MapReduce Performance Model for Hadoop 2.x

Daria Glushkova, Petar Jovanovic, Alberto Abello

21-03-2017
Agenda

• Introduction
• Background
• Related Work
• Proposed Solution
• Evaluation
• Conclusions and Future Work
The objective is to develop an efficient algorithm to estimate two measures of interest:
• The mean response time of individual tasks;
• The mean response time for a job.
Background: Hadoop Architecture

Hadoop 1.x vs Hadoop 2.x

- Pig, Hive, Others
  - MapReduce
  - HDFS

- MR, Pig, Hive, Others
  - YARN
  - HDFS

Dynamic resource allocation!
The YARN module consists of three main components:

- Global ResourceManager (RM) per cluster
- NodeManager (NM) per node
- Application Master (AM) per application
Background: Job execution process in YARN
Background: Resource management in Hadoop 2.x

• AM figures out its own resource requirements.

• AM asks for specific resources via a list of ResourceRequests objects.

The ResourceRequest object consists of the following elements:
• Priority of the request
• Locality constraints (node, rack locality or any)
• Size of each container required for that request
• Number of containers

<table>
<thead>
<tr>
<th>Number of containers</th>
<th>Priority</th>
<th>Size</th>
<th>Locality constraints</th>
<th>Task type</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>20</td>
<td>x</td>
<td>n1</td>
<td>map</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>x</td>
<td>n2</td>
<td>map</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>x</td>
<td>*</td>
<td>reduce</td>
</tr>
</tbody>
</table>
Two groups of approaches:

- **Static**
  
  Do not take into account the queuing delay due to contention at shared resources.
  
  - Herodotou [1]; - ARIA [2]; - TETRIS [3]

- **Dynamic**
  
  Do not consider the synchronization delays due to precedence constraints among tasks that cooperate in the same job (map and reduce phases).
  
  Vianna et al.[4]

Two techniques:

- Mean Value Analysis (MVA)
- Markov Chains

Common limitation: Use a fixed amount of slots per map and reduce tasks within one node.
Proposed Solution: 
Main Challenge 

Adapt existing performance model to Hadoop 2.x taking into consideration the dynamic resource allocation
Proposed Solution: Steps

A1: Initialize
A2: Construct Precedence Tree
A3: Estimate the intra and inter job's overlap factors
A4: Estimate task response time
A5: Estimate the average job response time
A6: Apply Convergence Test

Converged?
Yes
End
No
Proposed Solution: A1
Initialization

a) Using sample techniques - taking the average of task response time from job profile.
b) Obtain from existing cost models that can capture unit costs of map and reduce tasks.
Proposed Solution: A2 Building precedence tree

Captures the execution flow of the job using two types of primitive operators:

- P
- S
Proposed Solution: A2
Building precedence tree

The core rules and assumptions in timeline construction (related to job scheduling):

• RM has a Capacity schedule;
• AM Lifecycle of map task:
• AM Lifecycle of reduce task:
• We ignore late binding;

\[\text{Diagram of lifecycle events: pending, scheduled, assigned, completed}\]
Proposed Solution: A2 Building precedence tree

Rules and assumptions in timeline construction (related to resource management):

• In the resource request object, containers can have different priorities. There is no cross-application implication of priorities. Map tasks have higher priority than reduce tasks.
• Consider node locality constraints for map task and ignore locality constraints for reduce tasks;
• Check for slow start. If there are enough completed maps, schedule reducers;

How to divide the timeline into the phases: each start or end of task indicates the start of a new phase.
Proposed Solution: A2 Building precedence tree

Resource Request Object
Proposed Solution: A3 Estimation of the Intra- and Inter-job overlaps factors

For a system with multiple classes of tasks the queuing delay of task $i$ class due to class $j$ task is directly proportional to their overlaps. Two types of overlap factors:

- Intra-job $\alpha_{ij}$
- Inter-job $\beta_{ij}$
There are 2 alternative approaches to estimate the job response time:

- **Tripathi-based**
  Assumption: execution time of all tasks have Erlang or Hyperexponential distributions.

- **Fork/join-based**

  Execution time: 
  \[ H_k \cdot \max(T_i, T_j), \]
  where 
  \[ H_k = \sum_{i=1}^{s} \frac{1}{i}, \]
  \( s \) - is the number of child nodes

  The precedence tree is a binary tree

  \[ H_k = \frac{3}{2}, \forall k \]
Proposed Solution: A5: Estimation of task response time

To solve the queueing network models we apply Mean Value Analysis (MVA) [7]. MVA is based on the relation between the mean waiting time and the mean queue size of a system with one job less.
Proposed Solution: A6: Applying convergence test

1: if ($|R_i^{curr} - R_i^{prev}| \leq \epsilon, \forall i = 1, \ldots, N$) then
2: Calculate the Performance Metrics of the Algorithm;
3: Exit;
4: else
5: for $i := 1$ to $N$ do
6: $R_i^{prev} = R_i^{curr}$;
7: end for
8: Go to the Precedence Tree Construction Procedure;
9: end if
Experimental setup

