Impact of Data Transfer to Hadoop Job Performance - Architectural Analysis and Experiment -

Hyuma Watanabe
Graduate School of Library, Information and Media Studies, University of Tsukuba
hyuma@slis.tsukuba.ac.jp

Masatoshi Kawarasaki
Faculty of Library, Information and Media Science
University of Tsukuba
mkawa@slis.tsukuba.ac.jp

ABSTRACT

Hadoop is a distributed processing platform for analyzing large amount of data. It uses MapReduce framework for distributed parallel processing where Reduce computation uses all the relevant results of map computation performed in other nodes, thus needs massive amount of internode communication. This paper explores how the internode communication affects the overall job performance in Hadoop cluster and analyzes the mechanism of performance deterioration caused by internode communication. For this purpose, we built two kind of experimental Hadoop cluster using real machine in our laboratory and virtual machine of Amazon EC2 and tracked the progress of tasks which proceeds in a parallel way. As a result, we revealed that delay in one task caused by disk I/O or data transfer over the network propagates to other tasks and deteriorates job performance significantly. Furthermore, we found that speculative task execution is not necessarily successful when the task delay is caused by disk I/O.

Categories and Subject Descriptors
Computing methodologies~MapReduce algorithms

General Terms
Algorithms, Measurement, Performance, Verification.

Keywords
MapReduce, Data-intensive applications, Task Scheduling.

1. INTRODUCTION

With the expansion in the amount of Web information, there is an increasing demand for handling large-scale data called Big Data. Because these data can extend from several tens of terabytes to petabytes, it's difficult to handle these data by conventional ways. As a mean to deal with this problem, MapReduce [5] and its open source implementation 'Hadoop' [2], is widely deployed in recent years. Hadoop is a distributed processing platform which runs on a large cluster of commodity machines interconnected by Ethernet. It uses HDFS (Hadoop Distributed File System) for data storage and MapReduce for distributed processing framework.

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. Reduce computation can only start on a reduce task node when all the relevant map tasks are completed on map task nodes and their outputs are transferred over the network to the reduce task node. It is described in [4] that these data transfers can have a significant impact on job performance, accounting for more than 50% of job completion times.

In this paper, we explore how such interdependence affects job performance by executing multiple jobs on the experimental Hadoop clusters in our laboratory as well as in Amazon EC2 [1]. We track the progress of tasks which proceed in parallel and analyze how a delay in a particular task retards the progress of other relevant tasks and deteriorates the overall job performance. Based on these experiments, we clarify the mechanism of Hadoop performance deterioration.

According to our analysis, the main cause of job performance deterioration is the depletion of task slots due to the ineffective holding of task slots during the time of disk I/O or data transfer. As JobTracker that manages Hadoop task scheduling allocates a new task to a node which has free task slots, ineffective holding of task slots decreases the number of free task slots thus impedes subsequent tasks to start. This leads to job performance deterioration. Furthermore, speculative task execution has adverse effect if the delay of task progress is caused by disk I/O.

As for Hadoop performance analysis focused on data transfer, not so many studies have been made. Zaharia et. al address the need of data locality in Map task allocation and copy compute splitting to improve the dependence between map and reduce tasks. [8], [9] By Delay Scheduling, they succeeded in increasing the percentage of data local map tasks in order to reduce the network resource consumption and shorten the shuffle phase time. By Copy Compute Splitting that divides reduce phase into compute phase and real copy phase, they succeeded in resolving the "Slot Hoarding" problem where reduce slot just waiting for the end of the map task occupies reduce slot ineffectively. Orchestra [4] proposes a network bandwidth allocation scheme that improves fairness among Jobs running simultaneously. It also proposes a BitTorrent-like data broadcast scheme inside a cluster to reduce network load. LATE [6] proposes a new algorithm of speculative task execution that estimates task's finish time to decide which task should be backed up by speculative task.

The rest of this paper is structured as follows: In section 2, we overview Hadoop architecture. Section 3 provides the experiment setting, experiment results and their analysis. Section 4 discusses fundamental essentials against Hadoop performance deterioration. Section 5 concludes this paper.

