(DISTRIBUTED) STRUCTURED STORAGE FOR ATLAS

Mario Lassnig
CERN PH-ADP-CO/DQ2

mario.lassnig@cern.ch
ph-adp-ddm-lab@cern.ch

WLCG Database TEG

Preface: Say No to NoSQL

- Buzzword term to annoy RDBMS people
  - Correct CS term: *(Distributed) Structured Storage*
  - (many of them support SQL-like queries anyway!)
- Essentially two different models
  - "Big Data"-capable (=linear scale out) vs. the rest
- And then it's "feature matrix mix'n'match"
  - CAP Theorem
    - Consistency – Availability – Partition tolerance
  - ACID vs BASE
    - Basically available - Soft state – Eventual consistency
    - ZERO(W) – ANY(W) – ONE(RW) – QUORUM(RW) – ALL(RW)
  - Query vs MapReduce

Overview

- ATLAS relies heavily on Oracle
  - We have many use cases that put a high load on Oracle
    - Much of this load is constantly increasing requiring constant attention to optimisation in order to scale
    - Requires schema denormalisation, relaxation of constraints, service splitting, …
    - Oracle query optimiser does not always yield optimum execution plan requiring manual tuning
    - Oracle query plans are not always stable requiring manual interventions
  - Costly for developers, DBAs, DB experts, and hardware
- Structured Storage Initiative by the DQ2 Team, soon followed by PanDA and TDAQ
  - OLAP use cases: Query and compute *n* partitions for historical data aggregation and summarisation, with lookups from *m* other tables
  - Offload OLAP use cases from Oracle to the "right tool for the job"
- So, no, we don't want to get rid of Oracle for OLTP, we need it, but…
  - Structured storage is a good solution for our many OLAP cases
  - Reduce (human and machine) workload
  - Get results faster: non-invasive parallel data crunching
  - No need to buy special hardware
  - Community support and commercial support available
  - Operational cost is very low, so it's almost a free lunch

Again:
Structured storage is not and never has been "~~versus Oracle~~",
but to "complement Oracle"!

Some examples

- Document Stores
  - Semi structured data (schema=data)
  - e.g., *CouchDB*, *MongoDB*
- Graph Databases
  - Index free adjacency of data
  - e.g., *Neo4j*
- Key-Value
  - Schema-less, can be sorted or unsorted
  - e.g., *BigTable*, *SimpleDB*, *memcachedb*, *Kyoto Cabinet*, *Dynamo*, *Cassandra*, *Voldemort*, *Berkeley DB* (=*Oracle NoSQL*)
- Tabular Databases
  - Multidimensional sorted maps
  - e.g., *BigTable*, *HBase*
- Data Structure Stores
  - More complex datatypes than strings and numbers
  - e.g., *Redis, MongoDB*

Objective

- Evaluate as many different products as possible that can match ATLAS use cases, subject to
  - Development time
  - Operational cost
  - Support (commercial and community)
  - Performance characteristics

- So far, ATLAS has experience with
  - **Oracle** (*DQ2*, *PanDA*)
  - **Hadoop HDFS+HBase** (*DQ2*)
  - **Cassandra** (*DQ2*, *PanDA*, *TDAQ*)
  - **MongoDB** (*DQ2*)
  - **SimpleDB** (*PanDA*)

Important:
Most of us are not database administrators/experts.
We need to use the *right* tool for a given use case under time constraints.

Our project experiences

- DQ2 Accounting
- DQ2 Tracing
- DQ2 Popularity and Simulation
- DQ2 Log File Analysis
- PanDA Monitoring
- TDAQ Online Monitoring

DQ2 Accounting

(Mario Lassnig, Lisa Azzurra Chinzer)

- Break down usage of ATLAS data in DQ2
  - Free-form querying with history (difficult to predefine views)
  - Metadata based

    ```
    {nbfiles, bytes} := {project=data10*,datatype=ESD,location=CERN*}
    ```
  - Constant updates to the data, new keys, bulk operations…
  - Immense cost on Oracle
    - Developer time, DBA time, Hardware
- While we developed the Oracle solution, we evaluated other possible database backends
  - MongoDB, HBase
  - Can we do better?

DQ2 Accounting (MongoDB)

(Mario Lassnig, Lisa Azzurra Chinzer)

