A Process-aware Security Task Scheduling Algorithm

Tao Zhang[1], Yuanyuan Ma[1], Cheng Zhou[1],Tao Li[2]
[1]China Electric Power Research Institute (Nanjing)
NARI Road.No.8, 210003 Nanjing, China
[2]School of Computer Science, Florida International University
11200 SW 8th Street Miami, FL 33199, U.S.A
zhoucheng3@epri.sgcc.com.cn

Journal of Digital
Information Management

ABSTRACT: *In this paper, a process-aware security task-scheduling algorithm is proposed as IOAware. The algorithm evaluates the performance of hardware in computing nodes, and estimates properties of a task while it is being executed. In subsequent assignments, the algorithm assigns tasks based on the performance of the TaskTracker and the task properties. To verify the theoretical feasibility of the proposed IOAware scheduling algorithm, a schedule model is proposed to implement the method. After applying the scheduling module in a Hadoop cluster with multiple experiments, the results show that, compared with the performances of the FIFO, computing capacity-scheduling, and fair-scheduling algorithms in terms of four aspects: response time, localization ratio of the data, system throughput, and system resources, IOAware scheduling algorithm can reach up to the disk IO effect of the Shared computing nodes, effectively reduce the execution time of the tasks and improve the throughput of the cluster.*

Subject Categories and Descriptors
I.1.2 [Computing Methodologies]: Analysis of Algorithms; **E.4 [Coding and Information Theory]**

General Terms: Computing, Task Scheduling, Process Aware

Keywords: Hadoop, Security Task Scheduling, Process Aware

Received: 3 June 2014, Revised 19 July 2014, Accepted 25 July 2014

1. Introduction

Data must be read in the same order that compute nodes execute a Map task. However, a non-local task must be first localized, which consumes some network resources and local disk bandwidth [1]. Thus, we can effectively reduce the network bandwidth of a data node and the disk IO operations by reducing the migration of data during the MapReduce task. For the task itself, data localization is not needed if the data are exactly local. Obviously, this step will still need a certain amount of time if there are a lot of data. So we should improve the Map task's localization rate to reduce the single execution time [2].

This paper proposes a process-aware security task-scheduling algorithm called IOAware. In this algorithm, the total IO requests for the parallel execution of tasks in the compute node do not exceed the maximum disk IO load, because different IO request tasks are allocated to each compute node.

Every single CPU is used to execute these tasks, to improve the throughput rate of the system and reduce the average execution time..

2. IOAware

If the MapReduce task is executed on a heterogeneous cluster, it is vulnerable to restrictions in the compute node resources, and can create a competitive relationship with other tasks running in the compute node [3]. The most important competitive relationship is between the CPU use, network bandwidth utilization, and disk IO. In the Yahoo recommended configuration, each CPU core can support a Map task. So under normal circumstances there is no competition for CPU resources between different Map tasks. But there is competition between disk IO and network IO, and the CPU use, which is complementary [4]. If a Map task becomes limited by he IO, then it must stop the CPU calculations and wait for the operation to be completed. Therefore, in heterogeneous clusters, we must identify characteristics of nodes in the cluster and the task itself. In this paper, we considered the following four points for MapReduce tasks and clusters:

(1) The Hadoop cluster is heterogeneous, and the hardware configurations are not identical. However, its X86 architecture does not support ARM or GPU computing.

(2) The Map or Reduce tasks are single responsibilities, and not support the completion of multiple computing processes.

(3) Each compute node's information can be given by its configuration information, which is obtained using a heartbeat on the management node.

(4) Speculative execution on heterogeneous clusters can easily degrade the performance, so speculative execution is not possible.

2.1 Assessment of Properties of the Compute Nodes

There are more Map tasks in a job than Reduce tasks. Therefore, it is more reasonable to analyze the Map task when analyzing the properties of distributed computing nodes. Because the Map operations within a job are very similar, it is effective to take samples of how a Map task is reading data. The executions of each task contain a lot of information [5].

Analyzed from the perspective of Map tasks, the amount of RW-Data is the sum of the input and output:

$$Data = input + output \tag{1}$$

We define:

$$R_{disk} = \frac{Data}{F_{cpu} \times T_{exec}} \times \lambda \tag{2}$$

Then using equation (2) to quantitatively estimate the disk performance of the computing nodes. We can use this estimate to determine the relative disk rates and the literacy rates of the disk using reference values from some basis nodes.

In Equation (2),T_{exe} is the execution time of the Map task, and F_{cpu} is the rate of a single-core CPU. λ refers to the parallel factor; more simultaneously running tasks are represented by a larger value. However the disk IO speed is limited, so its value must be constrained within a certain range. There may be many parallel Map tasks on the same TaskTracker. More tasks increase the frequency of disk IO calls making an obstruction more likely, which affects the execution of tasks.

When a fully loaded disk processes data, the execution time is inversely proportional to the rates of reading and writing the data in each computing node, when compared to IO type of tasks with the same data [6]. The execution time should reduce as the disk literacy rate increases.

TaskTracker communicates with JobTracker using a heartbeat on Hadoop, and reports information about the current node, including the computing node memory, the amount of CPU, the total CPU frequency, memory restrictions of the configured Map and Reduce computing slots, and the action resource status.

$$F_{cpu} = \frac{F_{total}}{N_{cpu}} \tag{3}$$

where F_{total} is the total CPU frequency reported by a heartbeat, and N_{cpu} is total size of the CPU. This must then be normalized. We can then estimate the disk IO speed and execute the corresponding scheduling tasks according to the IO load of tasks in the task allocation.

