Hadoop in ATLAS

Mario.Lassnig@cern.ch

Overview

- ATLAS Distributed Computing has two main ways of database interactions
  - Transactional (PanDA Server, Rucio Server, DaTRI, …): real-time interaction
  - Analytical (PandaMon, BigPandaMon, Rucio UI, ADC Mon, …): monitor, report, and summary generation
- Transactional part well covered by CERN IT Database Group
  - With support from different other sections
  - Dedicated DBAs or DB developers in the experiments that support application developers
- Analytical part itself is also split into real-time monitoring and report generation
  - What's happening now? – covered the same way as the transactional part
  - What happened in the past *n* months? – collection of ingenuous hacks to not brake the transactional part
- Report and summary generation generally orthogonal to transactional performance

History

- Original problem that jumpstarted the use of Hadoop
  - Accounting of datasets in ATLAS DDM system with metadata, 2010
  - Put in production in 2011

- Relational database failed at this use case
- Three main problems
  - Orthogonal queries – optimisations for one report wrecked performance of another
  - Latency and contention – report generation interfered with the transactional part
  - Arbitrary string searches on wide columns – wrecks database CPU
- One minor problem
  - Temporary storage required for whole operation

- Technology evaluation of other technologies that were closer to requirements
  - Hadoop Hbase – Key/value store
  - Apache Cassandra – Key/value store
  - MongoDB – JSON store

Technology evaluation

- How to get data from relational database into non-relational database?
  - All open questions – industry was not doing that (non-relational from the get go)
  - Efficiently? – need a large enough setup of non-relational database (horizontal scaling)
  - Reliably? – we cannot afford to lose a high percentage of data (eventual consistency)

- DDM team custom-built a 12-node cluster in CERN IT data centre

	Cluster configuration
Nodes	12
Architecture	Linux x86_64
CPU Cores	96 (Intel Xeon 2.26 GHz, 8/node)
RAM	288 GB (24/node)
Storage	–
Storage Network	–
Disk	24 SATA (1TB each, 2/node)
Cache	–
Network	1 GigE

mongoDB

```
{
 _id: 'Main Account User',
 groups: ['group_a',
          'group_b',
          'group_c'],
 selections: {
              'select_a': 123,
              'select_b': abc
             }
}
```

Cassandra

```
'Groups' : {
    'Main Account User' : {
        'groups' : ['group_a',
                    'group_b',
                    'group_c']
    }
}

'Selections' : {
    'Main Account User' : {
        'select_a' : 123,
        'select_b' : abc
    }
}
```

APACHE HBASE

```
'Main Account User' : {
    'Groups' : {
        'groups' : ['group_a',
                    'group_b',
                    'group_c']
    },
    'Selections' : {
        'select_a' : 123,
        'select_b' : abc
    }
}
```

Technology evaluation

	MongoDB	Cassandra	Hadoop/HBase CDH4
Installation/ Configuration	Download, unpack, run	Download, unpack, configure, run	Distribution, Complex config
Buffered read 256	250'000/sec	180'000/sec	180'000/sec
Random read 256	20'000/sec	20'000/sec	20'000/sec
Relaxed write 256	10'000/sec	19'000/sec	16'000/sec
Durable Write 256	2'500/sec	9'000/sec	8'000/sec
Analytics	Limited MapReduce	Hadoop MapReduce	MapReduce, Pig, Hive
Durability support	Full	Full	Full
Native API	Binary JSON	Java	Java
Generic API	None	Thrift	Thrift, REST

However, life got in the way

- Maintaining an operational cluster of Hbase for ourselves was tedious
- We do not have the manpower to keep such a service production-ready
- Involved CERN IT and other interesting groups, that eventually resulted in the creation of lxhadoop (in 2013, operational 2014)

- In the meantime, we went back to the lowest common denominator
  - Daily dumps of Oracle table content in flat files on Hadoop
  - Mapreducing the files directly without Hbase
  - Writing the output to a multitude of small different files, one each per summary, available via HTTP

- Anyone who has written any mapreduce java program (granted, any java program) knows how tedious this is, especially with complex reporting
- Choice fell on a tool that created mapreduce programs for us: Apache PIG

