Residual Traffic Based Task Scheduling in Hadoop

Daichi Tanaka
University of Tsukuba
Graduate School of Library, Information and Media Studies
Tsukuba, Japan
e-mail: s1421593@u.tsukuba.ac.jp

Masatoshi Kawarasaki
University of Tsukuba
Faculty of Library, Information and Media Science
Tsukuba, Japan
e-mail: mkawa@slis.tsukuba.ac.jp

Abstract— **In Hadoop job processing, it is reported that a large amount of data transfer significantly influences job performance. In this paper, we clarify that the cause of performance deterioration in the CPU (Central Processing Unit) heterogeneous environment is the delay of copy phase due to the heavy load in the inter rack links of the cluster network. Thus, we propose a new scheduling method -Residual Traffic Based Task Scheduling- that estimates the amount of inter rack data transfer in the copy phase and regulates task assignment accordingly. We evaluate the scheduling method by using ns-3 (network simulator-3) and show that it can improve Hadoop job performance significantly.**

Keywords- distributed computing; Hadoop; MapReduce; job performance; network simulation.

I. INTRODUCTION

Owing to the rapid reduction in hard disk drive (HDD) cost and the rapid expansion in the variety of services, the amount of data volume in the world is ever increasing. Along with it, there are ongoing efforts worldwide to extract useful information from a large amount of data (big data) [1]. Under these circumstances, MapReduce [2] was developed as a distributed processing framework for big data. In MapReduce, data processing is performed in two stages, namely map stage and reduce stage. By performing parallel distributed processing in each stage, big data can be processed at high speed. Hadoop [3] is an open source framework of MapReduce that implements the functionalities to deal with problems such as fault tolerance, load distribution and consistency. As a result, Hadoop removed many difficulties of conventional distributed processing, thus making many enterprises such as Facebook, Yahoo and New York Times to use it for site management or log analysis.

Since Hadoop has become popular, research on improving Hadoop performance is being actively carried out in pursuit of a more efficient scheduling scheme [4]-[9]. However, these studies have been focusing on scheduling computation and storage resources, while mostly ignoring network resources. In [10], it is reported that data transfer time may account for more than 50% of total job execution time. In this paper, we explore how data transfer affects Hadoop job performance. In particular, we analyze how network congestion deteriorates overall job performance and, based on this analysis, we propose enhancements to Hadoop scheduler by introducing the concept of residual traffic over the network.

For the study of Hadoop performance, we need to build a large-scale experimental environment. On the other hand, the scale of the production environment of Hadoop is very large, having several hundred to several thousand nodes [1]. Since building the actual size of the cluster is not realistic, many studies described above use cloud services such as Amazon Elastic Compute Cloud (Amazon EC2) [11] to construct the experimental environment. In the cloud service, although the node performance (e.g., CPU and HDD) and the minimum bandwidth of inter-node links are guaranteed, the network topology is opaque. Furthermore, resource usage may be affected by other users of the cloud. Accordingly, we have developed Hadoop cluster simulator using ns-3 (network simulator-3) [12] [13] so that we can set the size and the topology of a network freely.

Using this Hadoop cluster simulator, we perform Hadoop simulation using sort benchmark. Through this experiment, we clarify that the cause of performance deterioration in the CPU heterogeneous environment is the delay of copy phase due to the heavy load in the inter rack links of the cluster network. Based on this analysis, we propose a new scheduling method -Residual Traffic Based Task Scheduling- that estimates the amount of inter rack data transfer in the copy phase and regulates task assignment accordingly. We evaluate the proposed scheduling method and show that it can improve Hadoop job performance significantly.

The rest of this paper is organized as follows. Section II provides an overview of Hadoop architecture and mechanism. Section III and IV describe the Hadoop cluster simulator and experiment discussing performance issues. Based on this, Section V proposes residual traffic based task scheduling to improve Hadoop performance and Section VI discusses its effectiveness. Section VII concludes this paper.

II. HADOOP OVERVIEW

A. Hadoop Structure

Hadoop consists of MapReduce engine and Hadoop Distributed File System (HDFS) as shown in Figure 1. Hadoop cluster is made of one MasterNode and many SlaveNodes. The MasterNode and the SlaveNode communicate with each other using the HeartBeat mechanism. The MasterNode receives HeartBeat message from a

SlaveNode at a constant frequency. It includes the state information of the SlaveNode.

In MapReduce engine, the MasterNode is called JobTracker and the Slave Node is called TaskTracker. When JobTracker receives a job from a user, it divides it into small tasks and assigns them to TaskTrackers. When a TaskTracker requests a task assignment to JobTracker by sending a HeartBeat message, the JobTracker assigns a task in response to this so that the number of assigned tasks does not exceed the number of taskslots whose value is pre-determined according to the performance of each TaskTracker node.

In HDFS, the MasterNode is called NameNode and the SlaveNode is called DataNode. When NameNode receives data from a user, it splits it into small file blocks (called chunks) having 64MB in size, and distributes them to DataNodes. At this time, NameNode produces replica to improve fault tolerance and reachability.

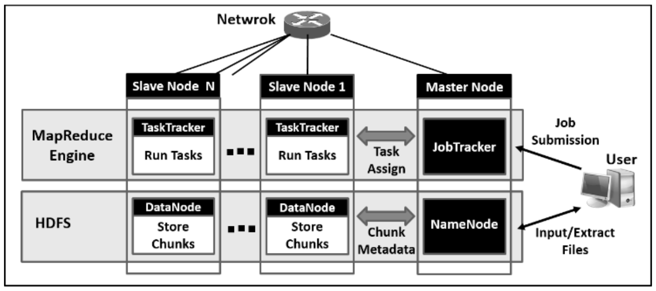

Figure 1. Hadoop Structure

B. MapReduce Procedures

MapReduce is a distributed processing framework for big data that processes data by using map function and reduce function that are defined by a user. Figure 2 shows MapReduce processing flow.