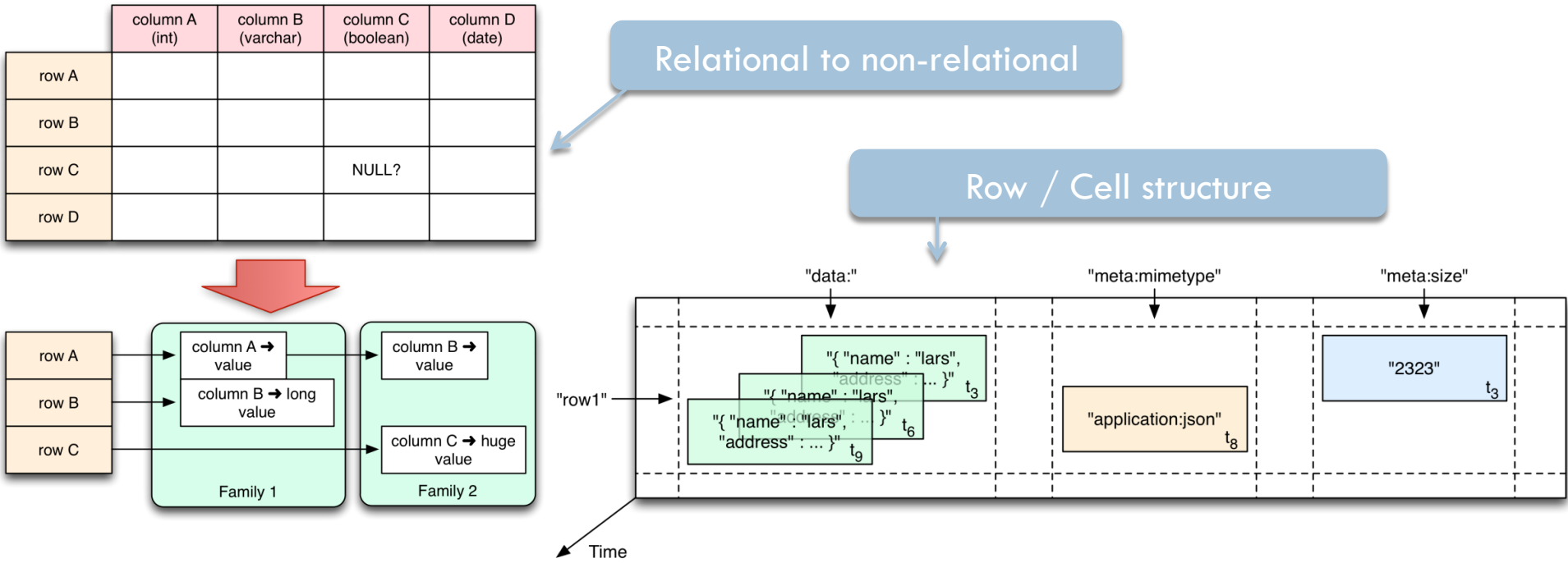
- Option 1:
  Keep source data in MongoDB and MapReduce into summary
  - Migrate catalogues: rich data model
  - However, MapReduce problematic
    - Apparently addressed in Mongo 2.x
    - Locking, concurrency, sharding
- Option 2:
  Only summary to MongoDB
  - Calculate summary in Oracle and write summary to MongoDB
  - Much better than Oracle's star schema summary
- Remarks
  - MongoDB took all data linearly
  - Full durability: 2500 upserts/sec
  - Relaxed durability: 6000 upserts/sec
  - Keeps all indexes in memory
    - 40 mio datasets fully indexed ~ 2GB
  - By now (November), even the Oracle summarization breaks sometimes
    - Locked statistics, and all other kinds of magic

```
PRIMARY> db.kvs.findOne({'_id': 'dc76b9b57360485f9938a318a77dd5f3'})
{
        "[otherfields]" : 104233247,
        "_id" : "dc76b9b57360485f9938a318a77dd5f3",
        "aglt2_userdisk" : {
                "rarchived" : "primary",
                "rgroup" : "/atlas/role=null",
                "rlength" : 3650135,
                "rnbfiles" : 2,
                "rowner" : "/o=grid-fr/c=fr/o=cea/ou=irfu/cn=nayanka bolnet",
                "ts" : ISODate("2011-05-11T16:01:21Z")
        },
        "cyfronet-lcg2_scratchdisk" : {
                "rarchived" : "default",
                "rgroup" : "/atlas/role=null",
                "rlength" : 1825068,
                "rnbfiles" : 1,
                "rowner" : "/o=grid-fr/c=fr/o=cea/ou=irfu/cn=nayanka bolnet",
                "ts" : ISODate("2011-04-11T22:11:07Z")
        },
        "hidden" : 0,
        "length" : 3650135,
        "locations": ["aglt2_userdisk", "cyfronet-lcg2_scratchdisk"],
        "name" : "user.nbolnet.0104233247.260895.lib._000601",
        "nbfiles" : 2,
        "owner" : "/o=grid-fr/c=fr/o=cea/ou=irfu/cn=nayanka bolnet",
        "replicas" : 2,
        "ts" : ISODate("2011-05-11T16:23:26Z"),
        "user" : "user",
        "username" : "nbolnet"
}
```

DQ2 Accounting (HBase)

(Mario Lassnig, Lisa Azzurra Chinzer)

- Current state: try to do it with Hadoop HBase
  - Proper MapReduce support, efficient data handling
  - Runs on clustered HDFS
- HBase is a database for multidimensional sparse columns with versioned cells
  - Not quite that obvious like the hashtables for MongoDB? Depends…
  - ColumnFamilies are a set of (possibly overlapping) columns that "belong together" (i.e., in the relational world this would be a view)



DQ2 Accounting (HBase)