2.2 Jobs Classification

Hadoop jobs can be divided into two types: CPU-bound and IO-bound. When these jobs are being executed, there are larges differences in their resource requirements for the computing nodes. The jobs can be roughly classified according to their properties. When executing Map-Shuffle, we can estimate the workload of the Map task using the ratio of MID (Map input data) to MOD (Map output data). Assuming that the ratio is ρ, we can calculate the value using the following equation:

$$MOD = \rho \times MID \tag{4}$$

Because the data are returned from Map's Shuffle process, SID (Shuffle input data) during the Shuffle satisfies can be considered as follow:

$$SID = MOD \tag{5}$$

Shuffle reorganizes the output files in the Map process using division, which is determined based on the number of Reduce tasks. So SOD (Shuffle output data) is a parameter concerning the division factor.

We assume that the load ratio (ρ) of each Map task is equal, because of the similarities among the properties of Map tasks from the same job. We assume that there are n Map tasks on the TaskTracker at the same time, that the Map task's execution time is the MTCT (Map task completed time), and that the literacy rate of the disk IO is the DIOR (disk IO rate).

When the Map tasks are being executed, we need disk operations for inputting the data slice, outputting the intermediate data, and inputting and outputting the Shuffle data. During the execution process, we assume that n

tasks are simultaneously running on the TaskTracker, and that they share disk IO. We assume that the product of n and the sum of MID, MOD, SID, and SOD is the total amount of data required by these tasks. Then, this amount divided by the tasks execution time is the average utilization rate of the disk. The task is CPU-bound if the average utilization rate of that disk is less than the average rate of the local disk. That is,

$$\frac{n \times (MID + MOD + SID + SOD)}{MTCT} = \frac{n \times ((1 + 2\rho)\, MID + \ SOD}{MTCT} < DIOR \qquad (6)$$

Disk IO is used more frequently during the execution of an IO-bound task. Additionally, because the disk IO usage and CPU usage are complementary, disk IO can encounter bottlenecks if the disk file read rate of the Map tasks is relatively high. This reduces the CPU utilization. There are two kinds of IO-bound tasks, which are dependent on the obstruction. In the first, the disk IO encounters its read and write bottleneck during the Map stage, i.e.,

$$\frac{n \times (MID + MOD)}{MTCT} = \frac{n \times (1 + \rho)\, MID}{MTCT} \geq DIOR \qquad (7)$$

In the second, the IO operation is less than the literacy speed and the IO is blocked during the Shuffle stage.

Equation (7) suggests that the Map task generated a large amount of disk IO operations before the Shuffle. There are n Map tasks competing for the disk, which obstruct the Map tasks and make them abandon their CPU time slices. Therefore, it takes more time to perform tasks. In the second case, Map tasks before the Shuffle process generate a large number of IO disk operations. When there are n Map tasks running in parallel, the IO disk encounters its bottleneck, leading to an obstruction. That is,

$$\frac{n \times (MID + MOD + SID + SOD)}{MTCT} = \frac{n \times ((1 + 2\rho)\, MID + SOD)}{MTCT} \geq DIOR \qquad (8)$$

2.3 Task Obstruction

Map tasks are locally executed on the computing node. Some factors influence its execution time, including CPU, memory, and disk rate [7-8]. To make a quantitative comparison, we define the comparative factor as follow:

$$R = \frac{Data}{\alpha\, Proc + \beta\, Disk} \qquad (9)$$

where α and β are rate factors, which respectively represent the proportion of CPU and disk affecting the implementation time in the task execution process, and $\alpha + \beta = 1.$ is the proportion of CPU after normalization. is the proportion of the disk estimate rate after normalization. We can use this equation to make a prediction based on the execution information of the completed Map tasks, and deduce the execution time of the current Map task in the computing node.

Thus, we can calculate the expected execution time of a Map task. Assuming that the task has been executed for T_{real}, and progressed by $Progress_{real}$. If the execution speed of the task is more than the threshold level, the compute node makes frequent calls to the disk IO. This results in the task becoming blocked, and seriously effects the execution of the task. The judgment method is shown as follow:

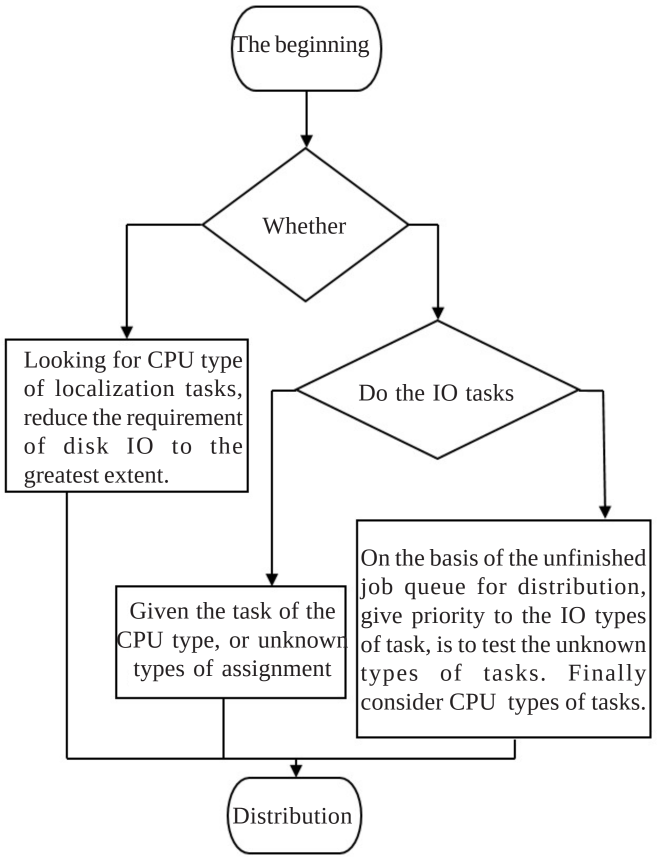
The beginning
Whether
Looking for CPU type of localization tasks, reduce the requirement of disk IO to the greatest extent.
Do the IO tasks
Given the task of the CPU type, or unknown types of assignment
On the basis of the unfinished job queue for distribution, give priority to the IO types of task, is to test the unknown types of tasks. Finally consider CPU types of tasks.
Distribution

