Using the In-Memory Columnar Store to Perform Real-Time Analysis of CERN Data

Maaike Limper
Emil Pilecki
Manuel Martín Márquez
About the speakers

• **Maaike Limper**
  • Physicist and project leader

• **Manuel Martín Márquez**
  • Data Scientist at CERN
  • Interests: Data Analytics and Big Data

• **Emil Pilecki**
  • Oracle DBA at CERN
  • Interests: Database High Availability and Performance
  • Previously: DBA Team Leader at Hewlett-Packard
CERN

- CERN - European Laboratory for Particle Physics
- Founded in 1954 by 12 Countries for fundamental physics research in a post-war Europe
  - Major milestone in the post-World War II recovery/reconstruction process
CERN openlab

- Public-private partnership between CERN and leading ICT companies
- Accelerate cutting-edge solutions to be used by the worldwide LHC community
- Train the next generation of top engineers and scientists.
A World-Wide Collaboration

Member States

- Austria: 96
- Greece: 182
- Slovenskia: 88
- Belgium: 101
- Hungary: 69
- Spain: 257
- Bulgaria: 75
- Israel: 61
- Sweden: 75
- Czech Republic: 202
- Italy: 1696
- Switzerland: 183
- Denmark: 50
- Netherlands: 153
- United Kingdom: 640
- Finland: 87
- Norway: 51
- France: 781
- Poland: 2250
- Germany: 1150
- Portugal: 106

Candidate for Accession

- Romania: 118

Associate Members in the Pre-stage to Membership

- Serbia: 41

Distribution of All CERN Users by Nationality on 14 January 2014
Fundamental Research

• Why do particles have mass?
  • Higgs Mechanism

• Why is there no antimatter left in the Universe?
  • Nature should be symmetrical

• What was matter like during the first second of the Universe, right after the "Big Bang"?
  • A journey towards the beginning of the Universe gives us deeper insight.
Fundamental Research

- What is 95% of the Universe made of?

![Diagram showing composition of the Universe]

- Dark Energy: 71.4%
- Dark Matter: 24%
- Atoms: 4.6%

TODAY
The Large Hadron Collider (LHC)

- **Largest machine in the world**
  - 27km, 6000+ superconducting magnets

- **Fastest racetrack on Earth**
  - Protons circulate 11245 times/s (99.9999991% the speed of light)

- **Emptiest place in the solar system**
  - High vacuum inside the magnets

- **Hottest spot in the galaxy**
  - During Lead ion collisions create temperatures 100 000x hotter than the heart of the sun;
CERN’s Accelerator Complex
150 Million of sensor
Control and detection sensors

Massive 3D camera
Capturing 40+ million collisions per second
Data rate TB per second
CMS Detector

Raw Data
- Was a detector element hint?
- How much energy?
- What time?

Reconstructed Data
- Particle Type
- Origin
- Momentum of tracks (4 vectors)
- Energy in cluster (jets)
- Calibration Information
Worldwide LHC Computing Grid

- Provides Global computing resources
  - Store, distribution and analysis

- Physics Analysis using ROOT
  - Dedicated analysis framework
  - Plotting, fitting, statistics and analysis
Data Analysis in Practice

- ROOT works with N-tuples
  - Specifically produced by physics groups
    - Physics objects, level of details, filter and pre-analysis steps
- Small Datasets
  - Copy files and run locally
- Large Datasets
  - WLCG infrastructure split the analysis in multiple jobs
  - Each job is sent to Grid site where input files are available
The Challenge

Can we replace the file based analysis with a model where the data is analyzed inside a centrally accessible Oracle Database?
The Challenge

- Physics-objects described by hundreds of variables
- Each query uses unique subset of variables
  - Cannot index all possible combinations
- Query performance typically limited by I/O reads
  - Root N-tuples Optimized to reduce I/O
The Challenge

- **Higgs+Z**: 40 variables
- **TTbar cutflow**: 262 variables

Total volume of I/O reads in GB:

- ttbar cutflow: 87 GB
- Higgs+Z: 154 GB
- DB: 317 GB

Analysis time (s):

- Higgs+Z from ntuples
- Higgs+Z from DB

Analysis time (s):

- ttbar cutflow from ntuples
- ttbar cutflow from DB
In-Memory Column Store

- New static pool in System Global Area
- Data in-memory – columnar format
- Huge performance boost for table scans!