(Mario Lassnig, Lisa Azzurra Chinzer)

- Current state
  - DDMLAB-hosted 10 node Hadoop cluster (with no special optimisation)
  - Full migration of Oracle content (400M rows, 20GB) into HBase data model (20M rows, 24GB): *2h 40m*
  - MapReduce once over data: *40 mins (random reads ~4K blocks, ~2.5MB/sec per node)*
    - *HDFS replication factor 4… (40*4)/60 (replication factor to a full 10, then MapReduce in 15min)*
    - *Compared with Oracle: 5-6 hours, if it finishes at all…*
    - *Can do all accounting summaries in one MapReduce run*
- Still ToDo
  - One-way synchronisation from Oracle to HBase
    - Not trivial
    - How to get data out of Oracle at the required Hertz and in proper order?
    - (Accounting not a realtime service, perhaps just dump the data once per day?)
  - Summary retrieval
    - Trivial
  - Ad-hoc queries need a slightly different data model than the MapReduce one
    - otherwise you would have to MapReduce ad-hoc, which requires some more machines

DQ2 Log Analysis (HDFS)

(Vincent Garonne, Mario Lassnig)

- Map-Reducing CSV files on Hadoop HDFS
  - Dump table from Oracle (DQ2 Traces)
  - Copy log files to HDFS (DQ2 Apache Logs)
- Use Hadoop Streaming API
  - read/write stdin/stdout
    - `mapper.py, reducer.py, wc, awk, perl, grep, ...`
  - can write results directly to HBase (with Java/Jython)
- Map-Reduce 75GB in 5 minutes
  - *with Java libraries it should be 2.5 times faster (memory allocation savings)*

Cluster Summary (Heap Size is 214.81 MB/3.56 GB)

Running Map Tasks	Running Reduce Tasks	Total Submissions	Nodes	Occupied Map Slots	Occupied Reduce Slots	Reserved Map Slots	Reserved Reduce Slots	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes	Excluded Nodes
0	0	58	10	0	0	0	0	60	60	12.00	0	0

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	709	0	0	709	0	0 / 25
reduce	100.00%	1	0	0	1	0	0 / 0

DQ2 Tracing

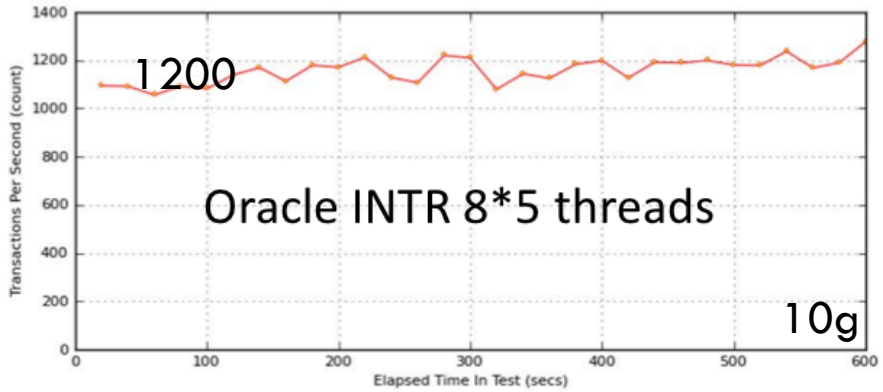
(Donal Zang, Vincent Garonne)

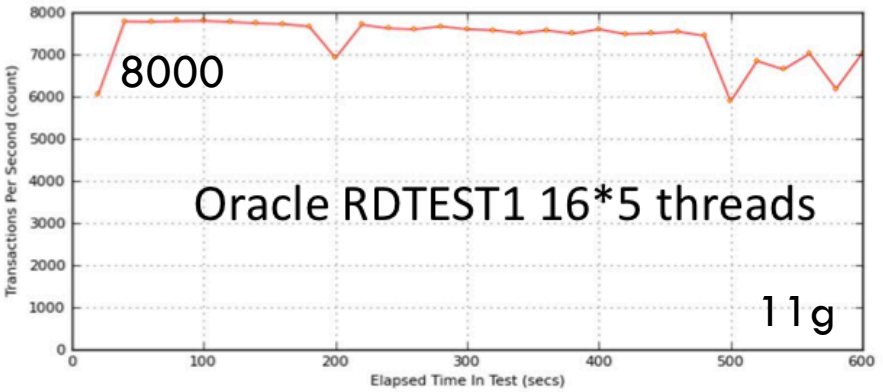
- Record relevant information about dataset/file access
  - *type, status, local site, remote site, file size, time, usrdn,…*
  - DQ2Clients (dq2-get/put), Panda Pilot, Ganga
  - Traces can be analyzed for many purposes
    - Data Popularity
    - DQ2 simulations
    - User behavior analysis
    - DQ2 system monitoring (including failures)
  - ~5 million traces every day
- Problem
  - All use cases need some sort of aggregation
  - Aggregation metrics can by very dynamic
  - On Oracle tens of minutes to hours with heavy IO
  - Can we go realtime?

