情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2011-DBS-152 No.6
Vol.2011-IFAT-103 No.6
2011/8/2

An Evaluation on Power Consumption and Performance Balancing Distributed Storage Systems

Hieu Hanh LE[†1] Satoshi HIKIDA[†1] Haruo YOKOTA[†1]

In distributed storage system, replicating data on multiple nodes for improving performance also extends power consumption of whole system. As the result, it becomes more important to control both power consumption and performance in storage system. Recently, a number of proposals that allows the system to be available to control power consumption have been proposed and confirmed to be success in certain circumstance and applications. In these proposals, system is made available to operate in multiple modes with containing different number of active datanodes that store data. Consequently, at each mode, the system obtains performance which is proportional with the number of active datanodes it contains. However, this approach faces a problem of performance degradation when the system needs to correct its data layout while moving up to high mode from power-saving modes if the dataset is updated during power-saving mode. In this paper, the function of data layout correction is added into two existing data placement methods in distributed storage system and the degradation in performance of these methods is then evaluated through empirical experiments.

†1 Department of Computer Science, Graduate School of Information Engineering and Science, Tokyo Institute of Technology

1. Introduction

Nowadays, balancing performance and power consumption of distributed storage in datacenters has gained a lot of interested with the expansion of data intensive services in cloud computing. Usually, in such kind of services, the provider has to store a huge amount of user's data, and responds to user's requests with high availability and performance. In order to guarantee such kind of characteristics, one of popular techniques is replicating data in different locations inside storing system. Data replication can not only prevent failure of each nodes in storing system, but also improve request's latency through parallel processing. Here, a node is defined as common computer machine that has at least processor, memory and hard disk. However, the increasing in the number of necessary nodes in order to apply data replication also expanding the power consumption of whole data storage.

Balancing both power consumption and performance can be achieved through data placement methods by controlling the total number of active nodes in the system 2),4), 5). These proposals are common in the way of managing energy in a group of nodes instead of controlling a single node individually. Specifically, at first the system divide all their storage nodes into certain small separated groups and then is organized to be able to operate in multiple gears that each gear containing a number of groups whose datanodes are active. The higher the gear is, the higher the number of activate nodes. At each gear, which contains different number of nodes, system is ensured to store all necessary data for guaranteeing the availability of systems. By controlling power management in the granularity of gear, those approaches are considered to be success in control the power consumption of whole system.

Although a number of proposals, introduced like above, are evaluated to be success at certain circumstances, however they have been still not compared with each other in the same environment yet. In 3), the authors reported the first comparison evaluation of two representative proposals, i.e. PARAID 2) and RABBIT 4) through empirical experiment on actual machines to compare their read performance while concerning system's power consumption. From this result, it was confirmed that both methods are good in balancing read performance characteristic and storage's power consumption.

Vol.2011-DBS-152 No.6
Vol.2011-IFAT-103 No.6
2011/8/2

It is also concluded that RABBIT slightly overcome PARAID because of better data distribution over datanodes storing data in system.

However, the above paper just dealed with the relationship between system's power consumption and performance over fixed dataset. If the dataset was modified when system contains a number of deactivate datanodes, then the original data layout policy is violated. As the result, system has to catch up with the newest status of the dataset through correction data layout when system moves to higher gear. In this gear, all deactivate nodes at previous lower gear are reactivated. In this case, system needs to deal with both correcting the data layout according to its fixed data placement policy and guaranteeing the availability of its service to users. The data layout correction process which depends on the amount of new data written newly to system is thought to have important effect to system's performance in such scenario.

The contribution of this paper is as follows.

First, we decide to develop new writing modules and data layout correction. The idea here is inspired by Write Offloading technique proposing in 7) that is considered as standard technique in distributed environment allowing some datanodes to be turned down to save power consumption. This technique's idea is summarized as firstly, to write new data to any available datanodes; secondly taking logs that describe this event; and finally correct the data layout later based on the log files.