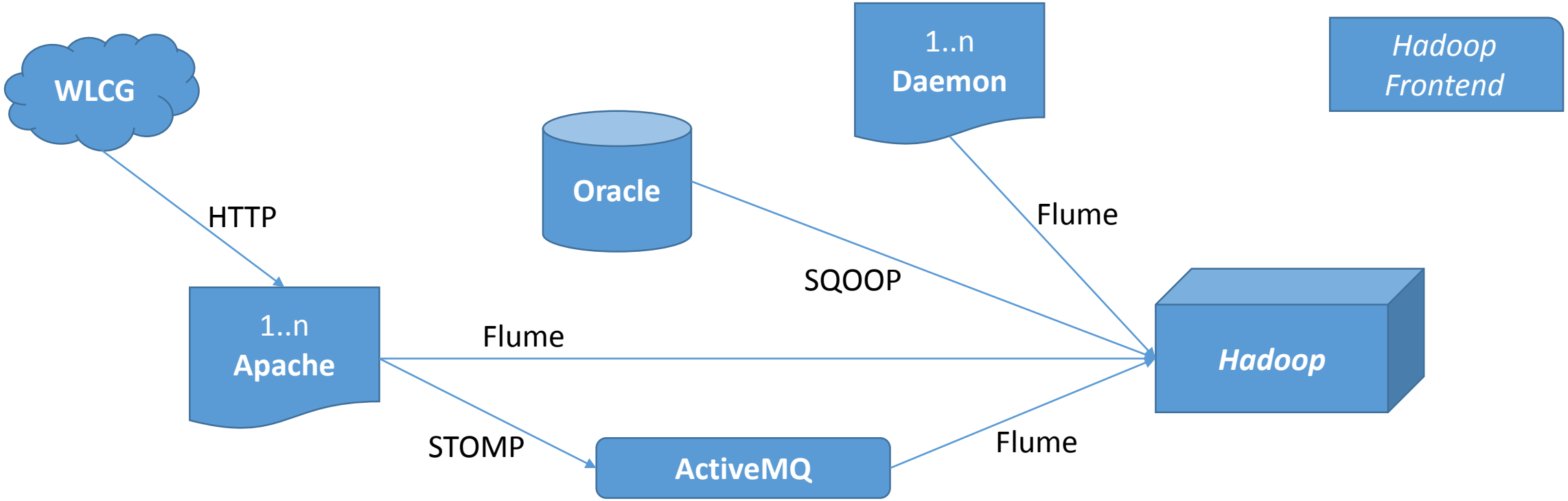
Hadoop/Pig/MapReduce in Rucio

Data sources

- Apache server logfiles (structured format)
- Daemon logfiles (unstructured format)
- Traces (JSON dictionaries)
- Oracle dumps (CSV files)

Data volume

- Logs *25 GB/day*
- Traces *5 GB/day (6 million traces)*
- Dumps *25GB compressed/day (1.5 billion rows)*



Hadoop/Pig/MapReduce in Rucio

- Daily jobs with Apache Pig on the data
- Pig is a data pipeline with a syntax “inspired” by SQL
- Generates native Java MapReduce jobs from PigScripts, runs them on the cluster

- Example workflow “generate a list of unique replicas of a file for all RSEs”
  - filter all files which only have one replica
  - split for (non-) deterministic storage endpoints
  - for the deterministic storage endpoints generate the path with a UDF (user-defined function)
  - merge everything together again
  - join with RSE table to get application-level storage endpoint names
  - store back onto HDFS in multiple files, but split the output per RSE automatically

Hadoop/Pig/MapReduce in Rucio

```
rses = LOAD '/user/rucio01/dumps/$CURRENT_DAY/rses/part-m-*.bz2' USING PigStorage('\t') AS (
        id: chararray,
        rse: chararray,
        rse_type: chararray,
        deterministic: chararray,
        volatile: chararray);

replicas = LOAD '/user/rucio01/dumps/$CURRENT_DAY/replicas/part-m-*.bz2' USING PigStorage('\t') AS (
        scope: chararray,
        name: chararray,
        rse_id: chararray,
        bytes: long,
        state: chararray,
        lock_cnt: long,
        adler32: chararray,
        created_at: chararray,
        accessed_at: chararray,
        path: chararray);
```

supports many different kinds of inputs

but only few datatypes

Hadoop/Pig/MapReduce in Rucio

```
-- group per file and count the number of replicas
      group_reps = GROUP d_reps BY (scope, dsn);
      count_reps = FOREACH group_reps GENERATE scope, dsn, d_reps, COUNT(d_reps) as num_reps;

-- filter out all non-unique replicas
      filter_unique = FILTER count_reps BY num_reps == 1;

-- there is only one entry left in bag, so flatten it
      flatten_reps = FOREACH filter_unique GENERATE FLATTEN(d_reps);
```

…

- Input: replica.csv.bz2: 12 GB, rse.csv: 15 KB
- Output: 700 bz2 files, 9 GB
- Runtime: 30 minutes

Hadoop/Pig/MapReduce in Rucio

- Example: Find set difference between two large tables, with a deadline

- Both tables hold 600 million rows each, 2x11GB bz2 (2x90GB raw)
- SQOOP dump from Oracle 4 hours, highly skewed parallelism due to the nature of the data
- Time to write Pig script with full outer join and validate on sample: 20 minutes
- Time to run Pig script on full dataset: 10 minutes
- Output: 40 million rows

- btw, the same query in Oracle, set to parallelism 4, wanted to create 1.5 PB of temporary storage and do nested loops (not enough memory for hash-based join). After about 8 hours the query ran out of rollback segments.

