IEICE TRANSACTIONS

on Information and Systems

DOI:10.1587/transinf.2014EDP7242

Publicized:2015/01/13

This advance publication article will be replaced by the finalized version after proofreading.

A PUBLICATION OF THE INFORMATION AND SYSTEMS SOCIETY

The Institute of Electronics, Information and Communication Engineers

Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3 chome, Minato-ku, TOKYO, 105-0011 JAPAN

PAPER

A Study of Effective Replica Reconstruction Schemes for the Hadoop Distributed File System

Asami HIGAI[†], ***Nonmember***, **Atsuko TAKEFUSA**[††], ***Member***, **Hidemoto NAKADA**[††], ***Nonmember***, ***and*** **Masato OGUCHI**[†], ***Member***

SUMMARY Distributed file systems, which manage large amounts of data over multiple commercially available machines, have attracted attention as management and processing systems for Big Data applications. A distributed file system consists of multiple data nodes and provides reliability and availability by holding multiple replicas of data. Due to system failure or maintenance, a data node may be removed from the system, and the data blocks held by the removed data node are lost. If data blocks are missing, the access load of the other data nodes that hold the lost data blocks increases, and as a result, the performance of data processing over the distributed file system decreases. Therefore, replica reconstruction is an important issue to reallocate the missing data blocks to prevent such performance degradation. The Hadoop Distributed File System (HDFS) is a widely used distributed file system. In the HDFS replica reconstruction process, source and destination data nodes for replication are selected randomly. We find that this replica reconstruction scheme is inefficient because data transfer is biased. Therefore, we propose two more effective replica reconstruction schemes that aim to balance the workloads of replication processes. Our proposed replication scheduling strategy assumes that nodes are arranged in a ring, and data blocks are transferred based on this one-directional ring structure to minimize the difference in the amount of transfer data for each node. Based on this strategy, we propose two replica reconstruction schemes: an optimization scheme and a heuristic scheme. We have implemented the proposed schemes in HDFS and evaluate them on an actual HDFS cluster. We also conduct experiments on a large-scale environment by simulation. From the experiments in the actual environment, we confirm that the replica reconstruction throughputs of the proposed schemes show a 45% improvement compared to the HDFS default scheme. We also verify that the heuristic scheme is effective because it shows performance comparable to the optimization scheme. Furthermore, the experimental results on the large-scale simulation environment show that while the optimization scheme is unrealistic because a long time is required to find the optimal solution, the heuristic scheme is very efficient because it can be scalable, and that scheme improved replica reconstruction throughput by up to 25% compared to the default scheme.

key words: *HDFS; distributed file system; replica; reconstruction; heuristic; optimization;*

1. Introduction

Large amounts of data generated from high-quality sensor networks, social network services, and high performance scientific experimental tools such as genome sequencers require efficient ”Big Data” management and processing in various fields of commerce and scientific computing such as high-energy physics and life information sciences. Distributed file systems, which manage large amounts of data over multiple commercially available machines, are widely used for such Big Data processing. To achieve high scalability and availability, a distributed file system consists of multiple data nodes, each depending on different system requirements, and each data node manages blocks of the data and their individual replicas. However, it is difficult to operate all of these data nodes without any failure. Some data nodes may be unstable due to system failure or maintenance.

In a distributed file system, data are replicated, and the data, including the replicas, are divided into data blocks and separately stored for reliability and availability. When a data node failure is detected, the data blocks stored in the data node are lost, and the access load of the other data nodes, which hold the lost data blocks, increases. Thus, the performance of data processing over the distributed file system decreases. Therefore, an important issue is the development of effective replica reconstruction that reallocates the missing data blocks to other stable data nodes to prevent such performance degradation.

The Hadoop Distributed File System (HDFS) [1], which is a part of the Apache Hadoop [2] project, has been a widely used open source distributed file system. In the HDFS replica reconstruction process, source and destination data nodes used to send and receive a missing data block are chosen at random. As a result, sending and receiving processes concentrate on a certain data node. To perform replica reconstruction effectively, it is necessary to balance the process workloads of each data node by choosing suitable source and destination data nodes.

To address this issue, we propose effective replica reconstruction schemes that aim to balance workloads of copying processes between source and destination data nodes. Our proposed schemes apply to a basic strategy based on a one-directional ring structure: each data node receives data blocks from the previous data node and sends data blocks to the next data node. Based on this strategy, the proposed schemes, optimization and heuristic aim to minimize the difference in the amount of transfer data of each data node and select a source data node, which holds a missing data block. In the optimization scheme, we define this replica reconstruction problem as 0-1 integer programming and solve the problem with an optimization solver. In the heuristic scheme, we select the source data node in a heuristic manner.

We have implemented the two proposed schemes in HDFS and evaluate our proposed schemes in an actual

[†]Department of Computer Science, Ochanomizu University
[††]National Institute of Advanced Industrial Science and Technology(AIST)

HDFS cluster composed of seven nodes [4]. We also investigate the performance in a large-scale simulation environment by using SimGrid [3], a discrete event simulator. From the experiments in the actual cluster, we confirm that the replica reconstruction throughputs of the proposed schemes improve by 45% compared to the HDFS default scheme, and the load of each data node can be balanced by eliminating the bias of data transfer. We also verify that the performances of the heuristic and optimization schemes are comparable. From the experiments in the large-scale simulation environment, we find that the optimization scheme is unrealistic because it takes a long time to find the optimal solution. The heuristic scheme is efficient in the large-scale environment as well because it can be scalable and the replica reconstruction throughput improves by 25% at most compared to the default scheme.

This paper is organized as follows: Section 2 describes the HDFS replica reconstruction scheme and its problems. Section 3 explains our proposed replica reconstruction schemes: an optimization scheme and a heuristic scheme. Sections 4 and 5 evaluate our proposed schemes on an actual HDFS cluster and a large-scale simulation environment. Section 6 introduces related work. Finally, we state our conclusions in Section 7.

