IEICE TRANSACTIONS

on Information and Systems

VOL. E98-D NO. 7
JULY 2015

A PUBLICATION OF THE INFORMATION AND SYSTEMS SOCIETY
The Institute of Electronics, Information and Communication Engineers
Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3 chome, Minato-ku, TOKYO, 105-0011 JAPAN

PAPER

System Status Aware Hadoop Scheduling Methods for Job Performance Improvement

Masatoshi KAWARASAKI[†a)], *Member and* Hyuma WATANABE[†], *Nonmember*

SUMMARY MapReduce and its open software implementation Hadoop are now widely deployed for big data analysis. As MapReduce runs over a cluster of massive machines, data transfer often becomes a bottleneck in job processing. In this paper, we explore the influence of data transfer to job processing performance and analyze the mechanism of job performance deterioration caused by data transfer oriented congestion at disk I/O and/or network I/O. Based on this analysis, we update Hadoop's Heartbeat messages to contain the real time system status for each machine, like disk I/O and link usage rate. This enhancement makes Hadoop's scheduler be aware of each machine's workload and make more accurate decision of scheduling. The experiment has been done to evaluate the effectiveness of enhanced scheduling methods and discussions are provided to compare the several proposed scheduling policies.

key words: *Hadoop, MapReduce, distributed computing, task scheduling, job performance*

1. Introduction

There is an increasing demand for handling large-scale data called Big Data. Because these data can extend from several tens of terabytes to petabyte, it's difficult to handle them in conventional ways. As a platform for big data analysis, MapReduce [1] and its open source implementation 'Hadoop' [2] is widely deployed in recent years.

As MapReduce runs over a cluster of massive machines, data transfer often becomes a bottleneck in job processing. Kandula [3] measured traffic in actual MapReduce datacenter to find that 86% of the links observe congestion lasting at least 10 seconds and 15% observe congestion lasting at least 100 seconds. Chowdhury [4] pointed out that data transfer time may consume more than 50% of total job execution time.

In general, distributed parallel processing is most efficient when the processing on each node is mutually independent. If there is any interdependence between nodes, it may cause delay and prolong the job processing time. MapReduce is actually such kind of model. In fact, reduce computation can only start on a reducer node when all the relevant map tasks are completed on mapper nodes and their outputs are transferred over the network. In this paper, we explore how such interdependence between nodes affects job performance. In particular, how the partial imbalance in the usage of cluster resources provokes overall performance deterioration.

In recent years, studies on Hadoop performance are becoming active. Zaharia [5] proposed "Delay Scheduling" to increase data local map tasks, thus suppress data transfer for receiving input data split from other node. Zaharia [6] also proposed "Copy Compute Splitting" in reduce task assignment to mitigate "Slot Hoarding problem" caused by delayed data transfer. Verma [7] proposed to break the barrier between map stage and reduce stage to improve performance. Xie [8] proposed a data placement scheme that adaptively balances the amount of data stored in each node to improve data-processing performance. Konwinski [9] proposed the speculative task excursion strategy called "LATE" that reflects estimated finish time of delayed tasks. Chen [10] proposed some improvements to LATE that use both the progress rate and the process bandwidth within a phase to select slow tasks. Polo [11] proposed resource-aware adaptive scheduling (RAS) that adjust the number of slots on each machine dynamically based on job profiling information as well as workload placement across them, to maximize the resource utilization of the cluster. Recently, YARN (Yet Another Resource Negotiator) [12] was presented as a new Hadoop architecture that separates resource management functions from the programming model, and delegates many scheduling-related functions to per-job components.

However, these solutions have focused on scheduling computation and storage resources, while mostly ignoring data transfer oriented congestion. Furthermore, current Hadoop's scheduling does not consider the real time system status for each machine.

This paper focuses on the influence of data transfer within Hadoop cluster. We run multiple jobs on our experimental Hadoop clusters in our laboratory as well as on Amazon EC2 [13] and track the progress of tasks which proceed in parallel. Through this experiment, we show that a delay in a particular task caused by data transfer oriented congestion at disk I/O and/or network I/O retards the progress of other relevant tasks and deteriorates the overall job performance [14]. Based on this result, we update Hadoop's Heartbeat messages to contain the real time system status for each node, like disk I/O and network usage rate, so that Hadoop's scheduler can be aware of each node's workload and make more accurate decision of scheduling. We evaluate the performance of the proposed methods using our experimental clusters to validate their effectiveness. Discussions are provided to compare the several proposed

Manuscript received November 19, 2014.
Manuscript revised February 9, 2015.
Manuscript publicized March 26, 2015.
[†]The authors are with University of Tsukuba, Tsukuba-shi, 305–8550 Japan.
a) E-mail: mkawa@slis.tsukuba.ac.jp
DOI: 10.1587/transinf.2014EDP7385

scheduling policies.

The rest of this paper is structured as follows: In Sect. 2, we overview Hadoop architecture. Section 3 provides the experiment setting, experiment results and their analysis. Section 4 discusses mechanism of Hadoop performance deterioration caused by data transfer oriented congestion at disk I/O or network I/O. Section 5 proposes some enhancements to Hadoop scheduler that reflect data transfer oriented congestion, Sect. 6 gives their experimental results and section and Sect. 7 provides discussions. Finally, Sect. 8 concludes this paper.

2. Hadoop Overview

2.1 Hadoop Architecture

A Hadoop cluster is constructed by interconnecting general-purpose computers that implements Hadoop. Hadoop is composed of Hadoop Distributed File System (HDFS) and MapReduce engine that runs on it. HDFS consists of a single NameNode that manages the name space of the entire file system and many DataNodes that store the actual file blocks in themselves. For each file block, replicas (three by default) are created in different DataNodes to improve data reachability and fault tolerance. MapReduce engine has a single JobTracker that manages a job from the user and assigns tasks to the TaskTrackers on each machine. In general, DataNode and TaskTracker are arranged in the same node so that a map task can be performed on the node having the input data split to be processed. Such map task is called “data local map task”.

MapReduce performs two functions, map function and reduce function, which are defined by the user. Figure 1 shows the MapReduce processing flow. When a job is submitted by a user, JobTracker divides the input data into splits. Then it generates a map task for each input data split and assigns a TaskTracker to process it. If the map task is not data local, the TaskTracker obtains the required data split from other nodes over the network. When the TaskTracker completes the map function for a given input data split, it writes the results to the local disk as a map output data.

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 data having the same key over the network and performs reduce function. The results are stored in HDFS. As reduce function cannot start until all the map tasks of a given job are completed, reduce task tends to cause job processing delay.