Graphics: www.oracle.com, Oracle In-Memory white paper
In-Memory Column Store

- Both row and columnar format simultaneously in memory
  - Buffer Cache for OLTP workload and data modifications (DML)
  - IM Column Store for analytics and reporting queries
  - Guaranteed transactional consistency
- Transparent for applications
  - No code change needed
- No storage overhead
  - Row format on disk

Graphics: Oracle In-Memory data sheet
IMC – Setup

- Very simple setup
  ```sql
  alter system set inmemory_size=128G scope = spfile;
  ```
- Database restart required
- Other parameters: data population, optimizer awareness...

```sql
SQL> show parameter inmemory

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>inmemory_clause_default</td>
<td>string</td>
<td>DEFAULT</td>
</tr>
<tr>
<td>inmemory_force</td>
<td>string</td>
<td>32</td>
</tr>
<tr>
<td>inmemory_max_populate_servers</td>
<td>integer</td>
<td>ENABLE</td>
</tr>
<tr>
<td>inmemory_query</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>inmemory_size</td>
<td>big integer</td>
<td>128G</td>
</tr>
<tr>
<td>inmemory_trickle_repopulate_servers_percent</td>
<td>integer</td>
<td>1</td>
</tr>
<tr>
<td>optimizer_inmemory_aware</td>
<td>boolean</td>
<td>TRUE</td>
</tr>
</tbody>
</table>
```

SQL> _
IMC – Memory

- Distribute memory between IMC and Buffer Cache

<table>
<thead>
<tr>
<th>MEMORY_COMPONENT</th>
<th>SIZE_GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGA Target</td>
<td>186</td>
</tr>
<tr>
<td>In-Memory Area</td>
<td>128</td>
</tr>
<tr>
<td>DEFAULT buffer cache</td>
<td>38</td>
</tr>
<tr>
<td>shared pool</td>
<td>12</td>
</tr>
<tr>
<td>java pool</td>
<td>3.5</td>
</tr>
<tr>
<td>large pool</td>
<td>3.5</td>
</tr>
</tbody>
</table>

- Data pool (1M chunks) and Metadata pool (64K chunks)

- V$INMEMORY_AREA view to monitor IMC pools

```
select POOL, ROUND(ALLOC_BYTES/1024/1024/1024,2) as "ALLOC_BYTES_GB",
       ROUND(USED_BYTES/1024/1024/1024,2) as "USED_BYTES_GB",
       POPULATE_STATUS
from v$inmemory_area;
```

<table>
<thead>
<tr>
<th>POOL</th>
<th>ALLOC_BYTES_GB</th>
<th>USED_BYTES_GB</th>
<th>POPULATE&gt;Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1MB POOL</td>
<td>101.99</td>
<td>.1</td>
<td>DONE</td>
</tr>
<tr>
<td>64KB POOL</td>
<td>25.98</td>
<td>.02</td>
<td>DONE</td>
</tr>
</tbody>
</table>
IMC – Data Population

**ALTER TABLE „electron“ INMEMORY;**

- Table can be loaded immediately (on database startup) or on-demand, when first accessed
- Data populated and re-populated in the background
- New processes IMCO, SMCO, Wnnn
- Optimizer automatically uses In-Memory table scans for fully populated tables
IMC – Data Population

- V$IM_SEGMENTS view to monitor in-memory area contents on segment level

```sql
SELECT segment_name, ROUND(SUM(bytes)/1024/1024/1024,2) "Orig. Bytes GB",
       ROUND(SUM(INMEMORY_SIZE)/1024/1024/1024,2) "In-memory GB",
       ROUND(SUM(bytes-bytes_not_populated)*100/SUM(bytes),2) "% In-memory",
       ROUND(SUM(bytes-bytes_not_populated)/SUM(INMEMORY_SIZE),2) "Compression Ratio"
FROM v$im_segments
GROUP BY owner,segment_name
ORDER BY 2 DESC;
```

<table>
<thead>
<tr>
<th>SEGMENT_NAME</th>
<th>Orig. Bytes GB</th>
<th>In-memory GB</th>
<th>% In-memory</th>
<th>Compression Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>electron</td>
<td>65.66</td>
<td>24.45</td>
<td>51.76</td>
<td>1.39</td>
</tr>
<tr>
<td>jet</td>
<td>32.27</td>
<td>15.08</td>
<td>98.45</td>
<td>2.11</td>
</tr>
<tr>
<td>muon</td>
<td>11.08</td>
<td>10.22</td>
<td>97.75</td>
<td>1.06</td>
</tr>
<tr>
<td>EF</td>
<td>3.22</td>
<td>.07</td>
<td>52.16</td>
<td>23.23</td>
</tr>
<tr>
<td>MET_RefFinal</td>
<td>2.53</td>
<td>2.7</td>
<td>100</td>
<td>.94</td>
</tr>
<tr>
<td>periodAllYear_v47-pro13-01_LE0</td>
<td>.13</td>
<td>.09</td>
<td>100</td>
<td>1.38</td>
</tr>
</tbody>
</table>
IMC - Compression

- Reduces the amount of extra memory needed for IMC
- 5 levels of compression
  - NOCOMPRESS → FOR DML → FOR QUERY LOW|HIGH → FOR CAPACITY LOW|HIGH
- 2x-20x compression
  - Depends on data type and distribution (number of unique values)
- Filters for In-memory table scans applied directly on compressed data
  - Except FOR CAPACITY compression
- Very low performance impact on queries
IMC – Cool Features

- In-memory Storage Index – like Exadata!
  - Skip scanning table sections based on filter predicates
- SIMD Vector Processing
  - Process multiple values in single CPU instruction
- In-memory Joins
  - With Bloom Filter transformation – replace join with a filter
- In-memory Aggregation
  - Special „Accumulator” aggregates data on the fly during the IMC big table scan
IMC – Questions

