SAMR: A self-adaptive MapReduce Scheduling Algorithm in Heterogeneous Environment

By Frank Chan
Goals

- MapReduce does not scale well in a heterogeneous environment
- Heterogeneous environments are distributed systems with nodes that vastly greatly in hardware
- SAMR: Self-adaptive MapReduce Scheduling Algorithm looks to accomplish this by using historical information for each node
Node classifications

• Mappers
  – Slow nodes
  – Fast nodes
• Reducers
  – Slow nodes
  – Fast nodes
• Each classification is based on the average of the total nodes for each type
• Map backup tasks are assigned on map fast nodes
• Reduce backup tasks are assigned on reduce slow nodes
Terms in the Paper

- Many tasks encompass 1 job
- There is only 1 DataNode and 1 NameNode

### Table I
CONCEPTIONS AND NOTIONS IN THIS PAPER

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NameNode</td>
<td>Records where data is stored</td>
</tr>
<tr>
<td>DataNode</td>
<td>Stores data</td>
</tr>
<tr>
<td>JobTracker(JT)</td>
<td>Manages MapReduce jobs</td>
</tr>
<tr>
<td>TaskTracker(TT)</td>
<td>Manages tasks</td>
</tr>
<tr>
<td>Job</td>
<td>MapReduce application</td>
</tr>
<tr>
<td>Map tasks(MT)</td>
<td>Tasks which run map function</td>
</tr>
<tr>
<td>Reduce tasks(RT)</td>
<td>Tasks which run reduce function</td>
</tr>
<tr>
<td>ProgressScore(PS)</td>
<td>Process score of a task</td>
</tr>
<tr>
<td>ProgressRate(PR)</td>
<td>Progress rate of a task</td>
</tr>
<tr>
<td>TimeToEnd(TTE)</td>
<td>Remaining time of a task</td>
</tr>
<tr>
<td>TrackerRate(TrR)</td>
<td>Progress rate of a TaskTracker</td>
</tr>
<tr>
<td>HISTORY_PRO(HP)</td>
<td>Weight of historical information</td>
</tr>
<tr>
<td>SLOW_TASK_CAP(STaC)</td>
<td>Parameter used to distinguish slow tasks</td>
</tr>
<tr>
<td>SLOW_TRACKER_CAP(STrC)</td>
<td>Parameter used to distinguish slow TTs</td>
</tr>
<tr>
<td>SLOW_TRACKER_PRO(STrP)</td>
<td>Maximum proportion of slow TTs</td>
</tr>
<tr>
<td>BACKUP_PRO(BP)</td>
<td>Maximum proportion of backup tasks</td>
</tr>
<tr>
<td>M1</td>
<td>Weight of first stage in MTs</td>
</tr>
<tr>
<td>M2</td>
<td>Weight of second stage in MTs</td>
</tr>
<tr>
<td>R1</td>
<td>Weight of coping data in RTs</td>
</tr>
<tr>
<td>R2</td>
<td>Weight of sorting in reduce tasks</td>
</tr>
<tr>
<td>R3</td>
<td>Weight of merging in reduce tasks</td>
</tr>
</tbody>
</table>
Progress Score Functions

• $M = \# \text{ of key/value pairs } T_i \text{ has processed}$
• $N = \# \text{ of key/value pairs } T_i \text{ has to still process}$

$$PS = \begin{cases} M/N & \text{For } MT, \\ 1/3 \times (K + M/N) & \text{For } RT. \end{cases}$$ (1)

$$PS_{avg} = \frac{1}{T} \sum_{i=1}^{T} PS[i]/T$$ (2)

For task $T_i$: $PS[i] < PS_{avg} - 20\%$ (3)

• Note: Progress score is a value from 0 to 1
Shortcomings of PS (Used by Late Scheduler)

• (1) Hadoop keeps values R1, R2, R3, M1, M2 but these can change depending on the hardware of the node, inherent to heterogeneous environments

• (2) Suppose $T_i$ needs 100 seconds to finish but has a progress score of 0.7 and $T_j$ needs 30 seconds to finish but has a progress score of 0.5  
  
  
  – If the $p_{\text{avg}} = 0.8$, by definition $T_j$ would get the backup task even though it doesn’t need it. (Time to complete is not accounted for)

• (3) On reduce tasks, if only 1 is considered slow and it’s the last one, assigning a backup task may not be needed (since it’s about to finish anyway)
SAMR: reading historical data and tuning parameters

