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 vast 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

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

Progress Score Functions

- M = # of key/value pairs T_i has processed
- N = # of key/value pairs T_i has to still process

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

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

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

- 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_{savg} = 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

Algorithm 1 SAMR algorithm

```
1: procedure SAMR
2:     input: Key/Value pairs
3:     output: Statistical results

4:     Reading historical information and tuning parameters using it
5:     Finding slow tasks
6:     Finding slow TaskTrackers
7:     Launching backup tasks
8:     Collecting results and updating historical information
9: end procedure
```

- 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

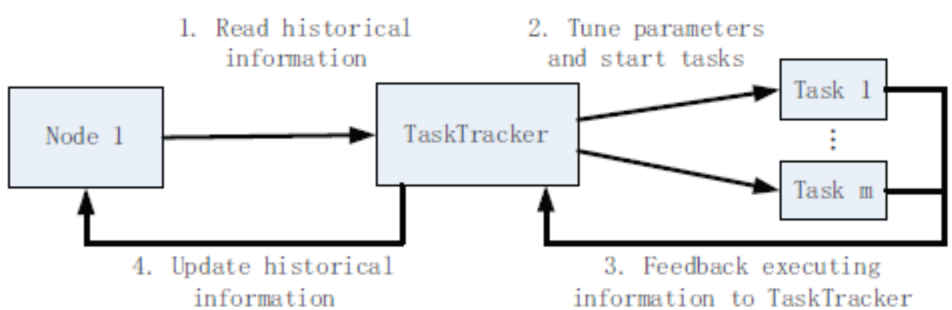

Figure 2. The way to use and update historical information

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)

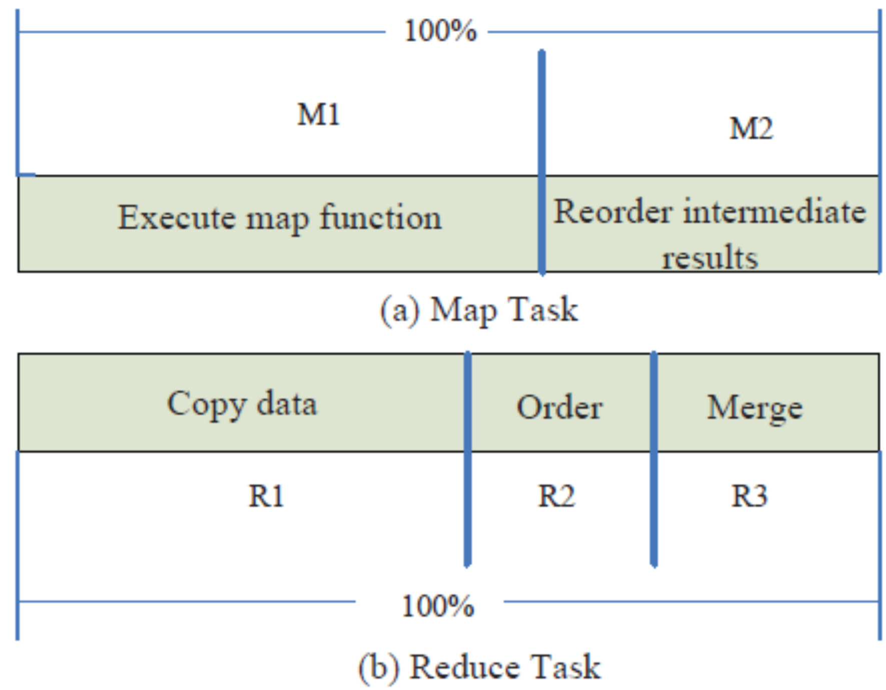

Figure 3. The two stages of *MT* and three stages of *RT*

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 evalued by equation 7

$$PR_i < (1.0 - STaC) * APR \qquad (6)$$

$$APR = \sum_{j=1}^{N} PR_j / N \qquad (7)$$

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)

$$TrR_{mi} = \sum_{i=1}^{M} PR_i/M \tag{8}$$

$$TrR_{ri} = \sum_{i=1}^{R} PR_i/R \tag{9}$$

$$ATrR_m = \sum_{i=1}^{N} TrR_{mi}/N \tag{10}$$

$$ATrR_r = \sum^{N} TrR_{ri}/N \tag{11}$$

$$TrR_{mi} < (1 - STrC) * ATrR_m \tag{12}$$

$$TrR_{ri} < (1 - STrC) * ATrR_r \tag{13}$$

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

$$BackupNum < BP * TaskNum \qquad (15)$$

SAMR: implementation details

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

$$SubPS = N_f / N_a \tag{16}$$

$$\text{For } MT\text{: } PS = \begin{cases} M1 * SubPS & \text{if } S = 0, \\ M1 + M2 * SubPS & \text{if } S = 1. \end{cases} \tag{17}$$

$$RT\text{: } PS = \begin{cases} R1 * SubPS & \text{if } S = 0, \\ R1 + R2 * SubPS & \text{if } S = 1, \\ R1 + R2 + R3 * SubPS & \text{if } S = 2. \end{cases} \tag{18}$$

