A Study of Effective Replica Reconstruction Schemes at Node Deletion for HDFS

Asami Higai
Ochanomizu University
2–1–1, Otsuka, Bunkyo-ku, Tokyo 112–8610, JAPAN
asami@ogl.is.ocha.ac.jp

Atsuko Takefusa
National Institute of Advanced Industrial Science and Technology(AIST)
1–1–1, Umezono, Tsukuba, Ibaraki 305–8568, JAPAN
atsuko.takefusa@aist.go.jp

Hidemoto Nakada
National Institute of Advanced Industrial Science and Technology(AIST)
1–1–1, Umezono, Tsukuba, Ibaraki 305–8568, JAPAN
hide-nakada@aist.go.jp

Masato Oguchi
Ochanomizu University
2–1–1, Otsuka, Bunkyo-ku, Tokyo 112–8610, JAPAN
oguchi@computer.org

***Abstract*—Distributed file systems, which manage large amounts of data over multiple commercially available machines, have attracted attention as a management and processing system 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 the removed data node held 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 in order 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 found 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 of the amount of transfer data of 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 evaluated them on an actual HDFS cluster. From the experiments, we confirm that the replica reconstruction throughput of the proposed schemes show a 45% improvement compared to that of the default scheme. We also verify that the heuristic scheme is effective because it shows performance comparable to the optimization scheme and can be more scalable than the optimization scheme.**

***Keywords*-HDFS; distributed file system; replica; reconstruction; heuristic; optimization;**

I. 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. In order 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 failures. Some data nodes may be unstable due to system failure or maintenance.

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

construction schemes, which aim to balance workloads of copying processes between source and destination data nodes. Our proposed schemes applies to a basic strategy based on a one-directional ring structure, that 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 of 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 HDFS cluster composed of seven nodes. From the experiments, we confirm that the replica reconstruction throughput of the proposed schemes improves by 45% compared to that of the default scheme and the load of each data node can be balanced by eliminating the bias of data transfer. We also verify that the performance of the heuristic and optimization schemes are comparable. We confirm that the heuristic scheme is very effective because it can be more scalable than the optimization 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. Section 4 evaluates our proposed schemes on an actual HDFS cluster. Section 5 discusses the problems to be addressed in the future. Section 6 introduces related work. Finally, we conclude in Section 7.

II. Replica Reconstruction for HDFS

A. Node Decommission and Deletion for HDFS

HDFS is a clone of the Google File System (GFS) [3] 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 of 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 is the minimum unit, 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. One is node decommission and the other is node deletion.

Node decommission is the way that a DataNode is removed from a cluster after replica reconstruction. The decommissioned node itself participates in the replica reconstruction process. This would be the case 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 causes errors frequently.

Node deletion is the way that a DataNode is removed from a cluster before replica reconstruction. The DataNode itself must not participate in the replica reconstruction process. This would be the case in which a DataNode is removed from a cluster unexpectedly. For example, due to node failure or trouble with a network connection,

B. HDFS Replica Reconstruction

When node decommission or deletion is detected, HDFS performs replica reconstruction, which copies the data the DataNode holds to the other DataNodes. The process proceeds by unit of blocks. NameNode makes all decisions regarding this block replication scheduling. NameNode chooses source and destination DataNodes for replications, and periodically transfers replication instructions to each source DataNode. The source DataNode transfers the blocks to the specified destination DataNode based on the instructions. The destination DataNode sends an acknowledgement to the source DataNode after finishing the copying of the block, and then the source DataNode sends an acknowledgement to the NameNode. This phase continues repeatedly until all of the blocks that are missing are replicated.

The number of instructions which 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 totalling 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 I.

Table I
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

C. HDFS Replica Reconstruction Issue

When all DataNodes belong to the same rack, NameNode selects a source DataNode from the DataNodes which hold a copy of the missing blocks, at random, and a destination DataNode, which does not hold the block, at random. This replica reconstruction scheme may cause a concentration of the data transfer process on a few DataNodes. In order 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 MByte 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. And then we counted the number of blocks that each DataNode received, in every one second. This information is derived from the hadoop-$user-datanode.log files in each DataNode. 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 MByte/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 II-B, so it seems an ideal state in which each DataNode is receiving two blocks during the experiment in terms of load balancing.

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 DataNode5 had received 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.

