A Study of Replica Reconstruction Schemes for Multi-rack HDFS Clusters

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 enable users to manage large amounts of data over multiple commodity computers, have attracted attention as a potential management and processing system for big data applications. The Hadoop Distributed File System (HDFS) is a widely used open source distributed file system. In the HDFS, multiple replicas are separately stored over the multiple data nodes for enhanced availability. When a data node failure is detected, replica reconstruction is performed. During this process, the access load of the other data nodes, which hold the lost data blocks, may increase, so that the overall performance of data processing over the distributed file system decreases. Therefore, an important issue is effective replica reconstruction in order to prevent such performance degradation. In addition, HDFS composed of multiple racks is needed to replicate the missing blocks on a different rack according to the HDFS replica placement policy, for the purpose of availability. We have to take into account network bandwidth and fault tolerance for such blocks which require data transfer between racks in the cluster. In this paper, we propose replica reconstruction schemes for a multi-rack HDFS cluster and evaluate the effectiveness of our proposed schemes in multi-rack cluster environments by simulation. In the proposed schemes, data transfer in a rack is performed based on a one-directional ring structure and inter-rack data transfer is performed in a round robin manner. We control streams between racks as giving the priority for the blocks which requires inter-rack transfer. The experiments show that the proposed schemes are effective for reduction of the execution time and improvement of the fault tolerance. We also confirm that the performance shows further improvement by controlling the number of streams between racks properly and the execution times of our proposed schemes show a 16% reduction in time required compared to that of the default scheme.**

***Keywords*-HDFS; distributed file system; replica reconstruction; data management;**

I. Introduction

Large amounts of data are generated from high quality sensor networks, social network services, and high performance scientific experimental tools, such as genome sequencer. Such ”Big Data” require efficient data management and processing in various fields of commerce and scientific computing, including high-energy physics and life information sciences. Distributed file systems, which enable users to manage large amounts of data over multiple commodity computers, are widely used for such Big Data processing. In distributed file systems, data are replicated and the data, including their replicas, are separately stored on a large amounts of data nodes for availability and reliability. When a data node failure has been detected, the data blocks stored on the faulty data node are lost. The access load of other data nodes, which also hold the lost data blocks, increases, so that the overall 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) [2], which is a part of the Apache Hadoop [1] project, is a widely used open source distributed file system. Large-scale HDFS clusters are generally configured in multiple racks to manage the large amount of data expected. In such a multi-rack configuration, replica data blocks are stored in two or more racks according to the HDFS replica placement policy in order to ensure data availability. Therefore, we have to take into account network bandwidth between racks and the fault tolerance of such blocks, which requires inter-rack data transfer. In our previous work [3], we found that the HDFS replica reconstruction scheme is inefficient because source and destination data nodes for replication are selected randomly, and we proposed replica reconstruction schemes for single rack HDFS clusters, in which data blocks are transferred based on a one-directional ring structure so that the difference in the amount of transfer data for each node is minimized. We confirmed that our proposals are effective in an actual single rack HDFS cluster. However, the study of

replica reconstruction schemes in large-scale environments is currently insufficient.

In this paper, we propose replica reconstruction schemes for a large-scale HDFS cluster configured of multiple racks. And we evaluate the proposed schemes in multi-rack environments by simulation using the SimGrid [5] simulator, which is a discrete event simulator for Grid and Cloud Computing. In the proposed schemes, the source data nodes for replication are selected to balance the load of each data node with respect to data transfer among data nodes in a rack, the destination data nodes are selected based on a one-directional ring structure for which we have already verified the effectiveness in [3]. On the other hand, with respect to inter-rack data transfer, the destination data nodes are selected from the data nodes included in the destination racks based on a round robin scheme. We schedule these missing blocks by controlling the number of streams between racks in the cluster and giving priority for the blocks based on consideration of the loads of networks and data nodes, and fault tolerance. From the experiments, we show that the proposed schemes are effective for the reduction of the execution times and improvement of the fault tolerance. In addition, we confirm that the performance shows further improvement by the reduction of the load of the destination node by setting the number of streams between racks to the number of nodes which belong to the rack, including the destination node. The execution time of one of our proposed schemes showed a 16% reduction compared to that of the default scheme.

