Algorithm Rebuilding and Performance Optimization of MapReduce in Hadoop

Zhao, Ben

(出版者 / Publisher)
法政大学大学院情報科学研究科
(雑誌名 / Journal or Publication Title)
法政大学大学院紀要. 情報科学研究科編 / 法政大学大学院紀要. 情報科学研究科編
(巻 / Volume)
9
(開始ページ / Start Page)
61
(終了ページ / End Page)
66
(発行年 / Year)
2014-03
(URL)
https://doi.org/10.15002/00010523

Algorithm Rebuilding and Performance Optimization of MapReduce in Hadoop

Zhao Ben
Computer and Information Science, Hosei University
Tokyo, Japan
Email: ben.zhao.3m@stu.hosei.ac.jp

***Abstract*—As a core component of Hadoop that is a cloud open platform, MapReduce is a distributed and parallel computing model based on mapping function for processing and generating large data sets. MapReduce abstracts business logic from implementation details, and provides powerful interfaces for programmers to use. It can mask the underlying specific implementation processes and efficiently reduce the distributed and parallel computing difficulty, and has high reliability, high scalability, high efficiency, high fault tolerance features. However, MapReduce mechanism itself is not perfect and mature, and needs to be improved the efficiency further. According to analyzing mapreduce principles and performance indicators, in heterogeneous environments, unreasonable resource scheduling, data transmission and system parameters are summarized. By rebuilding algorithms and optimizing performances, sliding window scheduling algorithm (SWSA), changing transfer protocol from HTTP to UDT and optimization of system configuration parameters are proposed. At last, this paper compares performance difference of MapReduce before optimization with after optimization. The algorithms are verified by experiments, rebuilding and optimization greatly improve performance of hadoop.**

Keywords—MapReduce; Hadoop; cloud computing; scheduling algorithm; transmission protocol; system parameter

I. INTRODUCTION

Under the stimulation of increasing the amount of users and data processed, problems of Internet applications follow. On the one hand, the vast amount of data requires huge scale storage resources as a basis. On the other hand, as net application dependent on data is increasing, the demand of capacity of processing massive data grows, the cost of maintaining data storage and processing data is increasingly higher. Under the application requirements of resources and computing capacity and related technology development, cloud computing as a new model is put forward [1].

Cloud computing [2][3] is an internet-based service commercial calculation model, which cloud be dynamically increased, used and delivered, and usually involves some dynamic, scalable and virtualized resources (computing power, storage space and information services) that are provided by Internet.

Hadoop developed by Apache, is an open-source distributed architecture. MapReduce is a parallel computing and distributed computing model of cloud computing for calculating and processing big data, meanwhile it is also a core sub-project of Hadoop [4].

In this paper, by rebuilding algorithm and optimizing performance, efficiency of processing big data in isomorphic and heterogeneous environment is improved.

II. CURRENT TECHNOLOGIES AND THEIR OPTIMIZATION

A. Current technologies

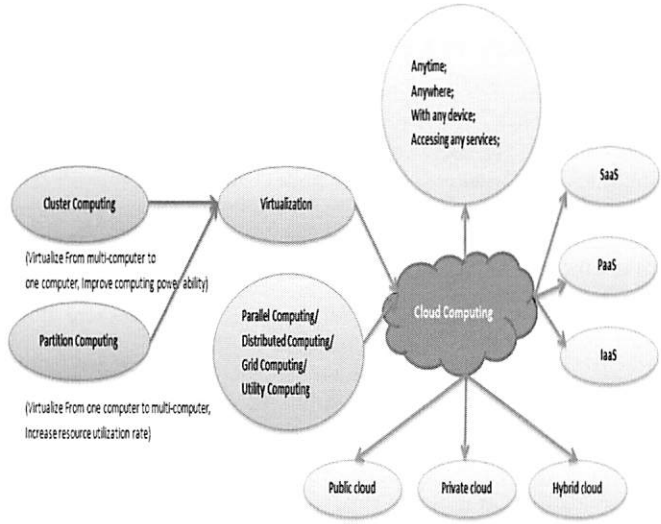


Fig. 1 Introduction of cloud computing

Cloud computing is a product of integration and development of traditional computer and network technology, such as parallel computing, distributed computing, grid computing, utility computing, virtualization and so on. And in virtualization technology, it includes cluster computing (virtualize from multi-computer to one computer, improve computing power ability) and partition computing (virtualize from one computer to multi-computer, increase resource utilization rate).