• How much performance can we gain with In-Memory Column Store for physics queries?
• How does it cope with numeric data used in physics?
  • int, float, double
  • Very high cardinality columns – almost every value is unique
• Is it really transparent?
  • e.g. no negative impact on DML operations
• Can the analysis now be performed in real-time?
IMC Tests – Methodology

• A workload consisting of all 4 benchmark queries
• Queries run in Parallel and some also in Serial mode
• Sample 115GB set of physics experiment data
• Multiple runs of the workload for each DB configuration
  • 4 undisturbed workload runs with a stable execution plan
  • Results averaged out of all test runs
  • 3-4 warmup runs before the actual test
IMC Tests - Environment

- **Server:** Dell PowerEdge R820
- **CPU:** Intel Xeon 2.7Ghz
  32 cores / 64 threads
- **Memory:** DDR3 256GB
- **Storage:** local SSD drive and NetApp NAS
- **OS:** Red Hat Enterprise Linux Server 6.5
- **RDBMS:** 12.1.0.2 Enterprise Edition - 64bit Production

Graphics: www.dell.com
IMC Tests – DB Configurations

• Row format only – Buffer Cache
  • With empty buffer (flushed before each query)
  • With pre-warmed big buffer – 160GB
  • With pre-warmed reduced buffer – 32/80GB

• In-Memory Columnar format – IM Column Store
  • In addition to Buffer Cache
  • All possible compression levels
  • BC/IMC pool sizes dependent on compression: total = 160GB
### IMC Tests – Compression Rates

- For tables used in physics analysis – 115GB in row format
  - "electron", "jet", "muon" → typical physics data: mixture of int, float, double
  - "Event Filter" → only boolean columns (mostly false), best compression
  - "Missing Energy" → table with float & double columns, worst compression

<table>
<thead>
<tr>
<th>Table: electron</th>
<th>jet</th>
<th>muon</th>
<th>Event Filter</th>
<th>Missing Energy</th>
<th>Total (all tables)</th>
<th>Size GB (all tables)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO MEMCOMPRESS</td>
<td>1.07</td>
<td>1.41</td>
<td>1.10</td>
<td>0.70</td>
<td>1.42</td>
<td>1.14</td>
</tr>
<tr>
<td>FOR DML</td>
<td>1.17</td>
<td>1.62</td>
<td>1.12</td>
<td>0.95</td>
<td>1.46</td>
<td>1.26</td>
</tr>
<tr>
<td>FOR QUERY LOW</td>
<td>1.39</td>
<td>2.11</td>
<td>1.06</td>
<td>21.75</td>
<td>0.94</td>
<td>1.51</td>
</tr>
<tr>
<td>FOR QUERY HIGH</td>
<td>1.60</td>
<td>2.77</td>
<td>1.30</td>
<td>30.08</td>
<td>1.49</td>
<td>1.82</td>
</tr>
<tr>
<td>FOR CAPACITY LOW</td>
<td>2.00</td>
<td>3.71</td>
<td>1.80</td>
<td>39.44</td>
<td>1.54</td>
<td>2.32</td>
</tr>
<tr>
<td>FOR CAPACITY HIGH</td>
<td>2.44</td>
<td>4.83</td>
<td>2.44</td>
<td>56.28</td>
<td>1.71</td>
<td>2.89</td>
</tr>
</tbody>
</table>
IMC Tests – Compression Rates

- For tables used in physics analysis – 115GB in row format
  - "electron", "jet", "muon" typical physics data: mixture of int, float, double
  - "Event Filter" only boolean columns (mostly false), best compression
  - "Missing Energy" table with float & double columns, worst compression

<table>
<thead>
<tr>
<th>Table:</th>
<th>NO MEMCOMPRESS</th>
<th>FOR DML</th>
<th>FOR QUERY LOW</th>
<th>FOR QUERY HIGH</th>
<th>FOR CAPACITY LOW</th>
<th>FOR CAPACITY HIGH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size GB (all tables)</td>
<td>1.07</td>
<td>1.14</td>
<td>1.12</td>
<td>0.70</td>
<td>1.42</td>
<td>1.26</td>
</tr>
</tbody>
</table>

- IMC Tests – Compression Rates
  - FOR DML
    - Size GB (all tables) 1.26
  - FOR QUERY LOW
    - Size GB (all tables) 1.51
  - FOR QUERY HIGH
    - Size GB (all tables) 1.82
  - FOR CAPACITY LOW
    - Size GB (all tables) 2.32
  - FOR CAPACITY HIGH
    - Size GB (all tables) 2.89

- NO MEMCOMPRESS
  - Size GB (all tables) 1.14

- FOR DML
  - Size GB (all tables) 1.26

- FOR QUERY LOW
  - Size GB (all tables) 1.51

- FOR QUERY HIGH
  - Size GB (all tables) 1.82

- FOR CAPACITY LOW
  - Size GB (all tables) 2.32

- FOR CAPACITY HIGH
  - Size GB (all tables) 2.89
IMC Tests – Population Time

- For tables used in physics analysis – 115GB in row format
- 32 populate processes
- SSD vs NAS storage