When a user submits a job, the JobTracker receives the input data split of the user data and generates map tasks. One map task is generated for one input data split. After the Job is turned on and the HeartBeat arrives at the JobTracker from a TaskTracker, the JobTracker assigns as many map tasks as the number of free map taskslot of the TaskTracker. In this case, a map task is assigned preferentially to the TaskTracker that holds the input data split to be processed. This map task is called a data local map task. If a map task is not data local, input data split needs to be obtained from other node over a network. After obtaining the input data split, the map function is executed. The result (map output) is stored in HDFS. When the map task is successfully completed, the TaskTracker sends a termination notice to the JobTracker.

When a part of map tasks of a given job (5% by default) are completed, JobTracker starts to assign reduce tasks to TaskTrackers. Reduce task collects all the map output having the same key over the network (copy phase) and performs reduce function. The results are stored in HDFS. When the reduce task is successfully completed, the TaskTracker sends a termination notice to JobTracker. JobTracker sends the Job completion notification to the user when it confirmed the end of all of tasks, and the job completes.

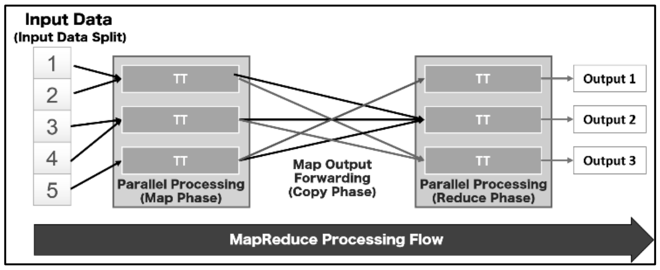

Figure 2. MapReduce Procedures

C. Hadoop Network I/O

During the MapReduce procedures, data transfer occurs in the following occasions.

1) *Transfer of input data split:* When assigned map task is not data local, input data split is obtained from other node. In this case, transmission of the size of input data split (by default 64MB) is generated.

2) *Transfer of map output data:* Reducer node receives all the relevant map outputs from mapper nodes. In this case, transmission of the total amount of map outputs is generated. We define this as resilient traffic in the later section of this paper.

3) *Transfer of reduce output replica:*

As reduce task output data (i.e., Job output data) is stored in the HDFS, transmission of copies corresponding to the number of replicas is generated. By default, three replicas are generated. One of the replicas is stored on the disk of the reducer node and the other two are stored in other nodes

III. HADOOP CLUSTER SIMULATOR

A. Design Principle

The objective of Hadoop cluster simulator is to provide experimental environment of Hadoop cluster network whose size and topology can be set freely. It simulates Hadoop job scheduling and task scheduling as defined in Hadoop version 1.1.2 and the three kinds of data transfer as described in Section II-C. We designed the simulator to keep track of cooperation with Hadoop scheduler behavior and network behavior.

B. Input Parameters

Although there are many parameters that can be set to actual Hadoop, the parameters that affect the performance of Hadoop are limited. The parameters used in our simulation are shown in Table I.

TABLE I. SIMULATION PARAMETERS

Category	*Component*
Job parameter	Scheduling method (FIFO, Fair), Job type (sort, search, index, etc.)
Configuration parameter	Number of taskslots, Chunk size, Number of replicas
Cluster parameter	Network topology, Number of nodes

Job parameters include Job Scheduling methods such as FIFO (First In First Out) Scheduling and Fair Scheduling, and Job types such as *sort*, *search* and *index*. They affect data transfer in the data generation timing and the amount of data. Specific values are shown in Table II and explained in section C.

The Configuration Parameters are the parameters that are set in the actual Hadoop configuration file. What affects data transfer is the block size when non-data local map task obtains input data split from other node and the number of replicas when the job output is copied to HDFS. The maximum number of task slots that is pre-allocated to each node affects the data transfer as well. If the number of slots is large, network traffic might occur in bursts and if the number of slots is not uniform among TaskTrackers, imbalance might occur in link bandwidth utilization.

The Cluster Parameters are the parameters that determine a network configuration of the cluster, such as the network topology and the number of nodes.

C. Modelling of Map Task and Reduce Task Processing

Key parameters that determine Hadoop performance are task processing time and output data size. We assumed the following:

1) Task Processing Time: Task processing time is determined only by the data size, not by the data content. Specific values are shown in Table II. These values were determined by reference to [12] and the actual measurement value in our experimental Hadoop cluster made of real machines [14].

2) Disk I/O Time: The time needed for disk I/O is negligibly small.

Output data size is calculated by "Filter" value. Filter means the ratio of output data size to input data size.

TABLE II. SPECIFIC PARAMETER VALUES

Parameters	*Job Type*		
	sort	*search*	*index*
MapFilter (%)	100	0-0.1	2-50
ReduceFilter (%)	100	100	2-50
MapProcessSpeed (sec/MB)	0.03	0.16	0.016
Reduce ProcessSpeed (sec/MB) (Sort phase)	0.016	0.016	0.016
Reduce ProcessSpeed (sec/MB) (Reduce phase)	0	0	0

Reduce task is divided into copy, sort and reduce phases. Reduce tasks can start when only some map tasks complete (by default 5%), which allows reduce tasks to copy map outputs earlier as they become available and hence mitigates network congestion. However, no reduce task can step into the sort phase until all map tasks complete. This is because each reduce task must finish copying outputs from all the map tasks to prepare the input for the sort phase.

