Spark Job Performance Analysis and Prediction Tool

Rekha Singhal
Problem Motivation

Application
Execution Time: 10 sec

Deployment

Development/Testing Environment

Production Environment

SLA: 60 sec

Application
Execution Time: 200 sec

After X years of deployment
Problem Solution

Development/Testing Environment

Production Environment
Cluster size, Data size, Platform configuration

Application Estimated Execution Time

Application

SLA: 60 sec

Performance Prediction Model

Application Estimated Execution Time
Spark Architecture

source:https://intellipaat.com/tutorial/spark-tutorial/spark-architecture/
Assumptions

- Development/Testing environment has at least one instance for each type of node in production environment

- Application representative small size data sets are available.

- Focus on 3 parameters - #executors, #cores per executor, ExecutorMemorySize

- Good network connectivity in the cluster
Approach

- Set up small size Spark cluster with one instance of each node type in production.
- Execute the application in development environment with given small data size.
- Collect Spark logs created during application execution.
- Parse the log and collect parameters used in the model.
- Build the prediction model using the collected measurements.
- Apply to model for give production environment – data size, Spark parameters and cluster size.
Application Execution on Spark

Application

Job1 → ... → JobN

S1 → Si → SM

E1 → ... → EK

T1 → ... → TL

K=Number of Executors

L=Number of cores per executor
Prediction of Application Execution Time

\[ \sum_{i=0}^{i=N} p_{Job_i} \]

\[ p_{Job_i} = JobStart_i + \sum_{j=0}^{j=SN_i} p_{Stage_i}^j + JobCleanup_i \]

‘i’th Job Estimated Execution Time in production environment

Estimated Execution time of ‘j’th stage of job ‘i’ in production
STAGE EXECUTION SIMULATOR for ESTIMATIONS !!
Stage Execution Behaviour

Spark Job Stages  Execution of tasks in an executor in stage Si
Stage Execution Behaviour

Spark Job Stages

Execution of tasks in an executor in stage Si

First Wave Tasks

Rest Wave Tasks

Task Execution Time

Core1
Core2
Core3
Core4

S0
S1
Si
SN
Task Execution Time Components

- Scheduler delay
- Serialization & de-serialization
- JVM Time
- Shuffle Time
- Computation Time
Task Execution Time Components

Scheduler delay - # Tasks, Task launch wave

Serialization & de-serialization - Block size

JVM Time - Processing type, Cores per executor, #executors per node

Shuffle Time - Data size per executor, Executor Memory

Computation Time - computation type, block size, data skew, heterogeneity
Task Computation Time Variability

![Graph showing task computation time variability with points at Bkt1 and Bktm.]
Performance Summary of Stage

- First Wave Average Scheduler Delay
- Rest Wave Average Scheduler Delay
- Number of tasks in each Bucket ‘p’
- Average computation time (duration) of each Bucket ‘p’
Task JVM Time Prediction

(a) Wordcount Application

(b) Terasort Application
Task Scheduler Delay Estimation

First Wave Scheduler delay increases linear to total number of tasks

Rest Wave Scheduler delay is same
Data Size per Task remains Same since Block Size same

\[
\text{ExecutorShuffleTime} = \text{AvgTskShuffleTime} \times \text{EstimatedExecutorTasks} + \text{SpillOverheads}
\]

Spilloverheads estimated by generating Spurious spills in constrained Development environment
Task Execution Time Estimation

Scheduler delay - prediction model

Serialization & de-serialization - from measurements

JVM Time - using prediction model based on measurements

Shuffle Time - prediction model

Computation Time - linear estimation of number of tasks in each bucket. Each bucket duration is average of tasks’ execution time in the bucket
Stage Execution Time Estimation

\[ p_{\text{Stage}_i^j} = \text{StageStart}_{i}^j + \max_{\text{on all cores}} \sum_{\text{TasksonCore}} \text{TaskExecutionTime} + \text{StageCleanup}_{i}^j \]
## Experimental Setup for Validation

<table>
<thead>
<tr>
<th>Configuration Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Executors</td>
<td>2, 4, 6</td>
</tr>
<tr>
<td>Number of cores per Executor</td>
<td>2, 4, 6</td>
</tr>
<tr>
<td>Executor Memory</td>
<td>4 GB</td>
</tr>
<tr>
<td>Data Size</td>
<td>10 GB, 20 GB</td>
</tr>
<tr>
<td>Cluster Size</td>
<td>2, 4</td>
</tr>
<tr>
<td>SQL1</td>
<td>Average on ‘lineitem’ column</td>
</tr>
<tr>
<td>SQL 2</td>
<td>Join of ‘lineitem’ and ‘order’</td>
</tr>
</tbody>
</table>
Model Validation: Wordcount

Wordcount: Actual vs. Predicted Execution Time

- Predicted Execution Time (ms) vs. Actual Execution Time (ms)
- The graph shows a linear relationship between the predicted and actual execution times.
Model Validation: Terasort
Model Validation: K-Means

K-Means: Actual vs. Predicted Execution Time

- Predicted Execution Time (ms)
- Actual Execution Time (ms)
Model Validation: SQL1 & SQL 2

**SQL1: Actual vs. Predicted Execution Time**

- Predicted Execution Time (ms)
- Actual Execution Time (ms)

**SQL2: Actual vs. Predicted Execution Time**

- Predicted Execution Time (ms)
- Actual Execution Time (ms)
Cost Model Accuracy

Average Prediction Error < 15%
Accuracy: ML Model vs Cost Model

Prediction Accuracy: ML Model vs Analytic Model

W*: Wordcount, T*: Terasort, K*: K-Means - 4 configurations on 20GB

- ML Model
- Analytic Model
Auto Tuning Algorithm

```
OptimizationModule(Input: DataSize, ClusterSize)
{
    Optimal_time = 9999;
    
    For Numexecutor = 1 to max cores in the Cluster do
        For Numcore_Executor = 1 to max cores on node do
            For Executormemory = Min Size to RAM size on node do
                If ValidConfiguration(cluster size, numexecutor, numcoreExecutor, Executormemory)
                    {
                        Time = PredictTimeModel(DataSize, ClusterSize, Numexecutor, NumcoreExecutor, Executormemory)
                        if Time < optimal_time
                            {
                                optimal_Numexecutor = Numexecutor
                                optimal_NumcoreExecutor = NumcoreExecutor
                                optimal_Executormemory = Executormemory
                                optimal_time = Time;
                            }
                    }
    }
}

Done
Done
Done
Return (optimal_Numexecutor, optimal_NumcoreExecutor, optimal_Executormemory)
```
Thank you

rekha.singhal@tcs.com