<table>
<thead>
<tr>
<th>Compression Level</th>
<th>Load Time SSD (s)</th>
<th>Load Time NAS (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO MEMCOMPRESS</td>
<td>3:24</td>
<td>35:57</td>
</tr>
<tr>
<td>FOR DML</td>
<td>3:28</td>
<td>36:04</td>
</tr>
<tr>
<td>FOR QUERY LOW</td>
<td>9:15</td>
<td>36:20</td>
</tr>
<tr>
<td>FOR QUERY HIGH</td>
<td>9:48</td>
<td>36:30</td>
</tr>
<tr>
<td>FOR CAPACITY LOW</td>
<td>20:20</td>
<td>48:34</td>
</tr>
<tr>
<td>FOR CAPACITY HIGH</td>
<td>26:57</td>
<td>49:24</td>
</tr>
</tbody>
</table>

Graph shows comparison of load times for SSD and NAS storage under various compression levels.
IMC Tests – Population Time

- For tables used in physics analysis – 115GB in row format
- 32 populate processes
- SSD vs NAS storage

<table>
<thead>
<tr>
<th>Compression Level</th>
<th>Load Speed SSD (MB/s)</th>
<th>Load Speed NAS (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO MEMCOMPRESS</td>
<td>576.7</td>
<td>54.5</td>
</tr>
<tr>
<td>FOR DML</td>
<td>565.6</td>
<td>54.4</td>
</tr>
<tr>
<td>FOR QUERY LOW</td>
<td>212.0</td>
<td>54.0</td>
</tr>
<tr>
<td>FOR QUERY HIGH</td>
<td>200.1</td>
<td>53.7</td>
</tr>
<tr>
<td>FOR CAPACITY LOW</td>
<td>96.4</td>
<td>40.4</td>
</tr>
<tr>
<td>FOR CAPACITY HIGH</td>
<td>72.8</td>
<td>39.7</td>
</tr>
</tbody>
</table>
IMC Tests – Population CPU

- **INMEMORY_MAX_POPULATE_SERVERS** parameter controls resources dedicated to IMC Store population
  - By default ½ of available CPU cores
IMC Tests – Physics Benchmark

• Query 1: „Electron Counter”
  • Count number of electrons meeting certain criteria

• Query 2: „Electron Filter”
  • Find good quality electron-positron pairs, calculate properties

• Query 3: „Interesting Events” (TTbar cutflow)
  • Find interesting collision events, meeting certain criteria

• Query 4: „Higgs Boson” (Higgs+Z)
  • Find collision events in which Higgs boson was produced
IMC Tests – Benchmark Query 1

- „Electron Counter”
- Big table scan (66GB) with filters

```sql
select count(*) from DATA12_8TEV."electron" where "pt">18000. and "eta"<1.5;
```

Execution Plan:

```
<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (CPU)</th>
<th>Time</th>
<th>Plan hash value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3317014141</td>
</tr>
<tr>
<td>1</td>
<td>SORT AGGREGATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>FX COORDINATOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>PK SEND QC (RANDOM)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>SORT AGGREGATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>FX BLOCK ITERATOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>TABLE ACCESS INMEMORY FULL</td>
<td>electron</td>
<td>6718K</td>
<td>51M</td>
<td>62386 (0)</td>
<td>00:00:00</td>
<td>1</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

6 - inmemory("pt">1.8E+004F AND "eta"<1.8E+004F)
   filter("pt">1.6E+004F AND "eta"<1.5E+004F)
```

In-Memory scan: electron TABLE ACCESS INMEMORY FULL

Fraction of a second
IMC Tests – Benchmark Results Q1

• „Electron Counter”

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Parallel 16 Execution</th>
<th>Serial Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row Format - Direct Path Read NAS</td>
<td>593.0</td>
<td>749.9</td>
</tr>
<tr>
<td>Row Format - Direct Path Read SSD</td>
<td>36.3</td>
<td>47.1</td>
</tr>
<tr>
<td>Row Format - Buffer Cache</td>
<td>2.3</td>
<td>34.3</td>
</tr>
<tr>
<td>IMC NO MEMCOMPRESS / DML</td>
<td>0.1</td>
<td>0.4</td>
</tr>
<tr>
<td>IMC FOR QUERY LOW / HIGH</td>
<td>0.1</td>
<td>0.4</td>
</tr>
<tr>
<td>IMC FOR CAPACITY LOW</td>
<td>0.1</td>
<td>0.5</td>
</tr>
<tr>
<td>IMC FOR CAPACITY HIGH</td>
<td>0.2</td>
<td>2.1</td>
</tr>
</tbody>
</table>
IMC Tests – Benchmark Results Q1

- "Electron Counter"

IMC vs Buffer Cache serial
86x faster!!!

IMC vs SSD Direct Path serial
118x faster!!!