Figure 1. Flow diagram for the IOA dispatcher task

$$\begin{cases} Rate_{expect} = \dfrac{1}{T_{expect}} \\ Rate_{real} = \dfrac{Progress_{real}}{T_{real}} \\ \dfrac{Rate_{except} - Rate_{real}}{Rate_{real}} \times 100\% > Threshold \end{cases} \qquad (10)$$

When a TaskTracker proposes assigning tasks, the JobTracker assigns tasks using scheduling algorithms. The scheduling algorithm in this article is determined by the running conditions of the machine when tasks are being allocated from the schedule. When the JobTracker receives "*heartbeat*" packages, it updates the execution information of the appropriate jobs. It then assigns new tasks to the compute nodes as appropriate. The flow diagram of the algorithm is shown in Figure 4. The specific strategies for assigning tasks are as follows.

Step 1: First, determine if there are seriously blocked IO-bound tasks, based on the processes of the tasks performed on the machine. If there have been serious obstructions, then execute the second step. If not, then carry out the third step.

Step 2: Because IO-bound tasks have been blocked, then there are reading and writing tasks that require serial read-write floppy disk operations. This prolongs the implementation of these tasks. However, the TaskTracker can calculate the slots, and can also add new tasks. To allow the TaskTracker to be as effective as possible and minimize the impact of the execution process of an existing task, we select data retention from the CPU-bound queue on the local DataNode, select less input and output data, and allocate tasks with high CPU utilization. If we cannot find this type of task, then we discard the wheel scheduling, and wait for the next dispatch to the TaskTracker.

Step 3: When the TaskTracker disk is used too often and a task block does not appear, we must consider the current overall situation of the tasks to be performed on the machine. If there are more TaskTracker IO-bound tasks than CPU-bound tasks, we implement Step four, otherwise we move to Step 5.

Step 4: If there is no blocking of IO-bound tasks, we can add some CPU-bound tasks to increase the CPU utilization. On the premise of not exceeding the disk IO bottleneck, these new tasks can be performed without significant impact; therefore, we chose from the CPU-bound task queue that uses local data. If there is no CPU-bound task, then we schedule a task from the unknown type wait queue that uses localized data. If we do not find a localized task, the schedule waits for the next task.

Step 5: If the machine is more IO-bound than CPU-bound, the task blocking has not occurred under normal circumstances. So, demanding tasks on the disk are given priority. We first select data from the IO-bound task queue with localization, then the localized wait queue, then from the unknown, and finally by considering the localization of selected tasks from the CPU-bound queue. If we cannot select an appropriate task, we relax the localization data requirements. We consider tasks that are within the same rack. If there are still no appropriate tasks, we consider the tasks across a rack.

3. Experiments and Analysis

To test the performance of the scheduling algorithm in heterogeneous clusters, we used computers that had been procured at different times. Their hardware configurations are given in Table 1.

Name	Global Settings	Map Slot	CPU Frequency (Frequency of core* The number of core)
Master	2 Cores, 2G Memory	/	/
Slave1	1 Core, 512M Memory	2	2.80GHz*1
Slave2	2 Cores, 1G Memory	2	2.93GHz*2
Slave3	2 Cores, 1G Memory	2	2.93GHz*2
Slave4	4 Cores, 2G Memory	4	2.33GHz*4

Table 1. Hardware configuration for experimental cluster

These computers were connected through the lab network, which has a network bandwidth of 100 M. To implement the computing cluster configuration, we used a virtual machine and an entity matching deployment scenario. JobTracker and NameNode used entities, and TaskTracker and DataNode used VMware virtual machines, with the networks configured in bridge mode. All of these machines were running the Ubuntu 10.10 operating system. Hadoop uses Java, so JDK Ubuntu was installed in this test system. This was to ensure that in follow-up experiments, all the users in the system with the same username and password could build the environment. The entire test network is shown in Figure 2.

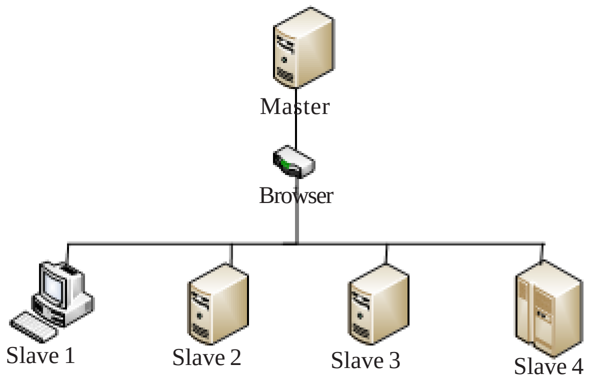

Figure 2. Experimental network

	Map Task	Reduce Task	Input Data
WordCount	39	1	2.5GB
Grep	36	1	5MB
TeraSort	16	1	1GB

Table 2. Jobs used in the experiments

To verify the scheduling model proposed in this paper, we used Hadoop security task scheduling for three test cases: WordCount, Grep, and TeraSort. Details for these three jobs are given in Table 2.

WordCount is a statistical count of identical words in the input data. We used the national level six library as the word generator source, which contains a total of 2.5 G of data and is stored in HDFS.