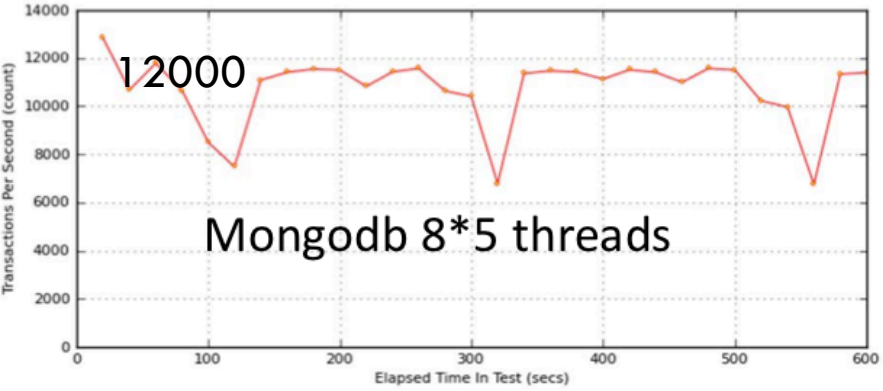
DQ2 Tracing (Cassandra)

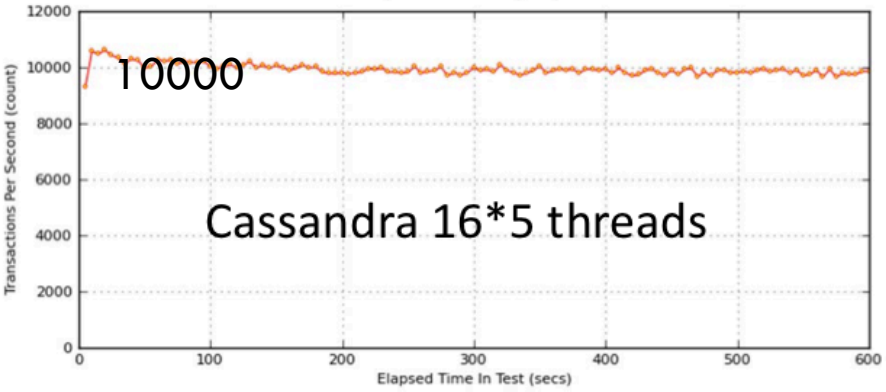
(Donal Zang, Vincent Garonne)

- Write performance tests
  - row-by-row insertion (~3KB/row)







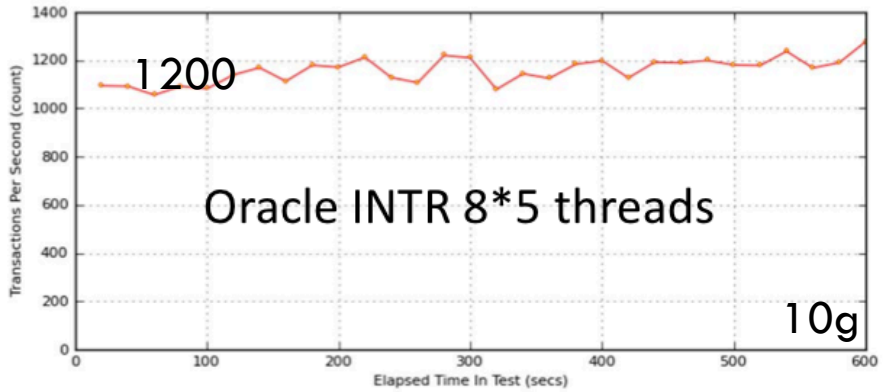


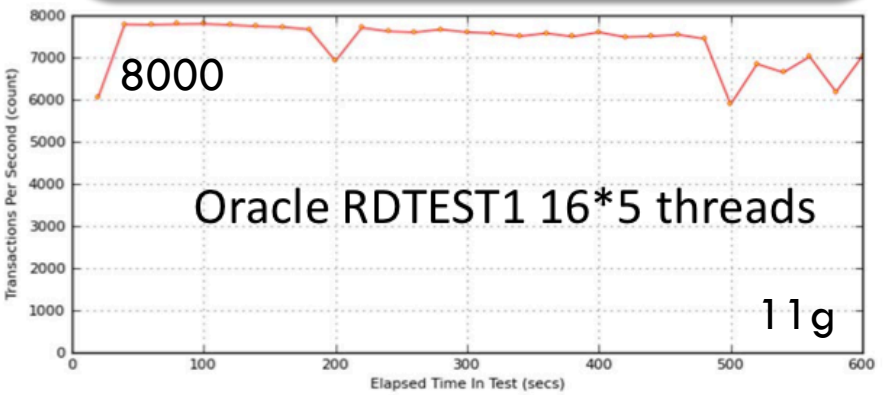
DQ2 Tracing (Cassandra)

