NameNode and DataNode Coupling for a Power-proportional Hadoop Distributed File System

Hieu Hanh Le, Satoshi Hikida, and Haruo Yokota

Department of Computer Science, Tokyo Institute of Technology, Japan
{hanhlh,hikida}@de.cs.titech.ac.jp,yokota@cs.titech.ac.jp

Abstract. Current works on power-proportional distributed file systems have not considered the cost of updating data sets that were modified (updated or appended) in a low-power mode, where a subset of nodes were powered off. Effectively reflecting the updated data is vital in making a distributed file system, such as the Hadoop Distributed File System (HDFS), power proportional. This paper presents a novel architecture, a NameNode and DataNode Coupling Hadoop Distributed File System (NDCouplingHDFS), which effectively reflects the updated blocks when the system goes into a high-power mode. This is achieved by coupling the metadata management and data management at each node to efficiently localize the range of blocks maintained by the metadata. Experiments using actual machines show that NDCouplingHDFS is able to significantly reduce the execution time required to move updated blocks by 46% relative to the normal HDFS. Moreover, NDCouplingHDFS is capable of increasing the throughput of the system that is supporting MapReduce by applying an index in metadata management.

Keywords: power-proportionality, HDFS, metadata management

1 Introduction

Energy-aware commercial off-the-shelf (COTS)-based distributed file systems for cloud applications are increasingly moving toward power-proportional designs, as the configuration of the systems is changeable on demand. Specifically, the system is designed to operate in multiple gears and each gear contains a different number of active nodes. Multi-gear operation is made possible through a number of recent works that focus on power-proportional data placement layouts [1, 2]. However, those works have not yet dealt with the reflecting of an updated data set that is modified (or appended) in a low gear mode when several nodes are powered off. In low gear, the currently active nodes should update the modified data instead of the inactive nodes. When the system moves to a high gear, to share the load equally to all active nodes, it is necessary to let the reactivated nodes catch up with the modification of the data set.

In addition to normal operations, the process of reflecting the updated data set increases several costs of metadata management (MDM) and data transference inside the system. Carrying out this process effectively is vital in realizing power proportionality for a distributed file system, such as the Hadoop Distributed File System (HDFS) [3], which is already widely used as a distributed file system for effective big data processing in the cloud. In the current HDFS architecture, reflecting updated files is ineffectively restrained at the NameNode because of access congestion in the metadata information of blocks.

This paper presents a novel architecture called the NameNode and DataNode Coupling HDFS (NDCouplingHDFS), which is designed to effectively reflect updated data in the power-proportional HDFS. NDCouplingHDFS couples MDM and data management to localize the range of blocks maintained by the metadata. Through this idea, the process is effectively distributed to multiple nodes as the load is shared among the nodes and each node can focus on its own work because all the necessary information is located locally.

Moreover, to raise the efficiency of reflecting updated data, it is preferable to eliminate the bottleneck of MDM at the single NameNode in a normal HDFS by using distributed MDM. Taking the locality of the file system into consideration, we suggest two approaches of distributed MDM based on a tree structure, namely static directory partitioning and the B-tree-based index method. In the first approach, we divide the namespace of the system among all the nodes, as each node will maintain a subpart of the directory hierarchy. In the second approach, we apply the parallel index technique, called Fat-Btree [4], which is used in current database management to manage the metadata of the file system. Our main contributions are the following.

- NDCouplingHDFS is proposed to solve the problem of reflecting updated (or appended) data sets when the power-proportional file system shifts from low gear to a higher gear.
- NDCouplingHDFS improves the IO throughput of the metadata operation of the HDFS by implementing distributed MDM with an index technique.
- An empirical experiment to evaluate NDCouplingHDFS is performed on actual machines. The empirical experimental results show that NDCouplingHDFS is able to significantly reduce the execution time to transfer updated blocks by 46% relative to a normal HDFS.

The remainder of this paper is organized as follows. Related work is introduced in Sect. 2. Section 3 describes our proposed system with the architecture and data flow. Section 4 presents a performance evaluation of our proposals. Conclusions and future work are discussed in Sect. 5.

2 Related Work

RABBIT [1] is the first work that aims to provide power proportionality to an HDFS by focusing on read performance. RABBIT uses the equal-work data layout policy using data replication. However, RABBIT does not yet consider the cost of reflecting updated data in low gear. Kim et al. [2] suggest a fractional replication method to achieve a balance between the power consumption and performance of a system. Their work considers the problem of identifying a suitable time to gear down and save power.