2. Hadoop Overview

2.1 Hadoop Architecture

Hadoop is composed of two elements; HDFS and MapReduce. HDFS includes a large number of DataNodes that actually store distributed data and one NameNode that manage them. In HDFS, data is handled on a block by block basis where each data block is replicated in three by default within the cluster, thereby improving robustness and reachability.

MapReduce also has a master-slave relationship between one JobTracker and many TaskTrackers. When the JobTracker receives a job from a user, it breaks the job down into its constituent tasks and performs task scheduling. Each TaskTracker performs allocated task. The number of map task and reduce task that each TaskTracker can handle is predetermined and managed by map task slot and reduce task slot, respectively.

TaskTracker and DataNode are allocated generally in the same machine so as to increase data local task.

2.2 MapReduce Process

MapReduce processing is performed by running a large number of Map and Reduce tasks in parallel on a computer cluster. A user can handle a data by specifying map and reduce functions to be performed by map and reduce tasks, respectively.

The map invocations are distributed across multiple machines by automatically partitioning the input data into a set of pieces called input split. A node running the map task acquires the input split to be processed from underlying HDFS. If the input split is present on the node, it can begin map function immediately. This situation is called "data local". If the input split isn't on the node, it is necessary to obtain it from other node over the network. Map task is allocated to a TaskTracker preferentially which possesses its input split in its underlying HDFS so as to produce data local map task. When the map task is completed, map output data is written out to a local directory (not HDFS).

When some map tasks of a given Job are completed (default 5%), JobTracker starts to allocate reduce tasks. Reduce task receives the map output over the network (Shuffle) and starts to perform reduce function only when it received all the relevant map outputs.

MapReduce job terminates when all the reduce tasks are completed. Output data from reduce function is stored in HDFS.

2.3 Speculative Task Execution

If a task is taking a very long time for some reason, JobTracker performs the same task in another TaskTracker 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
3. Speculative task for the task has not been launched yet

2.4 Hadoop Disk I/O

Through the job lifetime of Hadoop, the timing of disk I/O occurs as follows.

1. Retrieve map input split from HDFS: **Read/Write**.
2. Map function: **Read / Write**
3. Export map output data: **Write**.
4. Send/Receive map output data: **Read/Write**.
5. Reduce function: **Read / Write**
6. Export reduce output data: **Write**.

Furthermore, the reduce output, exported to HDFS after Job completion, is copied inside the cluster up to the number of HDFS replication. This causes network resource consumption and disk read/write in relevant nodes. Among these, 1, 2, 5, and 6 require data transfer as well.

3. Experiments

To explore how the data transfer affects the overall job performance in Hadoop cluster, we ran a series of Sort jobs which invokes massive data transfer. 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

We ran the experiments in two different cluster settings: Laboratory cluster and EC2 cluster. Specifications of each cluster are summarized in Table 1 and 2.

3.1.1 Laboratory cluster

Laboratory cluster (Lab cluster) is a single rack cluster which consists of one switch and seven nodes. It's heterogeneous in terms of disk and CPU performance as reflected in the number of task slots.

3.1.2 EC2 cluster

For all nodes of EC2 cluster, we used 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.

3.1.3 Hadoop Settings

According to "Hadoop: The Definitive Guide, 3rd Edition" [7], we increased the BlockSize of HDFS to 256 MB as a general arrangement for the clusters handling a large amount of data.