2.2 Hadoop Task Scheduling

JobTracker and TaskTracker communicate each other using HeartBeat mechanism. Task scheduling is performed using this mechanism as shown in Fig. 2. First, a TaskTracker requests task assignment to JobTracker by sending HeartBeat that includes the number of currently running tasks as well as the number of taskslot of the TaskTracker. The

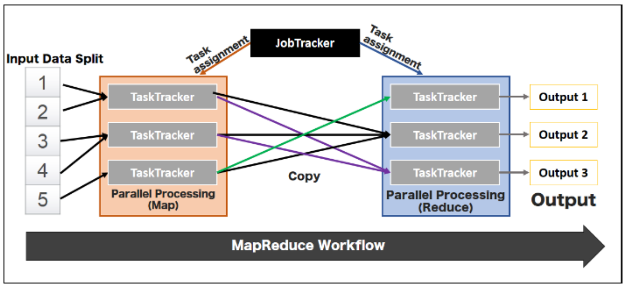


Fig. 1 MapReduce Workflow

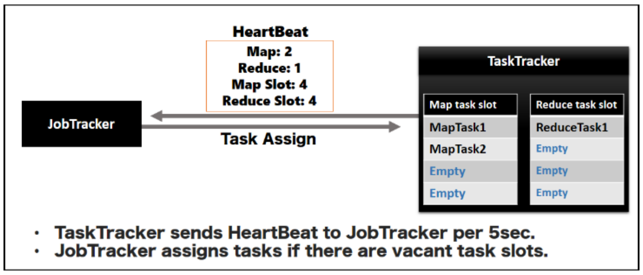


Fig. 2 HeartBeat Mechanism

JobTracker assigns a task in response to this so that the number of assigned tasks does not exceed the number of taskslot whose value is pre-determined according to the performance of each TaskTracker node. Task scheduling is performed as described above for each of map task and reduce task. The order of priority in which the task is assigned is (1) data local map task, (2) map task and (3) reduce task.

2.3 Job Scheduling

When the job scheduler is choosing the next job to run, it selects one with the highest priority. As FIFO scheduler does not support preemption, jobs are scheduled in the order of submission. The Fair Scheduler aims to give every user a fair share of the cluster capacity. When multiple jobs are submitted, free task slots are given to the jobs in such a way as to give each user a fair share of the cluster.

2.4 Data Transfer in Hadoop

In Hadoop, a large amount of data transfer occurs in network I/O and disk I/O. Regarding network I/O, data transfer over the network occurs in the following three occasions: (1) non data local map task receives input data split from other node, (2) reduce task receives map output from relevant map nodes (i.e., copy phase of reduce task) and (3) HDFS makes replica of the job result by sending the copy of reduce output data to other nodes.

On the other hand, disk I/O appears in the following occasions; (1) receiving map input data split from HDFS, (2) execution of map or reduce function, (3) sending and receiving map intermediate output, (4) writing map intermediate output or reduce output and (5) copying reduce output to

HDFS.

2.5 Speculative Task Execution

If a particular task is taking a long time for some reason, JobTracker makes other TaskTracker to perform the same task as a backup task. This is referred to a speculative task. If either of original task or speculative task is completed earlier, the other task is terminated immediately.

Conditions of launching a speculative task are as follow: (1) Progress of a task is late than the average minus 0.2 of the entire task, (2) The task has run for at least one minute, and (3) Speculative task for the task has not been launched yet.

3. Experiment of Hadoop Performance

To explore how the data transfer affects the overall job performance in Hadoop cluster, we performed experiments in two different cluster settings: Laboratory cluster (Lab cluster) and EC2 cluster. We added a simple modification to the original Hadoop program (version 1.1.2) to measure map output data transfer time for each map-reduce pair. In addition, we used *sysstat* command to measure network utilization, disk I/O usage and CPU usage of each node.

3.1 Experiment Environment

Specifications of Lab cluster and EC2 cluster are summarized in Tables 1 and 2.

Lab cluster is a single rack cluster which consists of one switch and seven nodes. It is heterogeneous in terms of disk and CPU performance. This heterogeneity is reflected in the number of task slots of respective nodes

EC2 cluster consists of twenty nodes by using m1.xlarge instance of Amazon EC2. Thus, it's homogeneous in terms of disk and CPU performance. As described in [4], a set of Amazon EC2 instances provides the network performance which is equivalent to a single rack cluster. We measured available network bandwidth between the pair of instances using *iperf*, and the result was 1∼1.2 Gbps.

Regarding the Hadoop setting, we increased the BlockSize of HDFS from its default 64MB to 256 MB. According to "Hadoop: The Definitive Guide, 3rd Edition" [15], 256MB is a general arrangement for the clusters that handles a large amount of data. As for the number of HDFS replication, we set it to 1 (default value is 3) because the scale of our Lab cluster is very small as compared to common cluster setting. In the EC2 cluster, we kept the default value 3.

As for job scheduler, we used Fair-Scheduler to create a typical environment of concurrent job execution.

We ran a series of sort jobs which invokes massive data transfer [16], [17]. Although job types may include *search* and *index* other than *sort*, *index and/or search* jobs generate only a small amount of data transfer. Because we focus on the impact of data transfer to job performance, the influence of *search* and /or *index* jobs is not of great importance. Sort job is often used as a benchmark for evaluating Hadoop cluster throughput [15]. Table 3 summarizes our job specifications. Input data is a 5GB random file. Jobs are launched every 5 seconds up to 5 jobs in Lab cluster and 16 jobs in EC2 cluster. These jobs run in parallel. We performed this experiment five times and recorded the results.

3.2 Job and Task Execution Time

Figures 3 and 4 show the experiment result about job execution time in each cluster. Despite the same job, there are

Table 1 Lab cluster specifications.

Name	Role	CPU	Core	RAM	HDD	Task slot
N0	JobTracker NameNode	Phenom II X4 920	4	4GB	1	N/A
N1	TaskTracker DataNode	Core i7 960	4	6GB	4	Map: 4 Reduce: 4
N2		Xeon E3 1240	4	4GB	3	
N3						
N4						
N5		Celeron G550	2	2GB	2	Map: 2 Reduce: 2
N6						

Table 2 EC2 cluster specifications.