This paper is organized as follows: Section 2 describes the HDFS replica reconstruction and the single rack replica reconstruction schemes we have already proposed in [3]. Section 3 explains the proposed multi-rack replica replication schemes. Section 4 evaluates the proposed schemes by simulation. Section 5 discusses the problems to be addressed in the future. Section 6 introduces related work. Finally, we conclude in Section 7.

II. HDFS REPLICA RECONSTRUCTION AND OUR PREVIOUSLY PROPOSED SCHEMES

HDFS is a part of the Apache Hadoop [1] project and a clone of the Google File System (GFS) [4] 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 HDFS 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 in the cluster. Their replica blocks are separately stored on the other DataNodes for fault tolerance.

A. Replica placement policy in multi-rack environments

When the HDFS cluster is configured as a single rack, all replicas are stored in the same rack. However, a cluster is generally configured using multiple racks having a large number of DataNodes in order to manage large amounts of data. Replicas are placed in two or more racks considering reliability, availability and network bandwidth utilization. In the latest versions Hadoop-2.2.4., they are placed according to the following replica placement policy.

First replica
: is placed on the DataNode where the writer to HDFS is located. If the writer is not in the cluster, a DataNode is selected randomly.

Second replica
: is placed on a DataNode which is randomly selected from a rack that is different from that of the first replica.

Third replica
: is placed on a DataNode which is randomly selected from the same rack as that of the second replica.

Further replicas
: are placed on a DataNode which is selected randomly.

B. Replica reconstruction process in HDFS

When a DataNode failure is detected, a replica reconstruction process is performed because the data blocks stored in the DataNode are lost. In the case where an HDFS cluster is configured as a single rack, the source and destination DataNodes are selected randomly, and the data transfer among the DataNodes in the rack is performed. On the other hand, in the case where an HDFS cluster is configured of multiple racks, the replica has to be placed in a suitable rack according to the replica placement policy described in the previous section. As an example, when the replication factor, which is equal to the total number of replicas in the HDFS cluster, is set to three and one DataNode fails, the state of the remaining replicas is one of the following two cases: One is the case where the remaining replicas are in the same rack and the other is the case where the remaining replicas are in different racks.

In the HDFS cluster, data transfer among DataNodes in the same rack is given higher priority than data transfer between racks on the assumption that inter-rack network bandwidth is smaller than aggregated intra-rack network bandwidth. Therefore, in the former case, intra-rack data transfer is performed to replicate the missing block as shown in Figure 1. On the other hand, in the latter case, inter-rack data transfer is performed to replicate the missing block as shown in Figure 2.

C. Our previously proposed replica reconstruction schemes for single rack environments

In our previous work, we proposed effective replica reconstruction schemes for single rack environments and showed that the proposed schemes were, in fact, effective [3]. The

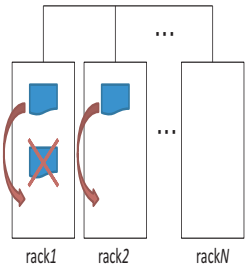

Figure 1. The intra-rack data transfer case where data transfer among DataNodes in a rack is performed.

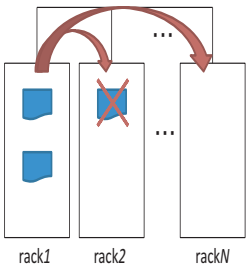

Figure 2. The inter-rack data transfer case where data transfer between racks in the cluster is performed.

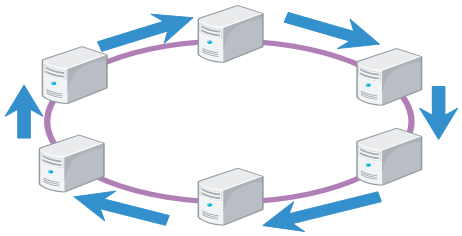

Figure 3. Data transfer based on a one-directional ring structure

replica reconstruction with the HDFS default scheme for single rack environments is inefficient because the data transfer is biased. In order to solve the issue, we proposed more effective replica reconstruction schemes eliminating the bias of the workload by transferring data based on a one-directional ring structure shown in Figure 3. Then, we intended to balance the amount of data blocks which each DataNode sends and receives, as well as to prevent disk I/O performance degradation due to the accesses from multiple DataNodes. We proposed two schemes: one was an optimization scheme formulated as 0-1 integer programming and the other was a heuristic scheme for obtaining the solution in a greedy manner.