IMC vs NAS Direct Path serial
1900x faster!!!
IMC Tests – Benchmark Query 2

- „Electron Filter”
- Big table scan with 2 self-joins and a join to time dimension
- physics analytic calculations of electron properties

```sql
select "RunNumber","EventNumber", el_sel_n,
    electron0."electron_i" as mu_id0, electron1."electron_i" as mu_id1,
    electron0."charge" as mu_charge0, electron1."charge" as mu_charge1,
    electron0."pt"/1000. as mu_pt0, electron1."pt"/1000. as mu_pt1,
    electron0."eta" as el_eta0, electron1."eta" as el_eta1,
    (case when abs(electron0."phi"-electron1."phi")<acos(-1.) then sqrt(Power(abs(electron0."phi"-electron1."phi"),2)
         +Power(abs(electron0."eta"-electron1."eta"),2)) else sqrt(Power(2.*acos(-1.) - abs(electron0."phi"-electron1."phi"),2)
         +Power(abs(electron0."eta"-electron1."eta"),2)) end) as DELTAR,
    ANALYSISTOOLS.PHYSANALYSIS.INV_MASS_LEPTONS(electron0."E",electron1."E",electron0."px",electron1."px",
    electron0."py",electron1."py",electron0."pz",electron1."pz")/1000. as INV_MASS
from DATA12_BTEV."periodAllYear_v47-pro13-01" 
    INNER JOIN (select "RunNumber","EventNumber", COUNT(*) as el_sel_n
              from "sel_electron_v" group by ("RunNumber","EventNumber")
              USING ("RunNumber","EventNumber")
    INNER JOIN "sel_electron_v" electron0 USING ("RunNumber","EventNumber")
    INNER JOIN "sel_electron_v" electron1 USING ("RunNumber","EventNumber")
where electron0."electron_i"<electron1."electron_i" and electron0."charge" != electron1."charge" and el_sel_n=2;
```
IMC Tests – Benchmark Query 2

- „Electron Filter”
- Big table scan with 2 self-joins and a join to time dimension
- physics analytic calculations of electron properties

In-Memory scan: electron
TABLE ACCESS
INMEMORY FULL

Combined with 2 index scans (buffer cache)
INDEX RANGE SCAN
IMC Tests – Benchmark Query 2

- „Electron Filter”
- Big table scan with 2 self-joins and a join to time dimension
- Physics analytic calculations of electron properties

In-Memory scan: electron
TABLE ACCESS INMEMORY FULL

Sub-query factoring – temporary table
LOAD AS SELECT (TEMP SEGMENT MERGE)
IMC Tests – Benchmark Results Q2

- „Electron Filter”

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Parallel 16 Execution</th>
<th>Parallel 16 with SubQ Factoring</th>
<th>Serial Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row Format - DP Read NAS</td>
<td>617.3</td>
<td>597.3</td>
<td>1260.5</td>
</tr>
<tr>
<td>Row Format - DP Read SSD</td>
<td>52.3</td>
<td>44.34</td>
<td>287.2</td>
</tr>
<tr>
<td>Row Format - Buffer Cache</td>
<td>31.3</td>
<td>21.9</td>
<td>251.5</td>
</tr>
<tr>
<td>IMC NO MEMCOMPRESS / DML</td>
<td>4.7</td>
<td>4.4</td>
<td>17.6</td>
</tr>
<tr>
<td>IMC FOR QUERY LOW / HIGH</td>
<td>4.7</td>
<td>4.4</td>
<td>17.6</td>
</tr>
<tr>
<td>IMC FOR CAPACITY LOW</td>
<td>8.3</td>
<td>4.7</td>
<td>17.8</td>
</tr>
<tr>
<td>IMC FOR CAPACITY HIGH</td>
<td>9.5</td>
<td>5.3</td>
<td>21.2</td>
</tr>
</tbody>
</table>

![Chart showing benchmark results for different configurations.](chart.png)
IMC Tests – Benchmark Results Q2

- "Electron Filter"

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Row Format - DP Read NAS</th>
<th>Row Format - DP Read SSD</th>
<th>Row Format - Buffer Cache</th>
<th>IMC NO MEMCOMPRESS / DML</th>
<th>IMC FOR QUERY LOW / HIGH</th>
<th>IMC FOR CAPACITY LOW</th>
<th>IMC FOR CAPACITY HIGH</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parallel 16 Execution</td>
<td>617.3</td>
<td>597.3</td>
<td>251.5</td>
<td>4.7</td>
<td>4.4</td>
<td>8.3</td>
<td>9.5</td>
</tr>
<tr>
<td>Serial Execution</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parallel 16 Execution with SubQ Factoring</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Serial Execution</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

IMC vs Buffer Cache serial
14x faster!!

IMC vs SSD Direct Path serial
16x faster!!