Hadoop/Pig/MapReduce in Rucio

- Example: Find arbitrary strings in all our logfiles and sort them based on time

- Usual time window is 2 weeks up to 3 months to investigate
  - logfiles stored per server per service per hour
  - roughly 14·25=350 GB to 60·25=1.5 TB of data to search in the default case
  - users are not very patient – naïve bash grep/xargs/parallel implementation causes grey hair when one does not know exactly what to search for (which is usually the case)
  - not useful for short searches, mapreduce instantation takes 20 seconds

- Potential solutions exist: Fulltext search engines like ElasticSearch/Lucene
  - Problematic, because one has to run one more additional (full-blown) service to provide a simple functionality
  - CERN IT already using ElasticSearch for another use case, they recommended not using it due to the high append/update rate of our data sources

- Native mapreduce distributed grep in Java
  - When parsing each line in the mapper, extract the timestamp and exploit it as the key
  - Actual string search per line is trivial
  - Keys are implicitly sorted by mapreduce over all inputs
  - Runtime (sub)linear with logfile size

Strategies for data modelling

- Objectives for any data model, to efficiently and coherently
  - … organize data
  - … abstract data
  - … store data
  - … retrieve data
  - … index data

Organize

- files and directories?
- datasets and containers?
- key-value pairs?
- native (programming) language data types?

Abstract

- file in userspace corresponds to physical file on disk or memory
- or multiple files? on multiple disks?
- on multiple nodes? how distributed?

Store

- provide a connector between abstraction and physical storage
- support native physical storage protocols

Retrieve

- provide a connector between physical storage and abstraction
- support native application data access protocols

Index

- how to find the data

Strategies for data modelling

- In the “big data” world, one all-encompassing data model is impracticable

- Multiple different sources of data
  - Application specific data
  - Server logs
  - Service logs
- Multiple formats of data
  - Cooperating systems use different internal data models
  - Log files between different services differ

- Two targets to be hit
  - Find the available pivots in the data sources
  - Make it easy to connect the pivots

Strategies for data modelling

- Straightforward dump of all different kinds of data not recommended
- Provided tools are geared towards several technologies – exploit them!

- Filesystem-like storage
  - Unstructured: Text
  - Structured: Avro, Sequence, Parquet, RCFile

- Database-like storage
  - HBase, HadoopDB: random-access key/value frontend to filesystem-like storage
  - Hive, Impala: SQL-like frontend to filesystem-like storage

- Compression
  - Streaming compression (lzo, gzip, snappy) – good for intra-mapreduce phases
  - Block compression (bz2) – good for input files
  - Create split-able output from structured storage, and streaming compression with each

- Deciding on the storage method upfront will pre-cure headaches down the road
- Recommendation if nothing fits: tab-separated text files with bzip2 compression

Strategies for data modelling

- Once the data storage has been sorted out, the tools provide capabilities to find pivots
  - primary keys in relational parlance
  - but not quite…

- The worst example

  2013-12-03T12:12:43.454161Z – ISO date format in one file

  12-03-2013 12:12 PM – US date format in another, with different precision

- Libraries exist for basically all tools to create/convert comparable datatypes on the fly
  - “waste” of CPU cycles for every single mapreduce run
  - however, usually the size of the data is the problem, not the processing
  - it might make sense to have periodic sanitation runs, to bring all data into the same format

- With one big exception: regular expressions!
  - Optimisation on regular expressions can yield performance increases of several orders of magnitude
  - Especially if you need backward group parsers, or elimination parsers

Strategies for data modelling

- Once the pivots are found it becomes easy to connect different data sources
  - Tools provide capabilities to “join” flat files
  - Or to merge data between different sources
  - With all different kinds of syntax and caveats (so you should not be too worried which one you choose)

- The important question is now: what are you going to do with all the data?
- There is currently no known project that makes visualization of big data easy/nice/possible

More information

- There is one application within ATLAS that is using Hbase now: Event Index
  - Trigger tables are being obtained from COMA
  - Uses Hbase to decode trigger information (convert trigger bits to list of trigger chains)
  - Trigger table information is requested from MapReduce or Scan and cached

- ATLAS Analytics Working Group
  - https://docs.google.com/document/d/1oLJ_JJPuLu6SZfK1P3j0m_imAJTBA81-4L23HxYbtj8/edit?usp=sharing&invite=CLGb2OQK
  - atlas-adc-data-analytics@cern.ch
  - https://its.cern.ch/jira/browse/ATLASMINER

- In cooperation with it-analytics-wg@cern.ch