Cloud computing has three types (public cloud, private cloud, hybrid cloud) and a three-tier service structure that includes IaaS (Infrastructure as a Service), PaaS (Platform as a Service), SaaS (Software as a Service). Meanwhile, cloud computing has the characteristics of accessing any services at anytime, anywhere, with any devices.

In this paper, the used key technologies are MapReduce and HDFS that are the core technologies of hadoop that is an open-source distributed architecture based on cloud computing.

1) Work principle of MapReduce

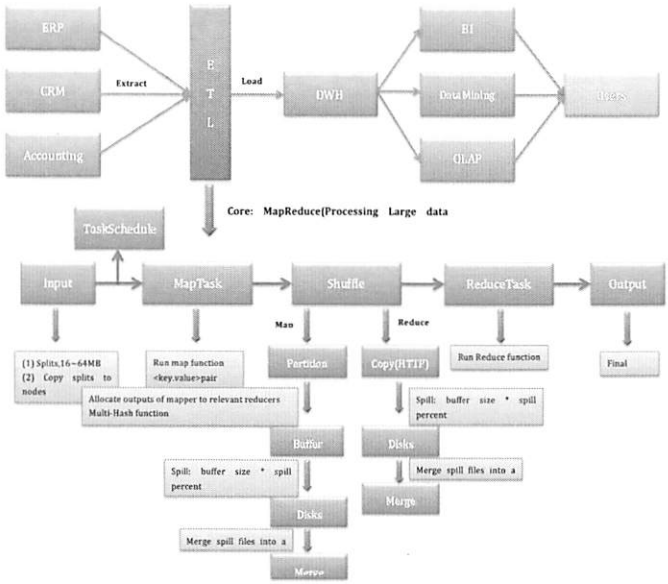


Fig. 2 Principle of MapReduce

Principle of MapReduce is divided into five parts, that are input, maptask, shuffle, reducetask and output. Meanwhile, taskschedule is between input and maptask. The cluster includes JobTracker (master node) and TaskTracker (slave node). JobTracker is responsible for scheduling and monitoring tasks, and TaskTracker is used to executing tasks, including map task and reduce task.

- Input: Original big data linking to Mapreduce library as input of user program is divided into M splits (16~64MB), and then splits are copied to other machines using fork within the cluster.
- Map task: TaskTracker assigned map tasks starts to read the corresponding splits as input data. Key-value pairs are extracted from input splits and passed to map function as parameters. Intermediate key-value pairs produced by map function are cached in memory.
- Shuffle: Intermediate key-value pairs as output of map tasks are allocated to relevant reducers using multi-hash function by partition function, cached in the buffer and periodically written to the local disk. These spill files including key-value pairs are sorted, combined, merged into a file, and copied to the corresponding reduce tasktrackers by HTTP protocol. In one reduce tasktracker, different keys of key-value pairs are sorted and combined, and spill files are merged into a file.
- Reduce task: Reduce tasktracker is responsible to running reduce funciton.
- Output: Output of reduce tasktracker is added to the output file. When all of the Map and Reduce functions are completed, master wakes up user program, and MapReduce call codes that return to user program.