IMC vs NAS Direct Path serial
72x faster!!
IMC Tests – Benchmark Query 3

- "Interesting Events" (TTbar cutflow)
  - 3 scans of a big table with filters and aggregations
  - 2x self-join, 100GB of data scanned
  - Sub-queries converted into views for simplicity

```sql
create or replace view "alljets" as (select * from "RunNumber","EventNumber","jet","E","emscale_eta","emscale_phi","emscale_pt","Etascale","EtaOrigin","PhiOrigin","MOOrigin","EMIES_EtaCorr","eta","phi","pt","isBadLoose","BCH_CORR_JET","BCH_CORR_CELL","Ijvtx","fl_w_SV0","fl_w_JetFitterCOMBNM", ANALYSTSTOOLS.MV1.mv1EvalJava("fl_w_IP3D","fl_w_SV1","fl_w_JetFitterCOMBNM","pt","eta") as "MV1","E" as "correctedE","pt" as "correctedPt",0 as "isMC" from DATA12_8TEV."jet");
create or replace view "notlvjets" as (select * from "alljets" where abs("emscale_eta")<2.5 and "correctedPt">25000.0 and "correctedE">0.);
... create or replace view "badjets_count" as (select "RunNumber","EventNumber",COUNT(*) as N from "badjets" GROUP BY ("RunNumber","EventNumber"));

select "RunNumber",COUNT(*) as "GRL_Events", COUNT(case when 1=1 and "goodjets_count".N>=4 then 1 END) as "GRL_4goodl_events", COUNT(case when 1=1 and "goodjets_count".N>=4 and "badjets_count".N IS NULL then 1 END) as "GRL_4good1_noBad1_events", COUNT(case when 1=1 and "goodjets_count".N>=4 and "badjets_count".N IS NULL and "taggedjets_count".N>=1 then 1 END) as "GRL_4good1_noBad1_1Tag1_events" from DATA12_8TEV."periodAllYear_v47-pro13-01" LEFT OUTER JOIN "goodjets_count" USING ("RunNumber","EventNumber")
LEFT OUTER JOIN "taggedjets_count" USING ("RunNumber","EventNumber")
LEFT OUTER JOIN "badjets_count" USING ("RunNumber","EventNumber") group by "RunNumber";
```
IMC Tests – Benchmark Query 3

- “Interesting Events” (TTbar cutflow)
  - 3 scans of a big table with filters and aggregations
  - 2x self-join, 100GB of data scanned
  - Sub-queries converted into views for simplicity

---

3x In-Memory scan: jet TABLE ACCESS INMEMORY FULL

No In-Memory joins HASH JOIN RIGHT OUTER
IMC Tests – Benchmark Results Q3

- „Interesting Events”

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Parallel 16 Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row Format - DP Read NAS</td>
<td>1254.7</td>
</tr>
<tr>
<td>Row Format - DP Read SSD</td>
<td>190.0</td>
</tr>
<tr>
<td>Row Format - Buffer Cache</td>
<td>148.8</td>
</tr>
<tr>
<td>IMC NO MEMCOMPRESS / DML</td>
<td>136.1</td>
</tr>
<tr>
<td>IMC FOR QUERY LOW / HIGH</td>
<td>136.5</td>
</tr>
<tr>
<td>IMC FOR CAPACITY LOW / HIGH</td>
<td>136.5</td>
</tr>
</tbody>
</table>
IMC Tests – Benchmark Results Q3

• „Interesting Events”