Table 1. Lab cluster Specification

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 Specification

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 Specification

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

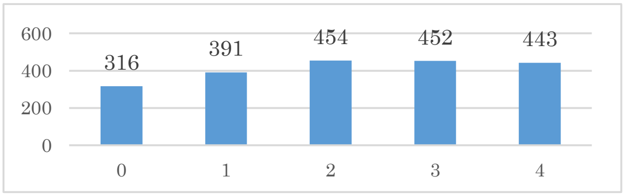

Figure 1. Job Execution Time (Lab. cluster)

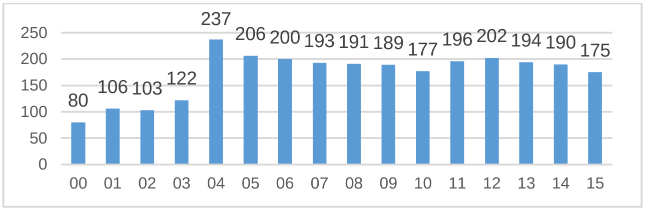

Figure 2. Job Execution Time (EC2 cluster)

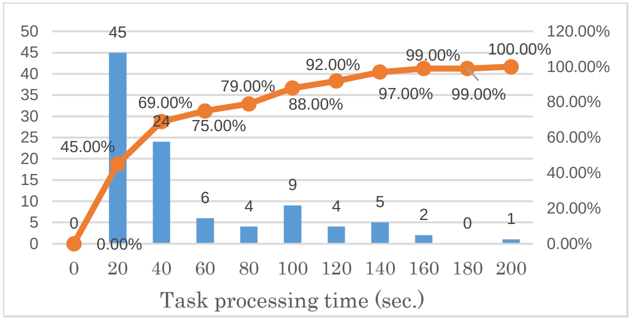

Figure 3. Cumulative distribution of Map task execution time (Lab cluster)

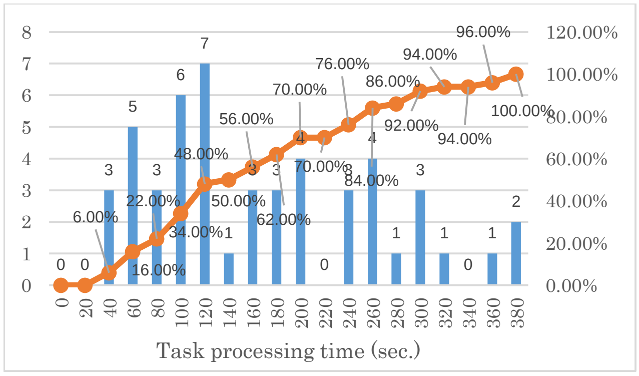

Figure 4. Cumulative distribution of Reduce task execution time (Lab cluster)

Regarding 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. As for EC2 cluster, we kept the default value 3.

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

3.1.4 Experiment Job

The jobs we used in the experiment are shown in Table 3. To explore the impact of data transfer in Hadoop performance, we used Sort job that generate intermediate data transfer in large amounts. [3] Input data is 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.

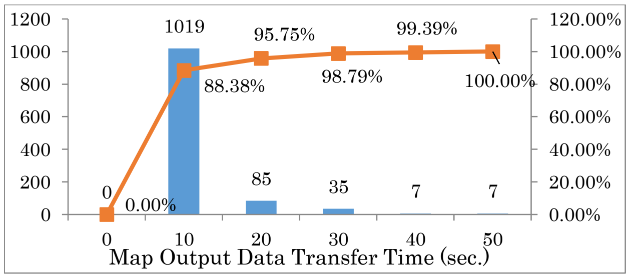

Figure 5. Cumulative Distribution of map output data transfer time (Lab. cluster)

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

3.2 Job and Task Execution Time

Figure 1 and 2 show the experiment results of job execution time in each cluster. Despite the same job, there are 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 in each cluster. The difference in task execution time is very large. The difference is even greater 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

Figure 3 and 4 shows 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 5 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 tailed 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 Figure 6. In Figure 6, mark "#" indicates a section of a map task execution. Similarly, mark "%" is the execution of copy in reduce task, and "!" is the execution of sort and reduce function in reduce task. Mark "-" is a killed task caused by speculative task.

In Job0 (see Figure 6 (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.