2) Running process of MapReduce

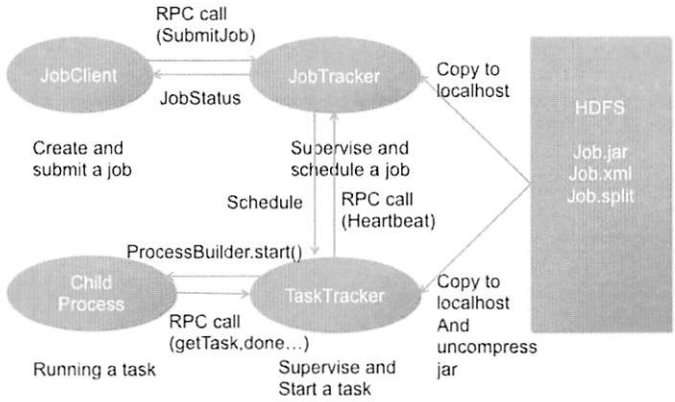


Fig. 3 Running process of MapReduce

3) HDFS

HDFS (Hadoop Distributed File System) is a distributed file system and has the characteristics of high fault tolerance [5].

HDFS is a master-slave structure, and consists of a namenode and some datanodes. Internal mechanism is that a file is divided into one or more blocks, which are stored in a set of data nodes. Namenode is used to manipulate the file of file namespace or directory operations such as open, close, rename, and so on. It also determines the mapping between block and datanodes. Datanode is responsible for handling read and write requests from the file system of users. Datanodes also perform block creation, deletion, and block copy instruction of namenode [6].

B. Optimization of current technologies

1) Rebuilding and optimization of scheduling algorithm (macroscopic)

In heterogeneous clusters, based on the original scheduling algorithm, sliding window scheduling algorithm is proposed (SWSA), can reasonably assign to the appropriate tasks, make full use of computing resources to solve the problem of load unbalance.

2) Rebuilding and optimization of computing process (microscopic)

a) transmission protocol (HTTP->UDT)

Transmission mode is improved using UDT, instead of HTTP, can improve data transmission rate from map end to reduce end.

b) Optimization of system configuration parameters

Analysis and optimization of system parameters, including data compression, reducing memory usage of reduce task, map/reduce task quantity that are running on tasktrackers, increasing the quantity of copier threads in shuffle phase.

III. Algorithm Rebuilding and Performance Optimization of MapReduce

A. Sliding Window Scheduling Algorithm (SWSA)

1) Common scheduling algorithm [7][8]

a) FIFO: scheduling algorithm of traditional hadoop

b) Fair Scheduling Algorithm: Facebook proposes this scheduling algorithm.

- Merits: fair
- Shortcoming (small job & big job): the longer running time of every task; the lower cluster throughput; (In a word, inefficiency). For small job: data localization will be weakened [9].

2) Define data structure of SWSA

```
class JobScheduling
 {
  int JobID;
  int JobCapacity; //Size of Job
  int priority;
  int TaskNum; //The number of task
  float AllowedTaskNum; // AllowedTaskNum decides the number of tasks executing in one job in one cycle
  boolean RunOrNot; //Whether the job is executed.
  int CurrentTaskLeft;
 }
```

3) Implementation process of SWSA

a) Request tasks

Tasktrackers check nodes conditions. If they find free nodes, they send request to jobtracker for allocating tasks.

b) Judge slow node or quick node

Jobtracker judges that this node is slow node or quick node. If this free node is quick node and Backup Taskqueue is not empty and the number of Backup Task is below upper limits, this Backup Task is allocated to this free node and return c) Task allocation strategy in one rotary cycle. If this free node is a slow node, return f) The "straggler" is executed.

c) Task allocation strategy in one rotary cycle

Jobtracker will get the first task of TaskQueue in sliding window, and judge map task or reduce task. If this map task is a small job (task), return d) Data localization strategy based on small job. Then parameters of TaskQueue and JobList are modified. If the value of CurrentTaskLeft is smaller than one, the CurrentJobPointer will add one.

d) Data localization strategy based on small job

if there is a smaller job,

if there is a local task running in this node, then allocate it;

if J.wait < T1, then continue to wait;

if T1 < J.wait < T2,

else if there is a task in this rack, allocate it;

else continue to wait;

if J.wait > T2, then allocate it in any node that could provides enough resources.

else allocate a task to BackupTaskQueue of this node.

