International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)
Web Site: www.ijettcs.org Email: editor@ijettcs.org
Volume 6, Issue 4, July- August 2017 ISSN 2278-6856

AIMS: Adaptive Improved MapReduce Scheduling in Hadoop

[1] D.Vasudha, [2] K.IshthaqAhamed

[1]PG Student, Computer Science and Engineering Dept, GPREC, Kurnool (District), Andhra Pradesh-518004, INDIA.

[2] Associate Professor, Computer Science and Engineering Dept, GPREC, Kurnool (District), Andhra Pradesh-518004, INDIA.

Abstract

Hadoop is a complete eco-system of open source projects that provide us the framework to deal with big data. Let's start by brainstorming the possible challenges of dealing with big data (on traditional systems) and then look at the capability of Hadoop solution. This approach lowers the risk of catastrophic system failure and unexpected data loss, even if a significant number of nodes become inoperative. Consequently, Hadoop quickly emerged as a foundation for big data processing tasks, such as scientific analytics, business and sales planning, and processing enormous volumes of sensor data, including from internet of things sensors. As an important extension of Hadoop, SAMR and ESAMR MapReduce scheduling algorithms take heterogeneous environment into consideration. However, it falls short of solving the crucial problem – poor performance due to the large data sets in which it computes progress of tasks. Consequently, neither Hadoop nor ESAMR schedulers are desirable in heterogeneous environment. To this end, we propose AIMS: an Adaptive Improved Scheduling Algorithm, which calculates progressof tasks dynamically and splits the files into multiple chunks, based on user inputs and adapts to the continuously varying environment automatically. When a job is committed, AIMS splits the input file into multiple chunks so that the process will be completed easily, then assigns them to a series of nodes. Meanwhile, it reads historical information which stored on every node and updated after every execution. Then, AIMSadjusts time weight of each stage of map and reducetasks according to the historical information respectively. Thus, it gets the progress of each task accurately and finds which tasks need backup tasks. What's more, it identifies slow nodes and classifies them into the sets of slow nodes dynamically. According to the information of these slow nodes, AIMS will not launch backup tasks on them, ensuring the backup tasks will not be slow tasks any more. It gets the final results of the fine-grained tasks when either slow tasks or backup tasks finish first. The proposed algorithm is evaluated by extensive experiments over various heterogeneous environment. Experimental results show that AIMS significantly decreases the time of execution up to 19% compared with Hadoop's scheduler and up to 10% compared with ESAMR scheduler..

Keywords: Map Reduce, Scheduling algorithm, He terogeneous environment, Self-adaptive, aims

1. INTRODUCTION

MapReduce is a programming model for processing large data sets with a parallel, distributed algorithm on a cluster.
MapReduce refers to two separate and distinct task. The first is a MAP job, which takes a set of data and coverts it into another set of broken down data with keys and value pairs. The REDUCE job then takes the output from a map as input and combines those data sets into smaller data set sequence.hadoop Common is the Core, which contains common utilities and libraries that support the other Hadoop subprojects and is the file system shell.
Hadoop gets insights from massive amounts of data and now you can see when done right, Hadoop can have a great impact on your entire enterprise.
Hadoop Features:

- Distributed, scalable, fault tolerant, high throughput which is needed for Big Data processing.
- HDFS is designed to support very large files so files are split into blocks
- 3 replicas for each piece of data by default
- Can create, delete, copy, but NOT update
- Designed for streaming reads, not random access
- Data locality: processing data on or near the physical storage to decrease

This process is scalable to thousands of nodes and petabytes of data, which is pretty impressive.
In this paper, an improved algorithm is proposed which gives better performance than SAMR that is *AIMS: an Adaptive Improved Scheduling Algorithm*. *AIM S*significantly improve Map Reduce in terms of saving time of execution as well as system resources. *AIMS* is inspired by facts that slow tasks prolong the execution time of the whole job and nodes requires various time in accomplishing the same tasks due to their differences, such as capacities of computation and communication, architectures, memorizes and power. Although Hadoop and SAMR also launch backup tasks for slow tasks, they cannot find the appropriate tasks which are really prolong the execute time of the whole job because the two scheduler always use a static way to find slow tasks. On the contrary, *AIMS* incorporates historical information recorded on each node to tune parameters and find slow tasks dynamically. *AIMS* can find slow tasks which needbackup task really. In order to save system resources, *AIMS* classifies slow nodes into map slow nodes and reduce slow nodes further. *AIMS* defines fast nodes and slow nodes to be nodes which can finish a task in a shorter time and longer time than most other nodes. Map/reduce slow nodes means nodes which execute map/reduce tasks using a longer time than most other nodes. In this way, *AIMS* launches backup map tasks on nodes which are fast nodes or reduce slow nodes.