Next, an evaluation of above modules is achieved through empirical experiment with PARAID and RABBIT, following the work in 3). The experiment is performed by using a benchmark that requires distributed Input/Output over large dataset and the performance degradation when system makes a change of its operation mode from low gear to high gear is reported. From the experiment result, because of better data placement which requires smaller amount of data needed to reallocate, PARAID was found to achieved faster execution time than RABBIT.

The paper is organized as follow. Section 2 gives a description of data placement methods of PARAID and RABBIT, a brief review of other approaches in balancing performance and power consumption in distributed storage systems. New functions of write data at low mode and data reallocation process when system upshifts to higher mode to serve a service which are newly implemented in PARAID and RABBIT will be explained in Section 3. Section 4 is planned to describe and report the experiment results on actual machines. The conclusion and future works will be summarized in Section 5.

2. Related Work

In this section, because PARAID and RABBIT's data placement methods were used in experiments, these methods are described. And then, some other proposals in this area are also introduced briefly.

2.1 PARAID and RABBIT Data Placement

Although not like RABBIT, PARAID was originally designed inside a RAID unit, the idea can be expanded to distributed environment that contains a large number of nodes connected through network. In this context, a node is defined as an normal computer machine including processor, memory and hard disks. In this part, we describe the modified skewed data placement in PARAID in order to apply in distributed environment.

Both of these two proposals are based from the idea of dividing the total number of nodes in the system into small separated groups. Consequently, the system then contains a certain number of gears that include number of groups. A low gear with small number of powered nodes is supposed to consume low power and vice versa.

Given a dataset D with total B blocks, a total number of nodes N are divided into G groups. Each group contains a different number of nodes. In detail, each node is symbolized as $n_{(g,i)}$, where g $(1 \leq g \leq G)$, i $(1 \leq i \leq N)$ indicate i-th of node at g-th group. For example, nodes $n_{(1,1)}$, $n_{(1,2)}$ belong to Group 1, while nodes $n_{(2,3)}$, $n_{(2,4)}$ belong to Group 2 and so on.

2.1.1 PARAID

PARAID is the first work to introduce the concept of gear-shifting based on load of system within a RAID unit. It utilized the idea of skewed striping pattern to adapt to the system load by varying the number of powered nodes.

PARAID takes advantage of replicating and striping data blocks in a skewed fashion to nodes. Then, nodes could be organized into a number of groups and a number of groups forms gear.

Vol.2011-DBS-152 No.6
Vol.2011-IFAT-103 No.6
2011/8/2

表 1 PARAID data placement

	Group 1		Group 2		Group 3	
Node	$n_{(1,1)}$	$n_{(1,2)}$	$n_{(2,3)}$	$n_{(2,4)}$	$n_{(3,5)}$	$n_{(3,6)}$
Gear 3	B_1	B_2	B_3	B_4	B_5	B_6
Gear 2	B_1	B_2	B_3	B_4	-	-
	$B_5(\frac{1}{4})$	$B_5(\frac{1}{4})$	$B_5(\frac{1}{4})$	$B_5(\frac{1}{4})$	-	-
	$B_6(\frac{1}{4})$	$B_6(\frac{1}{4})$	$B_6(\frac{1}{4})$	$B_6(\frac{1}{4})$	-	-
Gear 1	B_1	B_2	-	-	-	-
Gear 1	$B_3(\frac{1}{2})$	$B_3(\frac{1}{2})$	-	-	-	-
	$B_3(\frac{1}{2})$	$B_3(\frac{1}{2})$	-	-	-	-
	$B_3(\frac{1}{2})$	$B_3(\frac{1}{2})$	-	-	-	-
	$B_3(\frac{1}{2})$	$B_3(\frac{1}{2})$	-	-	-	-

In PARAID, at first, all data D of B blocks are allocated evenly to all nodes. Next, at lower gear that allows a number of nodes to be power off, in order to guarantee the availability, the data from nodes which are going to be deactivate will be evenly migrated to nodes that are active in next higher gear.