Grep is used to learn regular expressions, search jobs, and enter data using regular expressions in a user submitted search expression. This job contains two

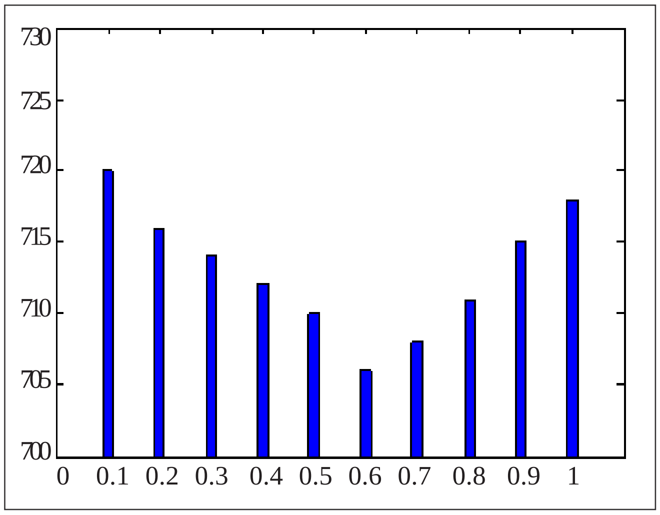

Figure 3. Comparison of the effect of different threshold levels on the response time

operations: Grep-Sort and Grep-Search. Grep-Search implements the regular expression searches. When the job finishes, the Grep-Search result is submitted to Grep-Sort, which sorts the data and returns it to the user. The experiment uses a randomly generated 5 MB of data.

For massive data, the sort operation can typically be used to measure the processing power of a distributed data processing framework. Terasort sorts Hadoop jobs. In 2008, it took Hadoop 209 s to successfully order 1 TB of the data, which won the first prize of good grades. Each Map task has a relatively large amount of data, so it cannot be saved in memory. The intermediate data can be saved to the local cache. The job has certain requirements for disk read and write IO speed.

We must specify a compute node according to the experimental reference coordinates of the whole cluster. Therefore, we selected the Slave2 reference coordinates, and used disk speed test tools on the local disk. After testing, the average read and write speed of the disk IO was 39 MB/S. The numerical value and machine name were used for parameter configuration in the IOAware scheduling algorithm in the configuration file.

3.1 Parameters

The scheduling algorithm for IOAware determines whether a machine performing certain tasks appears to be blocking an important parameter using a threshold level ($Threshold$). The threshold scheduling algorithm for IOAware uses the current task execution time to judge if the threshold has been exceeded, which indicates that the machine is blocked. The threshold is an unknown quantity, so we constantly changed it over the course of our experiments. This meant that we did not affect the performance of the IOAware scheduling algorithm. We based its values on many tests of the machines in the cluster.

Because the IOAware scheduling algorithm needs hardware information for the entire cluster, JobTracker and TaskTracker must be reset each time the configuration file is modified. After the daemon is restarted, several assignment submissions must be made to the scheduling algorithms, using "*heartbeat*" messages to collect configuration information for the whole cluster. Because there were only four test cluster machines, WordCount jobs were sufficient.

	WordCount	Grep	TeraSort
FIFO	711 sec	123 sec	294 sec
Capacity	710 sec	122 sec	300 sec
FairShare	712 sec	124 sec	301 sec
IOAware	706 sec	122 sec	296 sec

Table 3. Response times for an example job

	Map Input	Map Output	Shuffle Output	MTCT
WordCount	64 MB	49 MB	49 MB	36 sec
Grep	142 KB	5 KB	5 KB	5 sec
TeraSort	64 MB	64 MB	64 MB	8 sec

Table 4. Parameters of the example job using IOAware scheduling

In this experiment, we had to set a reasonable threshold interval. We set the initial value of the threshold to 0.1, and increased it to 1 in steps of 0.1. When the cluster was stabilized, we submitted the job 10 times, and recorded the average response time. The results of the experiment are shown in Figure 3. A threshold value of 0.6 minimized the average response time of the job.

3.2 Response Time

After obtaining an optimal threshold, we set the threshold of the configuration file to 0.6. We then carried out experiments to verify the validity of the IOAware security task scheduling algorithm. We submitted three example jobs 10 times, and recorded the average response time of the job. So that we could compare our results with other scheduling algorithms, in this experimental queue we did not set the computing power, we used the default job queue, we used 100% computing power for fair scheduling, the algorithm used the same configuration, there was no extra set up of multiple job waiting queues, and it used a separate job queue. The average response time of three-sample jobs are shown in Table 3.

The results show that the average response times for the three jobs were basically the same. The times taken by IOAware, FIFO, the capacity scheduling algorithm, and the fair scheduling algorithm were within a few seconds of each other. This is partly because the IOAware scheduling algorithm for the data localization ratio is higher than the other two, which reduces the time slices for downloading data, and reduces disk IO operations.