The amount of data transfer in the copy phase is determined by "(map output data size) / (the number of reduce tasks)". As the reduce output (i.e., job output) will be copied to other nodes by the HDFS, the data transfer amount generated at replication depends on the size of the reduce task output.

D. Validation of Developed Simulator

To validate the accuracy of the developed simulator, we performed comparative experiments with our experimental Hadoop cluster made of Amazon EC2[14]. The comparison scenario is shown in Table III. Validation was focused on scheduler behavior and network I/O of TaskTrackers.

TABLE III. COMPARISON SCENARIO

Parameter	*Component*
# Nodes	JobTracker : 1, TaskTracker :20
# Task slots	map slot = 4, reduce slot = 4
Clunk Size	64 MB (by default)
Job Type	sort
# Job	1
Job Size	10 GB

1) Scheduler Behavior: We validated the accuracy of task scheduling for the followings:

a) Priority handling of Data Local Map Task: Basically, map tasks are processed in the order of task ID. However, if a particular map task is not data local, the assignment of that map task is skipped.

b) Concurrent Processing of Map Tasks: If the number of map tasks of a given job is greater than the total number of map task slots within a cluster, all the map tasks cannot be assigned in one cycle. In this case, map task assignment is achieved in several cycles.

c) Start of Reduce Task: Reduce task can starts to be assigned when a certain amount of map tasks are completed.

d) Phased Processing of Reduce Task: Sort and reduce phases cannot be started until copy phase is completed.

By comparing the task progress gantt charts of Hadoop EC2 cluster and the developed simulator as well as examining simulation logs, we confirmed that these behaviors are accurately simulated.

2) Network I/O of TaskTrackers: We examined whether three kinds of data transfer as described in Section II-C occur at suitable timing. In our experiment, we measured the throughput at the network interface of a TaskTracker that is assigned one non data local map task and some reduce tasks. The result is shown in Figure 3. From Figure 3 and experimental log, we confirmed the following:

i. Transfer of input data split occurs at the timing of non-data local map task assignment.
ii. Transfer of map output data occurs at the timing of reduce task assignment.
iii. Transfer of reduce output replica occurs at the end of reduce task.

From the above, we have confirmed that this simulator correctly simulates Hadoop cluster behavior.

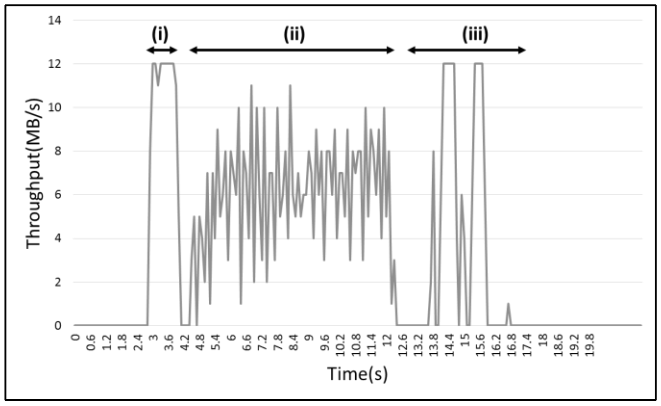

Figure 3. Network Input Throughput at a TaskTracker

IV. EXPERIMENT OF HADOOP PERFORMANCE

By using the developed simulator, we carried out Hadoop performance experiment.

A. Experiment Setting

The network model of experimental Hadoop cluster is shown in Figure 4. It has a tree structure (double-star type) that is common in many datacenters. It consists of 4 racks each of which includes 10 worker nodes. In total, there are 40 TaskTrackers (TTs) and one JobTracker (JT).

Each worker node is connected to Top of Rack switch (ToR-SW) by 1Gbps link. We call these links ToR-TT link in this paper. Each ToR switch is connected to Aggregate switch (AGG-SW) by 2.5Gbps link. As for the links between AGG-SW and Rack i, we call the uplink UP(i) and the downlink DOWN(i). Regarding CPU performance, we assumed heterogeneous environment. The CPU processing speed is faster in Rack 0, normal in Racks 1 and 2, and slower in Rack 3. The specific values are shown in Table IV. Conforming to CPU processing speed, the number of pre-allocated task slots in worker node is 8 for Rack 0, 4 for Rack 1 and 2, and 2 for Rack 3. We assumed homogeneous environment within each rack.

Job characteristics are summarized in Table V. In our experiment, a *sort* job having 5Gbytes was submitted every 5 seconds to the cluster. A total of 10 jobs were submitted in one experiment. We used sort benchmark that generates a large amount of output data. This is because we focused on the analysis of data transfer in Hadoop performance.

As for Job Scheduling, we used Fair Scheduler [5]. In Fair Scheduler, jobs are grouped into pools. When a slot needs to be assigned, the job that has been most unfair is scheduled. This time, we implemented to assign one pool for each job. Accordingly, all the running jobs are scheduled to obtain task slots equally.