Table 1 indicates a simple example of number of data at each node corresponding to system's operation gear through a storage cluster with six nodes, three groups and three gears. Gear 1 contains only active node in Group 1, Gear 2 needs all four nodes in both Group 1 and Group 2 be active and the highest gear, Gear 3, all six nodes are set to be activated. When the system move from Gear 3 to Gear 2 to save power, because all the data in Group 3's nodes are already replicated to active nodes in Group 1 and Group 2, the system is able to provide all the data in the dataset to clients using it.

Through the above techniques, PARAID is considered to be able to control the power consumption and performance of storage system.

2.1.2 RABBIT

RABBIT is a power-proportional distributed file system (PPDFS) that uses a cluster-based storage data placement to control power and performance of system.

Assuming that r replicas of B blocks of dataset D are desired to be stored to n nodes with G groups and G gears. At first, one replica of all B blocks are evenly stored in first primary p nodes at Group 1 (also called primary group). Consequently, each node in Group 1 contains $\frac{B}{p}$ blocks. The remaining $(r-1)$ replicas are distributed to $(N-p)$ nodes in the way that the node $n_{(g,i)}$, where $g > 2$ and $p < i \leq N$, stores $\frac{B}{i}$ blocks.

表 2 RABBIT data placement

	Group 1 (Primary)		Group 2 (Secondary)		Group 3 (Secondary)		
Node	$n_{(1,1)}$	$n_{(1,2)}$	$n_{(2,3)}$	$n_{(2,4)}$	$n_{(3,5)}$	$n_{(3,6)}$	$n_{(3,7)}$
Gear 3	$B(\frac{1}{2})$	$B(\frac{1}{2})$	$B(\frac{1}{3})$	$B(\frac{1}{4})$	$B(\frac{1}{5})$	$B(\frac{1}{6})$	$B(\frac{1}{7})$
Gear 2	$B(\frac{1}{2})$	$B(\frac{1}{2})$	$B(\frac{1}{3})$	$B(\frac{1}{4})$	-	-	-
Gear 1	$B(\frac{1}{2})$	$B(\frac{1}{2})$	-	-	-	-	-

Here, in the constraint of keeping number of replica r small with fixed number of nodes RABBIT can guarantee that the number of blocks stored by i-th node must not be less than $\frac{B}{N}$ for all $i \leq N$ when N nodes are active. Obeying this constraint makes it possible for the load to be shared equally among active nodes. Thus, the performance of the system is suggested to be linear with the number of powered nodes, i.e. implicitly corresponds to the power consumption of the system.

Table 2 shows an example of data placement and corresponding number of data at each node in RABBIT for a 7-node cluster with two primary nodes and 2-replica. It is well recognized that 5 nodes in Group 2 and Group 3 store are enough for storing the necessary data of second replication. Like PARAID, the system in RABBIT can operate in multiple gears. For example, as shown in Tab. 2, Gear 1 only includes the first two nodes belonging to Group 1, Gear 2 includes active disks in Group 2 joining with Group 1, and Group 3 activates all nodes inside the system.

2.2 Others

Like PARAID, GRAID 6) is a green storage architecture aiming for improving energy efficiency and reliability within RAID unit. In this proposal, the data mirroring redundancy of RAID10 is extended by incorporating a dedicated log disk. The function of this log disk is to store all updates since last mirror-disk update. Using these information for log disk, the system only needs to update the mirroring disks only periodically, thus being able to spin down all the mirroring disks to lower power mode most of the time to save energy.