The IOAware scheduling algorithm uses the demand for

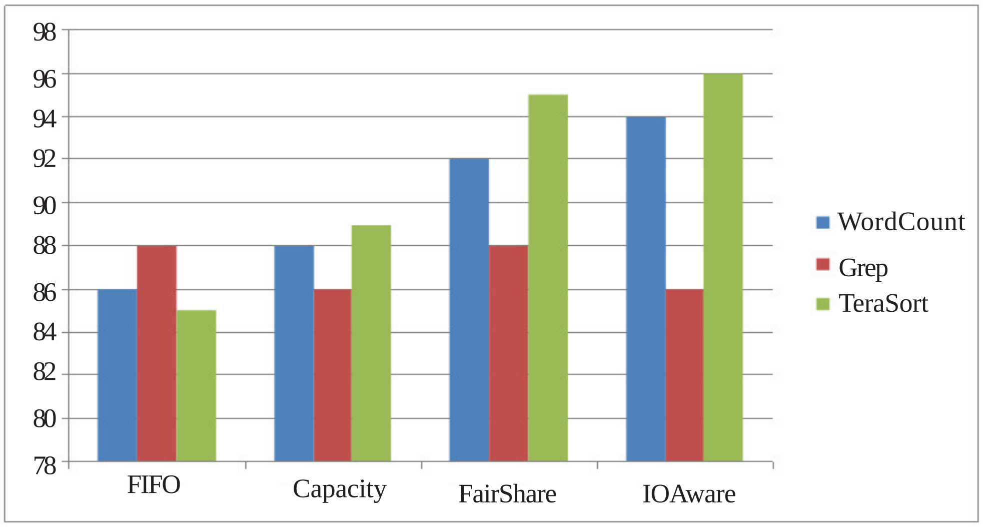

Figure 4. Data localization rate of example jobs

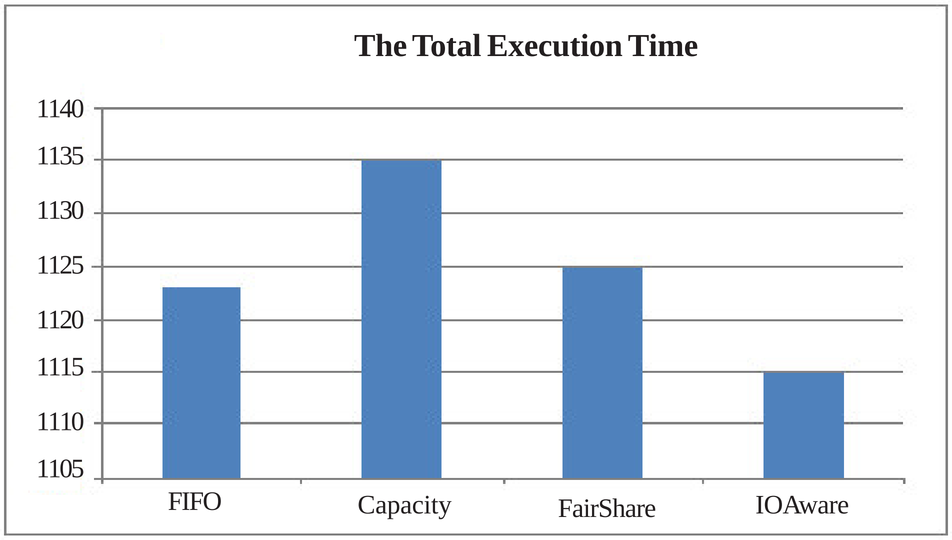

Figure 5. Total execution time

reading and writing data and the execution of tasks to determine a task's properties. Table 4 shows the results using IOAware to assign tasks according to the specific parameters in real time.

X is calculated according to Equation (X). WordCount requires CPU-bound tasks jobs. The jobs are initialized into the execution queue, then the system determines that they are CPU-bound tasks and moves them from the unknown type jobs queue to the job queue for CPU-bound jobs. Grep uses complex input pattern matching expressions, so it is a CPU-bound job. When the dispatcher recognizes that it is a CPU-bound task, it transfers it to the CPU-bound waiting queue. TeraSort requires a great deal of reading and writing to disk. The IOAware scheduling algorithm uses Equation (X) to determine that the TeraSort tasks are IO-bound, and transfers them to the IO-bound job queue.

3.3 Data Localization

Task input data localization can effectively reduce network bandwidth consumption, and disk IO operations can reduce the number of compute nodes. Therefore, we can enhance the localization rate of the data to effectively reduce the execution time of a task. We executed the example jobs X times to obtain averages.

As can be seen from Figure 4, the localization rate of the FIFO and capability scheduling algorithms was more than 80%. There were only four compute nodes in the experimental cluster, and two data backups. Therefore, it was likely that we would obtain input data from a TaskTracker that was performing tasks. However, the proportion of obtaining local data cannot be on behalf of these two algorithms' consideration for data localization when scheduling tasks. However the data localization rate of the fair scheduling algorithm was very high. For WordCount and TeraSort, the localization rate of the fair and IOAware scheduling algorithms were approximately 1% different, whereas for Grep, the data localization rate of the fair scheduling algorithm was more than that of IOAware.

We must consider the pressure that reading and writing

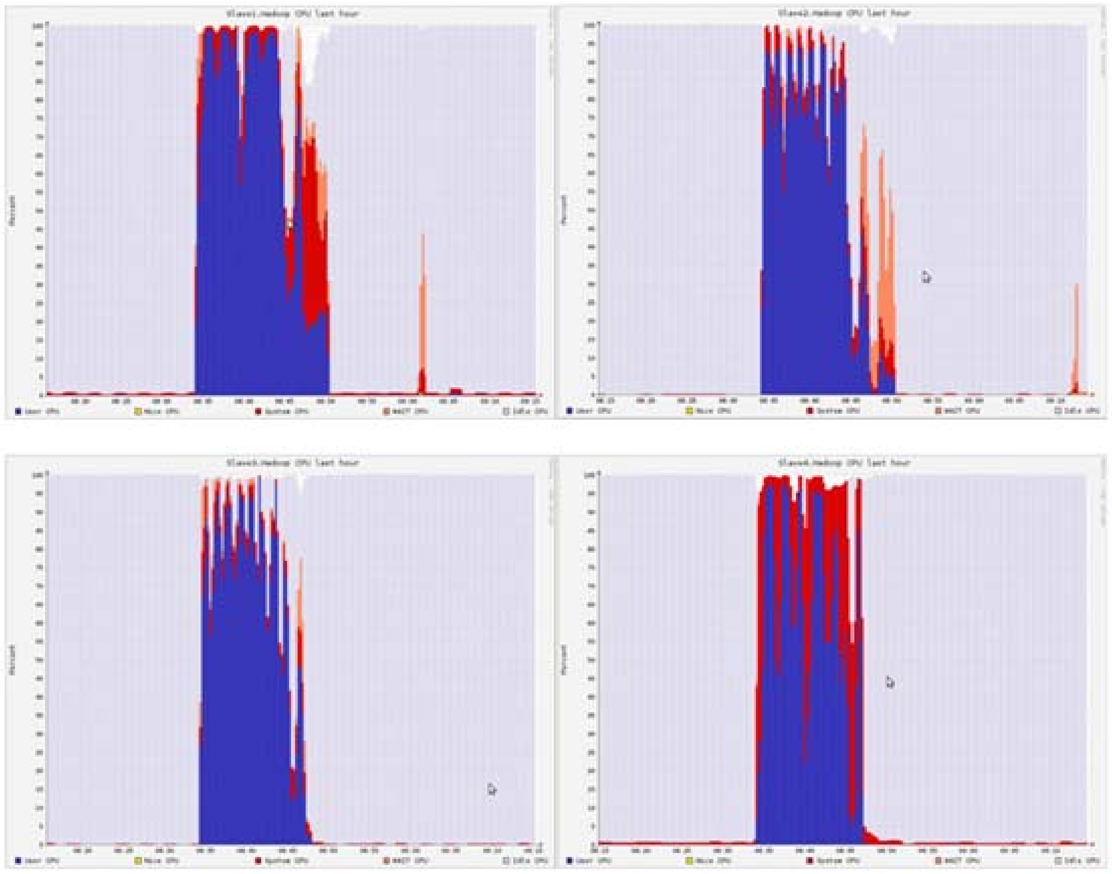
Figure 6. FIFO scheduling algorithm, CPU utilization