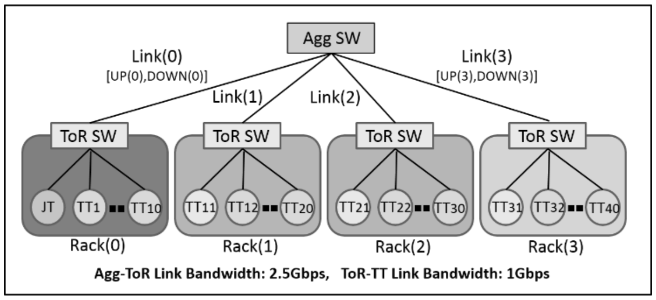

Figure 4. Experiment Cluster Architecture

TABLE IV. NODE PARAMETERS

Rack	*CPU performance*	*# Map/Reduce taskslot*	*Processing Speed (Relative Value)*
R0	Faster	8	2.0
R1, R2	Normal	4	1.0
R3	Slower	2	0.5

TABLE V. SIMULATION PARAMETERS

Category	*Component*	*Value*
Job Parameter	Scheduling Method	Fair Scheduling
	Job Type	Sort
	Job Size	5GB
Configuration Parameter	#Taskslots per TT	2~8
	Chunk Size	64MB(default)
	#Replication	3
Cluster Parameter	#TaskTracker (TT)	40

B. Experiment Results

Figure 5 shows the job execution time. We can see that job execution time becomes significantly long after Job5. Figure 6 shows the phase duration time of map task and reduce task (divided into copy phase and sort phase) of Job 0 and Jobs 7, 8 and 9. Regarding map task, we distinguish node local map task, rack local map task and remote map task.

For a given job, the meaning of phase duration time is as follows: Map Phase is the period from when the Job is turned on until the end of the final map task. Copy phase is the period from the start of the copy transmission of the first reduce task until the end of copy transmission of the last reduce task. Sort+Reduce Phase is the period from the beginning of reduce task that entered into the sort phase first until the end of the last reduce phase (i.e., the end of the job).

From Figure 6, we can see that Map Phase is very short and completed immediately in each job. On the contrary, copy phase is very long for Job 7, 8, 9 whose job execution time is long. Similarly, Job 7, 8, 9 take a very long time since Map Phase is finished to Sort+Reduce Phase begins, compared to Job 0.

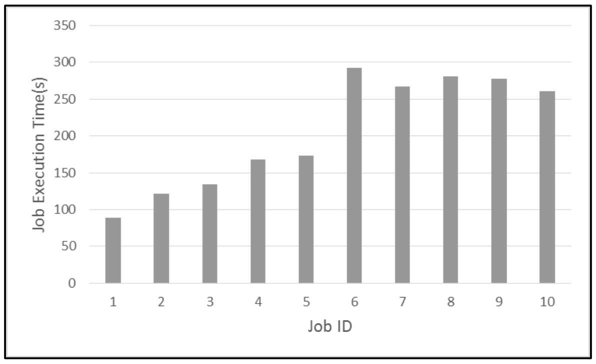

Figure 5. Job Execution Time per Jobs

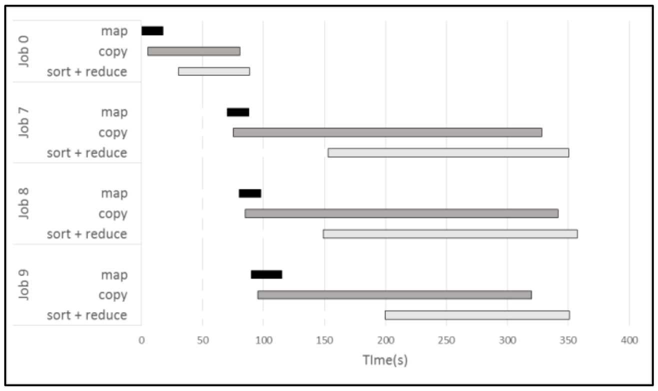

Figure 6. Phase Duration Time per Jobs（Job 0, Job 7, Job 8, Job9）

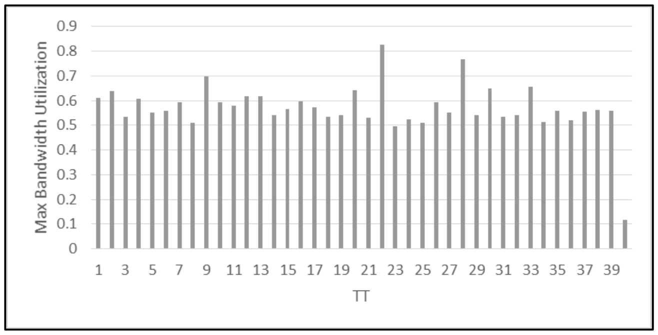

Figure 7. Max Bandwidth Utilization (TT-ToR)

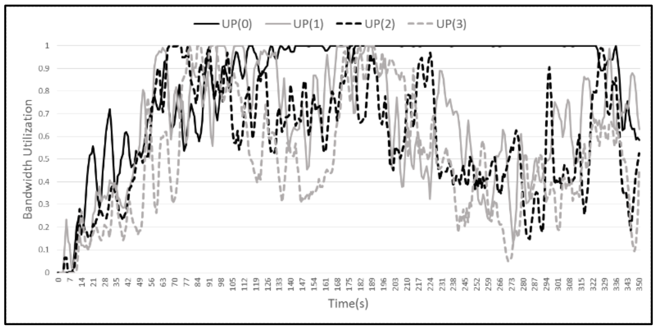

Figure 8. Uplink Bandwidth Utilization (UP)