Name	Role	CPU	Core	RAM	HDD	Task slot
N0	JobTracker NameNode	Xeon E5507 (Suitable)	4	15GB	5	N/A
N1∼ N19	TaskTracker DataNode					Map: 4 Reduce: 4

Table 3 Experimental Job specifications

Target Cluster	Type	Input Data	Map Tasks	Reduce Tasks	Concurrent Jobs
Lab. cluster	Sort	Random Files 5GB	20	10	5
EC2 cluster	Sort	Random Files 5GB	20	20	16

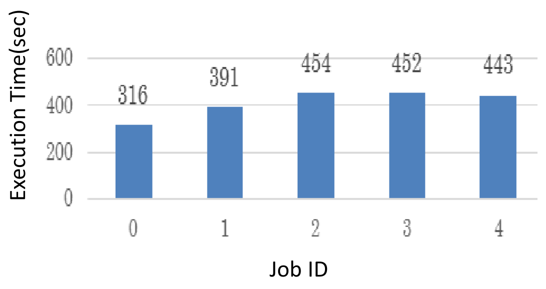


Fig. 3 Job Execution Time (Lab. cluster)

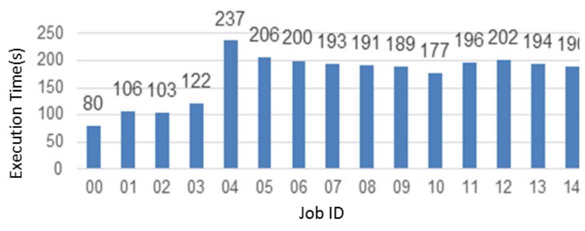


Fig. 4 Job Execution Time (EC2 cluster)

Table 4 Execution time (sec.) of map and reduce tasks.

	Map			Reduce		
Target Cluster	**Max.**	**Min.**	**Avg.**	**Max.**	**Min.**	**Avg.**
Lab. cluster	180.38	8.05	42.42.	366.66	25.59	159.54
EC2 cluster	63.57	7.64	21.51	99.33	24.82	51.61

Table 5 Map output data transfer time (sec.) of reduce tasks

Target Cluster	Max.	Min.	Avg.
Lab. cluster	43.416	0.032	4.25.
EC2 cluster	34.61	0.037	0.83

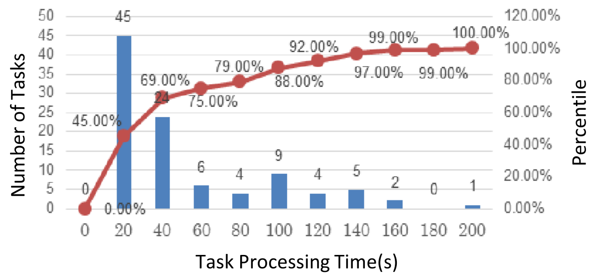


Fig. 5 Cumulative distribution of Map task execution time (Lab cluster)

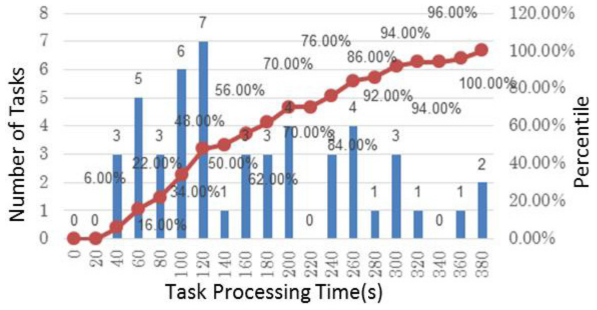


Fig. 6 Cumulative distribution of Reduce task execution time (Lab cluster)

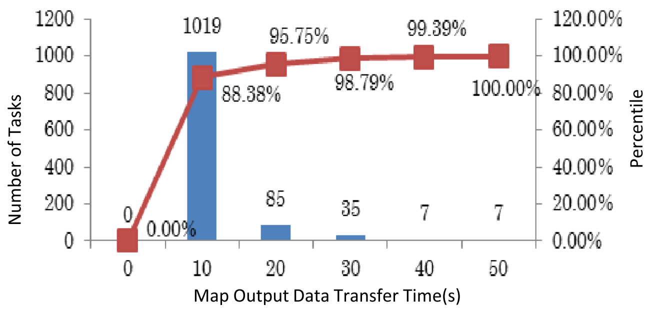


Fig. 7 Cumulative Distribution of map output data transfer time

differences in job execution time in both clusters. Table 4 shows the max, min., and average values of map task and reduce task execution time. The difference in task execution time is very large, especially in Lab cluster. Same holds true for the map output data transfer time as shown in Table 5. In the following sections, we analyze the causes of time consuming tasks in more detail.

3.3 Analysis of Hadoop Performance in Lab Cluster

Figures 5 and 6 show the distribution of task processing time in Lab cluster. Map task execution time has a long tail distribution where some take extremely long time. On the other hand, reduce task processing time is varied over a wide range. It should be noted that these data include speculative tasks. Figure 7 shows the distribution of map output data transfer time for each map-reduce pair. More than 88% of data transfers have completed within 10 seconds but others have long tail distribution.

To analyze what's happening about the time consuming tasks, we took a Gantt chart of task progress of Job 0, 1 and 2 in Lab cluster. The results are shown in Fig. 8. In Fig. 8, mark "#" indicates the execution of a map task. Similarly, mark "%" is the execution of copy in reduce task, and "!" is the execution of sort & reduce functions in reduce task. Mark "-" is a killed task caused by speculative task.

In Job0 (see Fig. 8 (a)), MapTask 6, 8, 13, 14, 18 and 19 took two to six times more execution time as compared to other map tasks. Because Job0 is the first job that enters the system, all the map tasks are data local. Accordingly, there is no cause for network congestion. Thus, it is envisaged that all the time consuming map tasks have been caused by disk I/O bottleneck. In fact, we found that disk usage maintained 100% during the time of these tasks.

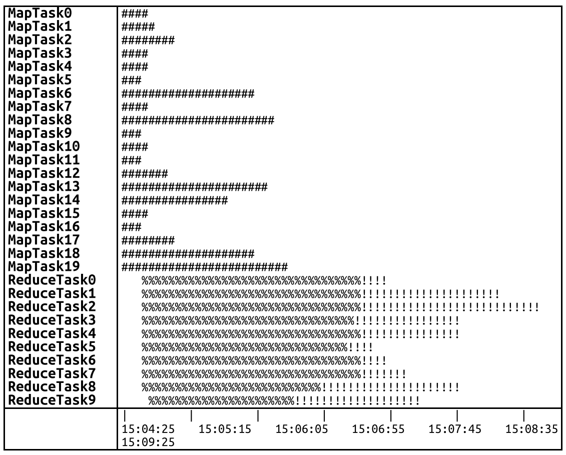
3.3.1 Disk I/O Bottleneck

Then, what are the causes of disk I/O bottleneck? Since there is no map function in sort job, the execution time of map task is equal to the time of receiving input split from HDFS, dividing it into partitions for reduce task, and writing them into node local disk. Each of node N1~N4 has 4 cores and 4 map task slots. Although node N1 has 4 HDDs, node N2~N4 has only 3 HDDs (see Table 1). Owing to this heterogeneity, disk I/O would easy to become a bottleneck in Lab cluster. We can observe similar phenomenon in other jobs as well. Therefore, heterogeneity in disk performance among nodes should have caused disk I/O bottleneck.