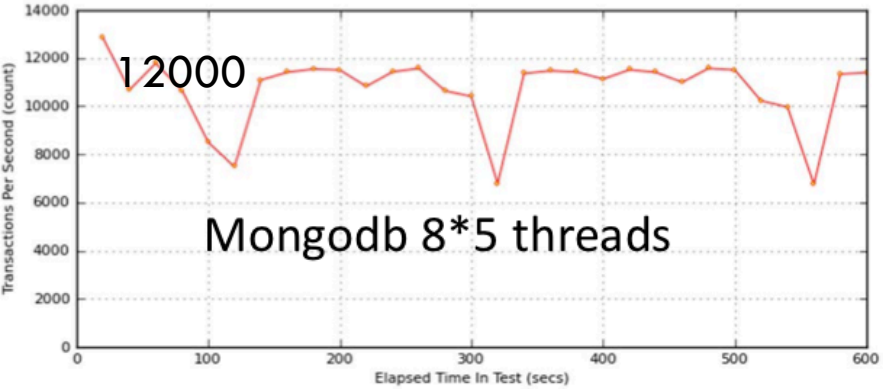
(Donal Zang, Vincent Garonne)

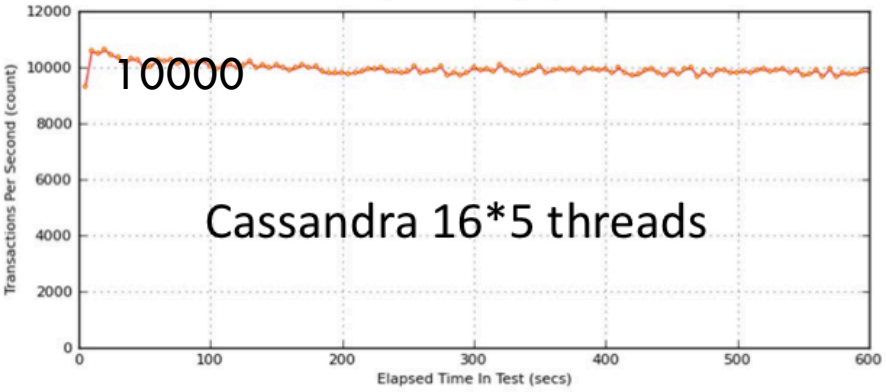
- Write performance tests
  - row-by-row insertion (~3KB/

Oracle NoSQL
(native Java, 1,5,16 threads)
~4200









DQ2 Tracing (Cassandra)

(Donal Zang, Vincent Garonne)

- Query results
  - One months traces (90M rows, 34GB)
  - Query 1: Total number of traces
  - Query 2: For each '%GROUPDISK%' endpoint, get the "Total Traces", "Write Traces", "Total Users"

Query	Oracle INTR	Oracle RDTEST1	Oracle RDTEST1 cache	Oracle production ADCR	Cassandra
Query 1	39 seconds	30 seconds	~1 second	1.14 hour	2.2 minutes
Query 2	47 seconds	30 seconds	~3 seconds	>5 hours	28.3 minutes

Parallel IO
(not allowed in production)

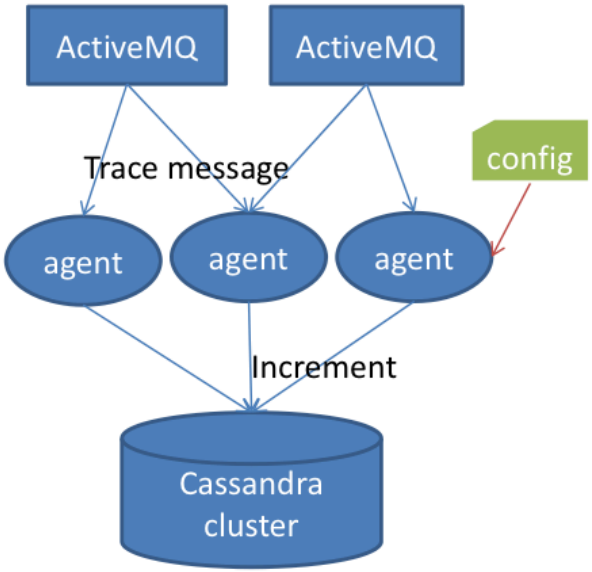
Random hash partition,
Good for durability and
spread, bad for range query

DQ2 Tracing (Cassandra)

(Donal Zang, Vincent Garonne)

- Exploit fast inserts to
  - a) build indexes in data model
    - >10k updates per second
  - b) use distributed counters

```
t_index = {
    '2011052017:remoteSite:eventType':{
          'CERN-PROD_DATADISK:put_sm:1304514380628696'      : 23444,
          'CERN-PROD_DATADISK:get:1304514380628697'         : 32232,
          'CERN-PROD_GROUPDISK:put_sm:1304514380628696'     : 43122,
          ...
       },
    ....
}
```



DQ2 Tracer Infrastructure

- Query example
  - count and sum of traces grouped by *site* and *eventType* in a specific time period

Oracle(production, ADCR)	Cassandra(use CF as index)
48 minutes (query t_traces)	10 seconds (query the index)

DQ2 Popularity and Simulation

(Thomas Beermann, Angelos Molfetas, Martin Barisits)