Designed as a power-proportional, distributed file system as RABBIT, SIERRA 7) is a replicated object store that allows storage servers to be put into low power states when load is low. This allows servers hosting inactive replicas to be powered down. For example, in three-way replicated system like in normal Hadoop Distributed File System

Vol.2011-DBS-152 No.6
Vol.2011-IFAT-103 No.6
2011/8/2

(HDFS) 1) or Google File System, SIERRA allows up to $\frac{2}{3}$ of the storage servers to be in standby. Comparing with RABBIT, SIERRA's data placement is simpler as all storage servers keep the same amount of data.

Kim et. al 5) suggest a fractional replication method in order to achieve balancing power consumption and performance of system. In this proposal, the data placement layout is inspired by PARAID as performing fractional replication and the decision of down shift operation modes to lower gear for saving power consumption by a probabilistic prediction model based on historical observations. However, this work still does not cover with power degradation problem at versus process, a movement of storage system's operation mode from low gear to high gear.

In 3), the authors reported an evaluation comparison between PARAID and RABBIT under similar distributed environment through empirical experiment. The idea of PARAID is extended from RAID unit to HDFS, a popular and open source distributed file system. From this paper, it is said that RABBIT, because of better data distribution gained better performance in read feature. However, the write modules and also the data layout correction when system restores from low gear to high gear are still not covered yet.

3. Data Catchup Process

In this section, focusing on the performance degradation occur when the system moves up from low gear to high gear with the existing of write request to dataset, the process catchup process from the time system accepts new write request at low gear, to the time it finishes a service requiring process over new dataset is describe. This paper follows the work done in 3) so the idea of data placement of PARAID is extended to be able to implement in HDFS, a distributed file system.

3.1 Overview of Catchup Process

Figure 1 describes an overview of catchup process. At first, considering that system operates in low gear with certain storing dataset. Then, it accepts requests that require writing new data to this dataset. Because in this gear, some of datanodes are in deactivate mode, in order to correspond to this write request, system is needed to implement write offloading technique in this step. Otherwise, new data could not be written completely in case of PARAID, or replicated fully in case of RABBIT.

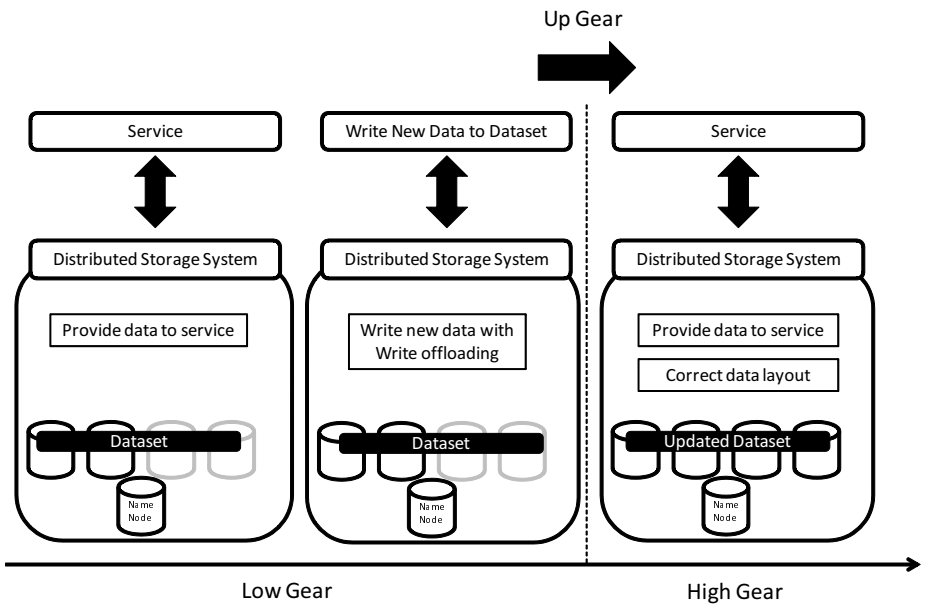

図 1 Overview of catchup process

Next, when a service requiring new dataset to be stored and desiring higher performance, system is changed to high gear. In this step, not only providing data to this service, but also data layout correction is needed inside storage system.