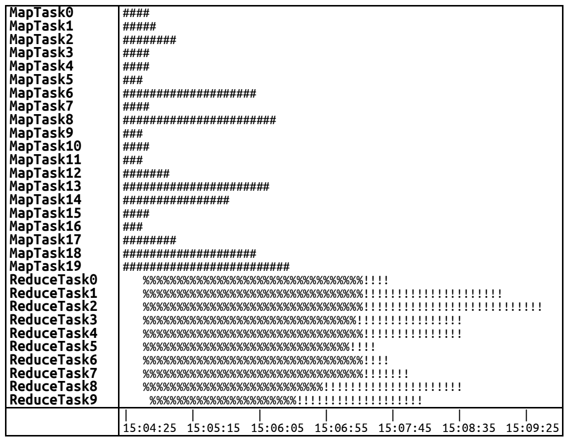

Figure 6 (a). Task progress of Job0 (Lab)

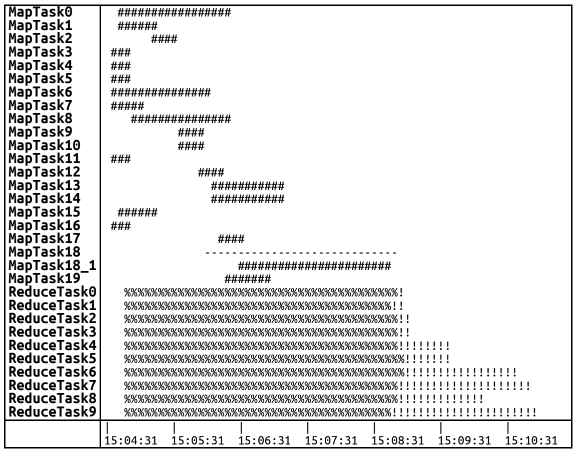

Figure 6 (b). Task progress of Job1 (Lab)

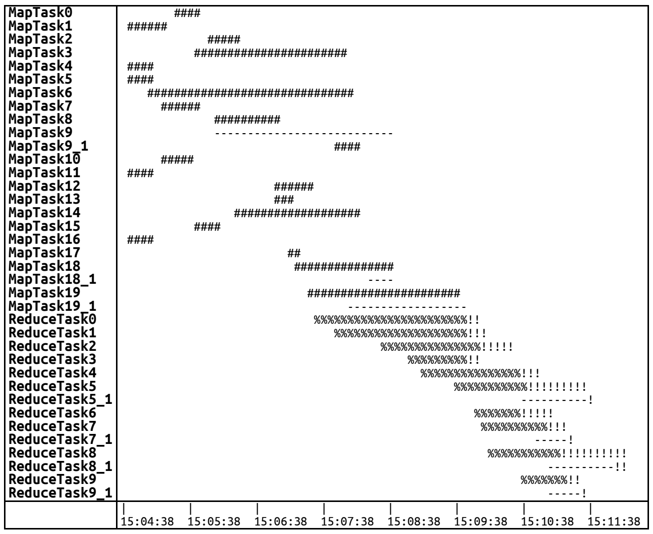

Figure 6 (c). Task progress of Job2 (Lab)

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. However, each of node N2~N4 has 4 cores and 4 map task slots but each has only 3 HDDs. Accordingly, 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 6 (c) shows that, in Job2, starting timing 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 are 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 to sort and reduce phase until all the map tasks have completed. The phenomenon can be observed in Figure 6 (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 [9]. This problem worsen reduce task slot starvation and cause a long delay in reduce task starting time which can be observed in Figure 6 (c).

3.3.3 Map Output Data Copy Time

As shown in Figure 5 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 of 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, following sort and reduce phase (as shown by "!" in Figure 6) would require frequent disk I/O.

3.3.4 Effect of Speculative Execution

3.3.4.1 Map Task

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 HDFS replication is 1 in Lab cluster, it further increase the congestion of disk I/O in the original node, thus prolong the execution time of both original and speculative tasks. Same phenomenon was also observed in other jobs.

3.3.4.2 Reduce Task

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 Section 2.3 do not work well when the cluster is suffering task slot starvation. As we can see from Figure 6 (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.