T1—— the time of waiting a free local node

T2 —— the time of waiting a free rack node

T1<T2 (the values of T1 and T2 need to be calculated in massive experiments)

e) Adjust the quantity of sliding window

If the pointer is pointed to the end of taskqueue of sliding window, Jobtracker will count the average workload of all tasktrackers.

AvgWorkLoad < LWorkLoad increase the number of sliding window

AvgWorkLoad > HWorkLoad decrease the number of sliding window

f) The "straggler" is executed

JobTracker calculates expected end time of all tasks. And in below upper limits of backup task case, the "straggler" of backup task is executed.

g) Update information

Updating information of job, and the pointer is moved to the front of a new TaskQueue and scheduling algorithm is executed again.

4) Parameter calculation

a) AllowedTaskNum

AllowedTaskNum value can determine the number of tasks executed by a job in a cycle.

Average task size: AvgTaskCapacity = JobCapacity / TaskNum

In a cycle, the quantity of data dealt with by Job[i] is Q[i]: q[i] = AvgTaskCapacity[i] * AllowedTaskNum[i]

So in a cycle, the total quantity of data dealt with is Q: Q =Σ AvgTaskCapacity[i] * AllowedTaskNum[i] (i=1...n)

Because priority [i] of Job[i] is proportional to q[i] of Job[i]

So AllowedTaskNum[i] * AvgTaskCapacity[i] / Q = priority[i] / Σ priority[i] (i=1...n)

Usually, the number of Map Task is far much than Reduce Task's. We can use the ability of dealing with data of Map Task instead of Task (TaskAbility). And data block in HDFS is called "BlockCapacity".

So Q = Max (TaskAbility * BlockCapacity, Num* blockSize * JobNum), (Num is the average quantity of tasks executed in a job in a cycle)

In the end, AllowedTaskNum[i] = Q * priority[i] / (AvgTaskCapacity[i] * Σ priority[j]) (j = 1...n)

b) LWorkLoad & AvgWorkLoad & HWorkLoad

If the number of sliding window is increased, (JobNum + Increment) * Num / TastAbility <= HWorkLoad;

Increment = (HWorkLoad - AvgWorkLoad) * TaskAbility / Num.

Simply, if the number of sliding window is decreased, Decrement = (AvgWorkLoad - LWorkLoad) * TastAbility / Num

c) threshold of progress value(judge "straggler")

This new method is based on Recent End Time (RET).

Principle of speculative execution: executing the backup task that is expected to preform for the longest time.

Because ProgressValue that is used for judging speed degree among tasks has blindness.

So I use "ProgressQuotiety" for meaning average speed. And ProgressQuotiety = ProgressValue / T

In the end, RET = (1 - ProgressValue) / ProgressQuotiety

d) ThresholdOfSlowNode(judge which tasktracker is slow node or quick node)

This method is to set slow node threshold. If the speed of node is lower than its Threshold, this node is a slow node, otherwise it is a quick node.

Threshold = ΣΣ ProgressValue[i](Task[j]) * 30% / n; (i=1...n,and n is the number of nodes ; j = 1...m and m is the number of tasks executed in every node)

B. UDT (change transmission protocol)

1) Introduction of UDT

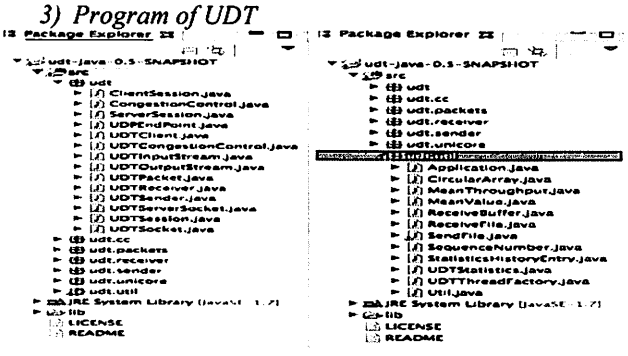
UDT (UDP-based Data Transfer Protocol) is a kind of Internet data transfer protocol. UDT is built on the UDP, and introduces new congestion control, and data reliability control mechanisms. It is a two-way connection-oriented application layer protocol and also supports reliable data streaming and partial reliable datagram transmission. As UDT is implemented completely on the UDP, it can also be used in other application areas addition to the high-speed data transmission.