2. Replica Reconstruction for HDFS

2.1 Node Decommission and Deletion for HDFS

HDFS is a clone of the Google File System (GFS) [5] developed by Google. HDFS is based on a master and worker architecture and consists of a single NameNode and multiple DataNodes. The NameNode stores the metadata for files and manages all the nodes in the cluster, and the DataNodes store data and perform MapReduce-based data processing. Each file is divided into blocks, which are the minimum units, and the blocks are replicated. Their replicas are separately stored on the other DataNodes for reliability and availability.

When a DataNode is removed from a cluster, HDFS keeps the number of replicas specified in a replication factor by replicating missing blocks from the DataNode to the other remaining DataNodes. There are two ways of removing nodes from a cluster for HDFS: node decommission and node deletion.

Node decommission is the means by which a DataNode is removed from a cluster after replica reconstruction. The decommissioned node itself participates in the replica reconstruction process, the situation in which nodes are removed from a cluster intentionally, for example, to shrink a cluster scale and initiate the shutdown of an unstable DataNode that frequently causes errors.

Node deletion is the means by which a DataNode is removed from a cluster before replica reconstruction. The DataNode itself must not participate in the replica reconstruction process, the situation in which a DataNode is removed from a cluster unexpectedly, for example, due to node failure or trouble with a network connection.

Table 1 Default values of the parameters relating to the replication process

Name	Description	Default value
N_work	The number of instructions which the NameNode can transfer at one time	2
N_stream	The number of blocks each DataNode can transfer at one time	2

2.2 HDFS Replica Reconstruction

When node decommission or deletion is detected, HDFS performs replica reconstruction, which copies the data that the DataNode holds to the other DataNodes. The process proceeds by units of blocks. NameNode makes all decisions regarding this block replication scheduling. We shows the step of the replication reconstruction process as follows:

(1) NameNode detects node decommission or deletion.
(2) NameNode chooses source and destination DataNodes for replications of the missing blocks and periodically transfers replication instructions to each source DataNode.
(3) Based on the instructions, the source DataNode transfers the blocks to the specified destination DataNode.
(4) The destination DataNode sends an acknowledgement to the source DataNode after finishing the copying of the block, and then the source DataNode sends an acknowledgment to the NameNode.

Step (2)-(4) continues repeatedly until all the blocks that are missing are replicated.

The number of instructions that the NameNode transfers to the DataNodes equals the product of the number of active DataNodes in the cluster and the `REPLICATION_WORK_MULTIPLIER_PER_ITERATION` parameter. We call this parameter N_work in this paper. The NameNode cannot provide instructions totaling more than this value. This scheduling process and the data transmission process are performed in parallel. The number of data blocks that each DataNode can transfer to the destination DataNode without receiving an acknowledgement equals the `dfs.max-repl-stream` parameter. We call this parameter N_stream in this paper. N_work is a hard-coded parameter in FSNamesystem.java of the ReplicationMonitor package. N_stream is a property we can specify explicitly. The default values of these parameters are shown in Table 1.

2.3 HDFS Replica Reconstruction Issue

When all DataNodes belong to the same rack, NameNode randomly selects both a source DataNode from the DataNodes that hold a copy of the missing blocks and a destination DataNode that does not hold the block. This replica reconstruction scheme may cause a concentration of the data transfer process on a few DataNodes. To clear this concern, we investigated the disk I/O throughput of each DataNode and the number of blocks that each DataNode received in the

HDFS replica reconstruction process using an HDFS cluster that consists of a single NameNode and six DataNodes connected by a Gigabit Ethernet switch. We use node deletion as the method for removing a DataNode from the cluster. In this experiment, the block size and replication factor are 64 MB and three, respectively.

We acquired the disk I/O throughput every one second using the Linux `iostat` command from all the DataNodes and calculated the simple moving averages of five seconds. We then counted the number of blocks that each DataNode received in every second. This information is derived from the Hadoop-$user-datanode.log files in each DataNode. The aggregated disk I/O throughput of the five remaining DataNodes is shown in Figure 1. The vertical axis represents the disk I/O throughput in MB/sec and the horizontal axis represents the time in sec. The number of blocks that each DataNode received is shown in Figure 2. The vertical axis represents the number of blocks and the horizontal axis represents the time in sec. N_stream is two as described in 2.2; thus, it seems to be an ideal state in terms of load balancing, in which each DataNode is receiving two blocks during the experiment.

However, Figure 2 shows that the numbers of blocks each DataNode is receiving are quite different and unstable. In the time period from 80 to 100 seconds, the number of blocks received by DataNode5 increases, that is, the destination DataNodes to be replicated are concentrated. At this time, the aggregated disk I/O throughput of all DataNodes decreases, and the overall replication process is stagnating. These results show that replica reconstruction with the default scheme for HDFS is inefficient because of unbalanced sending and receiving processes.

The default random scheme in the original HDFS is expected not only for balancing data transfer during recovery, but also for increasing data availability, balancing disk space usage between nodes, and improving the data processing performance during the normal operations of read/writes. However, in practice, we verified that unnecessary collisions occurred and the performance decreased during replica reconstruction because of the biased data transfer. Therefore, data availability, data locality and disk utilization of the default random scheme are not good due to this biased data transfer. These problems are resolved with the proposed scheme explained in the following section.

3. Proposal for an Effective Replica Reconstruction Scheme

To solve the replica reconstruction issue with the HDFS default scheme described in the previous section, it is necessary to balance the process workloads of each DataNode by choosing source and destination DataNodes properly. Therefore, we propose a scheduling strategy for replica reconstruction that aims to achieve efficient processing by choosing source and destination DataNodes based on a one-directional ring structure and balancing the workload. We propose two schemes: an optimization scheme and a heuristic scheme. In the optimization scheme, we define this replica reconstruction problem as 0-1 integer programming and solve the problem with an optimization solver. In the heuristic scheme, we select the source data node in a heuristic manner.

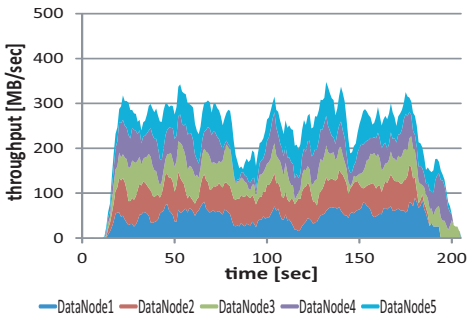

Fig. 1 Aggregated disk I/O throughput of five DataNodes.