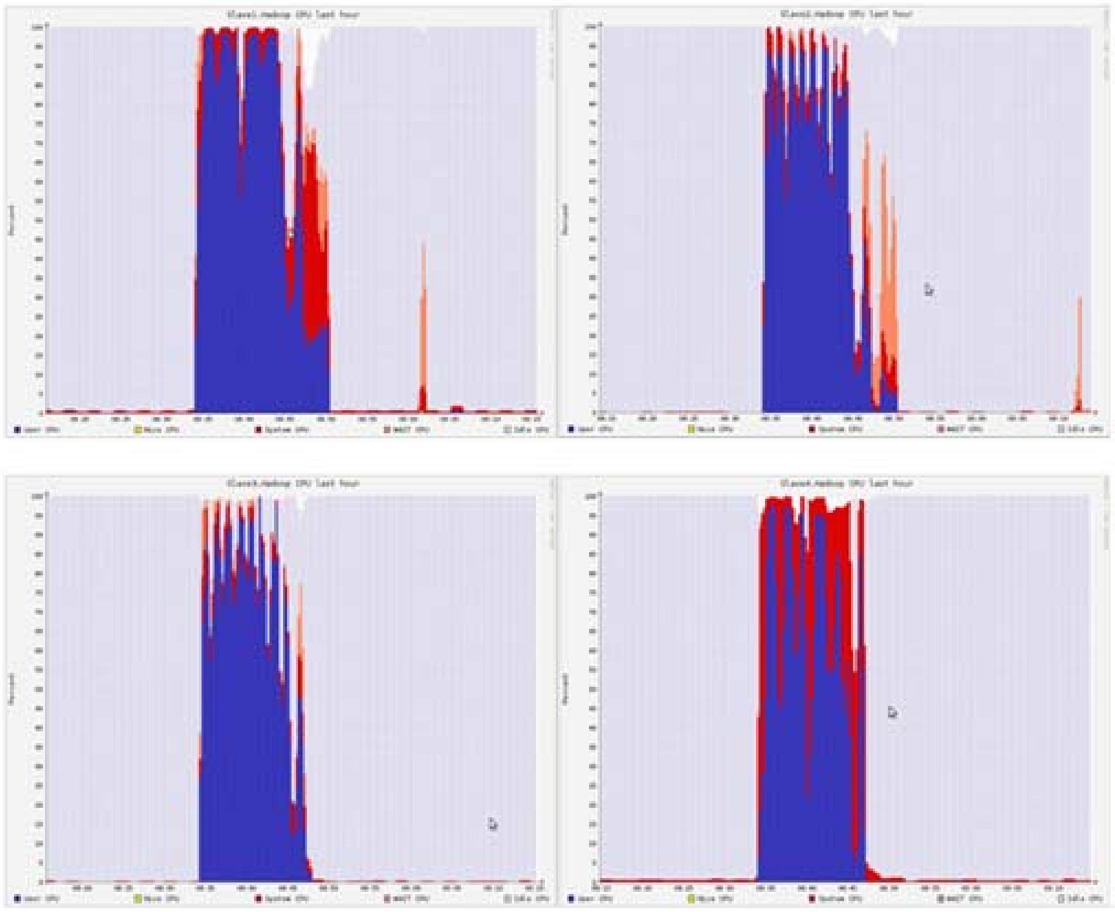
Figure 7. Capacity-scheduling algorithm, CPU utilization