We performed a set of experiments analyzing the job response time in terms of the following parameters:

• number of nodes: 4, 6, 8;
• size of input data: 1GB, 5GB;
• number of jobs (wordcount, sort) that are executed simultaneously in the cluster: 1, 2, 3, 4;
• Tripathi-based, Fork/Join-based algorithms.
Evaluation I

Input: 1GB; #jobs: 1

Input: 1GB; #jobs: 4

Fork/Join-based: 11-13%

Tripathi-based: 19-22%
Evaluation II

Input: 5GB; #jobs: 1

Input: 5GB; #jobs: 4

Fork/Join-based: 13.5%

Tripathi-based: 23%
The biggest values of errors: 17% and 25% for Fork/join and Tripathi-based
Conclusions

• We tackled the challenge of creating MapReduce Performance model for Hadoop 2.x, which takes into consideration queuing delays and synchronization.

• The average error of job response time estimation for standard block size is in the range of 11% and 13.5% (accuracy improvements over the original model for Hadoop with error 15%)

• Our future plans focus on the tuning of provided performance model in order to decrease the error of job response time estimation.

• Furthermore, we are planning to adapt our model to Spark.
Thank you for attention!
References

Computer Science Department Duke University, p. 19.

and allocation for mapreduce environments. In Proceedings of the 8th ACM
international conference on Autonomic computing, pages 235{244. ACM, 2011


and U. Dayal. Analytical performance models for MapReduce workloads. International


with precedent constraints. IEEE Transactions on Parallel and Distributed Systems,
Implementation of modified MVA:
The implementation of this algorithm was done extending Java Modeling Tool. We implemented two approaches: Tripathi-based and Fork/join-based for job response time estimation. The representation precedence tree was implemented extending the Stanford CS Education Library for binary trees.
Complexity Analysis

Complexity of MVA algorithm +
(Complexity of Precedence tree construction) * numOfIterations

Complexity of MVA algorithm: $O(C^2N^2K)$

Computation cost for the whole solution:

$O(C^2N^2K) + O(((m+r(m+1)) \times (n \times \max(pMaxMapsPerNode, pMaxReducePerNode))) \times \text{numOfIterations})$

The computational cost of the whole solution is dominated by the MVA algorithm that has polynomial complexity.
Algorithm 2 Response Time Estimation
/* Estimates the mean response time of each task class, assuming that the overlap factors $\alpha_{ij}, \beta_{ij}$ are given, $\forall i, j$ */

[S1] Estimate the Average Response Time of class $j$ task in center $k$ when the task population is given by $\bar{N} - \bar{1}_i$
Initialize $R_{jk}(\bar{N})\forall j \in 1..C$; $k = 1..K$ - the residence response time of task class $i$ in the center $k$.

$$R_{jk}(\bar{N} - \bar{1}_i) \approx \left\{ \begin{array}{ll} R_{jk}(\bar{N}) - \left( \frac{1}{N} \alpha_{ji} \right) - \frac{N-1}{N} \beta_{ji} \cdot \frac{S_{jk} \cdot R_{jk}(\bar{N})}{\sum_{k=1}^{K} R_{k,k}(\bar{N})}, & \text{if } j \neq i; \\
R_{jk}(\bar{N}) - \beta_{ji} \cdot \frac{S_{jk} \cdot R_{jk}(\bar{N})}{\sum_{k=1}^{K} R_{k,k}(\bar{N})}, & \text{if } i = j; 
\end{array} \right.$$ 

[S2] Estimate the Mean Queue Length at each queuing center

$$Q_{jk}(\bar{N} - \bar{1}_i) \approx \left\{ \begin{array}{ll} \frac{N_j \times R_{jk}(\bar{N} - \bar{1}_i)}{\sum_{k=1}^{K} R_{jk}(\bar{N} - \bar{1}_i)}, & \text{if } i \neq j; \\
\frac{\beta_{ij} \times R_{jk}(\bar{N} - \bar{1}_i)}{\sum_{k=1}^{K} R_{jk}(\bar{N} - \bar{1}_i)}, & \text{if } i = j; 
\end{array} \right.$$ 

[S3] Estimate the Average Queue Length as seen by arriving task $i$

$$A_{ik}(\bar{N}) = \frac{1}{N} \sum_{j=1, j \neq i}^{C} \alpha_{ij} Q_{jk}(\bar{N} - \bar{1}_i) + \frac{N-1}{N} \sum_{j=1, j \neq i}^{C} \beta_{ij} Q_{jk}(\bar{N} - \bar{1}_i)$$

[S4] Estimate the Mean Response Time at each center

$$R_{ik}(\bar{N}) = S_{ik}(1 + A_{ik}(\bar{N}))$$

[S5] Estimate the Total Response Time

$$R_i(\bar{N}) = \sum_{k=1}^{K} R_{ik}(\bar{N})$$