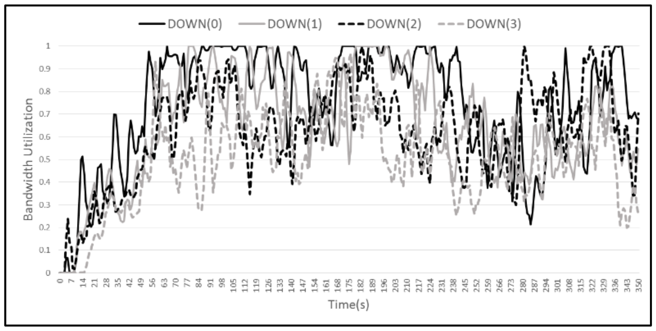

Figure 9. Downlink Bandwidth Utilization (DOWN)

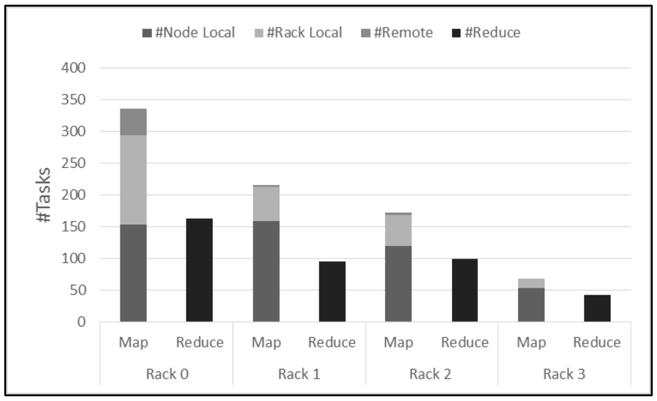

Figure 10. Number of Tasks per Rack

Next, in order to examine the factors that take time to copy phase, we measured the maximum utilization of the each TT-ToR downlink and the bandwidth utilization of UP(i) and DOWN(i) at AGG-ToR links. Each is shown in Figure 7, 8 and 9. From Figure 7, the maximum utilization of the TT-ToR links is 0.6 or at most 0.8. It can be seen that these links are underutilized through the Job execution time. By contrast, from Figure 8 and 9, UP(i) and DOWN(i) links are highly utilized. Especially, UP (0) link maintains 100% of utilization for a long period of time.

Figure 10 shows the number of processing Tasks per rack. Here, the Remote MapTask is a map task that is neither node local nor rack local. Since we are using slot-dependent scheduling scheme (a task is assigned immediately if a taskslot is opened), the number of processing tasks of each rack is proportional to the total number of slots allocated to the rack. From Figure 10, we can see that Rack 0 has processed a larger number of tasks compared to other racks. In particular, the number of map processing is remarkable.

Performance Deterioration Mechanisms

From the above experimental results, the performance deterioration is inferred to occur in the following manner. First, in CPU heterogeneous environment, there always is a rack having faster nodes than others. In map task, as most of the retention time of taskslot is CPU processing time, the faster rack processes a large amount of map tasks at high speed. It can be confirmed from Figure 10 that the map task processing is concentrated in Rack 0. This phenomenon is called "Task Stealing" [15]. If map tasks concentrate on a particular rack, a large volume of map output requests occur from inside and outside of the rack. As a result, the uplink of the faster rack becomes highly used. This can be confirmed by Figure 8. Bottleneck at UP (0) prolongs the copy phase of each reduce task, thus deteriorating Job performance.

From the above discussion, we conclude that the cause of performance deterioration in the CPU heterogeneous environment is the delay of copy phase due to the heavy load in the inter rack link. Inter rack data transfer is generated in the following three cases:

(1) Input data split transfer caused by Remote MapTask,
(2) Copy phase transfer caused by ReduceTask,
(3) Job output transfer caused by replication.

Among these, Remote MapTassks are very few, as seen from Figure 10. Replication is not under the control of HDFS and out of the scope of Task Scheduling. Thus, we propose a new scheduling method -Residual Traffic Based Task Scheduling- that reflects the inter rack data transfer in the copy phase in task scheduling.

V. Residual Traffic Based Task Scheduling

Based on the analysis described above, we propose enhancements to Hadoop task scheduling to improve Hadoop performance. Our proposal makes Hadoop scheduler aware of network congestion and regulates task assignment proactively. In this section, we propose the enhanced task scheduling algorithm.

A. Residual Traffic

To predict the inter rack link congestion status, we define residual traffic of Rack *i*. Before describing the residual traffic, we define the residual transmission amount and the residual reception amount. The residual transmission amount is the total amount of the map output data that was already generated by map tasks but have not being received by relevant reduce tasks. The residual reception amount is the total amount of the map output data that has not been received yet by relevant reduce tasks since their assignment.

Residual Up Traffic is the sum of the residual transmission amount of map output in each rack, and Residual Down Traffic is the sum of residual reception amount of running reduce tasks in each rack. Accordingly, ResidualUpTraffic (i) and ResidualDownTraffic (i) can be calculated as follows:

$$\text{ResidualUpTraffic(i)} = \sum_{j=1}^{number\_of\_map(i)} RemainData(MAP(i,j)) \quad (1)$$

$$\text{ResidualDownTraffic(i)} = \sum_{j=1}^{number\_of\_red(i)} RemainData(RED(i,j)) \quad (2)$$

Here, Map(i,j) is a map task that was assigned to Rack i in j-th order, RemainData (MAP) is the total amount of map output data of a given map task whose transmission is not completed, RED (i, j) is the ReduceTask that was assigned to Rack i in j-th order, and RemainData (RED) is the total amount of map output data that a given reduce task has not received yet. By calculating residual traffic, we can predict the load of inter rack links in the immediate future. If a rack has a large Residual Up Traffic, we can predict that the uplink from the rack is likely to congest. Similarly, if a rack has a large Residual Down Traffic, we can predict that the downlink to the rack is likely to congest.