2) Reason of changing from HTTP to UDT

HTTP (HyperText Transfer Protocol) is a TCP-based and most widely used network protocol on the Internet, and belongs to request/response model.

HTTP completes connection through three segments, this process is called three-way handshake (three-way handshake). And for every http request, a connection needs to be established, which greatly reduces data transfer speed. However UDT with these characteristics could compensate for the lack of HTTP and is more suitable for transferring intensive large data sets [10][11].

3) Program of UDT



C. Optimization of system parameters

Hadoop has nearly 200 system configuration parameters, and parameter optimization can effectively improve speed and efficiency [11].

1) data compression

Data compression for outputs of map task

The method to start this function:

mapred.compress.map.output: true

compress mode:

mapred.map.output.compress.codec: Gzip/lzo

According to test of the actual cluster, so we can draw the following conclusions:

- compression and decompression performance of LZO file is much better than Gzip files.
- For the same text file, Gzip compression reduces significantly disk space ratio better than LZO compression.

Therefore, we need to use the right compression algorithm under the appropriate environment. Usually, Gzip algorithm is typically used to compress file, because the purpose is to reduce the quantity of the file transmission and bandwidth costs.

2) reduce memory usage of reduce task

To reduce the needs of memory for reduce task and set more memory space for storing outputs of map task.

Mapred.inmem.merge.threshold

Mapred.job.reduce.input.buffer.percent

The default of this threshold is 0.8, and this threshold could be adjusted from 0.8 to 0.95. And when the data in the buffer reaches this threshold, the background thread will sort this data by key in the buffer, then write them to disk.

3) adjust map/reduce task quantity

{map/reduce}.tasks.maximum:

The maximum number of map / reduce task running on the tasktracker, and generally value is (CoreNum_Per_Node) / 2 ~ 2 * (CoreNum_Per_Node)

4) increase the quantity of copier threads in shuffle phase

The purpose is to improve the speed of transferring big data in the shuffle phase.

Mapred.reduce.parallel:

The number of copier threads is 5 (the default value). For the larger cluster, this value can be adjusted to 10 ~ 25.