The improved featured of this papers are as follow:

- AIMSwill use the history of previou executed
- AIMSwill perform the map tasks in different stategs to check the slow tasks.
- AIMSwill analyze and identify the reduce node and map nodes which are taking long time to complete .

The entire paper is described asbelow. Section II Will describe themain concepts of current algorithms those are used in current Hadoop world and their minus points.. Section III introduces the *AIMS* and reports the implementation details. Section IV describes the experimental results. Section V draws the conclusion with pointing out our future work.

2. RELATED WORK

In order to understand the programming model that is the basis of *AIMS*, this section provides a brief view of MapReduce. It first introduces the preliminary knowledge about MapReduce and then overviews the related work.

A. Basic conceptions in MapReduce

Map Reduce is a programming model enabling a great many of nodes to handle huge data by cooperation. In traditional Map Reduce scheduling algorithm, a Map Reduce application needs to be run on the Map Reduce system is called a *"job"*. A *job* can be divided into a series of *"Map tasks"(MT)* and *"Reduce tasks"(RT).* The tasks whichexecute map function are called *"Map tasks"*, and which execute reduce function are called *"Reduce tasks"*.

In a cluster which runs Map Reduce, nodes were classified into *"Name Node"* and *"Data Node"* from data storage aspect. There is only one *Name Node*, which records all the information of where data is stored. thereare lots of *Data Nodes* which store data. There are only one *"Job-Tracker"(JT)* and a series of *"Task Tracker"(TT).Job Tracker* is a process which manages jobs. *Task Tracker* is a process which manages tasks on the corresponding nodes. Table I lists all the conceptions and notions used in this paper.

MapReduce scheduling system involves six steps when executing a MapReduce job, illustrated in Figure 1 [1].

First, user program forks the MapReduce job. Second, master distributes *MT* and *RT* to different workers. Third, *MT* reads in the data splits, and runs map function on thedata which is read in. Fourth, these *MT* write intermediate key/value pairs into local disks. Fifth, *RT* read the inter-mediate results remotely, and run reduce function on the intermediate results which is read in. At last, these *RT* write the final results into the output files.

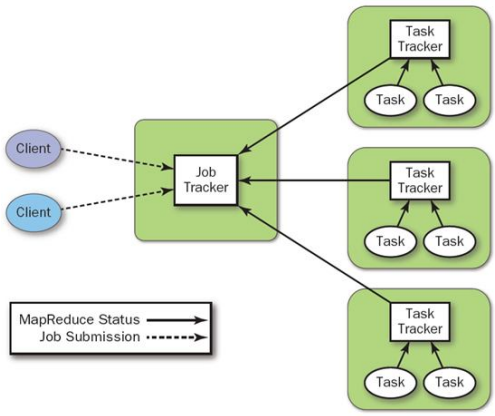


Figure 1. Overview of a MapReduce job

B. Map Reduce scheduling algorithm in Hadoop

Hadoop default scheduler starts speculative tasks based on a simple heuristic that compares each task's progress to the average task progress of a job.Hadoop chooses a task for it from one of three categories: Any failed tasks are given the highest priority. Non-running tasks are considered. For maps, tasks with data local to the node are chosen first.
Slow tasks that need to be executed speculatively are considered. To select speculative tasks, Hadoop monitors the progress of tasks using a Progress Score (PS) between 0 and 1. The average progress score of a job is denoted by PSavg.The Progress Score of the ith task is denoted by PS[i].
It supposes that the number of tasks which are being executed is T.
The number of key/value pairs need to be processed in a taskis N, the number of key/value pairs have been processed in the task is M, and the task has finished K stages (only for reduce task. There are three stages in a reduce task: copy data phase, sort phase and reduce phase). Hadoop gets PS according to the Eqs. 1 and 2 and launches backup tasks according to the Eq. 3.