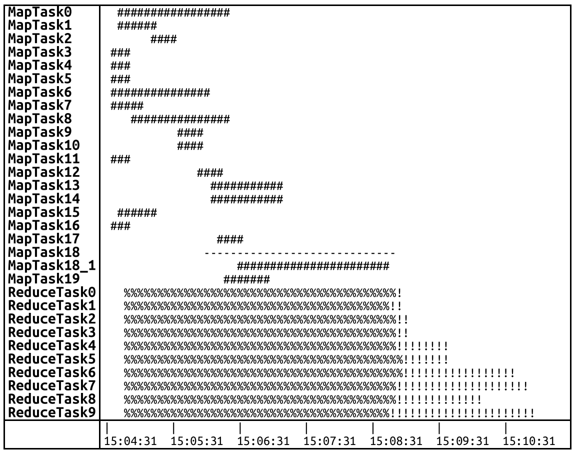
3.3.2 Reduce Slot Hoarding Problem

Figure 8 (c) shows that, in Job2, starting time of map tasks and reduce tasks vary widely. As explained below, this is because the number of free task slots within the cluster is decreasing and queued tasks are waiting for the termination of preceded tasks.

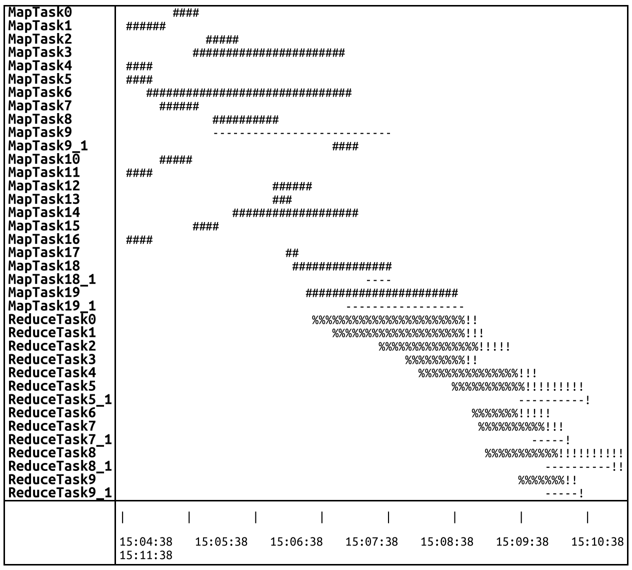
Reduce tasks begin allocated to Task Tracker nodes when 5% of total map tasks are completed. These nodes start to obtain map output over the network (i.e., copy phase). The imbalance in map task processing time influences this copy phase execution time of the reduce task. All reduce tasks cannot end their copy phase and proceed



(a)



(b)



(c)

Fig. 8 (a) Task Progress of Job0 (Lab Cluster), (b) Task Progress of Job1 (Lab cluster), (c) Task Progress of Job2 (Lab cluster)

to sort & reduce phase until all the map tasks have completed. The phenomenon can be observed in Fig. 8 (b) where all reduce tasks are waiting at copy phase for completion of MapTask18 without doing anything. This is exactly the "reduce slot hoarding problem" that is explained in [6]. This problem worsen reduce task slot starvation and cause a long delay in reduce task starting time which can be observed in Fig. 8 (c).

3.3.3 Map Output Data Copy Time

As shown in Fig. 7 and Table 5, most of map output data transfer ends within 10 seconds. Since 5 GB of input data is split into 20 map tasks and map output data are partitioned for 10 reduce tasks, each map output data which a reduce task receives become 25 MB. As link speed is 1Gbps, transfer time of each data file should not become so long. Nevertheless, some data transfer took more than 20 seconds. This is because of disk I/O congestion at the sender node which has map output data. In the receiver node which runs reduce task, disk I/O would not become a bottleneck because a reduce task first receives map output data on its memory. However, subsequent sort & reduce phase (as shown by "!" in Fig. 8) would require frequent disk I/O.

3.3.4 Effect of Speculative Execution

MapTask18_1 of Job2 is a speculative task for MapTask18 which is a data local map task. JobTracker recognized MapTask18 as slow task compared to other map tasks of Job2, and launched MapTask18_1 to make job execution time shorter. However, as MapTask18_1 is not data local map task because the number of HDFS replication is one in Lab cluster, it further increases the congestion of disk I/O in the original node, thus prolongs the execution time of both original and speculative tasks. Same phenomenon was also observed in other jobs.

Speculative reduce tasks: ReduceTask5_1, 7_1, and 9_1 of Job2 were terminated before they complete their tasks because their original tasks ended earlier. It means that these speculative tasks have used the task slot and resource of the cluster vainly. The conditions to launch speculative tasks as described in Sect. 2.5 do not work well when the cluster is suffering task slot starvation. As we can see from Fig. 8 (c), the original tasks were not taking long time for its completion compared to other tasks. They just started late. By the current conditions for speculative task execution, JobTracker recognizes late-started tasks as slow tasks, and executes unnecessary speculative tasks.

3.3.5 Summary

In Lab cluster, owing to the heterogeneity in disk performance among nodes, disk I/O bottleneck may appears in a particular node and prolongs map task execution time of the node. Delayed map task caused by disk I/O incurs reduce slot hoarding problem in relevant reduce nodes as well as useless speculative map task execution that accelerates disk I/O congestion.

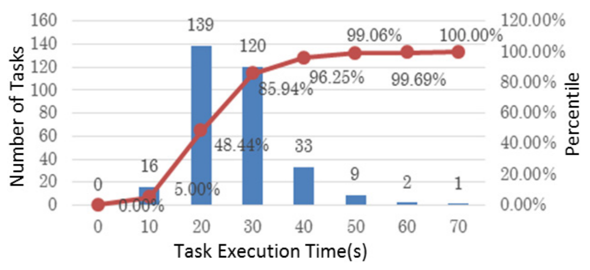


Fig. 9 Cumulative distribution of map task execution time (EC2).

