Monitoring WLCG with the *lambda* architecture

A new processing framework based on Hadoop (and friends)

Luca Magnoni
CERN IT
Outline

• Experiment Dashboard as use case
• WLCG Monitoring: challenges
• A new monitoring architecture
• Hadoop at work:
  • Data representation: make you choices
  • Performance results
• Real-Time layer: options and tools
Experiment Dashboard (ED)

- A common monitoring framework developed at CERN
- Multiple Targets:
  - Data activities
  - Job Processing
  - Services and Sites Availability
- Different Perspectives:
  - Users, Experts, Sites

Monitoring WLCG computing activities since 2006

14/04/15
CHEP 2015, Okinawa
What does it do? **Analytics.**

Collects and processes monitoring data to build custom visualization
Monitoring WLCG Data Activities

Monitoring events/day

Avg/Day
~ 20 Ml events
Current ED Data Pipeline

- ATLAS/Rucio servers
- FTS servers
- XRootD federations
  - GLEDs (aggregate & jsonify)
- HTTP
- Message Broker (ActiveMQ)
- ED Collectors (validate & enrich)
  - Raw data (~ 2 TB/year)
  - PL/SQL (process)
  - Statistics data
  - ED MVC Framework (serve result & build UI)
  - Web API
  - Web UI

Oracle

- Every 10 min

Raw data (~ 2 TB/year)

14/04/15

CHEP 2015, Okinawa
It works, but...

**Operational cost**
- Custom code/services
- Fragility
- Complexity
  - Too many transformation
  - Difficult to test/validate

**Propagation latency**
- Reports when file is closed
- Delays data forwarding

**Processing does not scale well with data volume**
- Fluctuations:
  - From few seconds to minutes/each run
- Difficult to improve complexity
  - Spikes > 10 minutes (affects UI)
- Reprocessing is expensive (i.e. days/week)
Architecture evolution

“80 percent of the development effort in a big data project goes into data integration and only 20 percent goes toward data analysis.” [Intel ETL White paper]

• **Goal:**
  - Scalable/Simplified/Mainstream

• **Challenges:**
  - Collect the raw data, process once and transform only when needed
    - More data (~ x10) to be archived and analyzed
    - Current PL/SQL jobs cannot make it
    - Hadoop/MapReduce seems appropriate, but it’s a new processing paradigm and infrastructure
Lambda architecture

• Taking inspiration from the Lambda architecture
• Implemented at Twitter
• Intuitively, one-technology-does-not-fit-all idea
  • Batch for slow, reliable and stateless processing
  • Real-Time for fast, complex and incremental computation
• Serve result from a dedicated serving layer

From the book:
Big Data
Principles and best practices of scalable realtime data system
By Nathan Marz and James Warren
ISBN: 9781617290343
New \textit{lambda}-style architecture

- \textbf{ATLAS/Rucio servers}
- \textbf{FTS servers}
- \textbf{XRootD federations}
- \textbf{UDP2JSON}
  - (forward JSON)
- \textbf{HTTP}

\textbf{Message Broker (ActiveMQ)}

\textbf{Flume}
- Real-time
  - Streaming agents: e.g. Spark
- Batch processing
  - Raw data on HDFS
  - Map Reduce

\textbf{Dashboard common library}
- (parse, tokenize, …)

\textbf{Statistics data}

\textbf{ED MVC Framework}
- (serve result & build UI)

\textbf{Web API}
\textbf{Web UI}

\textbf{HADOOP}
- \textit{Elasticsearch}, \textit{Oracle}

- \textbf{14/04/15 CHEP 2015, Okinawa}
New \textit{lambda}-style architecture

\textbf{Mainstream}
- Use existing tools and services
- Less code to maintain and operate

\textbf{Timeliness reporting}
- Reports all monitoring data as soon as it is produced

\textbf{Processing scales with data volume}
- Scale-out by design

FTS servers
XRootD federations
Message Broker (ActiveMQ)
UDP2JSON
(\textit{forward JSON})

\textbf{Flume}
- Real-time
- Batch processing
- Raw data on HDFS
- Map Reduce
- Dashboard common library
  (parse, tokenize, …)

\textbf{HADOOP}
- Elasticsearch
- Oracle

\textbf{New lambda-style architecture}

\textbf{ATLAS/ Rucio servers}
\textbf{UDP2JSON (forward JSON)}
\textbf{UDP2JSON (forward JSON)}
\textbf{UDP2JSON (forward JSON)}
\textbf{UDP2JSON (forward JSON)}
\textbf{UDP2JSON (forward JSON)}
\textbf{UDP2JSON (forward JSON)}
\textbf{UDP2JSON (forward JSON)}
\textbf{UDP2JSON (forward JSON)}
\textbf{UDP2JSON (forward JSON)}

\textbf{FTS servers}
\textbf{XRootD federations}
\textbf{Message Broker (ActiveMQ)}

\textbf{Processing scales with data volume}
- Scale-out by design

\textbf{Web API}
\textbf{Web UI}

\textbf{Statistics data}

\textbf{Web API}
\textbf{Web UI}

\textbf{Raw data on HDFS}

\textbf{Map Reduce}

\textbf{Dashboard common library}
  (parse, tokenize, …)

\textbf{Real-time}

\textbf{Batch processing}

\textbf{Reports all monitoring data as soon as it is produced}

\textbf{Processing scales with data volume}
- Scale-out by design

\textbf{Mainstream}
- Use existing tools and services
- Less code to maintain and operate

\textbf{Timeliness reporting}
- Reports all monitoring data as soon as it is produced

\textbf{Processing scales with data volume}
- Scale-out by design

\textbf{Mainstream}
- Use existing tools and services
- Less code to maintain and operate

\textbf{Timeliness reporting}
- Reports all monitoring data as soon as it is produced

\textbf{Processing scales with data volume}
- Scale-out by design
Now, make your choices!

How to store data on HDFS?
Do you need a schema? A serialization lib?

AVRO vs CSV vs JSON for 1 Day FAX data

Is your schema good?

File size (MB)

How to partition data?

|-- data
| |-- xrootd
| | |-- atlas
| | | |-- 2014
| | | | |-- 12
| | | | | |-- 01
| | | | | | | data.avro

Is compression a good idea?
Which algorithm?
Hadoop/MR* at work

Computation of Compressed/Uncompressed Avro, CSV and JSON files over different date ranges

Our choice: Avro, no compression

* CERN IT-DSS Analytix cluster (Hadoop/CDH 5.1), 8 nodes 32 cores/64GB, 7 nodes 4 cores/8GB
Real-Time layer: more choices!

- **Streaming frameworks**
  - Distributed (à la Hadoop)
  - Basic processing (filter, join, etc.)
  - Apache Storm, Apache Spark (Streaming), Apache Samza

- **In-memory engines**
  - Advanced processing (with DSLs like SQL or time interval algebra)
  - Esper, VoltDB

- **We are currently investigating:**
  - Apache Spark for streaming, potentially with Esper as operator for complex task
Conclusion

- Hadoop ecosystem fits well with WLCG monitoring
  - Scales with data volume
  - For Dashboard, imperative MR easier than SQL
  - Simpler architecture, less custom code and services
- The lambda idea enables “live-views” on WLCG
  - Keep all the data
  - Batch + streaming for fast and reliable processing
  - Ongoing investigation on Spark as ~ uniform platform
    - MR to Spark for batch trivial, done already (thanks to common lib)
- Plan: towards production
  - Migrate XRootD (FAX, AAA federations) and HTTP dashboards to new architecture by summer 2015
  - Focus then on FTS and DDM dashboard migration