IV. EXPERIMENTS AND RESULTS

A. Analysis of performance indicators

1) Task response time

2) Fairness

3) Accelerate Ratio

4) Fault Tolerance

B. Experiments

1) Test environment

a) Set up environment

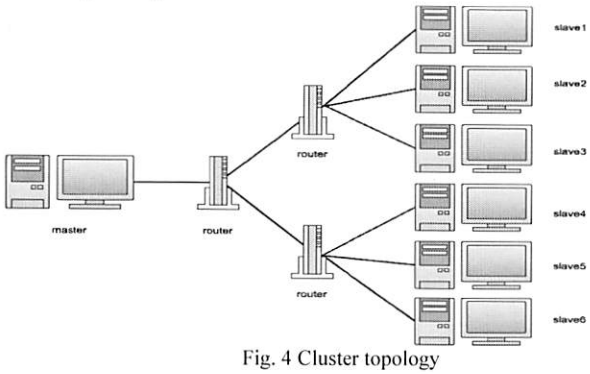


Fig. 4 Cluster topology

b) Cluster information

ID	Function	IP	Host Name	Hardware and Software Configuration			
				CPU	MEMORY	DISK	OS
1	master	192.168.1.10	zhaoben	Inter Core2 CPU 6600@2.40GHZ*2	4G	155.5GB 7200rpm	Ubuntu 12.04LTS
2	slave1	192.168.1.11	hosei44	Inter Core2 CPU 6600@2.40GHZ*2	4G	155.5GB 7200rpm	Ubuntu 12.04LTS
3	slave2	192.168.1.12	hosei11	Inter Core2 CPU 6600@2.40GHZ*2	4G	155.5GB 7200rpm	Ubuntu 12.04LTS
4	slave3	192.168.1.13	hosei55	Inter Core2 CPU 6600@2.40GHZ*2	4G	155.5GB 7200rpm	Ubuntu 12.04LTS
5	slave4	192.168.1.14	hosei22	Inter Core2 CPU 6600@2.40GHZ*2	4G	155.5GB 7200rpm	Ubuntu 12.04LTS
6	slave5	192.168.1.15	hosei33	Inter Core2 CPU 6600@2.40GHZ*2	4G	155.5GB 7200rpm	Ubuntu 12.04LTS
7	slave6	192.168.1.16	lane-ubuntu	Inter Core2 CPU 6600@2.40GHZ*2	4G	155.5GB 7200rpm	Ubuntu 12.04LTS
8	slave5	192.168.1.15	hosei66	Inter Core i7 2.9GHZ*2	8G	750GB 7200rpm	Ubuntu 12.04LTS
9	slave6	192.168.1.16	hosei77	Inter Core i7 2.9GHZ*2	8G	750GB 7200rpm	Ubuntu 12.04LTS

Fig. 5 Cluster configuration information

c) Used software information

Hadoop: Hadoop 1.1.2

Eclipse: eclipse-jee-indigo-SR2-linux-gtk-x86_64.tar.gz

JDK: JDK 1.7.0_25

2) Test methods and Test results

a) Sliding Window Scheduling Algorithm (SWSA)

- One Job: (Data size is 5.16GB)

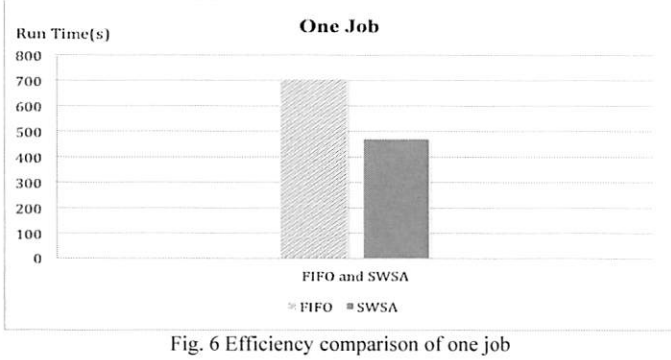


Fig. 6 Efficiency comparison of one job

Efficiency promotion: (707-472)/707=33.2%

- Multi jobs (the same priority)

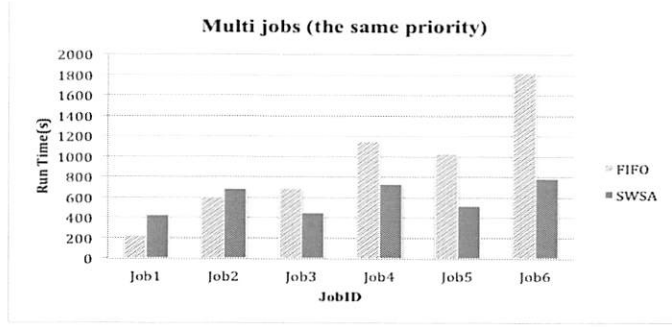


Fig. 7 Efficiency comparison of multi jobs with the same priority

The total run time of using FIFO: 5537s

The total run time of using SWSA: 3606s

Efficiency promotion: (5537-3606)/5537=34.9%

- Multi jobs (different priority)

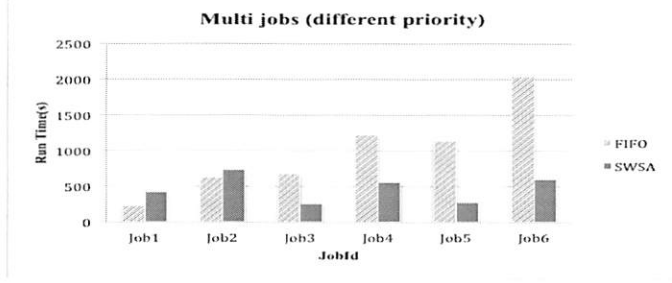


Fig. 8 Efficiency comparison of multi jobs with different priority

The total run time of using FIFO: 5955s

The total run time of using SWSA: 2878s

Efficiency promotion: (5955-2878)/5955=51.7%

b) UDT (change transmission protocol)