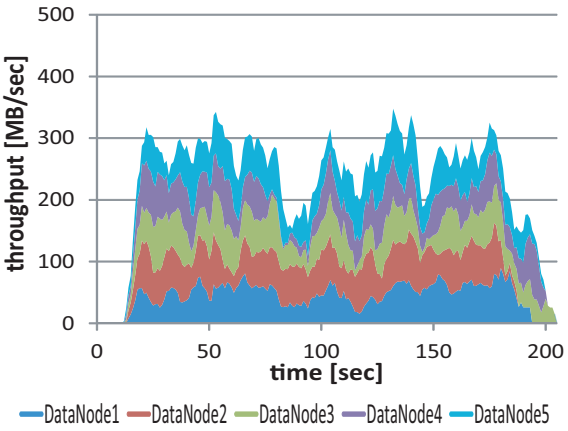

Figure 1. Aggregated disk I/O throughput of five DataNodes.

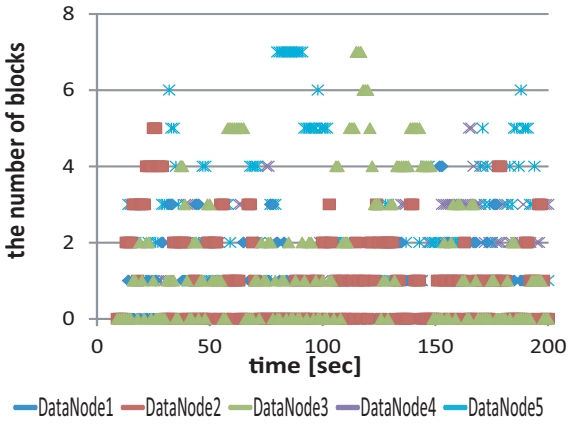

Figure 2. The number of blocks each DataNode received.

III. Proposal for an Effective Replica Reconstruction Scheme

In order 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 at processing efficiently by choosing source and destination DataNodes based on a onedirectional ring structure, and balancing the workload. We propose two schemes: one is an optimization scheme and the other is 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.

A. 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 of 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

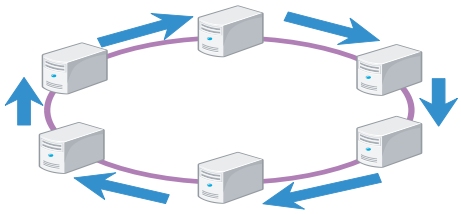

Figure 3. DataNodes arranged in a one-directional ring structure

proposed scheme only requires the number of blocks which each DataNode sends to be equal in order 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 replicas that are missing is completed. This can be realized by specifying a sufficiently large value to N_work described in section II-B.

B. Optimization Scheme

We formulate the replica reconstruction problem as 0-1 integer programming in the replication scheduling strategy we proposed in section III-A. As described in section III-A, all DataNodes are arranged in a one-directional ring structure with transfer of missing blocks to the next DataNode. The object 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 which are required to be replicated are denoted by D and B respectively. The total number of DataNodes, the total number of blocks which 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 is 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 one-directional ring structure is formulated as follows.

It is the optimization scheme that solves $X_{from,to,j}$ satisfying the above formulation and employs the result

$$Minimize$$

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

$$Subject\ to$$

$$All_{i,j} = Current_{i,j} + \sum_{from \in D} X_{from,i,j} \quad \forall i \in D, \forall j \in B \tag{2}$$

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

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

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

$$Current_{i,j} - \sum_{to \in D} X_{i,to,j} >= 0 \quad \forall i \in D,\ \forall j \in B \tag{6}$$

$$\sum_{j \in B} X_{from,to,j} <= M \cdot Adj_{from,to} \quad \forall from, \forall to \in D \tag{7}$$

$$\sum_{j \in B} X_{from,to,j} - N_{avg} \geq -z_i \quad \forall from, \forall to \in D \tag{8}$$

$$\sum_{j \in B} X_{from,to,j} - N_{avg} \leq z_i \quad \forall from, \forall to \in D \tag{9}$$

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

for replica reconstruction scheduling. Objective function (1) minimizes the difference of 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 the source DataNode has the block to transfer. Constraint (7) states DataNode $from$ and DataNode to are in the adjacency which DataNode $from$ can transfer to DataNode to in the one-directional ring. If there is no adjacency between DataNode $from$ and DataNode to, the number of blocks which 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.