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

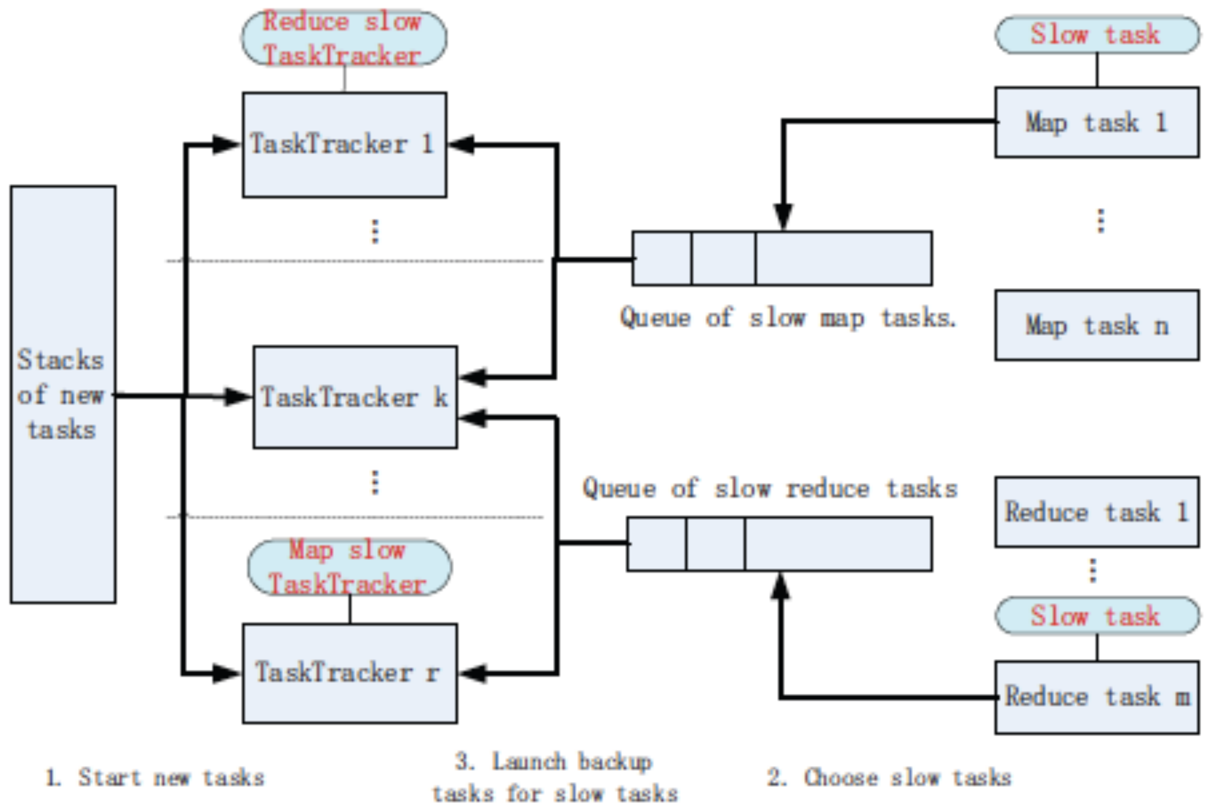

Figure 4. Overview on *SAMR*, *TTs* tries to launch new tasks first. If stack of new tasks is empty, they try to launch backup tasks for tasks in queue of slow tasks

Experimental Results

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

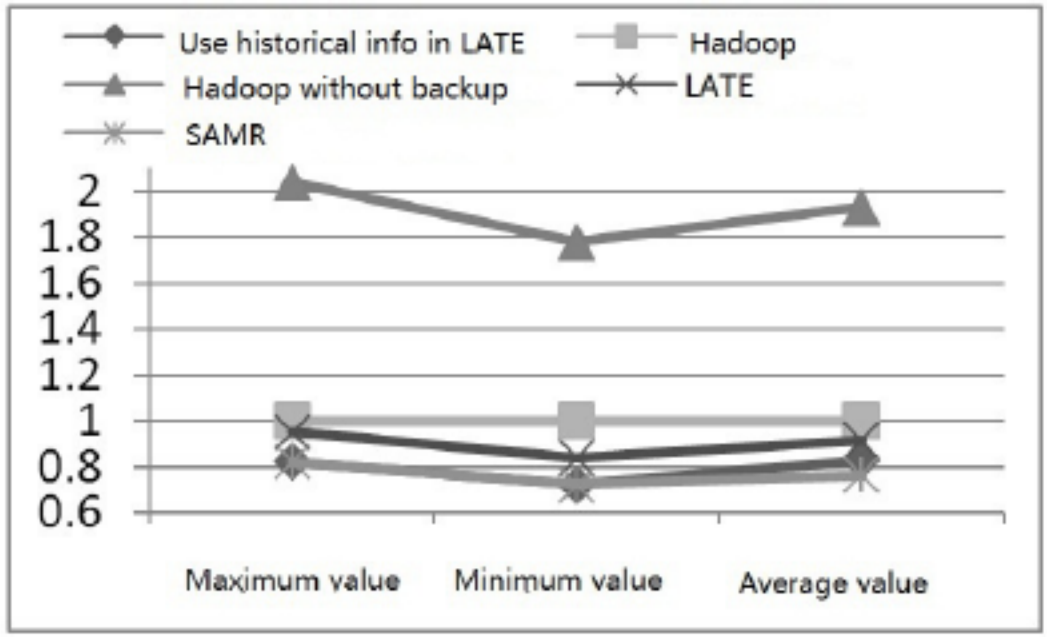

Figure 9. The execute results of "Sort" running on the experiment platform. Backup mechanism and Historical information are all very useful in "Sort". *SAMR* decreases time of execution about 24% compared to Hadoop.