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

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

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

If the Eq. 3 is fulfilled, Ti needs a backup task. The main shortcomings of this method include:

i. In Hadoop, the values of REDUCE1, REDUCE2, REDUCE3, MAP1, MAP2 are 0.33, 0.33. 0.34, 1 and 0 respectively. However REDUCE1, REDUCE2, REDUCE3, MAP1 and MAP2 are dynamic when tasks running on different nodes, especially in heterogeneous environment.
ii. Hadoop always launches backup tasks for those tasks of which PSs are less than P Savg20%. In this case, Hadoop may launch backup tasks for wrong tasks. For example, task Ti's PS is 0.7 and needs 100 seconds to

finish its work, while another task Tj's PS is 0.5, but only needs 30 seconds to finish its work. Suppose the average progress score P Savg is 0.8, the method will launch a backup task for Tj according to the Eq. 3. If we launch a backup task for Ti instead of Tj, it will save more time. What's more, tasks that PS is larger than 0.8 will have no chance to have a backup task, even though they are very slow and need a very long time to finish. This is because P Savg will never be larger than 1.

iii. Hadoop may launch backup tasks for fast tasks. For example, there are 3 RT Ri, Rj, Rk, and their PSs are 0.33, 0.66 and 0.66. In this case, P Savg = (0:33+0:66+0:66)=3 = 0:55. According to the Eq. 3, we should launch a backup task forRi. However, the second stage, sort stage, only needs a very short time in a real system. It is unnecessary to launch a backup task for Ri.

C. ESAMR Map Reduce scheduling algorithm

ESAMR Map Reduce scheduling algorithm always launches backup tasks for those tasks which have more remaining time than other tasks. Suppose a task T has run Tr seconds. Let PR denotes the progress rate of T, and TTE denotes how long time remaining until T was finished. ESAMR Map Reduce scheduling algorithm computes PR and TTE according to the Eqs. 4 and 5. PS in the Eq. 4 is computed according to the tasks, it often launches backup tasks for inappropriate tasks. This is because ESAMR cannot find TTE for all the running tasks correctly. One shortcoming of ESAMR, which is same to Hadoop, is the values of REDUCE1, REDUCE2, REDUCE3, MAP1, MAP2 are 0.33, 0.33. 0.34, 1 and 0 respectively.Another shortcoming ofESAMR MapReduce scheduling algorithm is it does not distinguish map slow nodes and reduce slow nodes. One node may executes MT quickly, but executes RT slower than most of other nodes. ESAMRjust considers one node either fast node or slow node, does not classify slow nodes further.

3..AIMS: AN ADAPTIVE IMPROVED SCHEDULING ALGORITHM

AIMS is designed with most important features required for mapreduce job execution. That leads to get the more accuracy of the job tracker. However, AIMS gets more accurate PSs of all the tasks by using historical information By using accurate PSs, AIMS finds real slow tasks and decreases more execute time compared with Hadoop an SAMR and ESAMR. Below Algorithm illustrates the process of AIMS. Sub section III-A describes how to read historical information and tune parameters using it. Subsection III-B describes how to find slow tasks. Subsection III-C describes how to find slow TTs. Subsection III-D describes when AIMS launches backup tasks. Subsection III-E describes the implementation details of AIMS.

Algorithm 1 AIMS algorithm

Fig.2 explains the entire flow of the AIMS algorithm step by step and explains the outcome of the algorithm.

. Implementation of AIMS

```
Aims()
{
Start_process()
CalculateWeightsMapTasks()
CalculateWeightsReduceTasks()
Node_heterogeneity()
FindSlowTasks()
CalculateProgressScore()
Backup_task()
End_process()
}
```

Figure 2.AIMS algorithm stages

This subsection introduces implementation details of AIMS. AIMS supposes the weight of "Execute map function" is MAP1, and the weight of "Reorder intermediate results" is MAP2, the weights of "copy data", "sort", and "merge" are REDUCE1, REDUCE2 and REDUCE3 respectively, which are illustrated in Figure.2

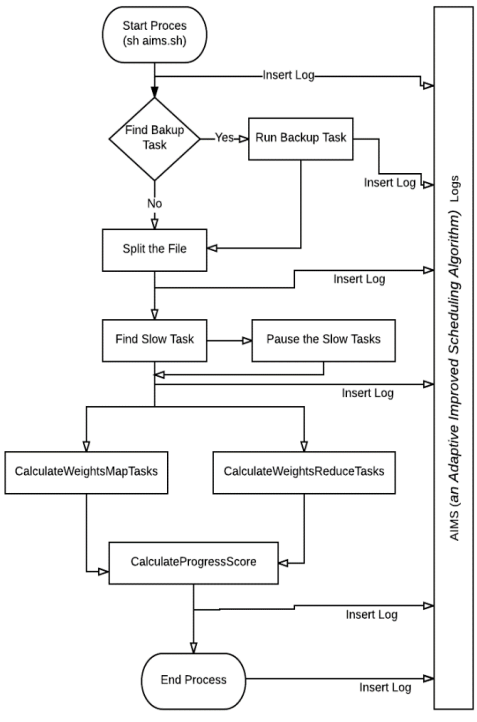
A. Start Process:

Aims algorithm will be invoked using aims.sh script with few parameters. Once the algorithm is invoked then all the reESAMRd methods and algorithm internal processes will be called and executed. Below is the syntax for aims algorithm and examples.