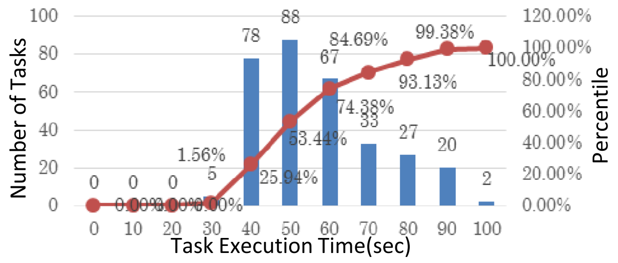


Fig. 10 Cumulative distribution of reduce task execution time (EC2)

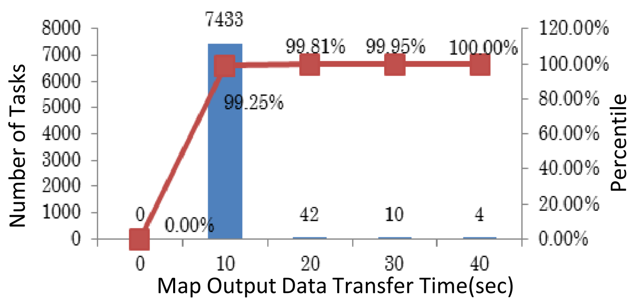


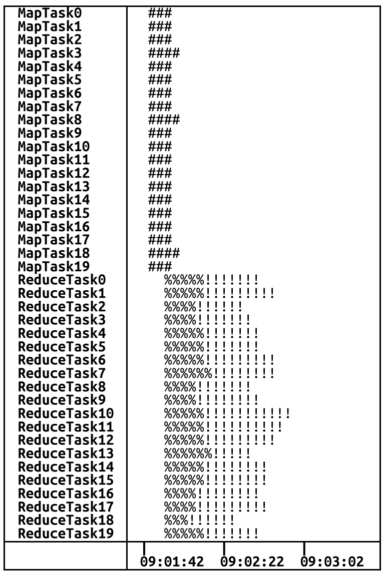
Fig. 11 Cumulative distribution of map output transfer time (EC2)

3.4 Analysis of Hadoop Performance in EC2 Cluster

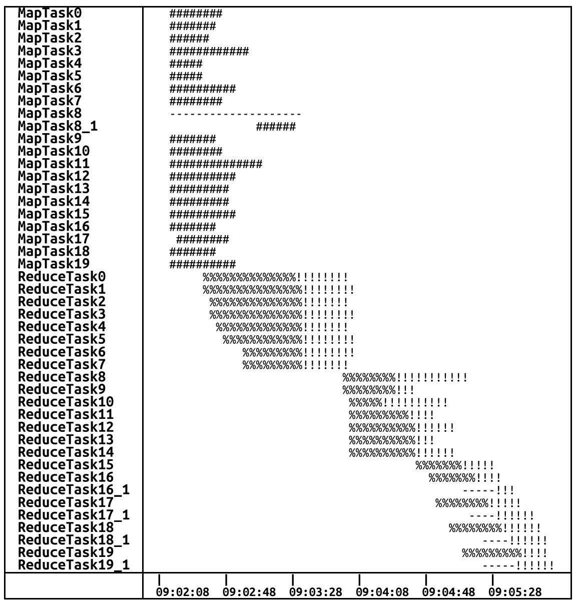
We made a similar analysis for the experimental results from EC2 cluster. Figures 9, 10 and 11 show processing time distribution of map task, reduce task and map output data transfer, respectively. Both of map and reduce tasks show a long tail distribution.

In EC2 cluster, we found a big gap in job execution time between job3 and its following jobs. Figure 12 (a), (b) is task progress Gantt chart of Job0 and Job4. As Fig. 12 (b) shows, task progress of Job4 is quite slow because of the delay of MapTask8 and the shortage of reduce task slot stemming from it.

We could not find the reason why MapTask8 took such a long time. There was room on the disk and CPU in the node on which MapTask8 was running, and since the task was data local, it should not have been affected by network I/O. At all events, MapTask8 was delayed significantly and its speculative task (MapTask8_1) was launched. But it seemed to be too late owing to the condition that the original task should have run for at least one minute before launching



(a)



(b)

Fig. 12 (a) Task progress of Job0 (EC2), (b) Task progress of Job4 (EC2)

its speculative task. As a result, reduce task slots were consumed ineffectively on the reduce nodes just waiting for the completion of MapTask8, thus invite reduce slot hoarding problem. Reduce slot hoarding retards the start of subsequent reduce tasks and provokes useless speculative reduce task executions, thus leads to job performance deterioration. Variation in reduce task starting time was caused by reduce slot starvation, and ReduceTask16_1, 17_1, 18_1, and 19_1 in Fig. 12 (b) were useless speculative reduce tasks. This phenomenon was observed in three of other jobs. The same phenomenon was also observed in Lab cluster.

Since EC2 cluster have many HDDs (five per node) and its node performance was uniform as compared with our Lab cluster (see Table 2), the large variation in task execution time was not observed. Still, more than 80% of relatively slow tasks were caused by disk IO, and the rest was caused by network congestion.

4. Performance Deterioration Mechanism

4.1 Deterioration Mechanism of Hadoop Performance

By summing up our analysis of experiment results, Hadoop performance deterioration is thought to occur by the following mechanism as shown in Fig. 13.

i. Map task is delayed caused by disk I/O congestion or map output data transfer is delayed caused by network congestion.
ii. Reduce task hoards a reduce slot without doing anything.
iii. Reduce slot starvation occurs and incurs a large variation in the start time of reduce tasks..
iv. Useless speculative reduce tasks are launched and consume I/O resources and reduce task slots of the cluster.
v. Useless speculative reduce tasks delay other tasks and worsen reduce slot shortage.

Figure 13 shows that the mechanism of job performance deterioration is the same regardless that it was caused by disk I/O congestion or network I/O congestion.

4.2 Impact of Imbalanced Node Performance

Regarding the nodes in our Lab cluster, there is a large variation in disk and CPU performance. In this cluster, map output data transfer takes longer time compared to EC2 cluster. This comes from the imbalanced node performance.

The high performance TaskTracker can complete relatively many map tasks. Since map output data exist only in the node which completed map task, data read-out request from many reduce tasks concentrates to this node. As a result, network link and disk I/O become congested thus delays relevant reduce tasks. Speculative task execution further promotes this congestion and prolongs the reduce tasks. To cope with this problem, map task scheduling needs to be performed considering the amount of map output data which has not been sent to reduce tasks yet. However, it will be better to build a Hadoop cluster with nodes having uniform performance.