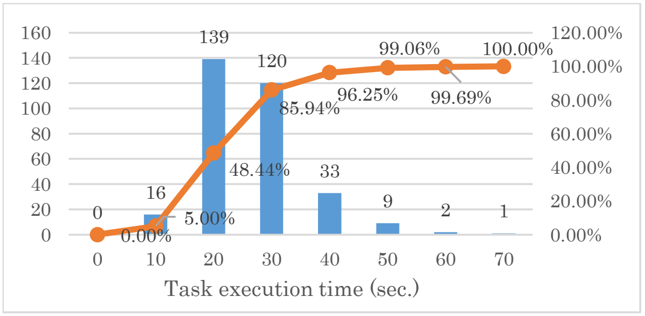

Figure 7. Cumulative distribution of map task execution time (EC2 cluster)

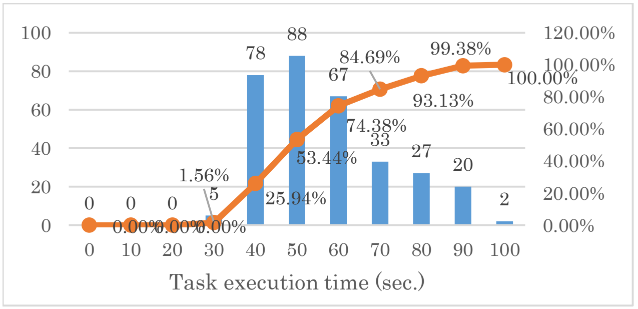

Figure 8. Cumulative distribution of reduce task execution time (EC2 cluster)

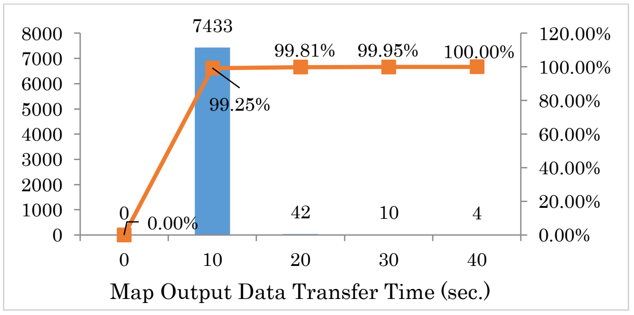

Figure 9. Cumulative distribution of map output data transfer time (EC2 cluster)

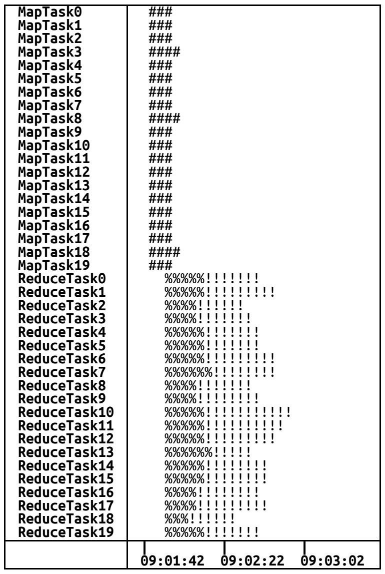

Figure 10 (a). Task progress of Job0 (EC2)

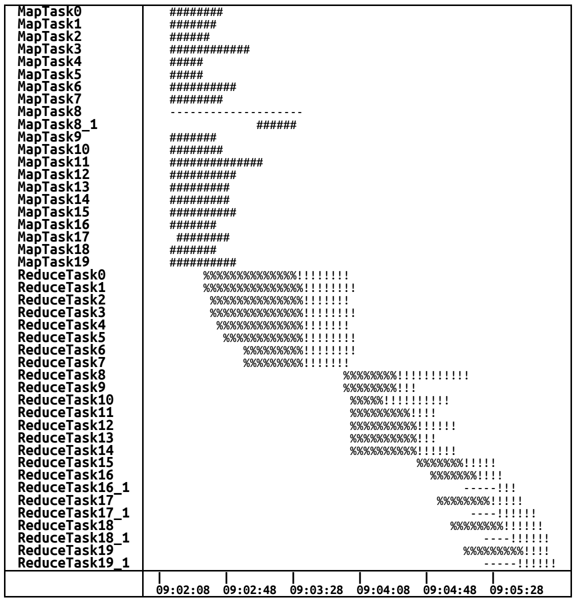

Figure 10 (b). Task progress of Job4 (EC2)

3.3.5 Summary

In Lab cluster, owing to the heterogeneity in disk performance among nodes, disk I/O bottleneck 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.