Once the command is executed then automatically all the processes will be called out to execute the given job (i.e. grep) and asks the users to proved the input split size that to be used for splitting the file. This increases the process speed

sh aims.sh <job><input_file><outpu_dir>
ex: sh aims.sh grep gprec.txt output2

B.Backup Tasks:



BackupTask method verifies any pending jobs are available that need to be executed before starting the actual job. If any pending jobs available then it resumes those jobs and then continue the actual processes as shown in Fig.2.

```
Backup_task()
for each pi=1..n
get process_status(pi)
if pi=fail
insert details(pi) into log_file
else
resume pi
```

AIMS uses a parameter, *HISTORY PRO(HP)*(range from of*MT* is 0, *REDUCE1*, *REDUCE2* and *REDUCE3* of *RT* are all 1/3.

If there are slow tasks, and Eq. 15 is fulfilled, a backup task can be launched when some of *TTs* are free. *BACKUP PRO(BP)*(range from 0 to 1) is used to definethe maximum proportion of backup tasks in all the tasks. Suppose the number of backup tasks is *BackupNum*, the number of all the running tasks is *TaskNum*. The Eq. 5 must be fulfilled in the system.

$$BackupNum < BP \quad TaskNum \tag{5}$$

Therefore, *MAP1* + *MAP2* = 1 and
REDUCE1 + *REDUCE2* + *REDUCE3* = 1.
MAP1, *MAP2*, *REDUCE1*, *REDUCE2* and *REDUCE3* are computed according to methodillustrated in Figure 2.
Suppose the number of key/value pairs which have been processed in a task is N_f , the number of overall key/value pairs in the task is N_a, the current stage of processing is *S* (limited to be 0, 1, 2), the progress score in the stage is*SubP S*.

```
Start_process():
if a job has completed PFM of its map tasks then
M1= CalculateWeightsMapTasks
M2=1-M1
end if
if a job has completed PFR of its reduce tasks then
< R1; R2 >= CalculateWeightsReduceTasks
R3=1-R1-R2
end if
slowTasks= FindSlowTask
run backup tasks for slowTasks
if a job has finished then
run K means algorithm to re-classify historical information into k clusters
end if
```

AIMS computes *PS* more accurate than Hadoop andESAMR, Because *MAP1*, *MAP2*, *REDUCE1*, *REDUCE2* and *REDUCE3* are tailored according to historical information. However, in Hadoop and ESAMR, *MAP1*, *MAP2*, *REDUCE1*, *REDUCE2* and *REDUCE3* are 1, 0, 0.33, 0.33, 0.34 respectively, which cannot adaptive to different environment. After getting exact *PS*, *AIMS* computes the remain time of all the running tasks, *T T E*, according to the Eq. 5. By this way, *AIMS* finds real slow tasks and launches backup tasks for these slow tasks on fast nodes of this kind of tasks consequently.

C. FindSlowTask:

```
FindSlowTasks()
set SlowTasks //a temp list to save all slow tasks
for each job that has completed PFM (or PFR) of its map (or reduce) tasks do
for each running task i of the job do
ProgressScore_i = CalculateProgressScore
ProgressRate_i = ProgressScore_i/Tr_i, where Tr_i is the time that has been used by the task
TTE_i= (1-ProgressScore_i)/ProgressRate_i
end for
ATTE=∑_{i=1}^{N} TTE_i/N , where N is the total number of running tasks of the job
for each running task i of the job do
if TTE_i-ATTE > ATTE * threshold then
slowTasks.add(ith task)
end if
end for
end for
return SlowTasks
```