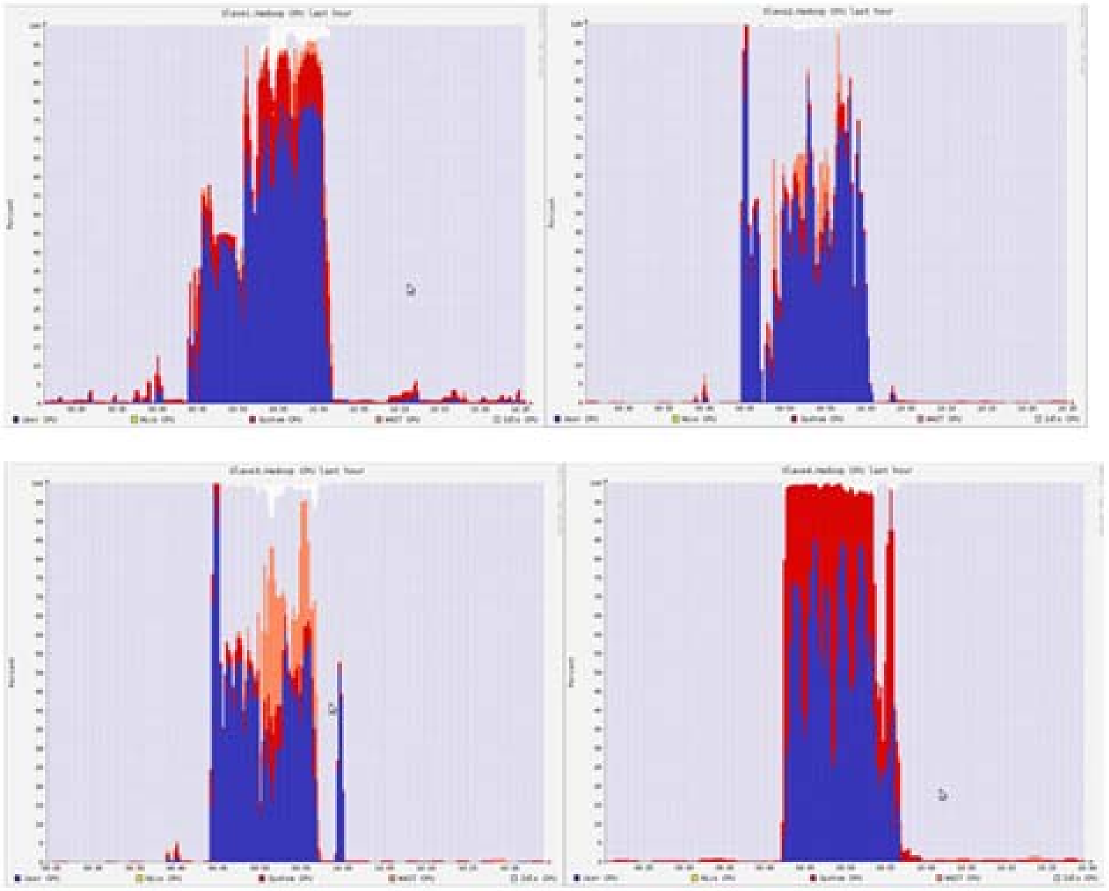

Figure 8. Fair-scheduling algorithm, CPU utilization

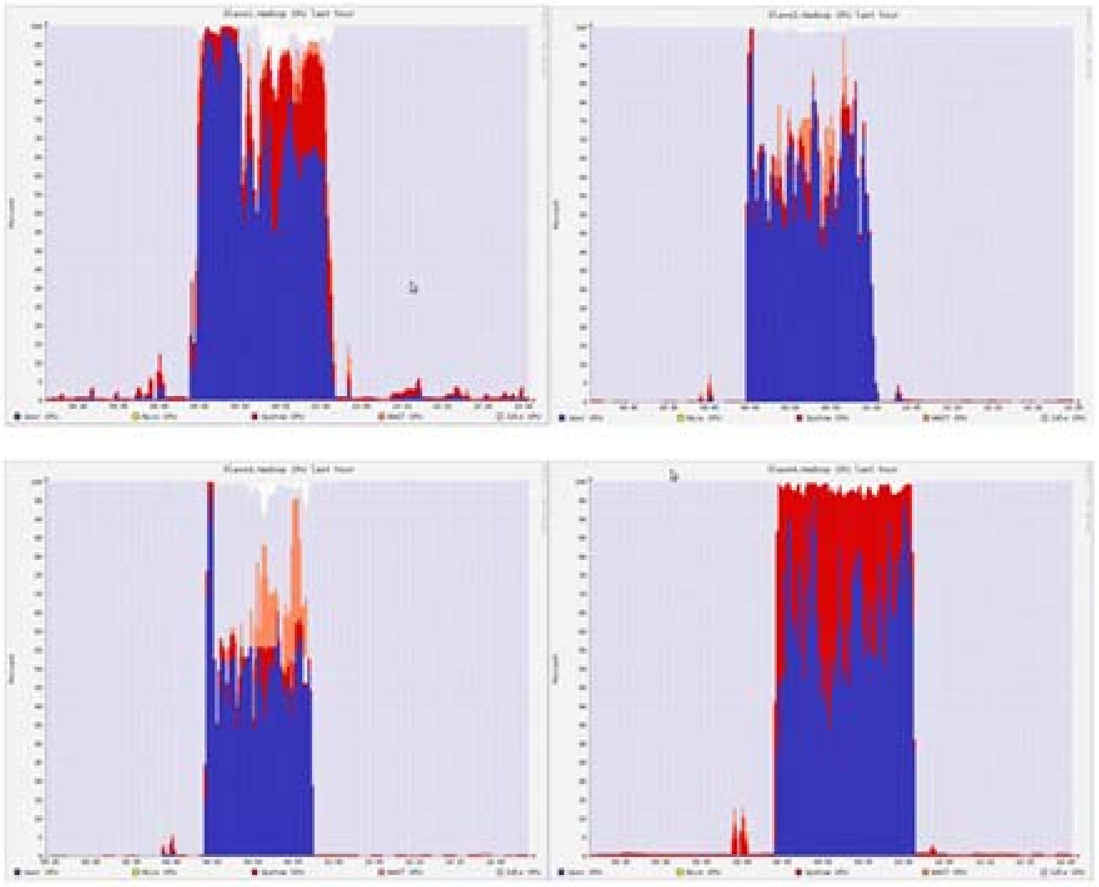

Figure 9. IOAware scheduling algorithm, CPU utilization

data puts on the compute nodes' disk, when the IOAware scheduling algorithm is in the process of security task scheduling. We must improve the data localization rate as much as possible, which can effectively reduce the required network bandwidth and the pressure on the local disk's reading and writing. Therefore, the IOAware scheduling algorithm's data localization rate is higher than that of the FIFO and capability scheduling algorithms.