The detail operation of above two processes is described in the following parts of this section.

3.2 Write New Data Using Write Offloading Technique

When system in low gear and has to deal with requests of writing new data to dataset which was already stored, it performs its data layout policy as defined according to its policy. Because system is operating in low gear, some of datanodes are powered off. As the results, certain parts of new data cannot be written to corresponding deactivated datanodes. In such cases, system randomly chooses another node from active nodes to

Vol.2011-DBS-152 No.6
Vol.2011-IFAT-103 No.6
2011/8/2

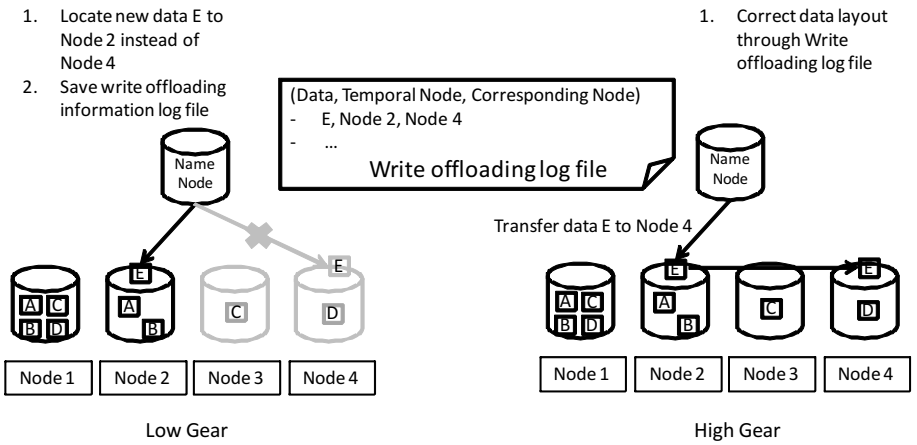

図 2 Data layout correction from write offloading

serve this request. The information of data, temporally chosen node and corresponding node is saved in a log file. This in formation will be retrieved when system decides to change its operation mode from low gear to high gear.

The example of write offloading process is shown in Fig. 2. In this example, according to data placement policy, Node 4 should serve write request of data E. However, during low gear, it is powered off, so the system decide to opt another node alternatively, here Node 2 is chosen. The system saves all of this information into a log file.

3.3 Data Layout Correction

When system changed to high gear to serve a service that requires processing over new updated dataset with higher performance desires, it needs to perform two functions. The first function in to transfer the data written temporally at temporal datanodes to corresponding nodes. It could be achieved by reading the information from log files. The second function is to serve the service which scans all new dataset storing in storage system. It is well noticed that, comparing with normal serving at high gear when there is no need of correcting data layout, the cost of allowing system to operate in multiple mode for power saving highly depends on the amount of this catchup data.

4. Experiments

The purpose of the experiment is to evaluate the effect of data catchup resulting to power degradation when system changes its operation from low gear to high gear with the occurrence of dataset updating.

4.1 Experiment Method

The functions of write offloading explained in Sec. 3 were added into HDFS source code. And the evaluation is achieved by using *grep* benchmarks accompanied with HDFS. This benchmark is a very typical application that utilizes Map-Reduce framework to find out from whole dataset elements that matches certain pattern. In this frame work, the dataset is split into multiple parts for multiple workers to perform an action of seeking for appropriate results. Worker needs to read all its responsible dataset so, when multiple workers perform their job at the same time, it could be implied that there exits certain distributed I/O to the storage system. As the result, the performance of system can be evaluated through the execution time of this benchmark.

At first, guaranteeing original data placement policy, system is operated at high gear and already stores certain dataset. Then we set the system to low gear and perform new requests that appending new data to old dataset. Next, we set the system's operation mode from low gear to high gear and run the benchmark. In this experiment, we designed two gears that Gear 1 contains only first three nodes to be active and Gear2 contains all seven nodes need to be powered on.