From our experiments, we verified that the heuristic scheme showed comparable performance to the optimization scheme with a calculation complexity same as that of the default scheme while the optimization scheme was impractical because the computation time required to find the optimal solution increased exponentially.

In the next section, we describe the issues that occur in the case of multiple racks and an extension of the proposed heuristic scheme.

III. Replica reconstruction schemes for multi-rack HDFS clusters

We extend our previously proposed heuristic scheme in the HDFS cluster to the case where the cluster is configured of multiple racks. The proposed schemes aim to perform the process effectively and improve the fault tolerance by balancing the amount of data transfer each DataNode sends and receives, controlling the number of streams between racks and designating the priority for the blocks.

A. The Selection algorithm for source and destination DataNodes in replica reconstruction

In the replica reconstruction for multiple racks, as we mentioned in section II-B, in the intra-rack data transfer case (Figure 1), data transfer is performed based on a one-directional ring structure, the same as in our previously proposed scheme. In the inter-rack data tranfer case (Figure 2), the destination rack is selected first, then a DataNode of the selected rack is assigned to the process in a round robin fashion. We aim to balance the workload of each DataNode by these strategies. In the case where an HDFS cluster is configured of three or more racks, with respect to data transfer between racks in the cluster, it is less likely to become a performance bottleneck because the destination rack can be selected from multiple racks. On the other hand, in the case that HDFS cluster is configured of two racks, there is a concern that performance may be degraded because data transfer in these racks is performed both between racks in the cluster as well as within a single rack. Therefore, in this paper, we focus on and propose replica reconstruction schemes in HDFS clusters configured of two racks and investigate the characteristics.

In the replica reconstruction, a source DataNode for replication is selected from the DataNodes which hold the missing data and a destination DataNode to be used for replication is selected from the DataNodes which do not hold the missing data. We call the set of DataNodes which holds the missing data ”Candidate Nodes.” And we call the racks other than the one including the deleted DataNode ”normal racks.” On the other hand, the rack including the deleted DataNode is called the ”failure rack.” Note that another DataNode in a failure rack is still capable of recovering replication data.

Namely, with respect to inter-rack data transfer, the rack including a source DataNode is ”normal rack” and another rack including a destination DataNode is ”failure rack.” When a failure rack receives a replica block from a normal rack, it is assigned to a DataNode of that failure rack in a round robin manner. The source and destination DataNodes for missing replicas are selected according to the following algorithm, in order to balance the workload of each DataNode by equalizing the number of blocks each DataNode has sent, with respect to intra-rack and inter-rack data transfer.

i Arrange the DataNodes in a ring structure logically, per rack.

ii Select the source and destination DataNodes for all blocks that need to be replicated as follows:

a) If intra-rack data transfer is possible:
Source: Select the DataNode with the smallest number of intra-rack transferred blocks from Candidate Nodes.

Destination: Uniquely determined by the ring structure in the rack.

b) Else if inter-rack data transfer is necessary:
Source: Select the DataNode with smallest number of inter-rack transferred blocks from Candidate Nodes.
Destination: Select the DataNode in the failure rack logically numbered in serial order with value of the remainder of "the total number of blocks transferred from the normal rack to the failure rack" divided by "the number of DataNodes included in the failure rack."

B. The scheduling schemes for replica reconstruction

With respect to the missing replicas, source and destination DataNodes were determined based on the above algorithm. In addition, we perform scheduling that controls the number of streams between racks by giving priority to the inter-rack data transfer blocks in order for fault tolerance.

1) Prioritized and Non-prioritized schemes: If a failure across the entire rack has occurred, the missing blocks whose remaining replicas are in the same (failure) rack are irreparable. Though it is less likely that such a failure would occur, fault tolerance can be improved by scheduling such missing blocks at the earliest opportunity. Therefore we propose a scheme whereby the missing blocks whose remaining replicas are in the same rack are scheduled on ahead by giving a higher priority to them than to the missing blocks whose remaining replicas are in different racks. We call this scheme the "Prioritized scheme." On the other hand, the scheme used when all missing blocks are scheduled in any order without distinguishing the missing blocks of these two states is called the "Non-prioritized scheme."

