The archive solution for distributed CMS WMAgent

Valentin Kuznetsov, Cornell University
Collaboration: 3800 people, 199 institutions, 43 countries

During 2017:
- processed 30 B raw events
- produced 16 B MC events
- transferred 4 PB/week with average transfer rates 2-6 GB/s
- deleted 85 PB (T1)/169 PB (T2) of least popular datasets
- replicated 20 PB (T1)/80 PB (T2) of most popular datasets

<table>
<thead>
<tr>
<th></th>
<th>T0</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk usage</td>
<td>21 PB</td>
<td>39 PB</td>
<td>54 PB</td>
</tr>
<tr>
<td>Tape usage</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU usage</td>
<td>326</td>
<td>425</td>
<td>1133</td>
</tr>
<tr>
<td>Databases</td>
<td>ORACLE, CouchDB, MongoDB, ...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Technologies</td>
<td>GRID, Cloud, XrootD, HDFS, Spark, ...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>C++, Python, C, Perl, Fortrans, Shell, Java, Go, ...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CMSSW: 190K commits, 1800 releases, 16M lines of code
CMS Data Management

Dynamo is a dynamic data-placement system moving PB of data among CMS sites.

PhEDEx is a CMS data-transfer management system.

Data Bookkeeping System is CMS main data-catalog for storing dataset, block, files, runs, lumis meta-data.

Data Aggregation System simplify user search queries across CMS data-services.

CRAB is a utility to submit analysis jobs to distributed computing resources.

Workflow Manager Agents responsible for splitting work jobs into chunks and sending them to CMS Global pool (HTCondor).

Request Manager is a main catalog for CMS production requests.

Monte Carlo Management system handles all MC requests.
CMS Workflow Management System

Central Production System

Jobs are distributed across GRID sites

WMAgents send jobs to process MC requests

Distributed WMAgents pull down work from WorkQueue

Global pool HTCondor

WorkQueue

Request Manager

Site A

Site B

Site C

Site D

All information about job are stored into FrameWork JobReport (FWJR)

WMAgents aggregate 500K docs/day

Job info are stored FrameWork JobReports
Requirements

- ~300K docs/day (10KB each), 3GB/day, 2TB/year
- Flexible schema and ability to extend it over time
  - unstructured JSON nested documents
- Flexible queries to look-up desired information
- Data aggregated across multiple metrics
- Web monitoring interface for job processing trends
- Have minimal impact on existing CMS infrastructure
Choices

- We decided to use **non-relational** data stores
  - Short-Term Storage is used to accumulate incoming data as fast as possible by storing them into document oriented MongoDB
  - Long-Term Storage is used to store data on HDFS file system
- We used **JSON** data-format for STS and **Avro** data-format for LTS
  - data consumed in JSON data-format, i.e. no changes to CMS codebase
  - data injected into HDFS in Avro (row-wise) data-format: schema evolution, language agnostic, compressible, append-able,
- We defined WMArchive **schema** upfront and convert data from STS to avto-data format before storing it on HDFS
- Separate data accumulation from data migration and clean-up procedure
- Interact with CMS DMWM stack via **RESTful** APIs
**Short-Term Storage (STS)**
- POST HTTP
- REST server
- MongoDB

**Migration STS->LTS**
- FWJR
- Avro file
- Local Disk
- Cronjob: collect docs from MongoDB
- 50K FWJR per single Avro file

**Long-Term Storage (LTS)**
- Cronjob: push Avro files to HDFS

**CERN MONIT**
- Data sent via AMQ
- Aggregation script
- Avro in JSON out
- Spark nodes

**Cronjob:** aggregate docs

Separate aggregation Pipeline
Data look-up

- For STS we rely on Mongo QL which supports reach syntax (query by value, patterns, value look-up in a lists, etc.). Here is an example of its syntax:

  ```json
  {"query": {"Job":re.compile(r"[a-z]+", "X.Y.Z":{"$in":[1,2,3]}, ...}}
  ```

- For LTS we rely on HDFS+Spark and Map-Reduce paradigm

  - user provide business logic to search or aggregate the data, we wrap it up into Python Spark job

- Large data volume can be processed relatively fast:

  - search results across one day of data in $O(10)$ sec, one month of data in $O(100)$ sec
Benchmarks