The effect of data catch up process leading to performance degradation of each method (PARAID, RABBIT) is evaluated through the execution time of *grep* benchmark over updated dataset when upshift system from low mode to high mode. The experiments were performed multiple times and memory cache was flushed between each time.

4.2 Experiments Environment

There are two elements in our framework.

First, in storage, we use a number of nodes which play a role of datanode as in HDFS. Each datanodes is an autonomous disk which contains processor, memory, hard disk and is designed for low power consumption.

Next, in order to implement data placement and to manage information about data

Vol.2011-DBS-152 No.6
Vol.2011-IFAT-103 No.6
2011/8/2

表 3 Nodes specification

	Namenode	Datanode
CPU	Intel Pentium 4 3.00GHz	Transmeta Efficeon TM8600 1.0GHz
Memory	SDRAM 2048MB	DRAM 512MB
HDD	SATA 320GB	IDE 100GB
Network Interface Card	1000Mb/s	100Mb/s
OS	Linux 2.6.18	Linux 2.6.18
Java	JDK-1.6.0	JDK-1.6.0
HDFS	0.20.2	0.20.2
Block Size	64MB	64MB

表 4 Dataset specification

Name	Value
File size	200MB
Old dataset	1000MB (5 files)
New data	200MB (1 files)

location such as which datanode is containing what data, a namenode is used. At this namenode, the source codes relating to data placement of normal HDFS are touched to make it available to implement the layout policy of RABBIT and PARAID. Furthermore, the function of implementing write offloading, setting operation mode of system through command line was added into namenode while datanode's functions was kept untouched.

The interconnect between datanodes and namenode is Extreme Network Summit 16101 Gigabit Ethernet switch.

The specification of namenode and datanodes are summarized in Tab. 3.

Here, the dataset is a text file contains a number of fix 4-word length phrases, and the pattern used in *grep* are generated randomly using Java-1.6.0. The size of storing dataset was fixed to 1GB while the size of new written dataset was fixed to 200MB. The block size used in HDFS was 64MB and the number of replicas of data using in RABBIT was set to 2.

4.3 Experiments Results

Figure 3 shows the experiment result with *grep* benchmark. The results were averaged from two times running. Normal shows the execution time of 1.2GB dataset when no catchup occurred. Catchup indicates the result starts from the point system change

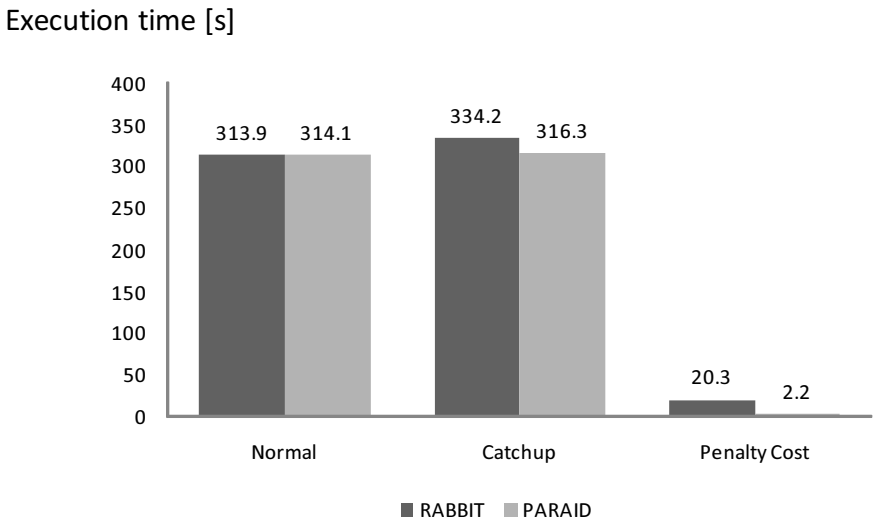

図 3 Experiment result with *grep* benchmark

to higher gear node to perform benchmark over updated dataset. Penalty cost describes the difference between Normal and Catchup results.