D.CalculateWeightsMapTasks:
if a node has finished map tasks for the job then calculate tempMAP1 based on the job's map tasks completed on the node

```
1.  CalculateWeightsMapTasks()
2.  if a node has finished map tasks for the job then
3.  calculate tempM1 based on the job's map tasks completed on the node
4.  M1=randomly chosen first stage weight M1 from the corresponding node's history
5.  beta=abs(tempM1-M1)
6.  for each M1[i] 2 the node^0s history, i=1.2,...,K do
7.  if abs(M1[i]-tempM1)<beta then
8.  M1=M1[i]
9.  beta=abs(tempM1-M1[i])
10. end if
11. end for
12. return M1
13. else
14. M1=∑_{i=1}^{k} M1[i]/k
15. return M1
16. end if
```

E. CalculateweightsReduceTasks:
if a node has finished reduce tasks for the job then calculate tempREDUCE1 based on the job's reduce tasks completed on the node

```
CalculateWeightsReduceTasks()
  if a node has finished reduce tasks for the job then
  calculate tempR1 based on the job's reduce tasks completed on the node
  calculate tempR2 based on the job's reduce tasks completed on the node
  < R1; R2 >=a randomly chosen R1 and R2 pair from the node's history
  beta = abs(tempR1  r1) + abs(tempR2  r2)
  for each R1[i] and R2[i] pair in the node's history, i=1,2,..,K do
  if abs(R1[i]-tempR1)+abs(R2[i]-tempR2)<beta then
  R1=R1[i]
  R2=R2[i]
  beta=abs(R1[i]-R1)+abs(R2[i]-R2)
  end if
  end for
  return< R1; R2 >
  else
      29. P
  R1=∑_{i=1}^{K} R1[i]/K
  R2=∑_{i=1}^{K} R2[i]/K
  return< R1; R2 >
  end if
```

F. Calculate Progress Score
SubP S=Nf /Na, where Nf is the number of key/value pairs which have been processed in a sub-stage of a task and Na is the total number of key/value pairs to be processed in a sub-stage of the task.

If HP is too large (close to 1), parameters in the current tasks are depend on historical information seriously. Thus specific situation of current job is overlooked by AIMS. Meanwhile, if HP is too small (close to 0), the parameters of current task are depend on the finished tasks which in the same job seriously. AIMS uses historical information following 4 steps, illustrated in Figure 2.

```
CalculateProgressScore()
SubP S=N_f /N_a, where N_f is the number of key/value pairs which have been
processed in a sub-stage of a task and N_a is the total number of key/value
pairs to be processed in a sub-stage of the task
if the task is a map task then
if the map task is on the  rst sub-stage then
P S = M_1 * SubP S
else
P S = M_1+M_2 * SubP S
end if
end if
```

First, *TTs* read historical information from the nodes where they are running on. The historical information in-cludes historical values of *MAP1*, *MAP2*, *REDUCE1*, *REDUCE2* and *REDUCE3*. Then, *TTs* tune*MAP1*, *MAP2*, *REDUCE1*, *REDUCE2*and*REDUCE3* according to his to rical information, *HP*, and information collected from the current running system. Consequently, *TTs* collect values of *MAP1*, *MAP2*,*REDUCE1*, *REDUCE2* and *REDUCE3* according to the real running information after the tasks finished. Finally, *TTs* write these historical in formation back to the xml stored on the nodes (line 8 in algorithm 1).

In addition, every *TT* read historical in formation from node which it is running on. There is not any additional communication is needed when read and update historical information, So *AIMS* is scalable.

Figure 3 illustrates the divisions and weights of stages in *MT* and *RT*. In Hadoop and ESAMR,*MAP1*of *MT* is 1.0,*MAP2*

For *TTi*, if it fulfills the Eq. 12, it is a slow map *TT*. If it fulfills the Eq. 13, it is a slow reduce*TT*.

$$T\,rR_{mi} < (1ST\,rC)\;\; AT\,rR_m \qquad (12)$$

$$T\,rR_{ri} < (1ST\,rC)\;\; AT\,rR_r \qquad (13)$$

According to the Eqs. 11 and 12, if *STrC* is too small (close to 0),*AIMS* will classify some fast *TTs* into slow *TTs*. If*STrC*is too large (close to 1),*AIMS* will classify some slow *TTs* into fast *TTs*.