TABLE VI. NUMBER OF ASSIGNABLE TASKS IN EACH REGULATORY LEVEL

Level	*Green*	*Yellow*	*Red*
#Assignable MapTasks	Available Map Slot	1	0
#Assignable ReduceTasks	1	0	0

B. Scheduling Algorithm

In the residual traffic based scheduling, JobTracker monitors the inter rack link bandwidth utilization, and if the usage has exceeded the threshold level, it adjusts the regulatory level in task assignment of each rack (i.e., UPRegulatoryLevel and DownRegulatoryLevel) referring to the residual transmission amount and/or residual reception amount. We discriminate three stages of regulatory level (Green, Yellow and Red) depending on the combined status of link utilization and residual traffic amount, so that we can change the way of task assignment accordingly. Table VI shows the number of tasks that can be assigned in each of the regulatory levels.

- *Regulatory level (Green):* Normal Task Scheduling. No regulation is applied.
- *Regulatory level (Yellow):* MapTask can be assigned one at most. ReduceTask cannot be assigned at all.
- *Regulatory level (Red):* No Map task or Reduce Task can be assigned.

Regulatory level is updated at every HeartBeat communication. Update algorithm for UPRegulatoryLevel of Rack *i* is shown below. DownRegulatoryLevel is updated similarly.

Algorithm Regulatory Level Update Algorithm

```
WHEN JobTracker receive a HeartBeat from a TaskTracker in Rack i
  IF UPRegulatoryLevel(i) = Green   THEN
    IF UpBandUsage(i) > StartTH
      IF ResidualUpTraffic(i) > Th_yr THEN
        Set UPRegulatoryLevel(i) to Red
      END IF
      IF ResidualUpTraffic(i) < Th_yr THEN
        Set UPRegulatoryLevel(i) to Yellow
      END IF
  END IF
END IF
  IF UPRegulatoryLevel(i) = yellow or red THEN
    IF UpBandUsage(i) < EndTh THEN
      Set UPRegulatoryLevel(i) to Green
    END IF
  END IF
END WHEN
```

Figure 11. Regulatory Level Update Algorithm

In the above, UpBandUsage(i) is the uplink bandwidth utilization of Rack i between ToR Switch and Aggregation Switch, DownBandUseage(i) is the downlink bandwidth

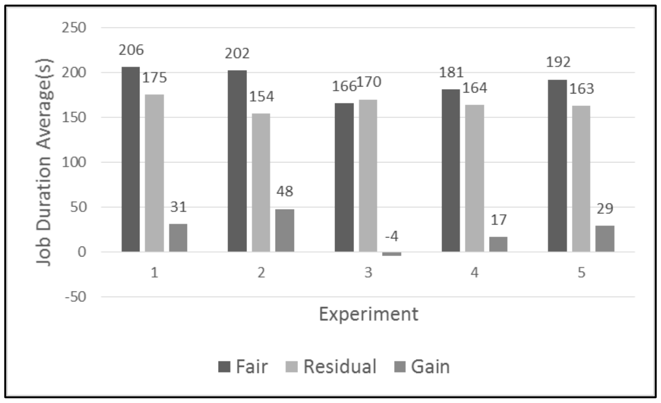

Figure 12. Average Job Execution Time in Each Experiment

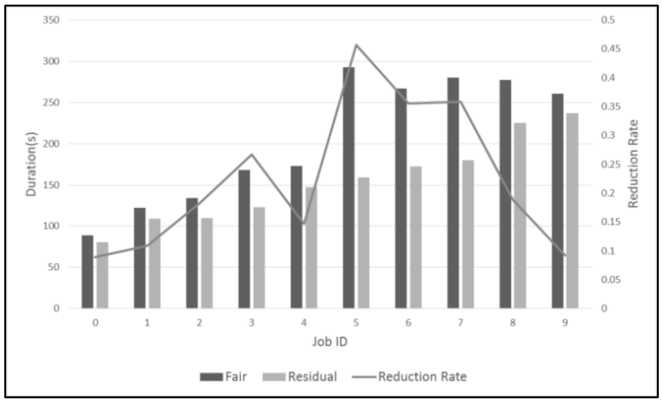

Figure 13. Job Execution Time Per Jobs

TABLE VII. RESIDUAL TRAFFIC LOAD BASED SCHEDULING PARAMETER

Parameter	*Value*
StartTH	0.8
EndTH	0.6
Th_yr	3.0

utilization of Rack i between ToR Switch and Aggregation Switch, StartTH is the bandwidth utilization threshold to start regulation, EndTH is the bandwidth utilization threshold to exit regulation, and Th_yr represents the boundary of residual traffic between strong regulation (Red) and weak regulation (Yellow).

StartTH and EndTH affect the strength and the duration of regulation. If StartTH is low, the regulation is easily invoked. If the difference between StartTH and EndTH is large, the duration of the regulation becomes long, because once the regulation is turned on, it is less likely to off. Th_yr also is a parameter related to the strength of the regulation. If Th_yr is low, regulatory level is likely to be Red and strong regulation is applied. Th_yr is a parameter that needs to be properly adjusted according to the execution environment.

VI. EVALUATION

In this section, we evaluate the Residual Traffic Based Scheduling described in Section V. In the experiment, we used the Hadoop cluster simulator described in Section III. We performed simulation five times by changing only the Scheduling scheme in the same scenario as preliminary experiments. The parameters used in the Residual Scheduling are shown in Table VII. The parameter values in Table VII were determined empirically to maximize the effectiveness of the proposed method in this execution scenario.