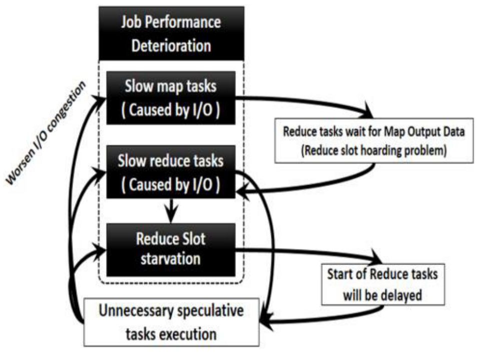

Job Performance Deterioration
Worsen I/O congestion
Slow map tasks (Caused by I/O)
Reduce tasks wait for Map Output Data (Reduce slot hoarding problem)
Slow reduce tasks (Caused by I/O)
Reduce Slot starvation
Start of Reduce tasks will be delayed
Unnecessary speculative tasks execution

Fig. 13 Job performance deterioration mechanism

4.3 Effect of Increasing Reduce Slots

In our experiment, many of the slot hoarding problems occurred in reduce phase. To cope with this problem, although using the copy compute splitting algorithm [6] is a good solution, it will also be effective to increase the number of reduce slot more than the number of cores of nodes when reduce function has little computation amount. This simple method can suppress reduce slot starvation by preventing "copy waiting reduce tasks" occupy all available reduce slots. In our settings, increasing reduce slot count by twice improved average job execution time by 20%.

4.4 Adverse Effect of Speculative Execution

As we described before, the conditions for launching speculative tasks that are used in current Hadoop implementation needs some improvement. The problems are as follow:

- When the reduce task in copy phase is delaying owing to congestions in sender's disk I/O or network I/O, speculative tasks worsen the situation and further deteriorates job performance.
- It misidentifies late started task as slow task.
- When the most tasks end within 1 minute, launch of speculative task execution would be too late.

To cope with these problems, following modifications to launching conditions would be effective.

- Take account of delaying reason. (If the delay is caused by unavoidable I/O bottleneck, speculative task should not be executed.)
- Prioritize and limit the number of concurrent speculative task.

5. Enhancement of Hadoop Scheduler

Based on the analysis in Sect. 4, we propose the enhanced Hadoop scheduling methods that proactively react to the congestion at disk I/O or network I/O so that the cluster performance is improved.

5.1 Basic Idea

Except for priority control and fairness control in job scheduling, current scheduler is only controlling the number of taskslots to be assigned so as not to exceed the preallocated fixed value for each TaskTracker node. However, the amount of node resources (e.g., CPU, disk I/O, network) that a particular task consumes is different depending on job type, task progress and data volume. As a result, current Hadoop scheduler cannot reflect the actual loading state of TaskTracker resources and may cause a large imbalance in the loading state between TaskTracker nodes in the cluster.

As explored in Sect. 3, congestion in a particular network link or disk I/O may retard the progress of relevant tasks and lowers the throughput of the entire cluster.

Based on these observations, for the purpose of shortening the Job execution time of Hadoop, we propose three kinds of Task scheduling methods that mitigate the resource usage heterogeneity within a cluster, namely "Receive Rate Scheduling", "Disk Rate Scheduling" and "Potential Reception Scheduling". The idea behind these three methods is the same. While maintaining current task-slot based scheduling, these methods take into account the actual usage of data I/O (network I/O and disk I/O) in the TaskTracker as a control index for task scheduling. If the data I/O usage rate of a given TaskTracker exceeds the pre-determined threshold value, some sorts of task are not assigned to the TaskTracker. Data I/O usage rate is transmitted to JobTracker by using HeartBeat mechanism.

5.2 Implementation of Proposed Methods

We made small modifications to the TaskTracker to add the capability to retrieve data I/O usage rate. We also modified HeartBeat to add the notification capability of the obtained usage rate information. We implemented our proposed methods by modifying the Fair Scheduler mounted on Hadoop version 1.1.2. We also made it possible to enable and disable each scheduling method or to set threshold value by describing in Hadoop configuration file.

5.3 Receive Rate Scheduling

This scheduling method monitors and controls the usage of reception bandwidth, so as not to transfer additional data through a congested link. At the time of receiving HeartBeat from a TaskTracker, the JobTracker calculates the reception bandwidth utilization (RBU) at the network interface and, if it exceeds the threshold value, it does not assign additional non-data-local map task nor reduce task to the TaskTracker. It is because these tasks oblige the TaskTracker to receive additional data immediately after task assignment. Data local map task is excluded from the control because it does not generate data transfer through the congested link. RBU is calculated as follows:

$$\mathrm{RBU} = 8 * \mathrm{Rd}/(\mathrm{T} * \mathrm{B}) \tag{1}$$

where Rd is the total amount of received data (byte) from the last HeartBeat transmission, T is the HeartBeat interval, and B is the network interface rate (bps).

5.4 Potential Reception Scheduling

In this scheduling method, JobTracker monitors the total volume of map output data (copy traffic) that each TaskTracker is receiving and, if it exceeds the threshold value, JobTracker does not assign additional reduce task to the TaskTracker. This is a proactive control so as not to send too much copy traffic to a specific TaskTracker. JobTracker does not assign non-data-local map task to the TaskTracker neither, but data local map task can be assigned, because the former requires the TaskTracker to receive additional data but the latter not.

TaskTracker sends the total volume of copy traffic that it is receiving as "Potential Reception Amount" to JobTracker using HeartBeat.

5.5 Disk Rate Scheduling

While above two scheduling methods relates to network I/O, this scheduling method aims at the dispersion of disk I/O workload. At the time of sending HeartBeat to JobTracker, TaskTracker calculates disk I/O usage rate based on disk I/O waiting time and sends it to JobTracker. If multiple disks are installed in a TaskTracker node, the average usage rate of all disks is used. When the average disk I/O usage rate exceeds the threshold value, JobTracker does not assign additional map task to the TaskTracker.

It should be noted that only map task is controlled by this scheduling method because map task receives map input data split in HDD that generates disk I/O workload immediately after task assignment. On the other hand, in the reduce task, the received map output data is stored in memory for merging and then recorded in HDD. Accordingly, there are more than several seconds of delay from the assignment of reduce task until the occurrence of disk I/O workload. As our method uses disk I/O usage rate at the time of HeartBeat transmission, disk I/O rate that this scheduler uses would be different from that when disk I/O workload actually occurs. Therefore, in order to avoid miss-regulation, reduce task is removed from the scope of this control.

