Plan

- Introduce TrendMiner
- Discuss context & user needs
- TrendMiner 1.0
- TrendMiner 2.0 (with *Apache Ignite*)
- Challenges
- Future work
Empower process and asset experts with advanced analytics to Analyse, Monitor and Predict the operational performance of batch, grade and continuous manufacturing processes.

We democratise analytics by giving insights to the people who need answers: the engineers and operators in the plant.
TrendMiner
TrendMiner

Context & Scale
- > 300M points per time series
- 10-40K active time series
- Source of data is generally very slow!

User Expectations
- Time to first result < 1s
- Higher resolution
- More active time series
- More advanced analytics

Responsiveness

Overall performance
TrendMiner 1.0

Focus on responsiveness (making TrendMiner more interactive)
TrendMiner 1.0

- Streaming back to UI
- Fast for small queries
- Slow for big queries

- File-based
- Not scalable

- Not scalable
TrendMiner 2.0

Focus on performance (making TrendMiner more efficient)
TrendMiner 2.0 - Caching

Time Series

Time Slices

\[ \text{IgniteCache<Key, Data>} \]

startDate \((t_i)\)  
endDate \((t_i)\)  
timeSeriesId  
[ts0, value0]  
[ts1, value1]  
[ts2, value2]
# TrendMiner 2.0 - Caching

## Time Slices

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-06-04</td>
<td>16:20:15.165</td>
<td>Point timestamp</td>
</tr>
<tr>
<td>2019-06-04</td>
<td>16:20:15.165</td>
<td>Slice by hour</td>
</tr>
<tr>
<td>2019-06-04</td>
<td>16:20:15.165</td>
<td>Slice by day</td>
</tr>
<tr>
<td>2019-06-04</td>
<td>16:20:15.165</td>
<td>Slice by month</td>
</tr>
<tr>
<td>2019-06-04</td>
<td>16:20:15.165</td>
<td>Slice by year</td>
</tr>
</tbody>
</table>
TrendMiner 2.0 - Affinity

Time Series X

$S_1$ $S_2$ $S_3$ $S_4$ $S_5$ $S_6$ $S_7$

Time Series Y

$S_1$ $S_2$ $S_3$ $S_4$ $S_5$ $S_6$ $S_7$

$S_i$ (all Time Series) $\rightarrow$ Node$_j$

e.g.: 2019-06-04
TrendMiner 2.0 - Compute Grid

Existing algorithms

Chronological swipe

Data Point

Data Point (meeting criteria)

Scalable algorithms = *IgniteCompute*

A. Split search (affects N slices)

B. Scatter "jobs" = *affinityCall*

C. Chronological swipe (single slice)

D. Post-process partial results (e.g. merging)
TrendMiner 2.0 - Result

Streaming back to UI
Fast for small queries
Fast for big queries

TrendMiner

Algorithms

Scalable

Source
Challenge - Multi-level Prioritisation

Search dimensions

❖ Time Series (single vs multiple series)

❖ Search window (single vs multiple time slices)

❖ Algorithm (visualisation vs descriptive vs predictive analytics)
Challenge - Multi-level Prioritisation

Ignite Capabilities

❖ *PriorityQueueCollisionSpi* *(grid.task.priority)* = 1 dimension!

MultiLevelPriorityQueueCollisionSpi (custom implementation)

❖ Still use *grid.task.priority*
  ❖ Priority degression factor = #(Time Series) X #(Time Slices)
  ❖ Urgency via "Service Levels" (0...N) = 2nd dimension
Multi-level Prioritisation (Example)
Multi-level Prioritisation (Example)

Historical search = Service Level 1

Queued tasks have a priority (degression factor)
Multi-level Prioritisation (Example)

New computation (max urgency) = Service Level 0

Compute node
Compute thread
Job queue
Multi-level Prioritisation (Example)

New computation (max urgency) = Service Level 0

Compute node
Compute thread
Job queue
Multi-level Prioritisation (Example)

- Compute node
- Compute thread
- Job queue
Multi-level Prioritisation (Example)
Multi-level Prioritisation (Example)

Historical search = Service Level 1

- Compute node
- Compute thread
- Job queue
Multi-level Prioritisation (Example)

Historical search = Service Level 1

First task > priority (no degression factor applied)
Future Work

❖ Improve scheduling efficiency (predictable job runtime)

❖ Prevent job starvation (e.g. job-stealing SPIs)

❖ Make all algorithms scalable

❖ Pave way for Ignite Native Persistence
Thank you!

javi.carretero@trendminer.com