Write Off-loading [5] is motivated by the goal of saving power through spinning down unnecessary disks. It allows write requests on spun-down disks to be

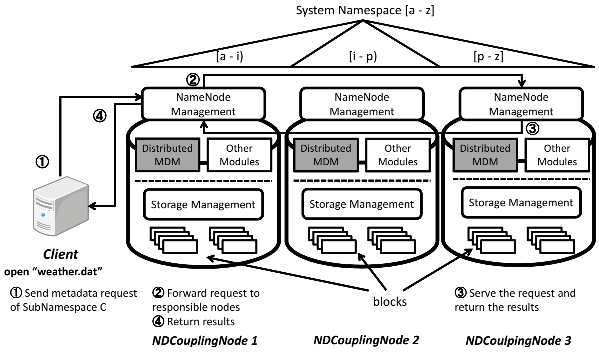


Fig. 1. A NameNode and DataNode Coupling HDFS architecture and data flow

temporarily redirected to other active disks in the file system. As a result, this technique lengthens the spin-down durations, thereby achieves additional power saving. Although not aiming to provide power proportionality, the idea could be considered as a solution for multigear file systems dealing with updated data when the system operates in low gear.

In previous work, we have taken into consideration the cost of updated data reflection relating to the size of moving data in a power-proportional HDFS [6]. As the size of moving data is small, the reflection process could be shortened.

3 NDCouplingHDFS

In this part, the assumptions employed in this paper is given. Then, the architecture of our system and two methods for distributed MDM are described. Finally, we present the system's behavior in reflecting updated data.

3.1 Assumptions and Conditions

In our proposal, we employed the following assumptions and conditions.

1. Data layout policy: The scope of this paper is limited to the MDM and the cost of reflecting updated data at power-proportional file systems. In low gear, the data from inactive nodes are replicated at other, active nodes.
2. Replication: When data are replicated at other nodes, their metadata are also replicated at the same node.
3. Failure: We suppose that all nodes in the system operate without failure.

3.2 Architecture and Data Flow of NDCouplingHDFS

The architecture and the data flow of NameNode and DataNode Coupling HDFS (NDCouplingHDFS) are shown in Figure 1. NDCouplingHDFS contains a cluster of NDCouplingNodes. There are two types of modules at each node in NDCouplingHDFS: the NameNode Management (NM) and the Storage Management (SM). The NM includes the new distributed MDM and other unmodified modules (such as Block Placement, Block Mapping) as in a normal HDFS. The important difference from a default HDFS is that the namespace of the file system is divided among all the nodes and the local distributed MDM only manages the metadata for files that are locally located. The SM at NDCouplingNode is the SM at DataNode in a normal HDFS.

Next, the data flow for the client interacting with NDCouplingHDFS is explained using Fig. 1. At first, the client randomly connects to a node to access the file system (open *weather.dat*). At this node, the request is forwarded to the corresponding node that contains the metadata of this file by distributed MDM. Then, the distributed MDM at this node looks for the file's metadata and sends the result back to the client. Finally, based on this result, the client opens connections to the responsible nodes to retrieve or store the file's blocks.

3.3 Distributed Metadata Management

In this part, we describes two approaches of employing distributed MDM to identify the responsible NDCouplingNode that contains the metadata for the accessed files.

Static Directory Partitioning Method. In this paper, we first try the static directory partitioning (SDP) method in distributing the namespace to multiple nodes in the system. Here, subparts of the directory hierarchy are manually assigned to individual nodes. All the nodes in the system have the mapping information about which node is responsible for what subpart of the file system directory. The system can process the request at most one hop to determine the appropriate nodes because the subparts of the hierarchy are treated as independent structures.

Fat-Btree-based Method. This method applies Fat-Btree to perform distributed MDM. Fat-Btree is an update-conscious parallel B-tree structure that was originally proposed in database management as an indexing technique for efficient data management [4, 7]. Because of the parallel tree structure, the distributed MDM based on Fat-Btree achieves higher performance for search query processing while maintaining good locality tracking of the file system.

Alternative Techniques. To realize good performance with distributed MDM, many recent systems distribute the metadata across multiple nodes utilizing distributed hash table [8, 9]. However, distributing metadata by hashing eliminates all hierarchical localities such as the POSIX directory access semantics.

3.4 Updated Data Reflection

Here, we describe the behavior of NDCouplingHDFS in serving the updated-data requests in low gear and reflecting the updated data when the system changes to high gear by reactivating a subset of nodes. In the normal HDFS, basically all the operations are similar however because there is only a single NameNode that is in charge of MDM, all the metadata operations are proccessed at the NameNode. Figure 2 shows an example of a four-node system in which each node maintains a subNamespace of the system. In low gear, Node 1 and Node 4 are inactive, and their maintenance data are consequently replicated at Node 2 and Node 3. During low gear, the part of the new updated data that is maintained by inactive nodes are reflected at predefined active nodes. Information about the data, the temporary node, and the intended node is saved into a Log file. In this example, Node 2 will update the data (here is $a1$) that should be updated by Node 1.