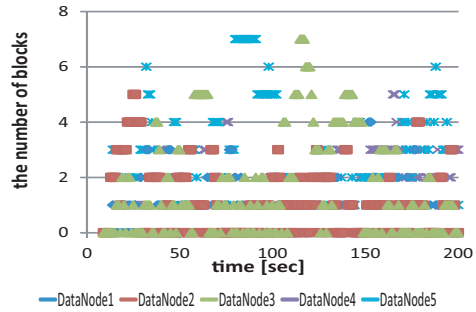

Fig. 2 The number of blocks each DataNode received.

3.1 Overview of the Basic Strategy

All DataNodes are assumed to be in the same rack. DataNodes are arranged in a ring structure, and each DataNode transfers data in one direction based on the ring structure shown in Figure 3. In this manner, the destination DataNode is always the next DataNode after each source DataNode in the ring structure. This one-directional ring strategy enables us to maintain a constant number of blocks that each DataNode is sending and receiving even if the transfer timing of each block is different. Further, because of the ring structure, the destination DataNode is determined uniquely by determining the source DataNode. Therefore, when choosing the source DataNode from which to send the missing blocks, our proposed scheme requires only the number of blocks that each DataNode sends to be equal to eliminate the bias of the sending and receiving process.

In replica reconstruction with the HDFS default scheme, the scheduling and data transfer are carried out in parallel. In our replica reconstruction with our proposed scheme, data transfer is carried out after the scheduling of all of the replicas that are missing is completed. This goal can be realized by specifying a sufficiently large value to N_work described in section 2.2.

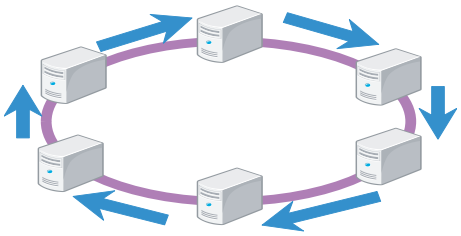

Fig. 3 DataNodes arranged in a one-directional ring structure

3.2 Optimization Scheme

We formulate the replica reconstruction problem as 0-1 integer programming in the replication scheduling strategy we proposed in section 3.1. As described in section 3.1, all DataNodes are arranged in a one-directional ring structure with transfer of missing blocks to the next DataNode. The objective of this replica reconstruction problem is to equalize the number of blocks that each DataNode sends, that is, to minimize the difference in the number of blocks that each DataNode sends.

We define the symbols. The sets of DataNode i and the sets of block j that are required to be replicated are denoted by D and B, respectively. The total number of DataNodes, the total number of blocks that are required to be replicated, and the replication factor are denoted by N_{dn}, N_b, and $N_{replica}$ ($\geq$2), respectively. The average number of blocks N_{avg}, which each DataNode sends, equals N_b/N_{dn}. The current block positions are represented by the matrix $Current_{i,j}$ ($i \in D$, $j \in B$). If DataNode i holds the block j, $Current_{i,j}$ equals 1, otherwise $Current_{i,j}$ equals 0. The adjacency of DataNodes is represented by the matrix $Adj_{from,to}$ ($from, to \in D$). If DataNode $from$ can send to DataNode to, $Adj_{from,to}$ equals 1. Otherwise, $Adj_{from,to}$ equals 0. The scheduling results of replica reconstruction are denoted by the variable $X_{from,to,j}$. If DataNode $from$ sends a block to DataNode to for replication, $X_{from,to,j}$ becomes 1. Otherwise, $X_{from,to,j}$ becomes 0. z_i is the variable that is used to minimize the difference in the number of blocks that each DataNode i sends. Now, replica reconstruction scheduling based on the one-directional ring structure is formulated as follows.

The optimization scheme solves $X_{from,to,j}$, satisfying the above formulation and employing the result for replica reconstruction scheduling. Objective function (1) minimizes the difference in the number of blocks that each source DataNode transfers. Equation (2) defines $All_{i,j}$, that is, the placement of all blocks after transfer. Constraint (3) states that the same block must not be arranged in the same DataNode in the placement of blocks after transfer. Constraint (4) states the total number of replicas of each block must equal $N_{replica}$. Constraint (5) states $X_{from,to,j}$ is 0 or 1. Constraint (6) states that the source DataNode has the block to transfer. Constraint (7) states that DataNode $from$ and DataNode to are in the adjacency that DataNode $from$ can transfer to DataNode to in the one-directional ring. If

$$Minimize$$

$$\sum_{i \in D} z_i \tag{1}$$

$$Subject\ to$$

$$All_{i,j} = Current_{i,j} + \sum_{from \in D} X_{from,i,j}$$

$$\forall i \in D, \forall j \in B \tag{2}$$

$$All_{i,j} \leq 1,\ \forall i \in D,\ \forall j \in B \tag{3}$$

$$\sum_{i \in D} All_{i,j} = N_{replica},\ \forall j \in B \tag{4}$$

$$X_{from,to,j} \in \{0,1\},\ \forall from, \forall to \in D, \forall j \in B \tag{5}$$

$$Current_{i,j} - \sum_{to \in D} X_{i,to,j} >= 0$$

$$\forall i \in D,\ \forall j \in B \tag{6}$$

$$\sum_{j \in B} X_{from,to,j} <= M \cdot Adj_{from,to}$$

$$\forall from, \forall to \in D \tag{7}$$

$$\sum_{j \in B} X_{from,to,j} - N_{avg} \geq -z_i$$

$$\forall from, \forall to \in D \tag{8}$$

$$\sum_{j \in B} X_{from,to,j} - N_{avg} \leq z_i$$

$$\forall from, \forall to \in D \tag{9}$$

$$z_i \geq 0,\ \forall i \in D \tag{10}$$

there is no adjacency between DataNode $from$ and DataNode to, the number of blocks that DataNode $from$ can transfer to DataNode to is 0. Otherwise, it is a positive value. M is a sufficiently large value, which does not exceed the total number of blocks, so we set M to N_b here. Constraints (8), (9) state the lower bound and the upper bound of the difference between the number of blocks and the average number of blocks N_{avg}, respectively. Constraint (10) states z_i is greater than or equal to 0.