6. Experiment of Enhanced Schedulers

To validate the effectiveness of proposed scheduling methods, we performed experiments on Lab cluster and EC2 cluster under the same conditions as in Sect. 3. The threshold values for the proposed scheduling methods are shown in Table 6. These values were obtained empirically to maximize the effect of each proposed method. They need to be adjusted depending on the size, topology and performance of the cluster as well as the job characteristics.

We compared in total five kinds of scheduling method; default Fair Scheduling, Receive Rate Scheduling, Potential Reception Scheduling, Disk Rate Scheduling and Integrated Scheduling that applies three proposed methods at the same time.

Table 6 Threshold Values in Proposed Scheduling Methods.

	Receive Rate Scheduling	Potential Reception Scheduling	Dsik Rate Scheduling
Lab cluster	160Mbps	100Mbytes	30%
EC2 cluster	300Mbps	150Mbytes	40%

6.1 Job Excursion Time

Tables 7 and 8 show the job execution time of above five scheduling schemes in Lab cluster and EC2 cluster, respectively. For each scheduler, the total number of submitted jobs is 35 for Lab cluster and 98 for EC2 cluster. We can see that Disk Rate Scheduling shortened the average job execution time by 5% in Lab cluster and by 6% in EC2 cluster, compared to the default Fair Scheduling.

Both of two scheduling methods that focus on network I/O, namely Receive Rate Scheduling and Potential Reception Scheduling, were effective in EC2 cluster rather than in Lab cluster. However, the reduction of the job execution time was only 2.6% in Lab cluster and less than 1% in EC2 cluster, compared to the default Fair Scheduling. The effect of Potential Reception Scheduling was limited and within the range of statistical variations. Furthermore, when applying the three proposed methods together, the job performance was slightly decreased as compared with the case of applying Disk Rate Scheduling only in both clusters.

6.2 Task Processing Time

Table 9 shows the average processing time of map task and reduce task in EC2 cluster for each scheduling method.

The Receive Rate Scheduling shortened map task processing time by 3.5% and reduce task processing time by 5% compared with default Fair Scheduling. The reason why the control effect of map task was poor was that non-data-local map task was only about 20%.

The Potential Reception Scheduling brought almost no performance improvement to map task, because it focuses on reduce task improvement. However, contrary to our expectation, control effect of this scheduling method to the reduce task was low.

The Disk Rate Scheduling improved the average processing time of map task significantly by 11% and that of reduce task by 6.5% in average. This is because, by preventing disk I/O congestion, it prevented the delay in reduce task processing that is caused by disk I/O congestion.

The Integrated Scheduling resulted in the superposition of the control effects provided by each scheduling scheme. It shortened the average task processing time by 9.4% in map task and 7.6% in reduce task.

As a whole, Disk Rate Scheduling was effective in shortening processing time of both map task and reduce task, thus shortened overall job execution time. Receive Rate Scheduling was effective in shortening reduce task processing time, thus shortened job execution time. In contrast, the effect of Potential Reception Scheduling was quite poor.

6.3 Distribution of Map/Reduce Task Processing Time

Figures 14 and 15 show the cumulative distribution of the map /reduce task processing time of the job that was the last job submitted to EC2 cluster, under the default Fair Scheduling and the combined Receive Rate and Disk Rate Scheduling, respectively. The job submitted at the end is normally most susceptible to the delay caused by the jobs submitted earlier. As these figures show, the proposed method shortens the job execution time by improving the worst value of map and reduce task processing time, thus achieves the reduction of task processing time by average 20% in map tasks and by

Table 7 Job execution time in Lab cluster

	Default	Receive Rate	Potential Reception	Disk Rate	ALL
Avg. Job time	465s	461s	464s	443s	456s
Max	563	584	574	582	575
Min	243	295	318	250	344
Δ	-	-4s	-1s	-22s	-9s

Table 8 Job execution time in EC2 cluster

	Default	Receive Rate	Potential Reception	Disk Rate	Receive Rate + Disk Rate	ALL
Avg. Job time	538s	534s	524s	507s	506s	511s
Max	616	596	595	600	575	589
Min	440	430	457	369	380	385
Δ	-	-14s	-4s	-31s	-32s	-27s

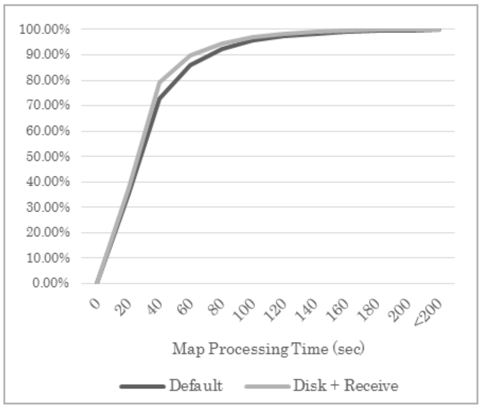


Fig. 14 CDF of map task processing time on EC2 Cluster

Table 9 Task processing time in EC2 cluster.

	Default		Receive Rate		Potential Reception		Disk Rate		Receive Rate + Disk Rate		ALL	
	Map	Reduce	Map	Reduce	Map	Reduce	Map	Reduce	Map	Reduce	Map	Reduce
Avg. Task time	33.27	318.49	32.06	301.99	32.55	316.26	29.54	297.84	30.70	299.61	30.13	294.26
Δ	-	-	-1.21	-16.50	-0.72	-2.23	-3.73	-20.65	-2.57	-18.88	-3.14	-24.22

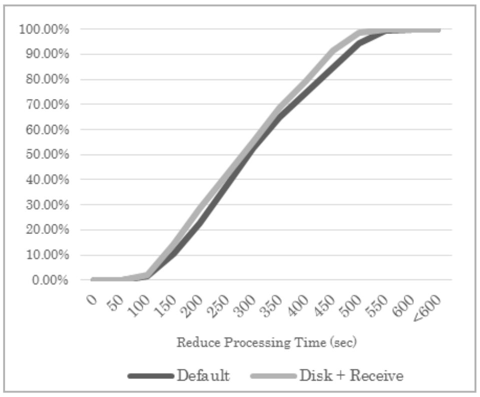


Fig. 15 CDF of reduce task processing time on EC2 Cluster

average 10% in reduce tasks.