2) Control of the number of intra-rack and inter-rack streams: There is a possibility that the network between racks may become the bottleneck if network bandwidth is occupied when data transfer is deluged between racks. The number of blocks each DataNode can transfer to the destination DataNode at one time is originally two. After receiving an acknowledgement from the destination DataNode, the source DataNode can transfer the next blocks. Accordingly, the maximum number of streams between racks is *"The number of DataNodes of a normal rack * 2*". Therefore, if the number of DataNodes in a rack is large, it is assumed that the network between racks is heavily congested when we use the Prioritized scheme. And the DataNodes of the failure rack, which are assigned to the destination DataNodes in the inter-rack data transfer case, have to perform not only intra-rack data transfer (sending and receiving), but also inter-rack data transfer (receiving only).

Therefore, in order to perform the data transfer of the replica reconstruction process effectively by controlling these processing loads, we propose two approaches; (a) controlling the number of inter-rack streams, (b) controlling the number of intra-rack streams.

In (a), when the actual number of streams between racks has reached the maximum number of inter-rack streams, data transfer within the rack is processed, even with prioritized scheme. In (b), until the data transfers between racks are completed, the number of sending streams of each DataNode in the failure rack is limited to one, although the number of sending streams of each DataNode is originally two. (b) is applicable only to the prioritized scheme which carries out the data transfer between racks first.

IV. Evaluation experiments

We evaluated the performance of the proposed replica reconstruction schemes for the HDFS cluster consisting of two racks, comparing that with the default HDFS scheme. In these experiments, we set the number of DataNodes in each rack to be the same at first, and then we deleted one of the DataNodes in one of the racks.

We measured the following:

Exp. 1. Execution times of Prioritized / Non-prioritized schemes.
The execution time of replica reconstruction without the stream control between racks.

Exp. 2. Fault tolerance of Prioritized / Non-prioritized schemes.
The progress of the replication of the blocks which are transferred between racks.

Exp. 3. Execution times of replication schemes with stream control.
The execution time of replica reconstruction with the stream control between racks. This experiment is to identify the optimum number of streams between racks.

A. Overview of experiments

For each scheme, we examined the performance of replica reconstruction by simulation. We used SimGrid[5], which is a simulator for distributed systems. Currently, SimGrid does not support disk processing simulations. Therefore, we represent disks as network links whose bandwidth is set to disk I/O performance, as shown in Figure 4. Table I shows the parameters we employed for the measurements. In the experiments, we set up an HDFS cluster consisting of two racks where the number of DataNodes are the same, and we delete one of the DataNodes of one of the racks. The number of DataNodes is set to 8, 16 and 32 per rack. We set the block size to 67 MB, since the default block size is 64 MB and transfer size, including the header, is 67 MB in the default setting of the HDFS. The number of blocks the deleted DataNode holds is set to be a constant average number of blocks which each DataNode sends during a replica reconstruction process, regardless of the number of DataNodes. We assume the network within a rack is 1 giga

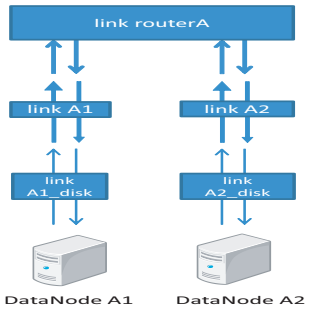

Figure 4. Network topology

Table I
MEASUREMENT PARAMETERS

The number of DataNodes	8*2, 16*2, 32*2
Block size	67MB
Replication factor	3
The number of blocks the deleted DataNode holds	80*(The number of all DataNodes excluding the deleted one)
Network bandwidth in a rack	125MB/sec (1Gbps)
Network latency in a rack	0.1msec
Network bandwidth between racks	1.25GB/sec (10Gbps)
Network latency between racks	0.1msec
Disk I/O performance	67MB/sec

Table II
THE REPLICA RECONSTRUCTION SCHEMES WE USED FOR EACH MEASUREMENT

Exp. 1. Exp. 2.	**Default** **Prioritized** **Prioritized w/ intra** (intra-rack stream control) **Non-prioritized**
Exp. 3.	**Prioritized w/ inter** (inter-rack stream control) **Prioritized w/ intra and inter** (intra-rack and inter-rack stream control) **Non-prioritized w/ inter** (inter-rack stream control)