A. Experiment Results

1) Job Execursion Time: After performing the experiments five times, we took the average of Job execution time for each Scheduling method. The result is shown in Figure 12. The average reduction in Job execution time was 24.2s and the reduction rate was about 13%. The maximum reduction in Job execution time was 48s and reduction rate was 23%. From

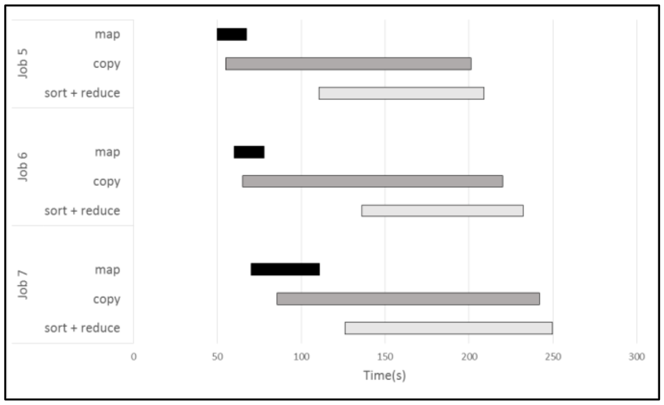

Figure 14. Phase Duration Time per Jobs (Residual Scheduling)

here, we use the results of the third experiment as a result of Residual Scheduling. Figure 13 shows the Job execution time of Fair and Residual Scheduling and the reduction rate. The average, maximum and minimum job execution time was 154s, 236s and 80s, respectively. The maximum reduction rate was 46% and it appeared in Job 5.

2) Phase Duration Time: To explore the cause of job execution time improvement, we analyse the duration of each phase. Figure 14 shows the phase duration time of Job 5, 6, and 7 whose reduction rate were large in Figure 13. In all of these jobs, the time difference between the end time of Map Phase and the start time of Sort + Reduce Phase is small, compared to Fair scheduling that is shown in Figure 6. To examine the factors that shorten this interval, we investigate the state of network link utilization during the time from 50s to 250s which corresponds to the copy phase of Jobs 5, 6 and 7.

3) Inter Rack Link Bandwidth Utilization: To evaluate the effectiveness of Residual Scheduling, we measured the bandwidth utilization of the uplink and downlink between

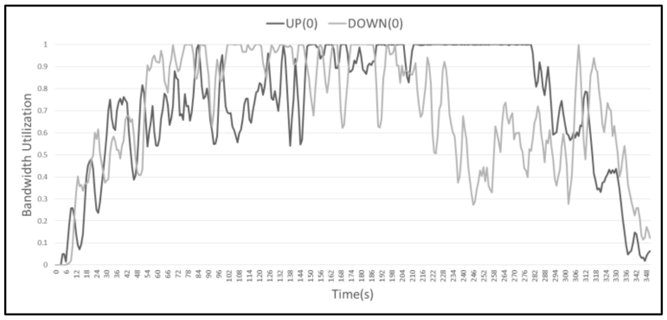

Figure 15. Bandwidth Utilization (Residual Scheduling)

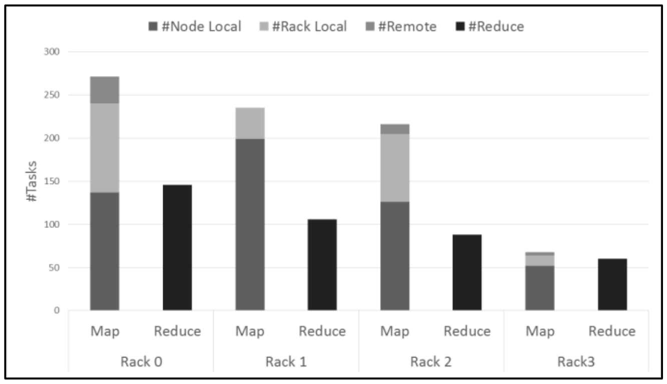

Figure 16. Number of Tasks per Rack

AGG-SW and ToR-SW of Rack 0 which was congested under Fair Scheduling. Figure 15 shows the measurement results. When compared with Figure 8 and 9, bandwidth utilization is improved during 50 ~ 250s. To investigate the reason of this improvement, Figure 16 shows the number of tasks per rack. When compared with Figure 10 which is the result of Fair Scheduling, the difference between Rack 0 (faster CPU*)* and Rack 1 & 2 (normal speed CPU) is shrinking.

4) *Residual Traffic:* Figures 17 and 18 show the Residual Up Traffic of each rack under Fair Schedeuring and Residual Scheduling, respectively. The difference in resudial traffic between racks is equalized under Residual Scheduling.

B. Effectiveness of Residual Traffic Based Scheduling

From Figures 17 and 18, we can see that the residual amount of transmission is regulated so as not to concentrate on Rack 0. This is also supported from the fact that the number of MapTasks assigned to Rack 0 is not concentrated as shown in Figure 16. However, even under the Residual Scheduling, the link bandwidth utilization UP (0) becomes 100% at the later stage of simulation as shown in Figure 18. This is because of the replication of job output performed by the HDFS and cannot be regurated by task scheduling. As we used Sort job, twice the amount of job size of data transfer occurs when the number of Replica is three, thus made UP(0) bandwidth utilization 100%.

