Applying scalable databases at CERN

8th XLDB Conference
May 20, Stanford, CA, USA
Kacper Surdy
Nature of large databases at CERN

Typically store time series data, examples:

- Logging systems
  - LHC log data: 50kHz archiving, 200 TB + 90 TB/year
- Control and data acquisition systems (SCADA)
  - LHC detector controls
  - Quench Protection System: 150kHz archiving, 2TB/day
- Grid monitoring and dashboards
Example of LHC log numerical data

<table>
<thead>
<tr>
<th>VARIABLE_ID</th>
<th>UTC_STAMP</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>906198</td>
<td>05-AUG-12</td>
<td>-0.0077</td>
</tr>
<tr>
<td>907888</td>
<td>05-AUG-12</td>
<td>-0.000140249</td>
</tr>
<tr>
<td>907936</td>
<td>05-AUG-12</td>
<td>-0.0000087134</td>
</tr>
<tr>
<td>907962</td>
<td>05-AUG-12</td>
<td>-0.0000042486</td>
</tr>
<tr>
<td>340739</td>
<td>05-AUG-12</td>
<td>2.2</td>
</tr>
<tr>
<td>340737</td>
<td>05-AUG-12</td>
<td>93.8</td>
</tr>
<tr>
<td>895035</td>
<td>05-AUG-12</td>
<td>-0.000008338</td>
</tr>
<tr>
<td>1193076</td>
<td>05-AUG-12</td>
<td>15.7</td>
</tr>
<tr>
<td>1214800</td>
<td>05-AUG-12</td>
<td>1</td>
</tr>
<tr>
<td>1214801</td>
<td>05-AUG-12</td>
<td>34</td>
</tr>
<tr>
<td>1214802</td>
<td>05-AUG-12</td>
<td>1</td>
</tr>
</tbody>
</table>
The problem we want to tackle

RDBMS provides us

• fast pinpoint data extraction with indexes
• good writing frequencies (up to 1MHz)

What is missing?

• ad-hoc analytics, e.g. signal correlations and aggregations
  • requires sequential scanning of the data sets
• throughput limited to 1 GB/s
  • on currently deployed shared storage RDBMS clusters
Technologies chosen for evaluation

Hadoop
- open scalable architecture
- multiple analytic tools available
- widely used

Impala
- SQL interface, ODBC/JDBC API available
- variety of supported file formats
- non MapReduce based
Different aspects of storing data

- Encoding: text vs binary
- Compression
- Partitioning
  - Vertical
  - Horizontal
Data format and compression is the key

Data size comparison – 8 days of ACCLOG

<table>
<thead>
<tr>
<th>Data Volume</th>
<th>Data format</th>
<th>and compression is the key</th>
</tr>
</thead>
<tbody>
<tr>
<td>1240 GB</td>
<td>CSV</td>
<td>no compression</td>
</tr>
<tr>
<td>1545 GB</td>
<td>SequenceFile</td>
<td>snappy</td>
</tr>
<tr>
<td>542 GB</td>
<td>Avro</td>
<td>bzip2</td>
</tr>
<tr>
<td>558 GB</td>
<td>Parquet</td>
<td></td>
</tr>
</tbody>
</table>

Test were done with several data file formats

2 compression types have been tested

bzip2 -> smallest files

Size of original data stored in a relational database - 649GB
Data format and compression is the key

Impala sequential scans of 8 days of ACCLOG data

Software used: CDH5.2+

Hardware used for testing: 16 ‘old’ machines

CPU: 2 x 4 x 2.00GHz
RAM: 24GB

Storage: 12 SATA disks 7200rpm (~120MB/s per disk) per host

Avro and Parquet are clean winners

CSV
SequenceFile
Avro
Parquet

Execution time

no compression  snappy  bzip2

snappy -> shortest execution times
Improving signal retrieval time

Problem: no indexes in Impala – full partition scan needed
  • With daily partitioning we have 40 GB to read

Basic solution: fine-grain partitioning
  • (year, month, day, signal id)

Concern: number of HDFS objects
  • 365 days * 1M signals = 365M of files per year
  • file size: 41KB only!

Solution: multiple signals data grouped in a single partition
Bucketing: proof of concept

Based on $\text{mod}(\text{signal\_id}, \ x)$ function

- where $x$ is tunable number of partitions created per day
- $(\text{year}, \ \text{month}, \ \text{day}, \ \text{mod( signal\_id, x)})$

```
<table>
<thead>
<tr>
<th>id, time, value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bucket 0</td>
</tr>
<tr>
<td>10000, 2015-01-09, 17</td>
</tr>
<tr>
<td>99000, 2015-01-09, 4</td>
</tr>
<tr>
<td>55000, 2015-01-09, 5</td>
</tr>
<tr>
<td>Bucket 15</td>
</tr>
<tr>
<td>10115, 2015-01-09, 5.6</td>
</tr>
<tr>
<td>10715, 2015-01-09, 9.8</td>
</tr>
<tr>
<td>Bucket 74</td>
</tr>
<tr>
<td>99074, 2015-01-09, 3.3</td>
</tr>
<tr>
<td>10074, 2015-01-09, 34</td>
</tr>
</tbody>
</table>
```
Bucketing: proof of concept

Based on $\text{mod}(\text{signal}_\text{id}, x)$ function

- where $x$ is tunable number of partitions created per day
- $(\text{year, month, day, mod(signal id, x)})$

And it works!

- 10 partitions per day = 4GB to read
- Data retrieval time was reduced 10 times
  (from 15s to less than 2s)

We have modified the Impala planner code to make
the function based partition pruning implicitly

- No need of explicit specification of a grouping function in
  ‘where’ clause
Conclusions

• Hadoop opens new use cases
  • many interfaces to the data
  • scalable
• Impala is good solution for our time series DBs
• choice of data format and compression type is critical
• customizations can be applied to improve performance