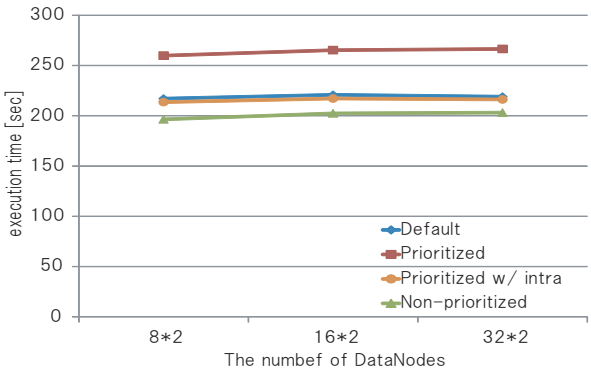

Figure 5. The execution times of Default, Prioritized w/ and w/o intra-rack stream control and Non-prioritized schemes.

Ethernet and the network between racks is 10 giga Ethernet. The disk I/O performance is set to 67 MB/sec based on actual peer to peer replica reconstruction experiments with two machines.

Table II shows the replica reconstruction schemes we used for Exp. 1., Exp. 2. and Exp. 3.. For Exp. 3., we measured the execution time of replica reconstruction changing the maximum number of streams between racks from one to twice the number of DataNodes in the rack.

B. Experimental Results

1) The execution times of Prioritized / Non-prioritized schemes: Figure 5 shows the execution time of replica reconstruction upon node deletion for each scheme. The vertical axis represents the execution time of the replica reconstruction in seconds, and the horizontal axis represents the number of DataNodes. Figures 6, 7, 8 and 9 show the progress of the replication of the blocks which are grouped into data transfer between racks, intra-normal rack and intra-failed rack with each scheme in the case where the number of DataNodes is 8*2. The vertical axis represents the progress of the replication as a percentage, and the horizontal axis represents the time in seconds. Table III shows the average replication time per block with each scheme in the case where the number of DataNodes is 8*2.

Figure 5 shows that the execution time of replica reconstruction is constant regardless of the number of DataNodes. The execution time of **Non-prioritized** shows a 10% reduction compared to that of **Default**. This is because the imbalance among the replica reconstruction processes is eliminated by the data transfer control based on the one-directional ring structure within racks and the round robin control between racks. Table III demonstrates that the average replication time per block is decreasing and replicas are reconstructed efficiently.

On the other hand, the execution time of **Prioritized** shows a 20% increase compared to that of **Default**. However, by adding control on sending streams of DataNodes within the failure rack (**Prioritized w/ intra**), we achieved comparable performance to that of **Default**. As discussed in section III-B2, the performance degradation in the uncontrolled Prioritized scheme (**Prioritized**) is caused by overload in DataNodes on the failure rack. On the DataNodes, disk write bandwidth restricts each stream throughput and slows down the replication. As a result, the process takes longer, as shown in Table III. On the other hand, according to Table III, the scheme with stream control (**Prioritized w/ intra**) exhibits a relatively short time for replication. However, total execution time is almost the same as that of **Default**. This is because data transfer within the normal rack cannot be carried out until data transfer between racks has been completed, as shown in Figure 8 denoted as 'Between racks.' Because the scheme gives higher priority to the replication of missing blocks whose remaining replicas are in the same rack.

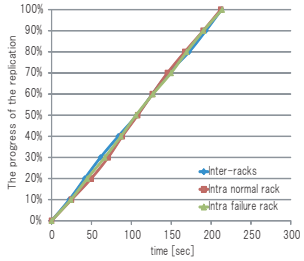

Figure 6. The progress of the replication with Default (The number of DataNodes:8*2)

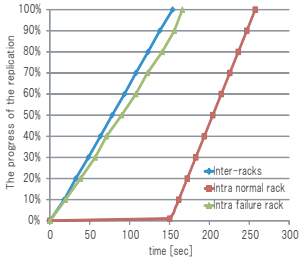

Figure 7. The progress of the replication with Prioritized (The number of DataNodes:8*2)

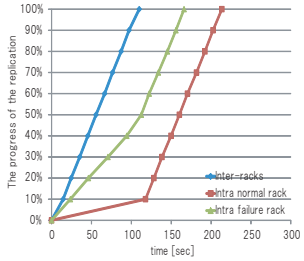

Figure 8. The progress of replication with Prioritized w/ intra-rack stream control (The number of DataNodes:8*2)