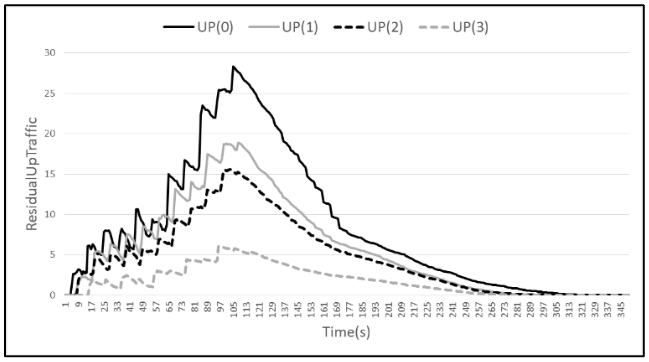

Figure 17. ResidualUpTraffic (Fair)

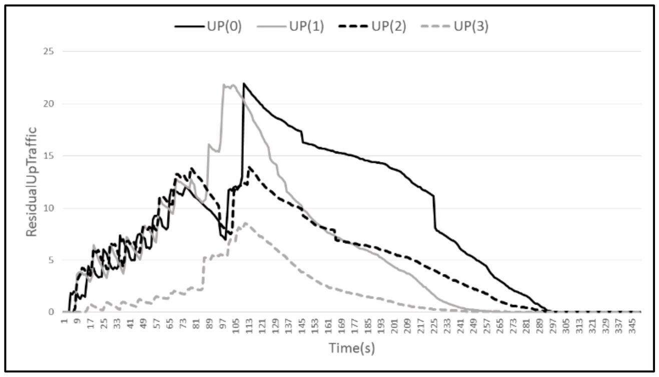

Figure 18. ResidualUpTraffic (Residual)

In summary, although it cannot cope with HDFS replication, the proposed Residual Scheduling can alleviate inter rack link load and shorten the duration of copy phase, thus achieving job performance improvement up to 23%.

C. Future Works

The proposed scheduler could not be fully compatible with replication procedure. This could be improved if we take into account the replication traffic in residual traffic. Also, in this paper we assumed sort jobs only, because it generates large amount of data transfer over the network. Mixture of other job types needs to be considered. Furthermore, implementation of proposed scheduling method needs to be studied.

VII. CONCLUSION

This paper discussed the impact of data transfer in Hadoop job performance. Using network simulation, it revealed the mechanism of job performance deterioration caused by the delay of copy phase due to the heavy load in the inter rack link of the cluster network. Based on this analysis, we proposed a new scheduling method -Residual Traffic Based Task Scheduling- that estimates the amount of inter rack data transfer in the copy phase and regulates task assignment accordingly. We evaluated this scheduling method and showed that the proposed method can improve Hadoop job performance significantly.

REFERENCES

[1] T. White. Hadoop: The Definitive Guide, 3rd Edition, O'Reilly Media / Yahoo Press,California, 2012.

[2] J. Dean, S. Ghemawat, “MapReduce: Simplified data processing on large clusters" Communications of the ACM 51.1 (2008): pp.107-113.

[3] The Apache Software Foundation. *Apache Hadoop.* [Online]. Available from: http://hadoop.apache.org 2015.1. 13.

[4] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, “Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling”. EuroSys conf., pp. 265-278, Paris, France, Apr. 2010.

[5] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, “Job Scheduling for Multi-User MapReduce Clusters”, Technical Report of EECS Department, University of California, Berkeley, 2009.

[6] A.Verma, B. Cho, N. Zea, I. Gupta, R. H. Campbell, "Breaking the MapReduce Stage Barrier", Cluster computing, 2013 – Springer

[7] J. Xie, S. Yin, X. Ruan, Z. Ding, Y.Tian, J.Majors, A. Manzanares, X. Qin, “Improving MapReduce performance through data placement in heterogeneous Hadoop clusters”, IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), pp 1-9, 2010

[8] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, I Stoica, “Improving MapReduce Performance in Heterogeneous Environments”, Technical Report of EECS Department, University of California, Berkeley, No. UCB/EECS-2009-183, Dec. 2009

[9] C. Qi, C. Liu, Z. Xiao, "Improving MapReduce Performance Using Smart Speculative Execution Strategy", IEEE Transactions on Computers, pp954-967, 2013

[10] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, I. Stoica, "Managing data transfers in computer clusters with orchestra" ACM SIGCOMM2011, pp98-109, 2011

[11] Amazon Web Services Inc. *Amazon Web Services, Cloud Computing: Compute, Storage, Datababase.* [Online]. Abailable from: http://aws.amazon.com 2015.1. 13.

[12] G. Wang, A. R. Butt, P. Pandey, K Gupta, "A simulation approach to evaluating design decisions in mapreduce setups." MASCOTS'09. Pp1-11, 2009.

[13] *ns-3.* [Online]. Available from: http://www.nsnam.org/ 2015.1. 13.

[14] H. Watanabe, M. Kawarasaki, “Impact of Data Transfer to Hadoop Job Performance - Architectural Analysis and Experiment -, ACMSE2014, Mar. 2014

[15] T. Chao, H. Zhou, Y. He, L. Zha, "A dynamic mapreduce scheduler for heterogeneous workloads." Grid and Cooperative Computing, 2009. GCC'09. Eighth International Conference on. IEEE, 2009.

[16] F. Ahmad, S. T. Chakradhar, A. Raghunathan, T. N. Vijaykumar, "Tarazu: optimizing mapreduce on heterogeneous clusters." ACM SIGARCH Computer Architecture News. Vol. 40. No. 1. ACM, 2012.