- Evaluate dataset popularity service on MongoDB
  - Has to join very large tables in Oracles
  - Queries take hours and summarisation is pre-defined
    → try a MapReduce approach
- Basic problem became apparent soon
  - MongoDB, due to its locking scheme, only supports very limited MapReduce functionality
  - Essentially unfit for this use case
- However, MongoDB seems better suited as an application backend
  - As a substitute, e.g., for MySQL, due to schema-less design
- Use it for DQ2 Simulation
  - Optimise storage models based on historical traces and popularity
  - Backend to a persistent steady state Genetic Algorithm
  - Blazingly fast and easy to work with due to “native” mapping of data structures
  - MongoDB locking not an issue for this use case
  - Lack of transactional support not problematic for this use case

PanDA Monitoring

(Maxim Potekhin, Torre Wenaus)

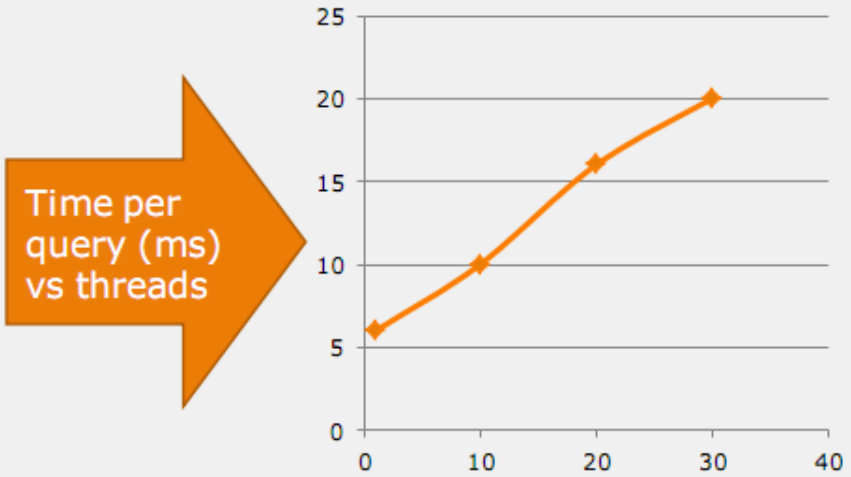
- Offload Oracle for PanDA log file analysis
  - Efficient support for time series and range queries essential
  - ~10Hz production write rate; reduced from previous heavier usage because of scalability problems
  - excessive Oracle load due to write rate
  - underutilizing the logger
    - a very convenient means of logging incidents, alarms, errors, warnings, informational logging
    - Heavily used by Panda server for logging its operations, reporting brokerage and PD2P decisions, …
- Two independent evaluations
  - Cassandra and SimpleDB

PanDA Monitoring (Cassandra)

(Maxim Potekhin, Torre Wenaus)

- Same approach as DQ2 Tracer
  - Exploit fast writes with composite indexes
  - e.g., PRODUSERNAME+JOBDEFID+DATE
- Saturated client machine (30 threads)
  - 3-node Cassandra cluster was able to serve 1500 random requests per second
  - ~2.5 times improvement to current Oracle solution
- Random query tests (10k PanDA jobs)
  - Oracle: ~100ms/request, regardless number of threads (1-30)
- Now in production with web frontend

Number of threads	Time per query	Server load: queries/sec
1	6ms	167
10	10ms	1000
20	16ms	1250
30	20ms	1500



PanDA Monitoring (SimpleDB)

(Maxim Potekhin, Torre Wenaus)

- SimpleDB is a cloud-based storage from Amazon
  - AWS pay-as-you-go for traffic and storage, first 1GB/month free
- Key-Value pairs are stored within *domains*
  - domain equals machine on AWS (implicit sharding)
  - 1billion attributes, 10GB, no item limit, 256 attributes per item, 1024 length, automatically indexed, string
- Automated extraction of Panda job and file data from Oracle and upload to S3 in operation since January 2011
  - SDB content: 131GB, 231M items, 34 domains
- SimpleDB domains sit in a specific AWS region
  - SDB direct write from CERN: ~300ms/write
  - SDB direct write from Virginia EC2: 70-100ms/write
- Heavily optimized for well-indexed and selective queries
  - Aggregation, summarisation lacking, bulk queries slow

PanDA Monitoring (SimpleDB)

