Black or White? How to Develop an AutoTuner for Memory-based Analytics

Mayuresh Kunjir and Shivnath Babu

Review by Ross Tooley
Background
Apache Spark

Bulk-synchronous parallel

Distributed on cluster
The optimisation problem

Input
- Program
- Data

Optimiser

Output

System-configuration parameters
- Containers per node
- Task concurrency
- Cache capacity
- Shuffle capacity
- New ratio
Existing Spark optimisers

- Guidelines for manual tuning
- Pure BO
  - Fekry et al.
- Genetic algorithms
  - Yu et al.
- Regression trees
  - Wang et al.
- And more...
Paper overview

Performance model

RelM

BO & RL

Evaluation
Design

RelM
Components

**Statistics**

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Containers per Node</td>
<td>1</td>
</tr>
<tr>
<td>M₁</td>
<td>Heap size</td>
<td>4404MB</td>
</tr>
<tr>
<td>CPUₘₑₜₚₜ</td>
<td>Average CPU usage</td>
<td>35%</td>
</tr>
<tr>
<td>Diskₘₑₜₚₜ</td>
<td>Average disk usage</td>
<td>2%</td>
</tr>
<tr>
<td>M₂</td>
<td>Code Overhead 90%ile value</td>
<td>115MB</td>
</tr>
<tr>
<td>M₃</td>
<td>Cache Storage 90%ile value</td>
<td>2300MB</td>
</tr>
<tr>
<td>M₄</td>
<td>Task Shuffle 90%ile value</td>
<td>0MB</td>
</tr>
<tr>
<td>M₅</td>
<td>Task Unmanaged 90%ile value</td>
<td>770MB</td>
</tr>
<tr>
<td>P</td>
<td>Task Concurrency</td>
<td>2</td>
</tr>
<tr>
<td>H</td>
<td>Cache Hit Ratio (the fraction of cached data partitions actually read from cache)</td>
<td>0.3</td>
</tr>
<tr>
<td>S</td>
<td>Data Spillage Fraction (the fraction of shuffle data spilled to disk)</td>
<td>0</td>
</tr>
</tbody>
</table>

**Enumerators**

<table>
<thead>
<tr>
<th>Task concurrency</th>
<th># containers</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1,1)</td>
<td>(1,2)</td>
</tr>
<tr>
<td>(2,1)</td>
<td>(3,1)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Initialiser**

\[
m_c = m_h \ast \min\left(\frac{M_c}{H + M_h}, 1 - \delta\right)
\]

\[
m_s = \min\left(\frac{M_s}{1 - \frac{S}{P}}, (1 - \delta) \ast m_h\right)
\]

\[
m_o = m_h \ast \frac{NR + 1}{NR} \ast m_e = m_h \ast \frac{1}{SR} \ast m_h
\]

\[
p_{CPU} = \frac{(1 - \delta) \ast 100}{n \ast CPU_{max} \ast P}, \quad p_{disk} = \frac{(1 - \delta) \ast 100}{n \ast Disk_{max} \ast P}
\]

\[
p_{norm} = \frac{(1 - \delta) \ast m_h}{M}, \quad p = \min(p_{CPU}, p_{disk}, p_{norm})
\]

**Arbitrator**

**Selector**

\[
U_C = \frac{M_1 + m_c + \rho \ast (M_{n} + m_2)}{m_{i_h}}
\]

Algorithm 1 ReM Arbitrator

**Input:** Configuration \(e = (M_c, M_p, m_c, m_e, m_{i_h})\), Safety factor \(\delta\)

1. if \((M_i + m_c) > (1 - \delta) \ast m_{i_h}\) then
2. return flagging insufficient memory
3. end if
4. while \((M_i + p \ast M_p + m_i) > m_{i_h}\) do
5. one of the following three in a round-robin manner:
6. I. Decrease \(p\) by 1 if \(p > 1\)
7. II. Reduce \(m_i\) by \(M_i\) ensuring that \(m_i > 0\)
8. III. Increase \(m_i\) by \(M_i\) ensuring \(m_i < (1 - \delta) \ast m_{i_h}\)
9. end while
10. Set shuffle memory \(m_i = \min(m_{i_h}, 0.5 \ast m_{i_h}/p)\)
11. Set output \(C = (M_c, M_p, m_c, m_e, m_{i_h})\)
Bayesian Optimisation and Reinforcement Learning
What’s the idea here?

1. Quasi-parameters

\[
q_1^x = \frac{M_l + \min(m_c^x, m_c) + p^x \times (M_u + \min(m_s^x, m_s))}{m_h^x},
q_2^x = \frac{M_l + m_c}{\min(m_0^x, m_c^x)},
q_3^x = \frac{p^x \times \min(m_s^x, m_s)}{0.5 \times m_e^x}
\]

\[
q^x = \{q_1^x, q_2^x, q_3^x\}
\]

2. Extend input space with q

\[
\{x_1, x_2, x_3, x_4, x_5, q_1^x, q_2^x, q_3^x\}
\]

3. Use RelM to predict quasi-parameters for each real configuration
Bayesian Optimisation

\[ GP(x, y) \]

\[ GP(x \cup q, y) \]
Comparison to BOAT

\[ GP(x \cup q, y) \]

Kunjir & Babu

BOAT

\[ GP(x, q_1) \quad GP(x, q_2) \quad GP(x, q_3) \]

\[ GP(q, y) \]
Reinforcement Learning (DDPG)
Results
Performance improvement

Figure 15: Runtime of every recommended configuration is scaled to the runtime of *MaxResourceAllocation*. Number of failed containers is shown on top of bars.
Convergence time

Figure 14: Training overheads of tuning policies. Number of iterations is shown on top of bars.
Looks good... any issues?

Overfitting
Comparisons
Comparing all optimisers

The general structure

Input → Modelling → Searching → Output
Contrasting search methods

- **Bayesian Optimisation**
  - BOAT
  - Kunjir & Babu
  - CherryPick

- **Reinforcement Learning**
  - DDPG
    - QTune
    - Kunjir & Babu
  - REINFORCE
    - Mirhoseini et al.
  - Thompson Sampling
    - Bao

- **Genetic**
  - REGAL

- **Population-Based Training**
  - Jaderberg et al.

- **Back-tracking**
  - TASO
Contrasting model methods

- **Expert model**
  - BOAT
  - TASO
  - QTune
  - Kunjir & Babu

- **Pure search-based**
  - CherryPick
  - Bao
  - Jaderberg et al

- **Auto-encoding**
  - REGAL (GNN)
  - Mirhoseni et al (FFN)