3.4 System throughput and resource utilization

System throughput can effectively indicate the total number of jobs that the system can perform over a period of time. To compare the speed of clusters performing tasks under the IOAware scheduling algorithm, we simultaneously submitted three examples to the cluster. The total execution time is shown in Figure 5.

From the figure, we can see that the IOAware scheduling algorithm effectively improved the throughput of the system. Under FIFO scheduling, the total execution time for the three operations was 1,123 seconds. Using the computing capacity scheduling algorithm, the total execution time for the three jobs was 1,135 seconds. The fair scheduling algorithm shares the resources among multiple queues, so the total execution time for the job is related to the order of the job submission and the job waiting queue, if we take multiple queue submissions for jobs. For example, if CPU-bound and IO-bound jobs are submitted to the same queue, the total execution time for these operations can be very small. However, if these two types of tasks compete for the same IO resources, the total execution time increases. In this experiment, the fair scheduling algorithm only used a separate queue, and the total execution time was 1,125 seconds, whereas the IOAware scheduling algorithm had a total execution time of 1115 seconds.

When these three example jobs were executed at the same time, the IOAware scheduling algorithm reasonably planned their execution order. The IOAware scheduling algorithm improved the proportion of the tasks' data localization, thus reducing the frequency of reading and writing to the disk. The IOAware scheduling algorithm can run different types of tasks on the same TaskTracker node so that the disk bandwidth is shared, which can effectively reduce the execution time of the task.

During the execution of the task, we monitored the cluster resource utilization using Ganglia. The FIFO algorithm uses the jobs' submission times to schedule tasks. When we submit WordCount, Grep, and then TeraSort, the CPU utilization peaks when the TaskTracker performs CPU-bound tasks, whereas when the TaskTracker performs IO-bound tasks, the CPU utilization is relatively low. This is because different jobs have different CPU requirements. Figure 6 contains information regarding CPU resource utilization using the FIFO scheduling algorithm.

However, the ability-scheduling algorithm did not assign multiple queues in this experiment. It assigned a default queue, that used 100% of the computing capacity of the cluster. The queue used a FIFO algorithm for its internal scheduling method. Therefore, the CPU utilizations of the capacity-scheduling and FIFO algorithms are basically the same. The CPU utilization depends on the task being executed, it was sometimes more than 80% and sometimes less than 15%. Figure 7 shows the CPU resource utilization for the computing capacity-scheduling algorithm.

The fair-scheduling algorithm supports multiple queues. Therefore, to reasonably use the system resources, we combine the different attributes of the tasks. In this experiment, WordCount and Grep were combined into the same queue, and TeraSort was in a different queue. The CPU utilization is shown in Figure 8.

IOAware considers the amount of data being read from and written to the disk when scheduling tasks. However, in general, too much reading and writing to disk reduces the CPU utilization, and less increases the CPU utilization. Therefore, controlling the frequency of reading and writing to disk within a certain range can effectively improve the overall CPU utilization. Because the IOAware scheduling algorithm combines IO-bound and CPU-bound tasks, and allows them to run on the same node, the peak period of CPU utilization is no longer than the FIFO or capacity-scheduling algorithms. Additionally, the vibration amplitude in the CPU utilization graph is no larger than the other two scheduling algorithms. The CPU utilization using the IOAware scheduling algorithm is shown in Figure 9.

4. Conclusion

In this paper, we proposed the IOAware scheduling algorithm. The algorithm analyzes Shuffle, reading and writing processes when tasks are being implemented. It determines properties of the tasks according to their demand for disk IO, and classifies them as either CPU-bound or IO-bound. It combines different types of tasks, reducing multiple simultaneous disk IO operations and the possibility of blocking the disk. When considering a task's properties, scheduling algorithms use improvements to the localization rate of the input data as an important indicator. Increasing the data localization rate can reduce the data transmission among multiple data nodes in the cluster, and the local machine's disk operations. This can effectively reduce the execution time of tasks.

We validated the proposed IOAware scheduling algorithm using some experiments. Compared with the performances of the FIFO, computing capacity-scheduling, and fair-scheduling algorithms in terms of four aspects: response time, localization ratio of the data, system throughput, and system resources, our experiments showed that, the IOAware scheduling algorithm can improve the localization rate of data, reducing the amount of data downloaded and the frequency of reading and writing disk. It can also effectively improve the system throughput, combine CPU and IO resources on the compute nodes, so that the disk utilization is controlled within a reasonable range. Then, the CPU and disk utilization will be reasonable.

References

[1] Rasooli A., Down D., (2011). An adaptive scheduling algorithm for dynamic heterogeneous Hadoop systems. *In*: Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research. IBM Corp, p. 30-4.

[2] Qian-mu Li, Jun Hou, Yong Qi. (2013).A classification matching and conflict resolution method on meteorological disaster monitoring information. *Disaster Advances,* 6 (1) 415-421.

[3] Fair Scheduler Guide. http://hadoop.apache.org/common/docs/current/fair_scheduler.html

[4] Moseley B., Dasgupta A., Kumar R., et al. (2011).On scheduling in map-reduce and flow-shops. *In*: Proceedings of the 23rd ACM symposium on Parallelism in algorithms and architectures. ACM, p. 289-298.

[5] Yang Xia, Lei Wang, Qiang Zhao, et al. (2011). Research on Job Scheduling Algorithm in Hadoop. *Journal of Computational Information Systems,* 7(16)5769-5775.

[6] Matthew L. M.,Brent N. C., David E. C. (2004). The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30 (7) 817-840.

[7] Zhong, M., Shen, K., Joel, S. (2008). Replication degree customization for high availability. A*CM SIGOPS Operating Systems Review,* 42 (4) 55-68.

[8] Pike, R., Dorward, S.,Griesemer ,R., Quinlan S. (2005). *Interpreting the Data: Parallel Analysis with Sawzall. Scientific Programming Journal*, 13 (4) 227-298.