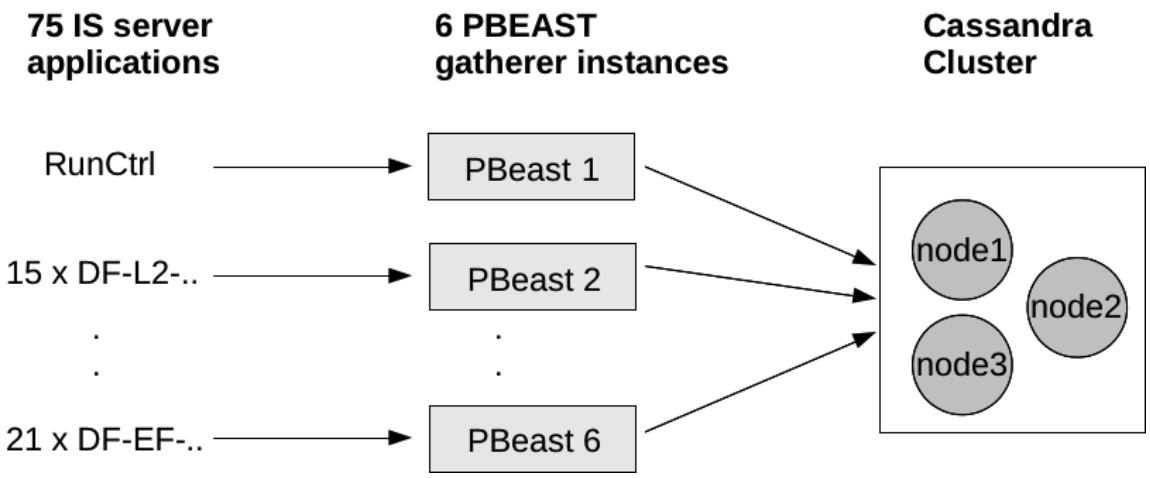
(Maxim Potekhin, Torre Wenaus)

- Best solution is actually
  - Python scripts mapreducing flat CSV files on EC2
  - Fraction of a second to produce any number of summaries
- The June bill
  - Large EC2 instance: ~$100
  - S3, 950GB, negligible request counts: ~$130
  - Data transfer: ~$80
  - SimpleDB storage, 88GB: ~$20
  - SimpleDB compute hours, 6000 hours: ~$830
- Conclusion
  - Real money, but very doable if cost/benefit ratio is good
  - Very little, if measured against fractional FTEs supporting in-house
  - However, still too many feature set limitations

TDAQ Online Monitoring

(Alexandru Dan Sioce)

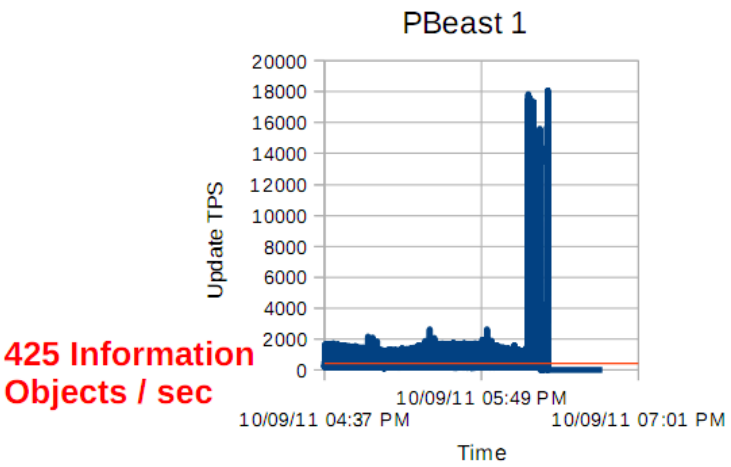
- Trigger and Data Acquisition (TDAQ)
  - suite of 30'000 applications
  - running on 3'000 machines
  - operate on the physics data, after detector, before disk
  - very high rate monitoring (~2500 Hertz)
  - until now, all this monitoring data was lost
  - need to store last *n* events with details, to investigate problems
- 3-Node Cassandra backend (4GB RAM, 4 Cores)