From this result, it is seen that in normal, both two nodes obtained similar results. However, the penalty cost of RABBIT well overcome the one of PARAID by 10 times, 20 seconds comparing with 2 seconds. It is well reasoned that in this case, because the amount of data needed to transfer inside storage system was much larger in RABBIT than in PARAID. There were only 8 MB needed to be write offloading in PARAID, while the corresponding number in RABBIT was 196 MB. It is explained by the fact that in this experiment, the number of replicas in RABBIT was set to 2, so it needs much more data than PARAID to be stored in the system. However, with no replicated data, PARAID is suggested not to be tolerant with the occurrence of failure in datanode.

Vol.2011-DBS-152 No.6
Vol.2011-IFAT-103 No.6
2011/8/2

5. Conclusion

Balancing performance and power consumption has become an important issue in distributed storage system area. For serving certain service over large amount of data storing in storage system, in order to achieve high performance, higher number of storage element is needed. However, it also makes the power consumption of whole system increased. Recently, a number of methods have been proposed to control the power consumption and system's performance. It makes the system can operate in multiple modes, however the power degradation occurred when system needs to catch up with new data while changing its operation mode from low gear to high gear is still not considered yet enough.

In this paper, we decided to choose two representative methods, i.e PARAID and RABBIT in this field and compared them in the context of power degradation when the system has to catch up newest status of storing dataset and serves service requiring process over this dataset. Following the previous work, we developed this functions in PARAID and RABBIT over popular and open source Hadoop Distributed File System. In low gear, we applied the idea write offloading technique to write new data to temporal datanodes and when system changed to high gear, data layout correction is fulfilled.

An empirical experiment through comparing execution time of *grep* benchmark that utilizes Map Reduce framework to find appropriate elements that matches certain pattern was performed. In both methods, the execution time of this benchmark at time catchup new data is needed and not needed were abstracted. From this experiment results, PARAID gained better performance comparing with RABBIT with faster execution time. The penalty cost for PARAID is 10 times better than RABBIT. One of considerable reasons is that because of its data placement policy, PARAID needs less amount of data to be relocated in data layout correction phase so, the bottleneck of system at that time was far behind one in RABBIT.

In the future, we would like to examine with larger dataset and another benchmark. Furthermore, because the power consumption in this paper was implied as the number of active datanodes used in storage system, we also would like to obtain real numeric results through real deactivating nodes.

Acknowledgements

This work is partly supported by Grants-in-Aid for Scientific Research from Japan Science and Technology Agency (A) (#22240005).

参 考 文 献

1) : Hadoop, `http://hadoop.apache.org`.
2) Charles, W., Mathew, O., Jin, Q., Andy, W. A.-I., Peter, R. and Geoff, K.: .
3) Hieu Hanh, L., Satoshi, H. and Haruo, Y.: Performance Comparison of Power-Proportional Approaches in Storage Systems through Empirical Experiment, *in the Proceedings of the 3rd Data Engineering and Information Management Forum* (2011).
4) Hrishikesh, A., James, C., Varun, G., Gregory R., G., Michael A., K. and Karsten, S.: *in the Proceeding of the 1st ACM Symposium on Cloud Computing*, New York, NY, USA.
5) Kim, J. and Rotem, D.: *Proceedings of the 14th International Conference on Extending Database Technology*, New York, NY, USA.
6) Mao, B., Feng, D., Jiang, H., Wu, S., Chen, J. and Zeng, L.: *in the Proceeding of IEEE International Symposium on Modeling, Analysis and Simulation of Computers and Telecommunication Systems, 2008. MASCOTS 2008.*
7) Thereska, E., Donnelly, A. and Narayanan, D.: SIERRA: A Power-Proportional, Distributed Storage System (2009).