C. Heuristic Scheme

Because it is impractical to implement the optimization scheme since it generally takes a long time to find the optimal solution, 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.

(1) DataNodes, which hold the block to be replicated, are the candidates to be designated as the source DataNode. However, if the next DataNode in the ring structure already holds the same block, the DataNode is excluded from the candidates.
(2) Give the priority k to the blocks to be replicated. For each replicated block, the priority k ($0 \leq k \leq$ replication factor-2) is calculated. k is the number of source DataNode candidates excluded in (1). Then the replicated blocks are grouped by k.
(3) Execute the following process for each group in descending order of priority k. For each block in each group, choose the source DataNode from the candidates. Here, the total number of times a DataNode has been chosen as the source DataNode is counted and the DataNode with the minimum total number is selected as the source DataNode.

After step (1), step (2) is executed for all of the replicated blocks, and then step (3) is executed. In step (3), the number of times each DataNode has been chosen as the source DataNode is balanced by scheduling the blocks whose number of candidates is fewer at first.

For example, when the replication factor is set to three, the number of candidates is two for each block. However, if the next DataNode in the ring structure holds the block already, the DataNode is excluded from the candidates, so that the number of the candidates equals one, and the source DataNode is determined uniquely. If such blocks are scheduled later in the scheduling process, it may happen that the DataNode as the source is chosen too often. To avoid the case, we define the priority k for each block.

IV. Evaluation Experiments

We implemented both the optimization scheme and the heuristic scheme into the replica reconstruction module in HDFS. In order 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
3) The computation time needed to find the optimal solution

1) indicates the data transfer rate among DataNodes for the reconstruction. 2) examines and evaluates the bias of the volume of processing for sending and receiving. 3) examines and evaluates if the optimization scheme is practical.

Table II
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	2GByte
HDD	73GByte SAS $\times$ 2(RAID0)
RAID Controller	SAS5/iR
Network	Gigabit Ethernet

Table III
Measurement parameters

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

A. Overview of Experiments

We used seven nodes on which we installed Hadoop-1.0.3 on an actual cluster. One of them is designated as a NameNode and the rest are DataNodes. TableII 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.

With respect to IV-1), 2) mentioned above, for each scheme, that is, the default scheme, heuristic scheme, and optimization scheme, we examined the performance of replica reconstruction at node deletion for block sizes 16, 32, 64, 128, 256 MByte. We define the replica reconstruction throughput as follows.

$$\begin{aligned}&\text{Replica reconstruction throughput } [\mathrm{MByte/sec}]\\ &=\frac{\text{the amount of data that the deleted DataNode holds } [\mathrm{MByte}]}{\text{execution time needed for replica reconstruction to complete } [\mathrm{sec}]}\end{aligned} \tag{11}$$

We are using the GLPK[4] optimization solver, which was provided free of charge for the optimization scheme. We copied five files of about 10 GByte 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 GByte, which corresponds to about 25% of the capacity of the entire cluster. Table III shows the parameters we used for the measurements. The amount of data each DataNode holds is balanced by a balancer which 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.

With respect to IV-3), we examined the computation time required to find the optimal solution of this 0-1 integer programming problem by changing the number of DataNodes and Blocks in the simulation. Table IV shows the number of DataNodes and Blocks we used. 800 blocks corresponds to the amount of data shown in Table III when the block size is set to 64 MByte.

Table IV
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

B. Experimental Results

1) Replica Reconstruction Throughput:
Figure 4 shows the replica reconstruction throughput at node deletion with each scheme. The vertical axis represents the replica reconstruction throughput in MByte/sec, and the horizontal axis represents the block size. When the block size is more than 64 MByte, the throughput is improved by the proposed schemes. The throughput of the heuristic scheme shows a 44% improvement compared to that of the default scheme, and that of the optimization scheme shows a 45% improvement. When the block size is smaller, such as 16 and 32 MByte, the throughput of each scheme does not differ. As we mentioned in section II-B, this is because the amount of data that each DataNode can transfer without receiving an acknowledgement 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. That is, there is still available disk bandwidth.

In order 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 V. The replica reconstruction throughput of the experiments is shown in Figure 5. The vertical axis represents the replica reconstruction throughput in MByte/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 show improvement, as shown in Figure 5. It can be seen from Figures 4 and 5, the replica reconstruction throughput of the heuristic scheme is comparable to that of the optimization scheme. Therefore the heuristic scheme is shown to be effective enough in this experiment environment.