- **STS**: data injection rate 2KHz
  - 1.5M documents translates into 15GB database size with 3.5 GB of index size

- **LTS**: data look-up via Spark job
  - 1 day of data (200K docs) needs 1min, 2 month of data (12M docs) needs 1 hour of processing time

- Single doc compression: JSON (25KB) ⇒ BSON (16KB) ⇒ Avro (7KB) ⇒ Avro.gz (1KB)
  - Multi-doc compression (use 10K docs): JSON (250MB) ⇒ BSON (160MB) ⇒ Avro (70MB) ⇒ Avro.bz2 (352KB)

- Final choice we store about 50-60K docs per single Avro to fit into 256MB block file constrain on HDFS
Current status

- The WMArchive system in production more than two year
  - one production and one testbed CERN VM (12 cores, 24GB RAM each)
- The data injection comes from 7 production WMAgent and 12 CRAB schedd nodes
- STS holds 3 months of data (tune-able parameter)
- We split STS/LTS into FWJR/CRAB collections
  - STS holds 2 separate collections for incoming docs and 2 collections for daily/hourly aggregated stats
    - each document has an internal state to indicate life-time of it in STS
- STS to LTS migration is done separately upon block completion (1 block contains ~60K docs and has 256MB size)
WMArchive data rate

- 7 production agents
- injection 24/7
  - 100k-1M docs per day
- docs migrated from STS to LTS once a day
- ~60k FWJR records per single (256MB) AVRO file
- Up-to-date we have 350M docs on HDFS (total size ~4TB)
Use cases

- WMArchive is used on daily basis by data-ops to identify problems with running workflows
  - identify failed workflow
  - consult dashboard for problematic site
  - identify issues by log look-up and exit codes
- Monitoring CMS production status
  - sites, campaigns, throughput metrics
- Data aggregation use-cases
Custom UI was designed to address data-ops needs for fine-grained queries:

- job state evolution
- event throughput
- exit codes and states
- workflow monitoring
- CPU, Storage, Memory metrics

CERN MONIT dashboards provide global views of time series metrics.
Custom cuts

Scope
Matches 158,16k jobs
from Mar 7, 2018 1:00 AM to
Apr 2, 2018 4:00 PM.
Collections
WMAgent: daily
WMAgent: hourly
CRAB: daily
CRAB: hourly

WORKFLOW
Filter by Workflow...

SITE
T2_US*

JOB TYPE
Run2017G

HOST
Filter by Host...

ACQUISITION ERA
Filter by Job Type...

EXIT CODE
Filter by Exit Code...

EXIT STEP
Filter by Exit Step...

TIMEFRAME
03/03/2018 - 04/02/20

Custom views

Metrics
Job State
CPU
Total Job Time
Total Job CPU
Min. Event Time
Max. Event Time
Avg. Event Time
Event Throughput
Total Loop CPU
Storage
Write
Read
Read Percentage ROOT
Read Rate
Read Operations

Summary
99.82% overall success rate

Job State Evolution

Event Throughput Evolution
Example

- Find log files in LTS for specific job/LFN while investigating failing workflows
  - very cumbersome operation and require multi-pass operation look-up in WMArchive document store
  - file resolution (which file belong to which processing chain step)
  - look-up log archive and log collect steps
  - input/output file matching

- User provides a JSON file with input parameters
  ```json
  {"spec" : { "lfn" : "file.root", "timerange" : [20180502, 20180520] } }
  ```

- Run spark job to process O(M) documents, data-look-up time ~ few minutes
  - we provide custom Map-Reduce code to perform this task efficiently on Spark platform
  - results show location of tar-ball on EOS
Summary

- WMArchive consists of loosely coupled layers for meta-data storage and archiving
  - we used different technologies to accommodate high-injection rate, schema evolution, large data-volume, flexible QL and search capabilities
  - custom UI along with global dashboards satisfies data-ops needs
  - in 2 years we accumulated 300M docs and will hit 1B in HL-LHC era
  - we didn’t experience any issues during production operation and run service on a single node
- WMArchive opens up possibilities to study users patterns and predict users behavior
  - it is part of larger effort in CMS to study resource utilization, see more in *Gaining Insight From Large Data Volumes with Ease* poster by Valentin Kuznetsov