3.3 Heuristic Scheme

It is impractical to implement the optimization scheme because it generally takes a long time to find the optimal solution. Thus, we propose a heuristic scheme to obtain replica reconstruction scheduling results. The heuristic scheme aims to equalize the number of blocks that each DataNode transfers. We describe the procedure below. First, we define the symbols. The sets of DataNode i and the sets of block j required to be replicated are denoted by D and B, respectively. As for the block j ($j \in B$), DataNodes which hold the remaining block j are belonging to the list $Candidates_j$ The scheduling priority is denoted by the variable $Priority_j$. $Priority_j$ equals to the number of elements in list $Candidates_j$. Smaller number of $Priority_j$ means higher priority. The total number of times DataNode i has been chosen as the source DataNode is denoted by the variable $Count_i$.

(1) Arranged all DataNodes in a ring logically like we

mentioned in section 3.1.

(2) Execute the following process (2-1) and (2-2) for all the blocks required to be replicated.
As for the block j,

(2-1) For the DataNode i that is included $Candidates_j$, if the next DataNode in the ring structure is included $Candidates_j$, that is the next DataNode already holds the block j, the DataNode i is excluded from the $Candidates_j$.

(2-2) Assign the number of the elements in $Candidates_j$ to $Priority_j$.

(3) Group by the number of $Priority_j$ and after that sort in descending order of the number of $Priority_j$.

(4) Execute for all of the blocks sorted. As for the block j, choose the DataNode with the minimum $Count_i$ from the $Candidates_j$ as the source DataNode and increment the $Count_i$. As a result, the destination DataNode is decided uniquely in the next DataNode in a ring structure.

Step (3), (4) mean that the blocks j which the source DataNode is decided uniquely are scheduled first in the process, and next, the blocks j which are able to choose the DataNode with the minimum $Count_i$ from the $Candidates_j$ as the source DataNode are scheduled. By doing so, it is possible to balance the total number of times each DataNode has been chosen as source DataNode effectively. In step (1), every time replica reconstruction occurs, the logical ring structure is reconfigured. Therefore there is no possibility that data arrangement is biased even if multiple replica reconstructions occur. In addition, our proposed scheme has the probabilistic aspects in terms of the configuration of logical ring structure, that is it is not necessarily determinism.

We show an example in the case that the replication factor is set to three and the number of DataNodes is five. Figure 4 shows a logical ring structure of DataNodes($d1$, ... , $d5 \in D$) and the arrangement of some blocks jn and $jn+1$(jn, $jn+1 \in B$) required to be replicated. As for the block jn, $Candidates_{jn}$ is $\{d1, d3\}$ and $Priority_{jn}$ is 2. As for the block $jn+1$, $Candidates_{jn+1}$ is $\{d5\}$ and $Priority_{jn+1}$ is 1. DataNode $d4$ is excluded from the $Candidates_{jn+1}$ because DataNode $d5$ that is the next one of DataNode $d4$ holds the block $jn+1$. The block $jn+1$ is scheduled and after that the block j is scheduled because scheduling is executed in decending order of the number $Priority$. As for the block $jn+1$, the $Candidates_{jn+1}$ includes only DataNode $d5$ so DataNode $d5$ is chosen as the source DataNode and the DataNode $d1$ which is the next node of DataNode $d5$ is decided as the destination DataNode uniquely. Next, as for the block jn, here we assume that $Count$ of DataNodes which are included in $Candidates_{jn}$, are $Count_{d1}$ = 10 and $Count_{d3}$ = 8 respectively. In this case, DataNode $d3$ is chosen as the source DataNode, and DataNode $d4$ is decided as the destination DataNode, which is the next node of DataNode 3.

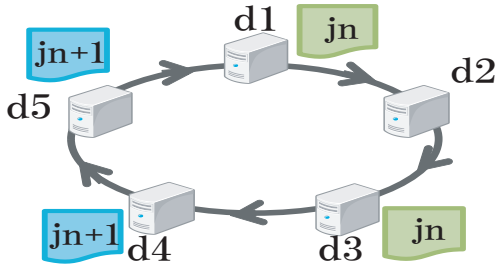

Fig. 4 Example of the arrangement of DataNodes and blocks

Table 2 HDFS cluster node specifications

OS	Linux 2.6.32-5-amd64 Debian GNU/Linux 6.0.4
CPU	Quad-Core Intel(R) Xeon(R) CPU @ 1.60GHz
Main Memory	2 GB
HDD	73GB SAS × 2(RAID0)
RAID Controller	SAS5/iR
Network	Gigabit Ethernet

4. Evaluation Experiments in an actual cluster

We implemented both the optimization scheme and the heuristic scheme into the replica reconstruction module in HDFS. To evaluate the performance of the replica reconstruction with each scheme, we measure the following:

(1) Replica reconstruction throughput
(2) The number of blocks each DataNode transfers

(1) indicates the aggregation data transfer rate among DataNodes for the reconstruction. (2) examines the bias of the volume of processing for sending and receiving.

We used seven nodes on which we installed Hadoop-1.0.3 on an actual cluster. Although the latest implementation is Hadoop-2.5.1, this version we used is almost the same with the latest one in terms of the replica placement policy and the method of replication. One of these nodes is designated as a NameNode and the rest are DataNodes. Table 2 shows the specifications of the nodes we used for the measurements. All nodes are connected by a Gigabit Ethernet and belong to a single rack.