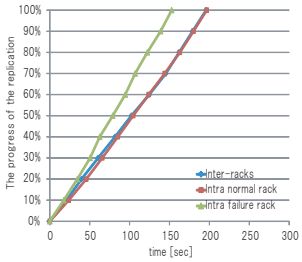

Figure 9. The progress of replication with Non-prioritized (The number of DataNodes:8*2)

2) The progress of replication of blocks which are involved in data transfer between racks: Figure 10 shows the progress of the replication of the blocks using each scheme which performed data transfer between racks during replica reconstruction upon node deletion. Figure 10 only shows the case with the number of DataNodes equals to 8*2. The cases with 16*2, 32*2 are omitted since they show the same tendencies. Figure 10, shows that data transfer between racks was completed very quickly in the cases of the 'Prioritized scheme' and the 'Prioritized scheme with streams within a rack'. Note that this is behavior intended to improve the fault tolerance. They have the disadvantage that the total execution time for replica reconstruction is longer, but when dealing with critical data, the behavior of these prioritized schemes is preferable. There is not much difference between **Default** and **Non-prioritized** with respect to the progress of the replication. This is because the replication of the blocks are processed in arbitrary order in both schemes.

3) The execution time of replica reconstruction with stream control between racks: Figures 11, 12 and 13 show the execution time of replica reconstruction using schemes with inter-rack stream control. The vertical axis represents the execution time of the replica reconstruction in seconds, and the horizontal axis represents the number of streams between racks.

Figures 11, 12 and 13 show that while the maximum

Table III
THE AVERAGE REPLICATION TIME PER BLOCK WITH EACH SCHEME (THE NUMBER OF DATANODES:8*2)

	Inter-racks	Intra normal rack	Intra failure rack
Default	5.65	3.11	6.68
Prioritized	6.08	3.95	5.91
Prioritized w/ intra	4.29	3.95	4.25
Non-prioritized	4.69	2.93	5.55

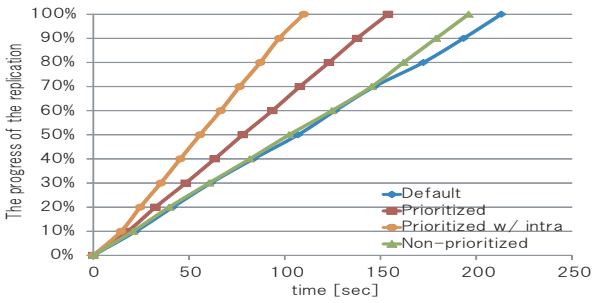

Figure 10. The progresses of replication of the inter-rack data transfer blocks (The number of DataNodes:8*2)

number of streams between racks is less than the number of DataNodes in the failure rack, the replica reconstruction time decreases along with the increase of the maximum number of streams between racks. All the schemes exhibit the same behavior in this area. This is because in this area, the data transfer between racks dominates the execution time, since there are not enough streams between nodes.

When the maximum number of streams between racks is equal to the number of DataNodes in the failure rack, i.e., each DataNode in the failure rack is receiving one block from the normal rack, the execution time of replica reconstruction is minimized. When the maximum number of streams is larger than the number of DataNodes, the execution time increases slightly as the maximum number of streams increases.

Note that the network bandwidth between racks is 1.25GB/sec, and hence nineteen streams can theoretically saturate the network between racks. However, Figure 13 shows that the optimum stream number is larger than nineteen. This means that the bottleneck here lies with the DataNodes in the failure rack rather than the data transfer between racks.

When the maximum number of streams between racks is larger, the execution time difference between **Non-prioritized w/ inter** and **Prioritized w/ inter** remains small. This is because it is rare that the actual number of streams between racks exceeds the number of DataNodes in the failure rack, since **Non-prioritized w/ inter** replicates each block in arbitrary order.