Figures 6 and 7 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, respectively. In Figure 6, the vertical axis represents the disk I/O

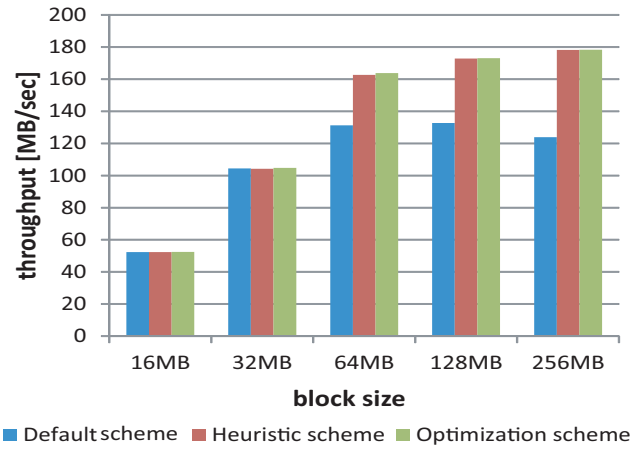

Figure 4. Replica reconstruction throughput at node deletion with each scheme

Table V
THE VALUE OF N_stream

Block size	N_stream
16MByte	8
32MByte	4

throughput in MByte/sec and the horizontal axis represents time in sec. In Figure 7, 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 which 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 to 2 or less and the disk I/O throughput of each DataNode is relatively equally high and stable.

2) The number of blocks each DataNode transfers:
Table VI 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 64 MByte 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 in each trial. Table VII shows the average standard deviation of each trial with each scheme. It can be seen from Table VII 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.

3) The computation time needed to find the optimal solution:
We evaluate by simulation the computation time needed to

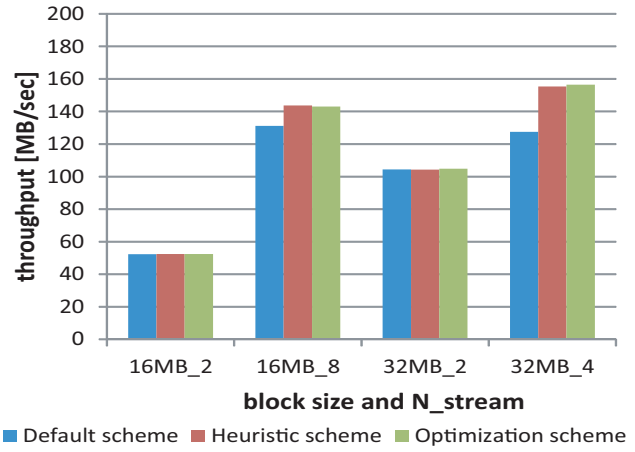

Figure 5. Replica reconstruction throughput at node deletion in the case of changing N_stream

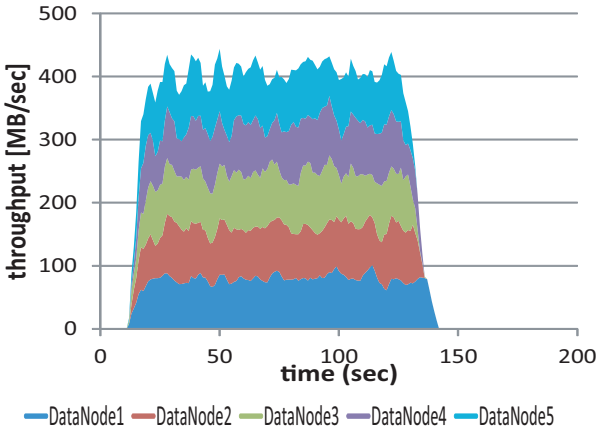

Figure 6. Aggregated disk I/O throughput of five DataNodes with the heuristic scheme.

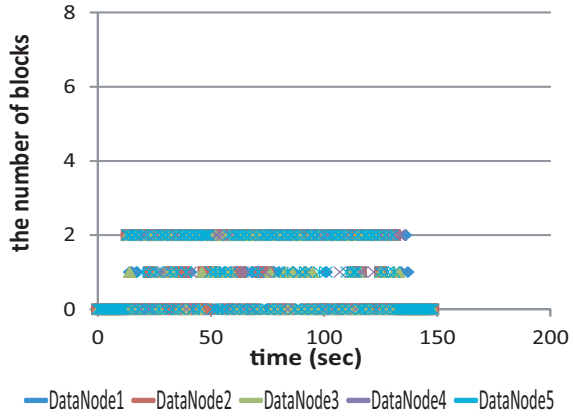

Figure 7. The number of blocks that each DataNode received with the heuristic scheme.