For each scheme (i.e., the default scheme, heuristic scheme, and optimization scheme), we examined the performance of replica reconstruction at node deletion for block sizes of 16, 32, 64, 128, and 256 MB. We define the replica reconstruction throughput as follows.

$$\text{Replica reconstruction throughput } [\text{MB/sec}] = \frac{\text{the amount of data that the deleted DataNode holds } [\text{MB}]}{\text{execution time needed for replica reconstruction to complete } [\text{sec}]} \quad (11)$$

We are using the GLPK[6] optimization solver, which was provided free of charge for the optimization scheme. We copied five files of approximately 10 GB to HDFS from the local disk using the put option in each trial. The replication factor is set to three, so the total amount of data, including replicas in HDFS, is approximately 150 GB, which

Table 3 Measurement parameters

Block size	16, 32, 64, 128, 256 MB
The number of DataNodes	six, including the deleted one
Replication factor	3
The amount of data in HDFS	50 GB*3(replication factor)

corresponds to approximately 25% of the capacity of the entire cluster. Table 3 shows the parameters we used for the measurements. The amount of data each DataNode holds is balanced by a balancer that is implemented as a Hadoop daemon before each trial. Therefore, in each trial, the amount of data each DataNode holds is almost the same, but the placement of each data item differs.

4.1 Replica Reconstruction Throughput

Figure 5 shows the replica reconstruction throughput at node deletion with each scheme. The vertical axis represents the replica reconstruction throughput in MB/sec, and the horizontal axis represents the block size. When the block size is more than 64 MB, the throughput is improved by the proposed schemes. The throughput of the heuristic scheme shows a 44% improvement compared to the throughput of the default scheme, and the throughput of the optimization scheme shows a 45% improvement. When the block size is smaller, such as 16 or 32 MB, the throughput of each scheme does not differ. As we mentioned in section 2.2, this difference occurs because the amount of data that each DataNode can transfer without receiving an acknowledgment from the destination DataNodes is limited by the number of blocks, N_stream. Therefore, in the case where block size is smaller, the amount of data to be processed at one time is less. In such a case, the DataNodes finish the processing instructed from the NameNode sooner and are in the idle state, waiting for the instructions of the replica reconstruction to be sent periodically from the NameNode. There is still available disk bandwidth. When the DataNodes have enough volume of data to write to their disk, our proposed schemes achieve better throughput because the number of instructions distributed among the DataNodes is well-balanced compared with the default case. However, when the DataNodes do not have enough volume of data, they are often in the idle state and disk bandwidth is not occupied enough with write data, and throughput of our proposed schemes is not increased in such a case.

To investigate the effectiveness of the proposed schemes in the case where the processing load is heavy even though block size is small, we examined the replica reconstruction at node deletion by changing N_stream as shown in Table 4. The replica reconstruction throughput of the experiments is shown in Figure 6. The vertical axis represents the replica reconstruction throughput in MB/sec, and the horizontal axis represents the block size and the value of N_stream. In the case where the process load is heavy even though block size is small, the throughput of the proposed schemes also shows improvement, as shown in Figure 6. Figures 5 and 6 show that the replica reconstruction throughput of the heuristic scheme is comparable to the replica reconstruction throughput of the optimization scheme. Therefore, the heuristic scheme is shown to be sufficiently effective in this experimental environment.

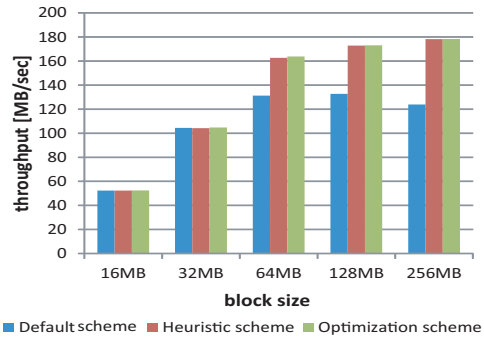

Fig. 5 Replica reconstruction throughput at node deletion with each scheme

Table 4 The value of N_stream

Block size	N_stream
16MB	8
32MB	4

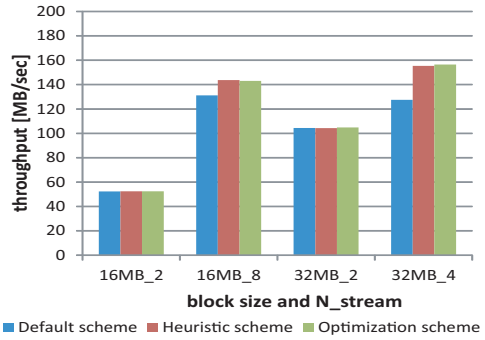

Fig. 6 Replica reconstruction throughput at node deletion in the case of changing N_stream

Figures 7 and 8 show the time series data of the disk I/O throughput of each DataNode and the number of blocks that each DataNode received with the heuristic scheme. In Figure 7, the vertical axis represents the disk I/O throughput in MB/sec and the horizontal axis represents time in sec. In Figure 8, the vertical axis represents the number of blocks received and the horizontal axis represents time in sec. As indicated in Figures 1 and 2, there was a large difference in the number of blocks that each DataNode received, and the disk I/O throughput was unstable in the case of the default scheme. However, with the heuristic scheme, the number of blocks each DataNode received is stable at 2 or less, and the disk I/O throughput of each DataNode is relatively equally high and stable.

4.2 The number of blocks each DataNode transfers

Taking into consideration of all blocks required to be

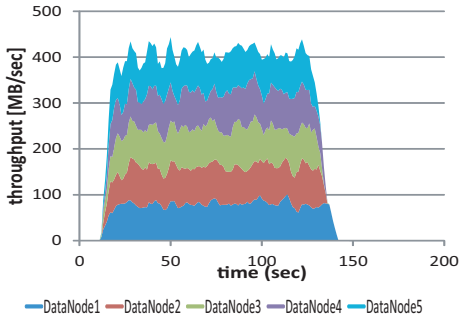

Fig. 7 Aggregated disk I/O throughput of five DataNodes with the heuristic scheme.

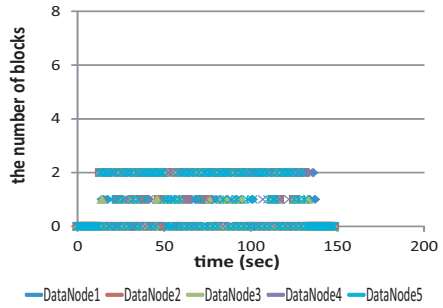

Fig. 8 The number of blocks that each DataNode received with the heuristic scheme.

replicated, there are a large number of combinations of the source and the destination DataNodes because the source DataNode is chosen from two DataNodes which hold the remaining blocks under the assumed condition (replication factor is set to three). The amount of transferred data each DataNode processes can be balanced by deciding the DataNode in a one-directional ring structure as a destination DataNode uniquely. Therefore, assuming a ring structure is reasonable for a well-balanced data transfer. However, even though both the optimization scheme and the heuristic scheme use the same ring structure, a result of the heuristic scheme is not the same as that of the optimization scheme but an approximate answer.