Table IV shows the reduction ratio of the execution

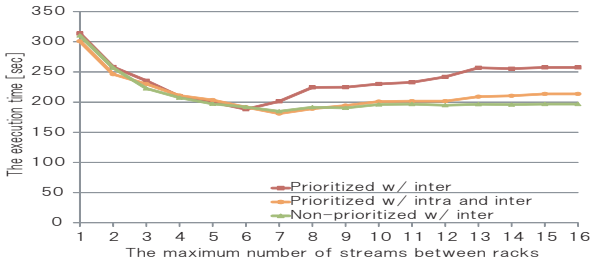

Figure 11. The execution time of replica reconstruction w/ inter-rack stream control (The number of DataNodes:8*2)

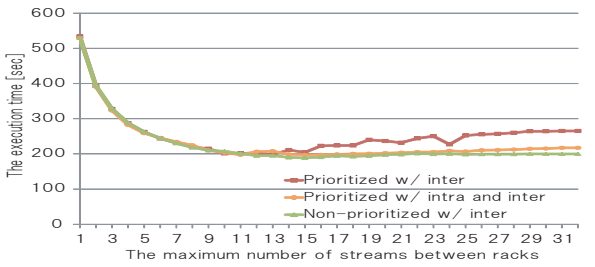

Figure 12. The execution time of replica reconstruction w/ inter-rack stream control (The number of DataNodes:16*2)

time compared to **Default** when each scheme exhibits the minimum execution time. While the reduction ratio without inter-rack stream control show a 10% reduction at most, the ratio with control show 16 % at most. We can conclude that proper inter-rack stream control is effective to minimize replication time.

V. Discussion

In this study, we have been discussing the assumption that HDFS clusters consist of two racks. In practice, HDFS may be utilized by clusters composed of three or more racks. When the remaining replicas of the missing blocks are in the same rack as shown in Figure 2, the block is replicated to a different rack according to the replica placement policy. When the number of racks is two, the destination rack is uniquely determined. On the other hand, when the number of racks is three or more, all racks, excluding the rack which includes the source DataNode, can be the destination rack. Therefore, we consider that inter-rack replication processes can be distributed over multiple racks and the bottleneck of the inter-rack network is reduced.

In addition, we have focused on balancing the load of each DataNode and tackled the challenge of finding effective replica reconstruction schemes. However, in an actual environment, the replica reconstruction process is performed in the background, so it is necessary to avoid the situation where this process occupies too much of the network bandwidth and reduces the performance of the foreground processes. Therefore, as future work, we are going to attempt to perform the replica reconstruction effectively using the

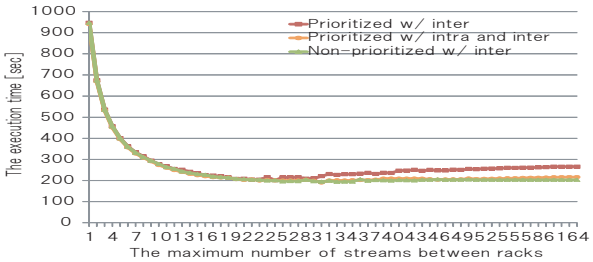

Figure 13. The execution time of replica reconstruction w/ inter-rack stream control (The number of DataNodes:32*2)

Table IV
The reduction rate of the execution time compared to the Default scheme

The number of DataNodes	8*2	16*2	32*2
Prioritized w/ intra	13.30%	10.29%	7.88%
Prioritized w/ intra and inter	16.64%	10.77%	13.01%
Non-prioritized w/ inter	14.89%	14.61%	11.29%

proposed schemes depending on the actual situation faced, while minimizing the influence on the foreground processes.

VI. Related work

A lot of replication strategies for data management have been proposed. In general, data are replicated and their replicas are stored in different data nodes for reliability and availability. And it is important to decide a suitable replication factor and replica placement.

Rahman, et al. [6], Wang, et al. [7], and Sato, et al. [8] proposed replication strategies based on file clustering for Grid file systems. In the clustering strategy described in [8], files are grouped according to each data processing task, based on the notion that the clustered files will be simultaneously used by another similar data processing task. 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. [9] 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 have to be placed and how many replicas have to be placed, based on network bandwidth and response time between regions. When a file is created, an access frequency number is calculated for each region and replicas are placed in the regions with a large access frequency number in a descending order. Furthermore, a site within the region the file has to be placed in 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. [10] proposed a replication strategy based on a dynamic file assignment according to access load. First,

they assign files, which are sorted by file size, to disks in a round-robin fashion so as to distribute the access load of all files evenly across all disks. Then, creation and deletion of replicas occurs according to the access load of all files and each disk. From the experiments, load balancing can be achieved in an environment where user access patterns change significantly.