3.4 Analysis of Hadoop Performance in EC2 cluster

We made a similar analysis for the experimental results from EC2 cluster. Figure 7, 8 and 9 shows 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 10 (a), (b) is task progress Gantt chart of Job0 and 4. As Figure 10 (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 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 Figure 10 (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, 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. Discussion

4.1 Deterioration Mechanism of Hadoop Job Performance

By summing up our analysis of experiment results, Hadoop performance deterioration is thought to occur by the mechanism as shown in Figure 11.

1. Map task delays caused by data I/O congestion.
2. Reduce task hoards a reduce slot without doing anything.
3. Reduce slot starvation occurs and incurs a large variation to the start time of reduce tasks..
4. Useless speculative reduce tasks are launched and consume I/O resources and reduce task slot of the cluster.
5. Useless speculative reduce tasks delay other tasks and worsen reduce slot shortage.

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 task tracker 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.

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 [9] may be ideal, 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 data 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.

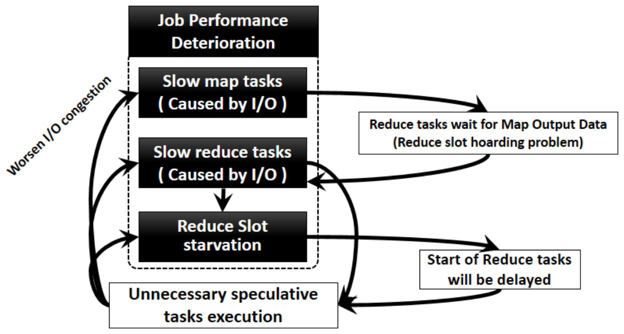
Job Performance Deterioration
Worsen I/O congestion
Slow map tasks (Caused by I/O)
Slow reduce tasks (Caused by I/O)
Reduce Slot starvation
Reduce tasks wait for Map Output Data (Reduce slot hoarding problem)
Start of Reduce tasks will be delayed
Unnecessary speculative tasks execution

Figure 11. MapTask Delay Propagation Effect

5. Conclusion

In this paper, we executed multiple jobs in parallel on experiment Hadoop clusters and monitored task progress to investigate the impact of data transfer on Hadoop job performance. We showed that delay at a particular task caused by data I/O bottleneck incurs delay in subsequent tasks and finally deteriorates job performance significantly. Based on these observations, we analyzed the mechanism of performance deterioration and revealed that the main cause is task slot hoarding in reduce node originally incurred by map task delay through disk I/O congestion. We further showed that speculative task executions bring adverse effects in some cases. Current conditions of speculative execution as implemented in existing Hadoop scheduler misidentify the reduce task that started late because of the shortage of task slots with a delayed reduce task. Although we explained the performance deterioration mechanism that is caused mainly by disk I/O congestion, the same mechanism applies for network congestion in shuffle phase. As for the measures against job performance deterioration, we suggested some methods. But the details of the methods and their effectiveness are the subject for further study.

6. REFERENCES

[1] Amazon EC2. http://aws.amazon.com/ec2.

[2] Apache Hadoop. http://hadoop.apache.org.

[3] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz. The Case for Evaluating MapReduce Performance Using Workload Suites. In MASCOTS, pages 390-399, 2011.

[4] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoika. Managing data transfers in computer clusters with orchestra. In SIGCOMM, pages 98-109, 2011.

[5] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In OSDI, pages 137-150, 2004.

[6] A. Konwinski. Improving MapReduce Performance in Heterogeneous Environments. Technical Report of EECS Department, University of California, Berkeley, 2009.

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

[8] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In EuroSys, 2010.

[9] M. Zaharia, D. Borthakur, J. Sen 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.