Table 5 shows the number of blocks that each DataNode transfers and the standard deviation, in the replica reconstruction at node deletion with each scheme. As an example, we pick up a single trial with a block size of 64 MB here. Because the data placement is different in each trial, the number of blocks that each DataNode transferred and the standard deviation are also different slightly different in each trial. Table 6 shows the average of the coefficient of variations of all trials with each scheme. It can be seen from Table 6 that the number of blocks that each DataNode transfers is balanced, and the bias of volume of processing for sending and receiving is eliminated by our proposed schemes. The heuristic scheme is also comparable to the optimization scheme in terms of the bias of sending and receiving processing; and therefore, we confirmed that the heuristic scheme is effective.

Table 5 The number of blocks that each DataNode transfers and the standard deviation.

	Default scheme	Heuristic scheme	Optimization scheme
DataNode1	78	78	78
DataNode2	79	79	79
DataNode3	84	78	79
DataNode4	85	79	79
DataNode5	67	79	78
Standard deviation	6.406	0.490	0.490

Table 6 The average of the coefficient of variations of all trials.

	Default scheme	Heuristic scheme	Optimization scheme
The average of the coefficient of variations	0.0945	0.0079	0.0054

Table 7 Measurement parameters

	The number of DataNodes	The number of blocks
The number of DataNodes changes where the number of blocks is fixed	5-25	800
The number of blocks changes where the number of DataNodes is fixed	10	800-4000

5. Evaluation Experiments in a large-scale environment by simulation

We evaluate the performance of the proposed schemes by simulation while the numbers of nodes and blocks are increasing. We measure the following:

(1) The computation time required to find the optimal solution
(2) Replica reconstruction throughput and the number of blocks each DataNode transfers

(1) examines whether if the optimization scheme is practical. (2) investigates whether if the heuristic scheme is scalable.

5.1 The computation time required to find the optimal solution

The optimization scheme is considered impractical because it generally takes a long time to find the optimal solution. Therefore, to evaluate the practicality of the optimization scheme, we examine the computation time required to find the optimal solution of this 0-1 integer programming problem by changing the numbers of DataNodes and blocks in the simulation. Table 7 shows the numbers of DataNodes and blocks we used: 800 blocks corresponds to the amount of data shown in Table 3 when the block size is set to 64 MB. The replication factor is set to 3, as in the previous section.

Figure 9 and 10 show the computation time required to

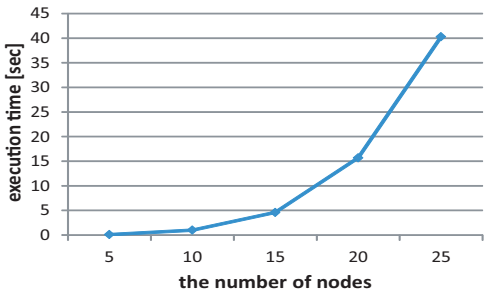

Fig. 9 Fixed number of blocks.

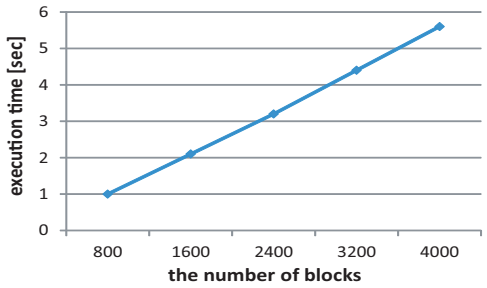

Fig. 10 Fixed number of nodes.

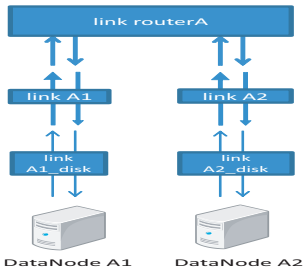

Fig. 11 Network topology

Table 8 Measurement parameters

The number of DataNode	6, 8, 16, 32
Block size	67MB
Replication factor	3
The number of blocks the deleting DataNode holds	80*(The number of DataNodes-1)
Network bandwidth	125 MB/sec
Network latency	0.1msec
Disk performance	67 MB/sec

find the optimal solution of the formulated 0-1 integer programming problem. Figure 9 shows the computation time when the number of DataNodes changes but the number of blocks is fixed. Figure 10 shows the computation time when the number of blocks changes but the number of DataNodes is fixed. From Figures 9 and 10, we confirm that the computation time needed to find the optimal solution increases exponentially with an increase in the number of nodes and linearly with an increase in the number of blocks.

Here we define the number of DataNodes as d, the number of all blocks as b_{all} and the number of blocks to be replicated as $b_{replica}$. The complexity of the optimization scheme is $O(b_{all} \cdot d^2)$. The complexity of the default scheme and the heuristic scheme is $O(b_{replica})$. In practice, because HDFS is used in a large-scale environment such as terabyte to petabyte scale, the number of nodes and the number of blocks are also very large. Therefore, we find that the optimization scheme cannot scale. The heuristic scheme is very efficient because it is possible to achieve replica reconstruction throughput comparable to that of the optimization scheme with a calculation complexity of $O(b_{replica})$.

5.2 Replica reconstruction throughput and the number of blocks each DataNode transfers

In the large-scale environment, we find that while the optimization scheme is impractical, the calculation complexity of the heuristic scheme is comparable to that of the default scheme. Therefore, we evaluate the scalability of the replica reconstruction processes by varying the number of DataNodes for the default scheme and the heuristic scheme using simulations.