IMC vs Buffer Cache
1.1x faster

IMC vs SSD Direct Path
1.4x faster

IMC vs NAS Direct Path
9.2x faster

Row Format - DP Read NAS
1254.7

Row Format - DP Read SSD
190.0

Row Format - Buffer Cache
148.8

IMC NO MEMCOMPRESS / DML
136.1

IMC FOR QUERY LOW / HIGH
136.5

IMC FOR CAPACITY LOW / HIGH
136.5

Parallel 16 Execution
IMC Tests – Benchmark Query 4

- „Higgs Boson” (Higgs+Z)
  - Join of 6 tables + 2 self joins
  - Over 200GB of data to be scanned
  - Very complex query with analytic calculations
  - Multiple layers of views for simplicity

```
create or replace view sel_EF_events as (select /*+ FULL("EF") */ "RunNumber","EventNumber"
  from DATA12_STEV_NAS."EF" where ("e24vhi_medium1"=1 or "e60_medium1"=1 or "e2e12Tvh_loose1"=1 or "mu24i_tight"=1 or "mu36_tight"=1 or "2mu13"=1));
...
create or replace view sel_muong_events as (select "RunNumber","EventNumber",ANALYSISSTOOLS.PHYSANALYSIS.INV_MASS_LEPTONS(
  muon0."E",muon1."E",muon0."px",muon1."px",muon0."py",muon1."py",muon0."pz",muon1."pz")/1000. as "DiMuonMass"
  from sel_mue_events INNER JOIN sel_muong muon0 USING ("RunNumber","EventNumber") INNER JOIN sel_muong muon1
  USING ("RunNumber","EventNumber") where muon0."muon_f"=muon1."muon_f" );

select "RunNumber","EventNumber","EventNumber","RunNumber","DiMuonMass","DiElectronMass","DisJetMass"
  from sel_muong_events FULL OUTER JOIN sel_electron_events USING ("RunNumber","EventNumber")
  INNER JOIN sel_jet_events USING ("RunNumber","EventNumber") INNER JOIN sel_MET_events USING ("RunNumber","EventNumber")
  INNER JOIN sel_EF_events USING("RunNumber","EventNumber")
  INNER JOIN DATA12_STEV_NAS."periodAllYear_v47-pro13-01" USING("RunNumber","EventNumber");
```
<table>
<thead>
<tr>
<th>VIEW</th>
<th>FILTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIN JOIN OUTER BUFFERED</td>
<td>MAIN JOIN BUFFERED</td>
</tr>
<tr>
<td>JOIN FILTER CREATE</td>
<td>JOIN FILTER CREATE</td>
</tr>
<tr>
<td>RX RECEIVE</td>
<td>RX RECEIVE</td>
</tr>
<tr>
<td>VIEW</td>
<td>VIEW</td>
</tr>
<tr>
<td>MAIN GROUP BY</td>
<td>MAIN GROUP BY</td>
</tr>
<tr>
<td>RX RECEIVE</td>
<td>RX RECEIVE</td>
</tr>
<tr>
<td>RX GROUP</td>
<td>RX GROUP</td>
</tr>
<tr>
<td>JOIN FILTER CREATE</td>
<td>JOIN FILTER CREATE</td>
</tr>
<tr>
<td>RX RECEIVE</td>
<td>RX RECEIVE</td>
</tr>
<tr>
<td>RX GROUP</td>
<td>RX GROUP</td>
</tr>
<tr>
<td>MAIN GROUP BY</td>
<td>MAIN GROUP BY</td>
</tr>
<tr>
<td>RX BLOCK ITERATOR</td>
<td>RX BLOCK ITERATOR</td>
</tr>
<tr>
<td>JOIN FILTER CREATE</td>
<td>JOIN FILTER CREATE</td>
</tr>
<tr>
<td>RX RECEIVE</td>
<td>RX RECEIVE</td>
</tr>
<tr>
<td>RX GROUP</td>
<td>RX GROUP</td>
</tr>
<tr>
<td>MAIN GROUP BY</td>
<td>MAIN GROUP BY</td>
</tr>
<tr>
<td>RX BLOCK ITERATOR</td>
<td>RX BLOCK ITERATOR</td>
</tr>
<tr>
<td>JOIN FILTER USE</td>
<td>JOIN FILTER USE</td>
</tr>
<tr>
<td>RX RECEIVE</td>
<td>RX RECEIVE</td>
</tr>
<tr>
<td>RX GROUP</td>
<td>RX GROUP</td>
</tr>
<tr>
<td>MAIN GROUP BY</td>
<td>MAIN GROUP BY</td>
</tr>
<tr>
<td>RX BLOCK ITERATOR</td>
<td>RX BLOCK ITERATOR</td>
</tr>
<tr>
<td>JOIN FILTER USE</td>
<td>JOIN FILTER USE</td>
</tr>
</tbody>
</table>

In-Memory scans: muon, electron, ...

TABLE ACCESS INMEMORY FULL

Bloom Filters for In-Memory join
JOIN FILTER CREATE

In-Memory join executed
JOIN FILTER USE

IMC Tests – Benchmark Query 4

- „Higgs Boson“ (Higgs+Z)
- Join of 6 tables + 2 self joins
- Over 200GB of data to be scanned
- Very complex query with analytic calculations
- Multiple layers of views for simplicity

In-Memory scans:

- muon, electron, ...
- TABLE ACCESS INMEMORY FULL

Bloom Filters for In-Memory join
JOIN FILTER CREATE

In-Memory join executed
JOIN FILTER USE
IMC Tests – Benchmark Results Q4

- „Higgs Boson“

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Parallel 16 Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row Format - DP Read NAS</td>
<td>3094.2</td>
</tr>
<tr>
<td>Row Format - DP Read SSD</td>
<td>197.8</td>
</tr>
<tr>
<td>Row Format - Buffer Cache</td>
<td>53.9</td>
</tr>
<tr>
<td>IMC NO MEMCOMPRESS</td>
<td>55.5</td>
</tr>
<tr>
<td>IMC FOR DML</td>
<td>32.3</td>
</tr>
<tr>
<td>IMC FOR QUERY LOW / HIGH</td>
<td>26.2</td>
</tr>
<tr>
<td>IMC FOR CAPACITY LOW / HIGH</td>
<td>27.6</td>
</tr>
</tbody>
</table>
IMC Tests – Benchmark Results Q4

- „Higgs Boson”

IMC vs Buffer Cache
2.1x faster!

IMC vs SSD Direct Path
7.6x faster!