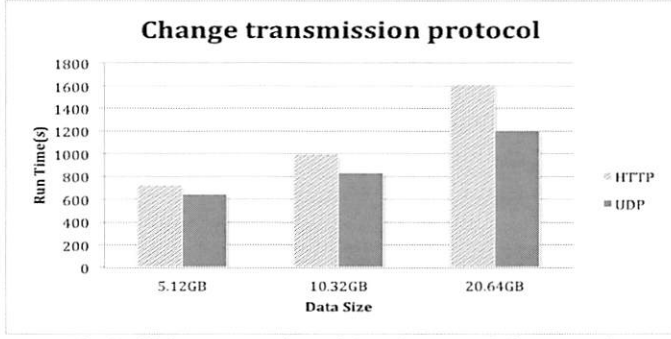


Fig. 9 Efficiency comparison of changing transmission protocol

Efficiency promotion (5.12GB): (725-647)/725=10.8%

Efficiency promotion (10.32GB): (992-831)/992=16.2%

Efficiency promotion (20.64GB):(1611-1204)/1611=25.3%

c) Optimization of system parameters

Set value of system parameters:

- Data compression

Mapred.compress.map.output: true

compress mode:

mapred.map.output.compress.codec: Gzip

- Reduce memory usage of reduce task

Mapred.inmem.merge.threshold

Mapred.job.reduce.input.buffer.percent: 0.95

- Adjust map/reduce task quantity

The number of map in cluster is 18.

The number of reduce in cluster is 2.

- Increase the quantity of copier threads in shuffle phase

Mapred.reduce.parallel: 10

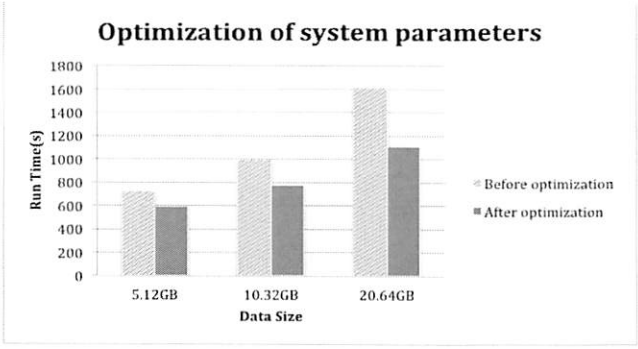


Fig. 10 Efficiency comparison of operating system parameters with different data size

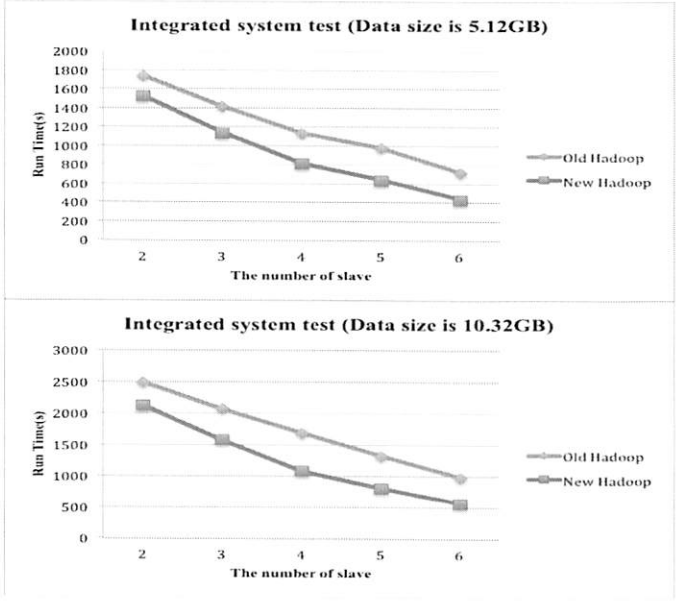
Efficiency promotion (5.12GB): (725-593)/725=18.2%