We use SimGrid [3], which is a discrete event simulator for distributed systems. Because SimGrid has not supported a disk process simulation, additional links with bandwidth equal to disk I/O throughputs are configured in the simulation environments, as shown in Figure 11. The table 8 shows the measurement parameters we used. The number of DataNodes is set to 6, 8, 16 or 32 including the deleted node. The case in which the number of DataNodes is set to 6 is compared with the evaluation in the actual cluster. The block size is set to 67 MB because the default block size in HDFS is 64 MB, and the transfer size including the header is 67 MB. The number of blocks the deleted DataNode holds is set to be proportional to the number of DataNodes to equalize the average number of blocks each DataNode transfers, regardless of the number of DataNodes. Disk performance is set to 67 MB/sec. We obtained this value when examining the performance of the one-to-one replica reconstruction process in the preliminary experiment in the actual environment. Figure 12 shows the replica reconstruction throughputs at node deletion. The vertical axis represents the replica reconstruction throughput in MB/sec, and the horizontal axis represents the block size. Figure 12 shows that the replica reconstruction throughputs for the both schemes increase when the number of DataNodes increases. The throughput of the heuristic scheme improves 25% at most compared to the default scheme.

Table 9, 10 show that the standard deviation of the number of blocks each DataNode transfers and the average of the coefficient of variations of all trials in the case of the actual environment using six DataNodes we evaluated in section 4 and the large-scale environment using thirty-two DataNodes respectively. The values of the large-scale environment are almost same as those of the actual one. Therefore, we find that both schemes are scalable and the heuristic scheme is effective in the large-scale environment.

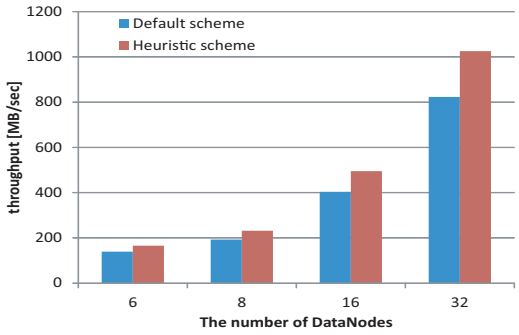

Fig. 12 Replica reconstruction throughputs

Table 9 The comparison of the standard deviation of the number of blocks each DataNode transfers.

	Default scheme	Heuristic scheme
actual environment	6.406	0.490
large-scale environment	8.274	0.775

Table 10 The comparison of the average of the coefficient of variations of all trials.

	Default scheme	Heuristic scheme
actual environment	0.0945	0.0079
large-scale environment	0.1007	0.0098

6. Related Work

6.1 Replication strategies

Many replication strategies for replica management have been proposed. In general, data are replicated, and their replicas are stored on different data nodes for reliability and availability. It is important to select the replication factor properly and to determine where to place each replica.

Rahman et al. [7], Wang et al. [8], and Sato et al. [9] proposed replication strategies based on file clustering for Grid file systems. In the clustering strategy described in [9], files are grouped according to each data processing based on the notion that the clustered files will be simultaneously used by another data processing. Then, the replication times for each file are minimized under the given storage capacity limitations. The experiments showed the proposed strategy was more efficient than a strategy that did not group related files.

Sashi et al. [10] proposed a replication strategy for a region-based framework based on the popularity of files over a geographically distributed Grid environment. By calculating the access frequency of each file, they determine in which region the replicas must be placed and how many replicas must be placed based on network bandwidth and response time between regions. When file f is created, the access frequency is calculated for each region, and replicas are placed in the regions in a descending order of the access frequency. Furthermore, the site within the region in which the file must be placed is determined by considering the number of requests and the response time. Therefore, this strategy increases the data availability and also reduces the number of unnecessary replications.

Tjioe et al. [11] proposed a replication strategy based on a dynamic file assignment according to access load. First, they assign files, which are sorted according to file size, to disks in a round-robin fashion to distribute the load of all files evenly across all disks. Then, creation and deletion of replicas occur according to the load of all files and the load on each disk. From the experiments, load balancing can be achieved in an environment where user access patterns change significantly.

Wenfeng et al. [12] proposed a replication strategy based on the response time of each request. Their strategy determines replica allocation and the number of replicas for each data to satisfy the requirement of response time for acquiring each replica from every node and minimize the replica degree, the number of replicas for each data, at the same time. Compared to other strategies, the proposed strategy could satisfy the response time requirements of every node in a single request and also reduce the number of replicas. Moreover, the proposed strategy maximally reduced the total request response time and improved the overall system performance.

As described above, most of the replication strategies mainly consider an access time, a storage capacity, and a replication time. However, the overhead of a replication process on each node itself is not adequately considered.

6.2 Network topology

Felix et al. [13] investigated distributing an OS image to all machines efficiently in a large-scale cluster. They investigated three logical network topologies: a star topology, an n-ary spanning tree, and a multi-drop-chain. Figure 13 shows each network topology. The blue nodes indicate the source of each data transfer, and the green nodes denote a switch (a connection point between all nodes). The above images of each topology show the logical connectivity of all nodes. From the evaluation experiments, the star topology suffered from heavy link congestion at the server link in the case of an increase in the number of nodes. The n-ary spanning tree could not replicate data into multiple streams efficiently enough because of the limitation of network bandwidth. The multi-drop-chain could replicate data regardless of increasing nodes and network bandwidth. Therefore, these authors concluded that the multi-drop-chain topology was efficient to adopt a large-scale cluster.

Therefore, we assume that data transfer based on the one-directional ring structure we applied is effective.