Table VI
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	7.162	0.548	0.548

find the optimal solution of the 0-1 integer programming problem formulated. Figure 8 shows the computation time when the number of nodes changes but the number of blocks is fixed. Figure 9 shows the computation time when the number of blocks changes but the number of nodes is fixed. From Figures 8 and 9, we confirm that the computation time needed to find the optimal solution increases exponentially with the increase of the number of nodes and linearly with the increase of the number of blocks. Here, we define the number of nodes as d and the number of blocks as b. The complexity of the optimization scheme is $O(b \cdot d^2)$, on the other hand, the complexity of the default scheme and the heuristic scheme is $O(b)$.

In practice, because HDFS is used in large-scale environment such as terabyte and petabyte scale, the number of nodes and the number of blocks is also huge. Therefore, it is found that the optimization scheme cannot scale. On the other hand, the heuristic scheme is very efficient because it is possible to achieve replica reconstruction throughput comparable to the optimization scheme with a calculation complexity of $O(b)$.

V. DISCUSSION

In this study, we have focused on eliminating the bias of data transfer and tackled the challenge of finding effective

Table VII
AVERAGE STANDARD DEVIATION OF THE NUMBER OF BLOCKS THAT EACH DATANODE TRANSFERS.

	Default scheme	Heuristic scheme	Optimization scheme
Average standard deviation	8.337	0.697	0.481

replica reconstruction schemes. However, in practice, the replica reconstruction process is performed in the background, so it is necessary to avoid the situation that this process occupies the bandwidth and reduces the performance of the foreground process. Therefore, as future work, we are going to attempt to perform the replica reconstruction effectively while minimizing the influence on the foreground process. And also we have been discussing the assumption all nodes belong to a single rack. In practice, HDFS is operated by configuring multiple racks. When there are multiple racks in the HDFS cluster, multiple replicas are

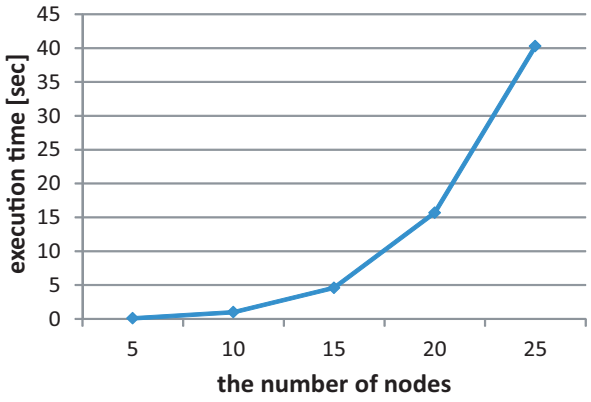

Figure 8. Computation time when the number of nodes changes, but the number of blocks is fixed.

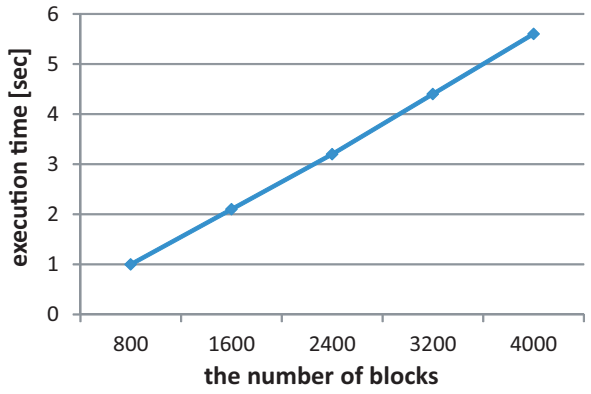

Figure 9. Computation time when the number of blocks changes, but the number of nodes is fixed.

placed according to the following replica placement policy: The first replica is placed on the same node as the client. But if the client is running outside the cluster, a node is chosen at random. The second replica is placed on a different rack from the first, chosen at random. The third replica is placed on the same rack as the first, but on a different node chosen at random. Further replicas are placed on random nodes in the cluster. Therefore, with respect to replica reconstruction we must consider the process between racks. We think we can apply the replication scheduling strategy we proposed even if there are multiple racks in an HDFS cluster. We are assuming two one-directional ring structures: one is a ring between racks in a cluster, and the other is a ring between nodes in each rack. In replica reconstruction, if there is a DataNode that holds a replica of missing blocks in the same rack as the removed node, data transfer is executed based on the one-directional ring between nodes in the rack, and if not, data transfer is executed based on the one-directional ring between racks in the cluster.