7. Discussion and Future Issues

7.1 Characteristics of the Combined Scheduling

As can be seen from Tables 8 and 9, the combined scheduler of three scheduling methods reveals longer job execution time than the Disk Rate Scheduler only. However, looking at Table 9, the average reduce task processing time itself is made shorter. This can be interpreted that, by operating two types of scheduling namely Receive Rate Scheduling and Potential Reception Scheduling, reduce task assignment was over-regulated.

Therefore, we implemented Receive Rate Scheduling and Disk Rate Scheduling under the same conditions. As a result, the average job execution time was shortened by 32 seconds. When implementing both Receive Rate and Potential Reception Scheduling, it might be possible to prevent excessive control by relaxing threshold values of each scheduling method.

7.2 Poor Performance of Potential Reception Scheduling

As described in Sect. 6.2, the effect of Potential Reception Scheduling was very poor. This scheduling method aims to achieve proactive control to mitigate network I/O congestion in TaskTracker caused by map output data reception. But the network I/O load of TaskTracker is not only due to copy traffic of reduce task. Percentage of input traffic to the map task and replication traffic of HDFS is often critical. We consider these to be the cause of poor performance of Potential Reception Scheduling.

However, we still think that proactive control like this scheduling method would be effective and we are paving the way to improve it.

7.3 Influence of Network I/O Bottleneck

The effect of the proposed scheduling methods that focused on network I/O was limited. Many studies point out that data transfer influences significantly to Hadoop job processing performance. In spite of this, the number of and the duration of network congestion that has been observed in our experiments were very small. One reason for this is that network congestion was hard to occur in our experiment environment. In Lab cluster, as disk I/O bandwidth of the TaskTracker was very small, disk I/O congestion occurred before the network congestion occurs. In EC2 cluster, as many nodes having a bandwidth of more than 1Gbps are connected in star topology, rack-to-rack links that tend to be a bottleneck do not exist compared to the Hadoop typical cluster made up of a plural racks. The scheduling method focused on network I/O would be more effective in actual Hadoop cluster. Once severe network I/O congestion occurs, the proposed scheduling methods would evade the reduce slot hoarding problem and reduce slot starvation, thus prevent performance deterioration (see Fig. 13).

8. Conclusion

This paper discussed the impact of data transfer in Hadoop task scheduling and analyzed the mechanism of performance deterioration caused by disk I/O congestion or by network I/O congestion. It further showed that speculative task executions bring adverse effects in some cases. Based on these observations, this paper proposed three kinds of system status aware Hadoop Task scheduling methods, two methods focusing on network bandwidth usage rate and one method focusing on disk I/O usage rate. We validated their effectiveness in our experiment clusters using sort benchmark. Further improvement of Hadoop job performance would be expected by reflecting real time status of other resources than disk I/O and/or network I/O in the system to task scheduling like our proposals.

Today, Solid State Disks (SSDs) are common especially for data processing purposes like MapReduce. The use of SSDs in the experimental cluster is a subject for further study. This may mitigate disk I/O congestion. The use of resource-based scheduling like YARN as a Hadoop platform is also a subject for further study.

References

[1] J. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters,” In OSDI, vol.51, no.1, pp.107–113, San Francisco, USA, Jan. 2008.

[2] The Apache Software Foundation, “Apache Hadoop,” http://hadoop.apache.org

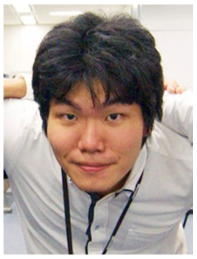
[3] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, “The Nature of Datacenter Traffic: Measurements & Analysis,” Proc. ACM SIGCOMM conf. on Internet measurement conference, pp.202–208, Chicago, USA, Nov. 2009.

[4] M. Chowdhury, M. Zaharia, J. Ma, M.I. Jordan, and I. Stoika, “Managing data transfers in computer clusters with orchestra,” Proc. ACM SIGCOMM conf., vol.41, no.4, pp.98–109, Toronto, Canada, Aug. 2011.

[5] 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, April 2010.

[6] 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.
[7] A. Verma, B. Cho, N. Zea, I. Gupta, and R.H. Campbell, "Breaking the MapReduce Stage Barrier," Cluster computing, vol.16, no.1, pp.191–206, 2013.
[8] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares, and 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.
[9] A. Konwinski, "Improving MapReduce Performance in Heterogeneous Environments," Technical Report of EECS Department, University of California, Berkeley, no.UCB/EECS-2009-183, Dec. 2009
[10] Q. Chen, C. Liu, and Z. Xiao, "Improving MapReduce Performance Using Smart Speculative Execution Strategy," IEEE Trans. Comput., vol.63, no.4, pp.954–967, 2014.
[11] J. Polo, C. Castillo, D. Carrera, Y. Becerra, I. Whalley, M. Steinder, J. Torres, and E. Ayguadé, "Resource-aware Adaptive Scheduling for MapReduce Clusters," Middleware 2011, vol.7049, pp.187–207, Springer, 2011.
[12] V.K. Vavilapalli, A.C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache Hadoop YARN: Yet Another Resource Negotiator," Proc. 4th annual Symposium on Cloud Computing - SOCC '13, pp.1–16, 2013.
[13] Amazon Web Services, Inc., "Amazon Web Services, Cloud Computing: Compute, Storage, Datababase," http://aws.amazon.com
[14] H. Watanabe and M. Kawarasaki, "Impact of Data Transfer to Hadoop Job Performance - Architectural Analysis and Experiment -, ACMSE2014, March 2014.
[15] T. White, Hadoop: The Definitive Guide, 3rd Edition, O'Reilly Media/Yahoo Press, California, 2012.
[16] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "The Case for Evaluating MapReduce Performance Using Workload Suites," Proc. MASCOTS, pp.390–399, Singapore, July 2011.
[17] O'Malley, Owen, "Terabyte sort on apache hadoop," Yahoo, http://sortbenchmark.org/Yahoo-Hadoop.pdf, May 2008.

Masatoshi Kawarasaki received the B.E., M.E. and Ph.D degrees in Electrical Engineering from Kyoto University in 1975, 1977 and 1991, respectively. From 1977 to 2004, he was with NTT and was engaged in the research of networking architecture and network control. He moved to University of Tsukuba as a professor in 2004. His research interests locate in ubiquitous networking, including virtual networking, mobile health system and distributed processing platform. He is a member of IEEE, IEICE and JAMI.



Hyuma Watanabe received the B.S. and M.S. degrees in Informatics from University of Tsukuba in 2012 and 2014, respectively. He engaged in the development of mobile health system and the study of Hadoop performance improvement. He is now with Microsoft Development, Ltd.