IMC vs NAS Direct Path
118x faster!
IMC Tests – OLTP Benchmark

- OLTP schema from one of the physics databases (CMS)
  - High concurrency workload with ~300 simultaneous sessions
  - Mixed DML and SELECT queries – mostly with index access path
- Real Application Testing – 1h of peak activity captured
- Replay comparison: row format (BC) vs In-Memory Store
  - IMC with 3 compression levels: DML / QUERY HIGH / CAPACITY HIGH

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Replay Duration (s)</th>
<th>DB Time (s)</th>
<th>CPU Time (s)</th>
<th>Physical Reads (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROW FORMAT</td>
<td>1733</td>
<td>3172</td>
<td>2571</td>
<td>97.9</td>
</tr>
<tr>
<td>IMC FOR DML</td>
<td>1707</td>
<td>3427</td>
<td>2895</td>
<td>7.9</td>
</tr>
<tr>
<td>IMC FOR QUERY HIGH</td>
<td>1718</td>
<td>3449</td>
<td>2888</td>
<td>7.5</td>
</tr>
<tr>
<td>IMC FOR CAPACITY HIGH</td>
<td>2037</td>
<td>3929</td>
<td>3400</td>
<td>7.1</td>
</tr>
</tbody>
</table>
IMC Tests – OLTP Benchmark

- OLTP schema from one of the physics databases (CMS)
  - High concurrency workload with ~300 simultaneous sessions
  - Mixed DML and SELECT queries – mostly with index access path
  - Real Application Testing – 1h of peak activity captured
  - Replay comparison: row format (BC) vs In-Memory store

IMC with 3 compression levels:
- DML / QUERY HIGH / CAPACITY HIGH

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Replay Duration (s)</th>
<th>DB Time (s)</th>
<th>CPU Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row Format Only</td>
<td>1733</td>
<td>3172</td>
<td>2571</td>
</tr>
<tr>
<td>IMC for DML</td>
<td>1707</td>
<td>3427</td>
<td>2895</td>
</tr>
<tr>
<td>IMC for Query High</td>
<td>1718</td>
<td>3449</td>
<td>2888</td>
</tr>
<tr>
<td>IMC for Capacity High</td>
<td>2037</td>
<td>3929</td>
<td>3400</td>
</tr>
</tbody>
</table>

Configuration:
- ROW FORMAT
- IMC FOR DML
- IMC FOR QUERY HIGH
- IMC FOR CAPACITY HIGH
Physics Analysis NOW – with IMC

- In-memory Column Store provides significant performance boost for physics queries
- Very positive benchmark results

```
with "sel_electron" as (select /* PARALLEL(16) */ "electron_j", "RunNumber", "EventNumber", "E", "px", "py", "pz", "charge", "pt", "phi", "eta" from DATA12_STEV
where ("author"='1' or "author"='3') and "cl_etas"/"cos(eta)" > 10000. and (abs("cl_eta")>0 and abs("cl_eta")<2.47) and (abs("cl_eta")<1.37 or and BITANO("CQ",1446)>0 and abs("trackZ0pvnbiased") < 2. and "HighPF"='1')
    select "RunNumber", "EventNumber", "sel_electron0.electron_j", as mu_id0, electron1."electron_j" as mu_id1, electron0."charge" as mu_charge0, electron0."pt"/1000. as mu_pt0, electron1."pt"/1000. as mu_pt1, electron1."eta" as mu_eta1, electron1."phi" as mu_phi1, electron0."eta" as el_eta0, electron1."eta" as el_eta1, electron0."phi" as el_phi0, (case when abs(electron0."phi"-electron1."phi")=acos(-1.) then sqrt(Power(abs(electron0."phi"-electron1."phi"),2)+Power(abs(electron0."phi"-electron1."phi"),2)) else sqrt(Power(2.*acos(-1.) - abs(electron0."phi"-electron1."phi"),2)+Power(abs(electron0."eta"-electron1."eta"),2)) end) as DELTAR,
    ANALYSISSTOOLS.PHYSANALYSIS.INV_MASS_LEPTONS(electron0."E",electron1."E",electron0."px",electron1."px",electron0."py",electron1."py",electron0."charge",electron1."charge",as el_sel_n from "sel_electron" group by ("RunNumber","EventNumber")) USING
    INNER JOIN [select "RunNumber","EventNumber",COUNT(*) as el_sel_n from "sel_electron" group by ("RunNumber","EventNumber")) USING
    INNER JOIN "sel_electron" electron0 USING ("RunNumber","EventNumber") INNER JOIN "sel_electron" electron1 USING ("RunNumber","EventNumber")
where electron0."electron_j"<electron1."electron_j" and electron0."charge" is electron1."charge" and el_sel_n=2;
```

<10s
Conclusions

- In-Memory Column Store is both powerful and easy to use.
- Significant performance improvements of analytic-type queries on the same hardware:
  - On average 10x for fast SSD storage.
  - On average 100x for slower spindles.
- No negative impact on OLTP, high concurrency workloads.
- More data can go to memory due to the columnar format with compression:
  - Very low performance impact of higher compression levels.
Conclusions

Do we plan to use In-Memory Column Store at CERN?

Will definitely consider it in future projects!

Many good use-cases identified
Maaike: Maaike.Limper@cern.ch
Manuel: Manuel.Martin.Marquez@cern.ch
Emil: Emil.Pilecki@cern.ch

See also: http://db-blog.web.cern.ch/