VI. Related Work

A. Replication strategies

A lot of 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. And it is important to decide the replication factor properly, as well as where to place each replica.

Rahman et al. [5], Wang et al. [6], and Sato et al. [7] proposed replication strategies based on file clustering for Grid file systems. In the clustering strategy described in [7], 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 replication times for each file are minimized under given storage capacity limitations. From the experiments, they showed the proposed strategy was more efficient than a strategy that did not group related files.

Sashi et al. [8] proposed a replication strategy for a regionbased 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 have to be placed and how many replicas have to 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 with the large in a descending order of the access frequency. Furthermore, in which site within the region the file has to be placed is determined by considering the number of requests and the response time. Therefore, their strategy increases the data availability and also reduces the number of unnecessary replications.

Tjioe et al. [9] 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 so as to distribute the load of all files evenly across all disks. Then, creation and deletion of replicas are occurred 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. [10] proposed a replication strategy based on a response time of each request. It determines replica allocation and the number of replicas for each data in order to satisfy the requirement of response time acquiring each replica from every node, and minimize the replica degree, the number of replica for each data, at the same time. From the experiments compared to other strategies, the proposed strategy could meet with every node 's response time requirements of a single request and also reduced the number of replica degree. Moreover, it reduced the total request response time at most and improved the overall system performance.

As described above, most of replication strategies mainly

consider an access time, a storage capacity, and a replication time. However, an overhead of a replication process on each node itself is not considered well.

B. Network topology

Felix et al. [11] 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 10 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 pictures of each topology show a 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 increasing the number of nodes. And the n-ary spanning tree could not replicate data into multiple streams efficiently enough because of the limitation of network bandwidth. On the other hand, the multi-drop-chain could replicate data regardless of increasing nodes and network bandwidth. Therefore, they concluded the multi-drop-chain topology was efficient to adopt a large-scale cluster.

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

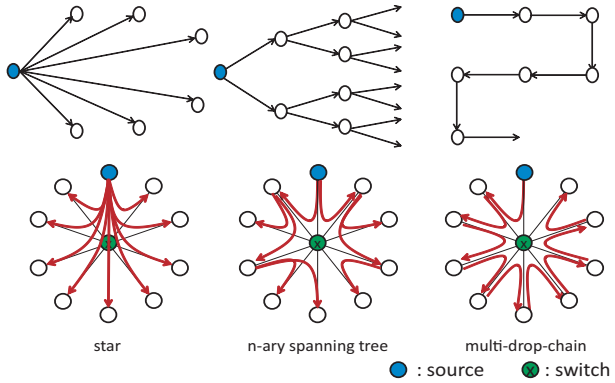

Figure 10. Network topologies described in [11]

VII. Conclusion

In replica reconstruction with the default scheme for HDFS, it is an issue that inefficient processing occurs during the replica reconstruction because the processing is concentrated on some of the DataNodes, even if source and destination DataNodes are chosen at random. To address this issue, we proposed two effective replica reconstruction schemes intended to balance the workloads of each DataNode by choosing source and destination DataNodes properly. Our proposed replication scheduling strategy requires that DataNodes are assumed to be arranged in a ring and data blocks are transferred based on a one-directional ring structure to minimize the difference of the amount of transfer data of 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. From the experiments, we confirmed that the replica reconstruction throughput of the proposed schemes showed a 45% improvement compared to that of the default scheme, and the load of each DataNode can be balanced by eliminating the bias of data transfer. We also verified that the heuristic scheme was comparable to the optimization scheme with respect to the replica reconstruction throughput and balancing the load of each DataNode. Furthermore, we confirmed that the heuristic scheme was very effective because it can be more scalable than the optimization scheme.

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] 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.

[4] GLPK. http://www.gnu.org/software/glpk/.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] J. Tjioe, R. Widjaja, A. Lee, and T.Xie, "DORA:A Dynamic File Assignment Strategy with Replication," International Conference on Parallel Processing 2009.

[10] 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.

[11] 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.