<table>
<thead>
<tr>
<th>Algorithm 1 SAMR algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: procedure SAMR</td>
</tr>
<tr>
<td>2: input: Key/Value pairs</td>
</tr>
<tr>
<td>3: output: Statistical results</td>
</tr>
<tr>
<td>4: Reading historical information and tuning parameters using it</td>
</tr>
<tr>
<td>5: Finding slow tasks</td>
</tr>
<tr>
<td>6: Finding slow TaskTrackers</td>
</tr>
<tr>
<td>7: Launching backup tasks</td>
</tr>
<tr>
<td>8: Collecting results and updating historical information</td>
</tr>
<tr>
<td>9: end procedure</td>
</tr>
</tbody>
</table>

- Generally, R1 R2 R3 M1 M2 are generated statically by default
- Step 8 in the algorithm now dynamically changes these values
SAMR: reading historical data and tuning parameters

- SAMR uses the HISTORY_PRO (HP) to tune the parameters.
- If HP is close to 1, then SAMR doesn’t do anything, historical information is already kicking in.
- If HP is close to 0, then SAMR uses the collected task information to update the historical data (see Figure 2).
- Historical information includes M1, M2, R1, R2 and R3 which let’s the TaskTracker tune it’s own values accordingly (explained in the next section) and updates the node.

Figure 2. The way to use and update historical information
SAMR: How parameters are weighted for Map Tasks and Reduce Tasks

- Also, each value M1, M2... have a weight associated with it. For instance, M1 ~ 60% and M2 ~ 40% (Figure 3)
SAMR: How to find slow tasks

• A task is considered slow if $T_i$’s $PR_i$ is less than equation 6 where $STaC = SLOW\_TASK\_CAP$

• The average progress rate (APR) is evaluated by equation 7

$$PR_i < (1.0 - STaC) \times APR$$  \hspace{1cm} (6)$$

$$APR = \sum_{j=1}^{N} \frac{PR_j}{N}$$  \hspace{1cm} (7)$$
SAMR: How to find slow TTs

- SLOW_TRACKER_CAP (STrC) is used to classify fast and slow TaskTrackers
- Note: There is 1 TT per node
- Suppose there are N TTs
- For map tasks, the rate of the $i^{th}$ tracker is denoted as $TrR_{mi}$ (equation 8)
- Similarly, the average rate of all map task trackers is denoted as $ATrR_m$ (equation 10)
- Equations 12 and 13 determines a fast or slow TT (one for map and one for reduce tasks)

\[
TrR_{mi} = \sum_{i=1}^{M} PR_i / M \tag{8}
\]
\[
TrR_{ri} = \sum_{i=1}^{R} PR_i / R \tag{9}
\]
\[
ATrR_m = \sum_{i=1}^{N} TrR_{mi} / N \tag{10}
\]
\[
ATrR_r = \sum_{i=1}^{N} TrR_{ri} / N \tag{11}
\]
\[
TrR_{mi} < (1 - STrC) * ATrR_m \tag{12}
\]
\[
TrR_{ri} < (1 - STrC) * ATrR_r \tag{13}
\]
SAMR: when to launch backup tasks

- BACKUP_PRO (BP) defines the max proportion of backup tasks to all tasks
- BackupNum is the number of Backup Tasks
- TaskNum is the total number of tasks

\[ \text{BackupNum} < \text{BP} \times \text{TaskNum} \] (15)
SAMR: implementation details

- $N_f = \#$ key/value pairs already processed for a task
- $N_a = \#$ key/value pairs in total for a task

\[
SubPS = \frac{N_f}{N_a}
\]

For $MT$: $PS = \begin{cases} 
M_1 \times SubPS & \text{if } S = 0, \\
M_1 + M_2 \times SubPS & \text{if } S = 1.
\end{cases}$ \hspace{1cm} \text{(17)}$

For $RT$: $PS = \begin{cases} 
R_1 \times SubPS & \text{if } S = 0, \\
R_1 + R_2 \times SubPS & \text{if } S = 1, \\
R_1 + R_2 + R_3 \times SubPS & \text{if } S = 2.
\end{cases}$ \hspace{1cm} \text{(18)}
SAMR: implementation details

- At step 1, the PR’s and remaining time of task (TTE) is computed.
- During step 2, TT’s can assign backup tasks as needed/available.

Figure 4. Overview on SAMR, TT’s tries to launch new tasks first. If stack of new tasks is empty, they try to launch backup tasks for tasks in queue of slow tasks.
Experimental Results

- 5 algorithms tested
  - HADOOP w/o backup mechanism doubled
  - Regular scheduling in HADOOP
  - LATE (7%)
  - LATE w/ Historical Information (15%)
  - SAMR

Figure 9. The execute results of “Sort” running on the experiment platform. Backup mechanism and Historical information are all very useful in “Sort”. SAMR decreases time of execution about 24% compared to Hadoop.