TASKSLOWFIND(STrP)(range from 0 to 1) is usedto define the maximum proportion of slow *TTs* in all the

According to the Eq. 6, if *STaC* is too small (close to 0), *AIMS* will classify some fast tasks into slow tasks. If*STaC* is too large (close to 1),*AIMS* will classify slow tasks into fast tasks.
TTs. Suppose the number of slow TTs is Slow Tracker Num, the number of all the TTs is Tracker Num. The Eq. 15 must be fulfilled in the system.

$$\text{SlowTrackerN um} < \text{ST rP T rackerN um} \quad (14)$$

If the Eqs. 12 and 14 are fulfilled at the same time, AIMS views the TT as a map slow TT.
AIMS can launch a backup task on *Task Tracker T* T_j fortask T_i only when T_i is a slow task which fulfills the Eq. 6, *T* T_j is not a slow *TT*, according to the Eqs. 12 and 13, and the number of backup tasks is less than the maximum number of backup tasks, which was got according to the Eq. 15.

AIMS schedules tasks in the following 3 steps, illustratedin Figure 4.
First, all the *TTs* obtain new tasks from stack of new tasks according to data locality property. Then, the *TTs* compute *PR* and *TTE* for all the tasks running on it. Next, the

algorithm finds which tasks are slow *MT* or slow *RT*. Consequently, these slow tasks are inserted into correspondent queue of slow tasks (queue of slow *MT* or queue of slow *RT*). Meanwhile, if stack of new tasks is empty, the *TT* tries to find slow task in the queue of slow tasks, and launches backup task. Only when the *TT* is not a map/reduce slow *TT*, it can launch backup tasks for *MT*/*RT*.

4. EVALUATION

We have verified the AIMS effectiveness using Hadoop 2.3.0 and jdk1.8 and two large data sets of 600MB each.

The configuration parameters for AIMS are given below:
Jobname: We can directly pass the job name like grep, wordcount,pietc to the aims.sh Inputname: we can mention the input file that can be used for processing Output directory: the directory where the jobs output files are placed

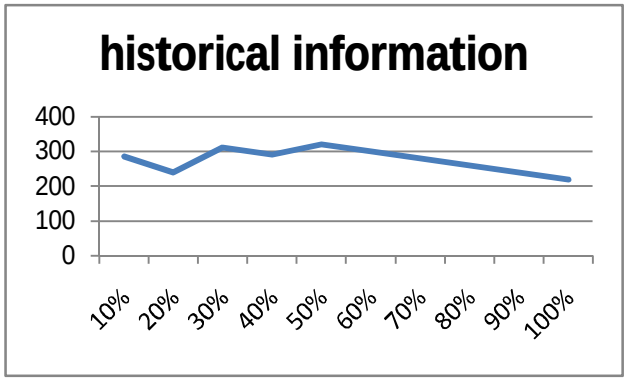


Figure 5. Affection of "HP" on the execute time, "HP" does not affect execute time too muchto the method mentioned in [20]. The detail experimental environment are showed in tableII.

A.Experimental environment

We establish experimental environment by using virtual machines on Ubuntu 16.04 and Oracle Virtual Box. This virtual Box is installed withThe version of JDK is 1.8.0.10, and the version of Hadoop is 2.3.0. The AIMS is implemented based on Hadoop 2.3.0. Because we cannot get the primary version of ESAMR MapReduce scheduling algorithm, so we implement a new one, according

Table II PROFILES OF EXPERIMENTS

No. of copies *TT*/node	m/r slots on *TT*s		Benchmarks
3	1	3/3	*"grep", "Sort","WordCount"*

The benchmarks used in these experiments are examples in Hadoop: "Grep", *"Sort"* and *"WordCount"*, because ESAMR also uses the two programs as benchmarks. The two benchmarks show key characteristic of MapReduce clearly.

B. Correctness of historical information

In order to verify the correctness of historical information used in *AIMS*, we list historical information and information collected from the real system in Figure 8. For either *MT* or *RT*, the weights of stages recorded in the historical information are not far from the weights collected from the real system. The weights of all the stages are far from the constant weights in Hadoop and ESAMR.

C. Performance of AIMS

In order to evaluate performance of AIMS, We compare performance of six different Map Reduce scheduling Backup mechanism and Historical information are all very useful in "Sort". AIMS decreases time of execution about 24% compared to Hadoop.
algorithm by running Sort and Word Count ten times each. The six algorithms are Hadoop without backup mechanism, scheduling algorithm in Hadoop, LATE, SAMR, ESAMR using historical information and AIMS. Parameters used in AIMS are gotten from experiments in subsection IV-B. primary slow RT, and hence saving a lot of time. By finding real slow tasks and launching backup tasks on fast nodes, AIMS has archived better performance than all the other MapReduce schedulers.
Figure 7 shows the efficiency of *AIMS* when running *Sort* benchmark. We uses the execute time of Hadoop as thebaseline, and finds that Hadoop without backup mechanism spends about double time in executing the same job. ESAMR decreases about 11% execute time, ESAMR using historical information mechanism decreases about 21% execute time, *AIMS*decreases about 22% execute time compared toHadoop. This is because *RT*spend a long time in *Sort*. Backup tasks on fast nodes for slow *RT* can finish in a shorter time than the