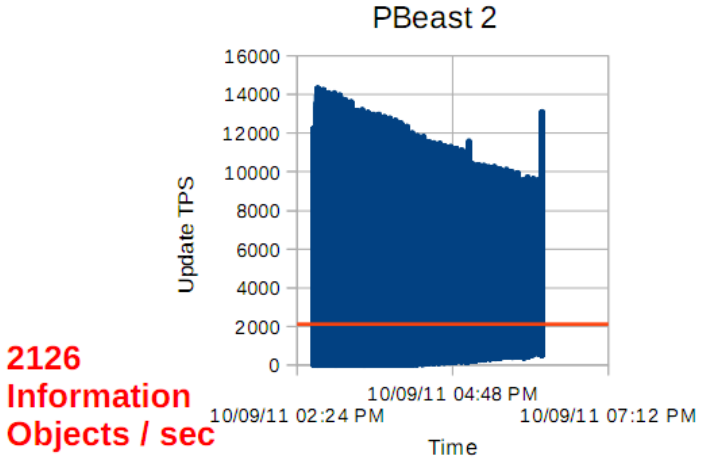


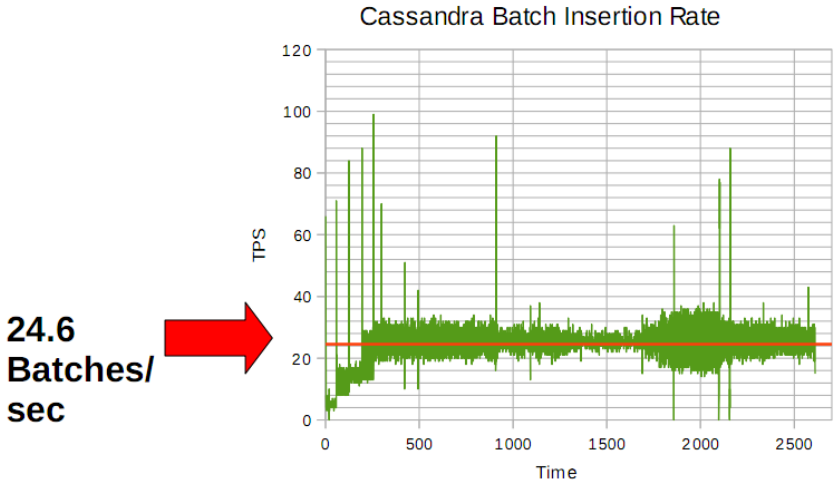
TDAQ Online Monitoring

(Alexandru Dan Sioce)

- 24.6 batches/sec with 226KB/batch ~ 5.6 MB/sec
- Very good at taking in time series data coming at various rates with occasional bursts







What do we need?

- Logging and monitoring
  - High write rates
- Data analytics
  - Complex computations over lots of data
- Content and Summary retrieval
  - Fast lookup
- Application backend
  - Low latency

What do we need?

- Logging and monitoring
  - High write rates
    → *Cassandra*, *MongoDB*, *HBase*
- Data analytics
  - Complex computations over lots of data
    → *HBase*, *MongoDB*, *Cassandra*
- Content and Summary retrieval
  - Fast lookup
    → *Cassandra*, *MongoDB*, *HBase*, *SimpleDB*
- Application backend
  - Low latency, schema-less design, “native” data structures
    → Cassandra, *MongoDB*

What do we need?

- Logging and monitoring
  - High write rates
    → *Cassandra*, ~~*MongoDB*~~, ~~*HBase*~~
- Data analytics
  - Complex computations over lots of data
    → *HBase*, ~~*MongoDB*~~, ~~*Cassandra*~~
- Content and Summary retrieval
  - Fast lookup
    → *Cassandra*, *MongoDB*, *HBase*, ~~*SimpleDB*~~
- Application backend
  - Low latency, schema-less design, "native" data structures
    → Cassandra, *MongoDB*

With 1 writer

With Java

Limited MapReduce

No native MapReduce

Only free up to 1GB/month, Location-dependent

But what about...

- Hypertable, Riak, Voldemort, Redis, CouchDB, ...

  Probably good choices for given use cases as well, but

  1. *we have no operational experience with them*
     (except some corner-case trials)

  2. *we went for the products with the largest market-share, most active communities, and most reliable roadmaps over the next years*

- Didn't have time to try everything
- Highly evolving area
  - e.g., MongoDB locking&MapReduce apparently solved in recently released versions
  - we have to stay flexible → Backend-independent data formats if possible!

Recommendations

- ***CERN should actively support structured storage systems with hardware, development and FTEs***
  - But not by providing a common shared DB service for everyone, like with Oracle
    - If you want this, then you haven't understood these systems
  - Instead, offer choice:
    → CERN hosts a pool of barebone nodes, and experiments request nodes
    → Product experts (FTEs) offer help
    - e.g. "PanDA Monitoring needs 4 Cassandra Nodes" or "DQ2 needs 4 Hadoop Datanodes"
- ***Why can this work?*** (i.e., Google, Facebook, Amazon, Twitter, Yahoo... do it like this)
  - The architecture of these systems does not require "online administration with a DBA"
  - If something is broken, take out the faulty node and replace it. Nodes resync automatically.
- ***Work required ?***
  - Construction of images to be deployed on a barebone machine by experts
    - e.g., Hadoop Namenode, Hadoop DataTaskNode, Cassandra Node
    - no special requirements on the hardware; no virtual machines; not necessarily SLC
    - images (even puppet modules) exist already (e.g., Cloudera), commercial support available
  - Deployment of a monitoring dashboard for these nodes
    - application specific dashboards exist already from/for each product
    - system dashboard already hosted by CERN (nagios, lemon)
  - No excessive operational support needed except
    - Basic product maintenance (new versions, cluster restarts, firewalling, etc...)
    - Wipe and reinstall of a faulty node with an image
    - Hardware replacement of faulty hardware

Recommendations

- ***One size DOES NOT fit all***
  - Too many different use cases
  - Structured storage systems are designed to keep ongoing operational cost low
  - Proper images+dashboards could be done within weeks

- ***Two major products (game changers) should be supported***
  - All of these systems can do efficient key-value lookups,
    but only Hadoop and Cassandra scale out without explicit partitioning/sharding

  - ***Hadoop HDFS + HBase***
    - Large-scale analytics (data aggregation, correlation, and summarisation)
    - Distributed storage

  - ***Cassandra***
    - Time-based data (log file analysis, time series)
    - Low-latency application backend

We need this technology!
(And many others probably as well.)