL & CouchDB usage

MySQL

- Contains relational data, e.g. job definitions, input/output file mappings
- Job states: created, running, completed
- Used for strongly correlated information, e.g. file N processed by job J, produces file M

CouchDB

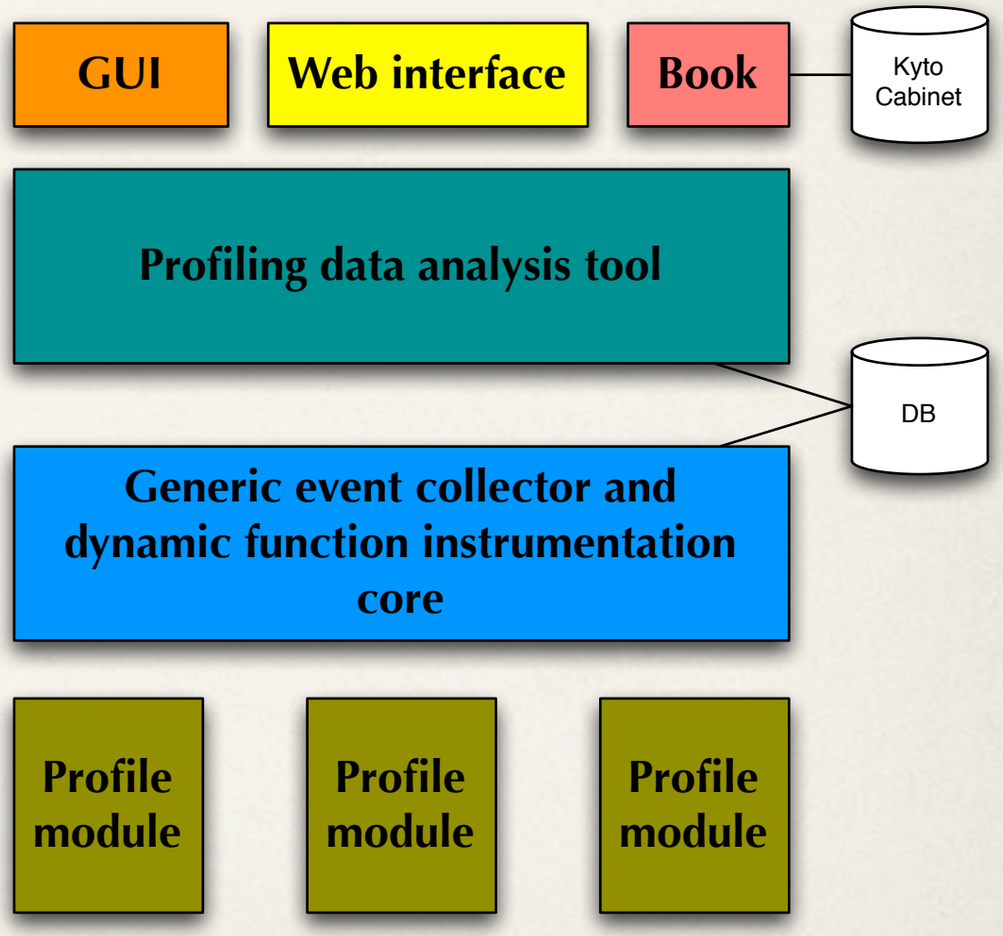
- Contains non-structural data, e.g. meta information about jobs as they progress
- State change times, completion reports
- Aggregated monitoring information
- Reduce & sum over time

WMAgent performance

- Under commissioning phase now
- Single Agent is capable of high rate of jobs: ~10Hz
- For Tier1 tests
  - 10-20K job executing in agent, 7500 batch slots
  - MySQL: few MB, 20M rows
  - CouchDB: ~6GB JobDump docs, 12GB in views

IgProf & KyotoCabinet

- IgProf (Ignominous Profiler) is a stand-alone tool for measuring and analyzing application memory and performance characteristics
- O(100) profiles/build, O(100M) of keys into simple DB
- IgProf uses SQLite and KyotoCabinet
  - SQLite to store build profiles
  - Kyoto to analyze profile results (compare multiple one)



Why KyotoCabinet?

- KyotoCabinet is a library of routines for managing a database
  - Choose your DB type depending on your app (HashDB, DirDB, etc)
- The database is a simple data file containing records, each is a pair of a key and a value
- Runs very fast, elapsed time to store/search 1M records ~1 sec for hash or B+ tree databases
  - Multi-thread safe, supports transaction and ACID properties
- Written in C++ and provides API for C/C++, Python, Ruby, Java

Dark side of the moon

MongoDB	CouchDB	KyotoCabinet
✤ performance greatly degrades if indexes are not fit in RAM ✤ corruption issues (in sharding environment) were reported in a past on MongoDB forum	✤ data is uncompressed (requested feature) and there is no automated data/ index compression ✤ map-reduce views can be large and slow if poorly constructed ✤ error messages can be opaque (erlang is not user friendly)	✤ quite new in a “marker” ✤ developed and supported by couple of individuals ✤ crashes and corrupted data are reported (but rare)

Summary

- Usage of NoSQL is only started in CMS
  - So far it is driven by use cases
- We carefully evaluated existing technologies (including RDMS ones) and pick up NoSQL solutions to fit our application requirements
- The NoSQL solutions represent complementary stack of software which co-exist with RDMS naturally