7. Conclusion

In replica reconstruction with the default scheme for HDFS, inefficient processing occurs during the replica reconstruction because the processing is concentrated on some of the DataNodes, even if the source and destination DataNodes

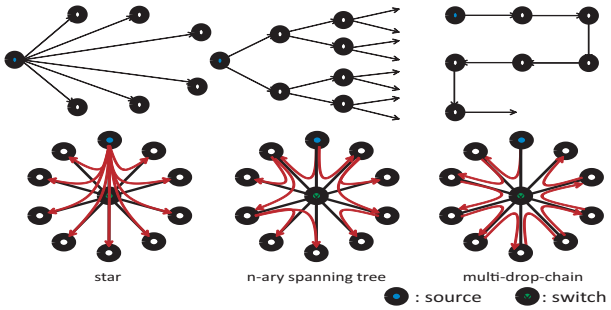

Fig. 13 Network topologies described in [13]

are chosen at random. To address this issue, we proposed two effective replica reconstruction schemes intended to balance the workloads of each DataNode by properly selecting source and destination DataNodes. Our proposed replication scheduling strategy asuumes that DataNodes are arranged in a ring, and data blocks are transferred based on a one-directional ring structure to minimize the difference in the amount of data transfer at each DataNode. Based on this strategy, we proposed two replica reconstruction schemes: an optimization scheme and a heuristic scheme. We have implemented the proposed schemes in HDFS and evaluated them on an actual HDFS cluster and a large-scale simulation environment. From the experiments, we confirmed that the both proposed replica reconstruction schemes outperformed the HDFS default scheme, the heuristic scheme was comparable to the optimization scheme, and the heuristic scheme was very efficient in a large-scale environment.

In this study, we have focused on eliminating the bias of data transfer and tackled the challenge of identifying effective replica reconstruction schemes. However, in practice, the replica reconstruction process is performed in the background, so it is necessary to avoid the situation in which the replica reconstruction process occupies computing and network resources and reduces the performance of the foreground process. Therefore, as future work, we will attempt to perform the replica reconstruction effectively while minimizing the influence on the foreground process. In addition, we will attempt to extend our proposed schemes for the replica reconstruction in which HDFS is operated by multiple racks because HDFS is composed of multiple racks in larger-scale environments.

References

[1] Dhruba Borthakur. ”HDFS Architecture,” 2008 The Apache Software Foundation.
[2] Tom White, Hadoop: The definitive guide, trans. Ryuji Tamagawa. O’Reilly JAPAN, 2010.
[3] SimGrid. http://simgrid.gforge.inria.fr/
[4] Asami Higai, Atsuko Takefusa, Hidemoto Nakada, Masato Oguchi, ”A Study of Effective Replica Reconstruction Schemes at Node Deletion for HDFS,” 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp.512-521, Chicago, IL, USA
[5] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung (October 2003), ”The Google File System,” 19th Symposium on Operating Systems Principles (conference), Lake George, NY: The Association for Computing Machinery, CiteSeerX: 10.1.1.125.789, retrieved 2012-07-12.
[6] GLPK. http://www.gnu.org/software/glpk/.
[7] Rashedur M.Rahman, Ken Barker, Reda Alhajj, ”Study of Different Replica Placement and Maintenance Strategies in Data Grid,” In Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid, pp.171-178, 2007.
[8] Y. Wang and D. Kaeli, ”Load balancing using grid-based peer-to-peer parallel I/O,” In Proceedings of IEEE International Conference on Cluster Computing, pp.1-10, 2005.
[9] Hitoshi Sato, Satoshi Matsuoka, and Toshio Endo, ”File Clustering Based Replication Algorithm in a Grid Environment,” In Proceedings of the 9th IEEE International Symposium on Computing and the Grid (CCGrid2009), pp.204-211, Shanghai, China, May 2009.
[10] K. Sashi, Antony Selvadoss Thanamani, ”A New Replica Creation and Placement Algorithm for Data Grid Environment,” International Conference on Data Storage and Data Engineering, pp.265-269, 2010.
[11] J. Tjioe, R. Widjaja, A. Lee, and T.Xie, ”DORA:A Dynamic File Assignment Strategy with Replication,” International Conference on Parallel Processing 2009.
[12] W.F. Wang, W.H. Wei, ”A Dynamic Replica Placement Mechanism Based-on Response Time Measure,” Proc. of IEEE International Conf. on Communications and Mobile Computing, pp.169-173, 2010.
[13] Felix Rauch, Christian Kurmann, Tomas M.Stricker, ”Partition Cast — Modelling and Optimizing the Distribution of Large Data Sets in PC Clusters,” Euro-Par 2000, LNCS 1900, pp.1118-1131, 2000.

Asami HIGAI Asami Higai received B.E. from Ochanomizu University in 2013. She is currently a student of master’s program in the Department of Computer Science at Ochanomizu University. Her research field is parallel and distributed computing. She is a student member of IPSJ.

Atsuko TAKEFUSA Atsuko Takefusa received B.E., M.E. and Ph.D. (Sci.) from Ochanomizu University in 1996, 1998 and 2000, respectively. From 2000 to 2002, she was a JSPS research fellow (DC2) at the Tokyo Institute of Technology. She stayed at the University of California, San Diego in the United States as a visiting researcher in 2000-2001. She was an assistant professor at Ochanomizu University in 2002 to 2005. She became a researcher at the National Institute of Advanced Industrial Science and Technology (AIST) in 2005. She has been a senior researcher at AIST since 2013. Her research field is parallel and distributed computing including Grid, Cloud and HPC. She is a member of ACM, IPSJ and IEICE.

Hidemoto NAKADA Hidemoto Nakada received his Ph.D. degree from the University of Tokyo in 1995. He joined the Electrotechnical Laboratory in 1995, which was merged into the National Institute of Advanced Industrial Science and Technology (AIST) in 2001. He also served as a visiting associate professor at the Tokyo Institute of Technology from 2001 to 2005. His interests lie in the area of parallel / distributed computing, including Grid and Cloud technologies. He is a member of ACM and IPSJ.

Masato OGUCHI Prof. Masato Oguchi received B.E. from Keio University, M.E. and Ph.D from the University of Tokyo in 1990, 1992, and 1995 respectively. In 1995, he was a researcher in the National Center for Science Information Systems (NACSIS) - currently known as National Institute of Informatics (NII). From 1996 to 2000, he was a research fellow at the Institute of Industrial Science, University of Tokyo. He stayed at Aachen University of Technology in Germany as a visiting researcher in 1998 - 2000. In 2001, he became an associate professor at the Research and Development Initiative in Chuo University. He joined Ochanomizu University in 2003 as an associate professor. Since 2006, he has been a professor at the Department of Information Sciences, Ochanomizu University. His research field is in network computing middleware, including high performance computing as well as mobile networking. He is a member of IEEE, ACM, IEICE, and IPSJ.