Efficiency promotion (10.32GB): (992-774)/992=23%

Efficiency promotion (20.64GB):(1611-1114)/1611=30.9%

d) Integrated system test

The number of slave node is from 2 to 6; Data size includes 5.12GB, 10.32GB and 20.64GB.



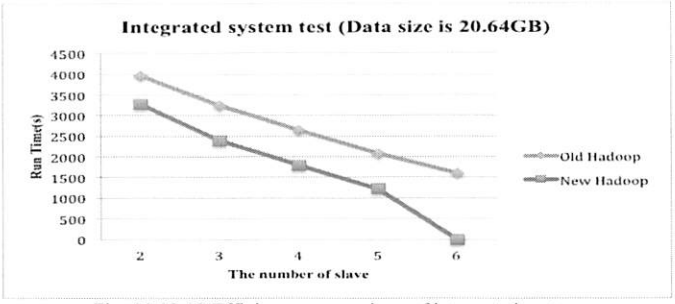


Fig. 11 12 13 Efficiency comparison of integrated system with different the quantity of slave node and different data size

The degree of performance improvement will increase, when the amount of data and the number of computer in the cluster increase.

V. CONCLUSIONS

In this paper, rebuilding scheduling algorithm (SWSA) improve task respond speed, load balance and fairness in the scheduling process of MapReduce. Meanwhile, it proves that new scheduling algorithm (SWSA) has scalability in the cluster. By replacing the transfer protocol and optimizing system configuration parameters, they improve further performance of MapReduce. In short, through this research, the study of this paper could prove that it is helpful to improve the performance of mapreduce.

ACKNOWLEDGMENT

The author would like to thank Professor Koike for his patiently guide and advice to this research. Also the author would like to thank CIS of Hosei University for supporting.

REFERENCES

[1] G. Yang, "The Application of MapReduce in the Cloud Computing," in 2011 2nd International Symposium on Intelligence Information Processing and Trusted Computing (IPTC), 2011, pp. 154–156.

[2] "Cloud computing," Baidu Baike. 14-Jul-2013.

[3] "Cloud computing," Wikipedia, the free encyclopedia. 14-Jul-2013.

[4] White T, Hadoop Definitive Guide, Beijing: Tsinghua University, 2010.

[5] A. Oriani and I. C. Garcia, "From Backup to Hot Standby: High Availability for HDFS," in 2012 IEEE 31st Symposium on Reliable Distributed Systems (SRDS), 2012, pp. 131–140.

[6] "HDFS," Baidu Baike. 14-Jul-2013.

[7] S. Selvarani and G. S. Sadhasivam, "Improved cost-based algorithm for task scheduling in cloud computing," in 2010 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), 2010, pp. 1–5.

[8] Zhang Mimi, "Performance analysis and improvement optimization of MapReduce model in Hadoop," University of Electronic Science and Technology of China, 2010.

[9] Q. Cao, Z.-B. Wei, and W.-M. Gong, "An Optimized Algorithm for Task Scheduling Based on Activity Based Costing in Cloud Computing," in 3rd International Conference on Bioinformatics and Biomedical Engineering , 2009. ICBBE 2009, 2009, pp. 1–3.

[10] R. K. L. Ko, M. Kirchberg, B.-S. Lee, and E. Chew, "Overcoming Large Data Transfer Bottlenecks in RESTful Service Orchestrations," in 2012 IEEE 19th International Conference on Web Services (ICWS), 2012, pp. 654–656.

[11] Peng Fuquan, Jin Canghong, Wu Minghui and Ying Jing,"Optimization and reconstruction shuffle in MapReduce", China SciencePaper, Vol7,No.4,April,2012.