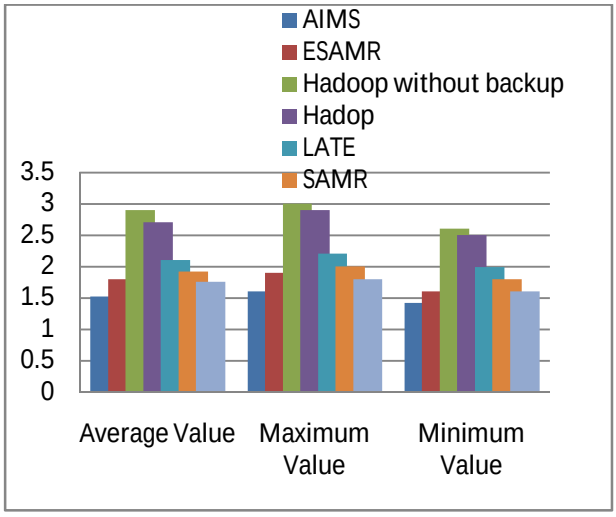


Figure 7. The execute results of "Sort" running on the experiment platform

Backup tasks and Historical information are not very useful in "WordCount". AIMS decreases time of execution about 21% compared to Hadoop

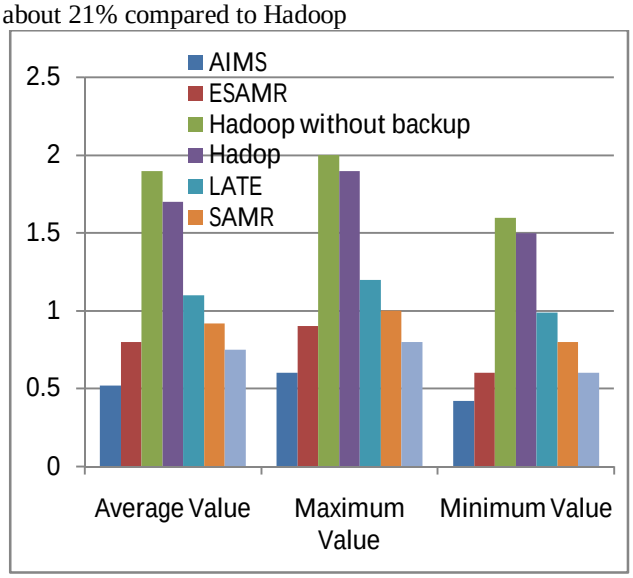


Figure 7. The execute results of "WordCount" running on the experiment platform.

In Figure 8, Hadoop without backup mechanism spends just a little more time than Hadoop when running *WordCount* benchmark. ESAMR scheduling algorithm can decrease about 10% , ESAMR scheduling algorithm with historical information mechanism can decrease about 16% execute time, *AIMS* can decrease about 19% execute time compared to Hadoop. This is because *RT* spends a shorter time in *WordCount* than *Sort* benchmark and backup tasks for slow tasks cannot save too much time.

5. CONCLUSION AND FUTURE ENHANCEMENTS

Here, in this paper, we have proposed and implemented *AIMS*: an Adaptive Improved Scheduling Algorithm, which uses file split file mechanism and classifies slow nodes into map and reduce slow nodes. The experimental results have shown the performance of the AIMS: an Adaptive Improved Scheduling Algorithm. The algorithm decreases the execution time of MapReduce jobs, especially in large data sets. The algorithm selects slow tasks and launch backup tasks accordingly while classifying nodes correctly, and saving a lot of system resources.

The proposed algorithm can be enhanced further for below activities.At initial stage, this algorithm focuses only on how to account for data locality when launching backup tasks, because data locality may remarkably acceler-ate the data load and store. Second, *AIMS* is considering a mechanism to incorporate that tune the parameters should be added. Third, *AIMS* will be evaluated on various platforms by first evaluated on rented Cloud Computing platform

REFERENCES

[1] J. Dean and S. Ghemawat, "Mapreduce: simplied data pro-cessing on large clusters," in OSDI 2004: Proceedings of 6thSymposium on Operating System Design and Implemention,(New York), pp. 137–150, ACM Press, 2004.