Wenfeng, et al. [11] proposed a replication strategy based on the response time for each request. This strategy determines replica allocation and the number of replicas for each data block in order to satisfy the requirement due to the response time of acquiring each replica from every node, and to minimize the replica degree and the number of replicas for each data block, at the same time. From the experiments, the proposed strategy was able to meet with every node's response time requirements for a single request and also reduced the value for replica degree. Moreover, it reduced the maximum total request response time and improved the overall system performance.

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

On the other hand, Suzuki, et al. [12] proposed algorithms to perform wide-area simultaneous file replication between PC clusters effectively. They mentioned that it was important to prevent the network performance from degrading and to prevent disk I/O performance from being deteriorated due to the concentration of accesses in order to replicate the file data between PC clusters effectively. They formulated the file replication between PC clusters as a selection problem of the file transferred and a scheduling problem of transfer order by using graph theory. They proposed two algorithms to solve these problems: one is based on a linear programming approach and the other is based on a greedy approach. In addition, in the case where the process is concentrated on a certain node, they proposed a dynamic replica creation algorithm whereby another node in the source cluster to which the node belongs, sends replica files instead of the busy node. From their experiments, they showed that the proposed algorithm is effective in simultaneous multiple file replication between PC clusters. Although they take into account inter-cluster replica creation based on the load of each node, it is different from our replication schemes, which enable effective data replication in both intra-rack and inter-rack data transfer environments.

VII. Conclusion

We proposed effective replica reconstruction schemes for HDFS clusters configured using two racks and evaluated the proposed schemes by simulation. In replica reconstruction for multi-rack environments, we have to consider the loads of networks and DataNodes, and fault tolerance. Therefore, we extended our single-rack replica reconstruction schemes based on one-directional ring data transfer, and attempted to control streams of inter-rack data transfer and give a higher priority for blocks required for inter-rack data transfer. From the experiments, we showed that the proposed schemes are effective for the reduction of execution times and improvement of fault tolerance. We also confirmed that the performance shows further improvement due to the reduction of the load on the destination node by setting the number of streams between racks to the number of nodes which belong to the rack, including the destination node.

References

[1] Tom White., "Hadoop: The definitive guide," O'Reilly JAPAN, 2009.

[2] Dhruba Borthakur. "HDFS Architecture," The Apache Software Foundation, 2008.

[3] Asami Higai, Atsuko Takefusa, Hidemoto Nakada and Masato Oguchi. "A Study of Effective Replica Reconstruction Schemes at Node Deletion for HDFS," Proc. the 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid2014), pp.512–521, 2014.

[4] Sanjay Ghemawat, Howard Gobioff and Shun-Tak Leung. "The Google File System," Proc. 19th Symposium on Operating Systems Principles (SOSP'03), pp. 29–43, 2003.

[5] SimGrid. http://simgrid.gforge.inria.fr/

[6] Rashedur M.Rahman, Ken Barker and Reda Alhajj. "Study of Different Replica Placement and Maintenance Strategies in Data Grid,", Proc. the Seventh IEEE International Symposium on Cluster Computing and the Grid, pp.171–178, 2007.

[7] Y. Wang and D. Kaeli. "Load balancing using grid-based peer-to-peer parallel I/O," Proc. IEEE International Conference on Cluster Computing, pp. 1–10, 2005.

[8] Hitoshi Sato, Satoshi Matsuoka and Toshio Endo, "File Clustering Based Replication Algorithm in a Grid Environment," Proc. the 9th IEEE International Symposium on Computing and the Grid (CCGrid2009), pp. 204–211, 2009.

[9] K. Sashi and Antony Selvadoss Thanamani. "A New Replica Creation and Placement Algorithm for Data Grid Environment," Proc. International Conference on Data Storage and Data Engineering, pp. 265–269, 2010.

[10] J. Tjioe, R. Widjaja, A. Lee and T.Xie. "DORA: A Dynamic File Assignment Strategy with Replication," Proc. International Conference on Parallel Processing (ICPP'09), pp. 148–155, 2009.

[11] W.F. Wang and W.H. Wei, "A Dynamic Replica Placement Mechanism Based-on Response Time Measure," Proc. IEEE International Conf. on Communications and Mobile Computing, pp. 169–173, 2010.

[12] Katsunori Suzuki and Osamu Tatebe, "Replication Scheduling for a Set of Files between PC Clusters," IPSJ Transactions on Advanced Computing Systems, Vol.3, No.3, pp. 113–125, 2010.