[2] J. Dean and S. Ghemawat, "MapReduce: a flexible data processing tool," Communications of the ACM, vol. 53, no. 1,72–77, 2010.

[3] J. Varia, "Cloud architectures," White Paper of Amazon,jineshvaria.s3.amazonaws.com/public/cloudarchitectures-varia.pdf, 2008.

[4] L. Barroso, J. Dean, and U. Holzle, "Web search for a planet: The Google cluster architecture," IEEE Micro, vol. 23, no. 2, pp22–28, 2003.

[5] L. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," ACMSIGCOMM Computer Communication Review, vol. 39, no. 1, 50–55, 2008.

[6] R. Buyya, C. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility," FutureGeneration Computer Systems, vol. 25, no. 6, pp. 599–616,2009.

[7] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in SOSP 2003: Proceedings of the 9th ACMSymposium on Operating Systems Principles, (New York, NY,USA), pp. 29–43, ACM, 2003.F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach Burrows, T. Chandra, A. Fikes, and R. Gruber, "Bigtable: A distributed storage system for structured data," in Proceed-ings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2006), 2006.

[8] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and Kozyrakis, "Evaluating mapreduce for multi-core and mul-tiprocessor systems," in HPCA 2007: Proceedings of the 2007IEEE 13th International Symposium on High Performance Computer Architecture, (Washington, DC, USA), pp. 13–24,IEEE Computer Society, 2007.

[10] M. de Kruijf and K. Sankaralingam, "Mapreduce for the cell b.e. architecture," tech. rep., Department of Computer Sciences, University of WisconsinCMadison, 2007.

[11] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang, "Mars: a mapreduce framework on graphics processors," in PACT 2008: Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, (NewYork, NY, USA), pp. 260–269, ACM, 2008.

[12] M. Schatz, "CloudBurst: highly sensitive read mapping with MapReduce," Bioinformatics, vol. 25, no. 11, p. 1363, 2009.

[13] S. Zhang, J. Han, Z. Liu, K. Wang, and S. Feng, "Spatial Queries Evaluation with MapReduce," in Proceedings of the2009 Eighth International Conference on Grid and Cooper-ative Computing-Volume 00, pp. 287–292, IEEE ComputerSociety, 2009.

[14] M. Zaharia, D. Borthakur, J. Sarma, K. Elmeleegy,Shenker, and I. Stoica, "Job scheduling for multi-user mapreduce clusters," tech. rep., Technical Report UCB/EECS-2009-55, University of California at Berkeley, 2009.
[15] C. Tian, H. Zhou, Y. He, and L. Zha, "A dynamic MapReduce scheduler for heterogeneous workloads," in Proceedings ofthe 2009 Eighth International Conference on Grid and Coop-erative Computing-Volume 00, pp. 218–224, IEEE ComputerSociety, 2009.
[16] P. Elespuru, S. Shakya, and S. Mishra, "MapReduce system over heterogeneous mobile devices," in Proceedings of the7th IFIP WG 10.2 International Workshop on Software Tech-nologies for Embedded and Ubiquitous Systems, pp. 168–179,Springer, 2009.
[17] C. Jin and R. Buyya, "MapReduce programming model for .NET-based distributed computing," in Proceedings of the15th European Conference on Parallel Processing (Euro-Par 2009), Citeseer, 2009.
[18] Yahoo,"Yahoo! hadoop tutorial." http://public.yahoo.com/gogate/hadoop-tutorial/start-tutorial.html.
[19] Hadoop, "Hadoop home page." http://hadoop.apache.org/.
[20] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving mapreduce performance in heteroge-neous environments," in 8th Usenix Symposium on OperatingSystems Design and Implementation, (New York), pp. 29–42,ACM Press, 2008.
[21] Xiaoyu Sun, "an enhanced self-adaptive mapreduce scheduling algorithm", Spring 5-4-2012

Author

D.Vasudha received the bachelor's. Degree in Computer Science and Engineering from Jawaharlal Nehru Technological University-Ananthapur and currently pursuing Master's in Computer Science from G.Pulla Reddy Engineering College, Kurnool, and A.P. Her research interests big data Analysis

Sri K.IshthaqAhamed is currently an Associate Professor in Dept.of Computer Science& Engineering in G.Pulla Reddy Engineering College, Kurnool, A.P and done his PhD from Rayalaseema University, Kurnool. His research interests include Artificial Intelligence and Soft Computing