When the system changes to high gear by reactivating nodes (Node 1 and Node 4), the following four-step operations are carried out.

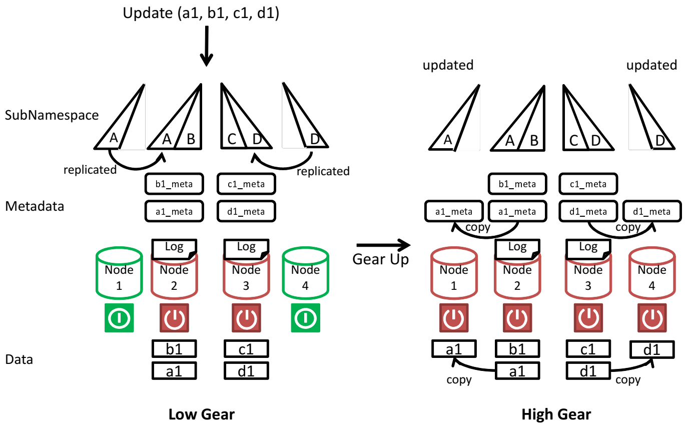


Fig. 2. Operations at updated data reflection processes of NDCouplingHDFS

Step 1: Transfer updated metadata. The active nodes check the Log files and transfers only the different metadata to the reactivated nodes.

Step 2: Issue block transfer commands. Next, the MDM searches for updated file blocks using the information in Log file. It then issues the block transfer command by filling the block transfer queue of each SM with the block and destination node paired information. After each constant heartbeat, the SM receives a command and transfers the blocks to the destination nodes. There are two considerable approaches for issuing a command. The **sequential issuance method** repeats the above search-and-issue operation for each transferred file, while the **batch issuance method** first looks for all the blocks and their destination nodes and then places them into a queue.

Step 3: Transfer updated blocks. When the SM receives the command issued by MDM, it sends the blocks to the destination nodes. However, in the current implementation in this part of the HDFS, for each block, the system has to open a new connection to the destination node. In order to reduce the cost of opening new network connection, we suggest the **batch transfer method** which sends all the blocks through just a single connection. The current implementation in the HDFS is called the **sequential transfer method**.

Step 4: Reflect updated metadata. The MDM updated the metadata for the newly arrived files as in the default HDFS based on the notifications from SM.

4 Experimental Evaluation

We carried out an empirical experiment with actual machines to verify the effectiveness of NDCouplingHDFS in terms of reducing the cost of updated-data reflection when the system shifts to higher gear. Next, we examined the effectiveness of distributed MDM relating to the scalability of metadata operations.

4.1 Updated-data Reflection

To verify the effectiveness of each contribution proposed in Sect. 3, we prepared five configurations which are formed from the combinations of distributed MDM, command issuance method and block transference method. Table 1 shows the characteristics of these configurations.

Table 1. Characteristics of the configurations used in updated-data reflection experiments

Configuration	**NormalHDFS**	**SSS**	**SBS**	**SBB**	**FBB**
Metadata management	Centralized	SDP	SDP	SDP	Fat-Btree
Command issuance	Sequential	Sequential	Batch	Batch	Batch
Block transference	Sequential	Sequential	Sequential	Batch	Batch
Updated metadata transference	-	○	○	○	○

Table 2. Experimental environment

# Gears	2
# nodes Low Gear	8
# nodes High Gear	16
# updated files	16000
file size	1MB

Table 3. Specification of a node

CPU	TM8600 1.0GHz
Memory	DRAM 4GB
NIC	1000 Mb/s
OS	Linux 3.0 64bit
Java	JDK-1.7.0

Table 4. HDFS information and parameters

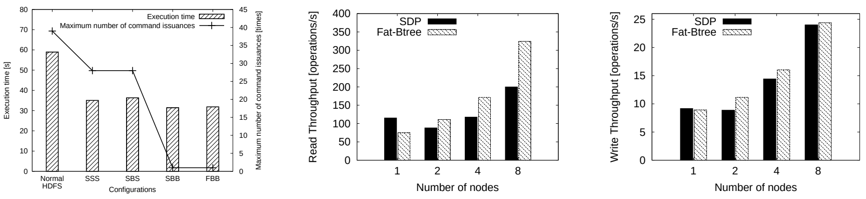
version	0.20.2
max.rep-stream	100
heartbeat_interval	1

Experimental Environment. We compare the proposed NDCouplingHDFS with the normal HDFS by changing the configuration of the system (Tab. 2). Both systems operate in two gears, a Low Gear and a High Gear with different number of active nodes (eight and 16 nodes). For **NormalHDFS**, there is one further node to be in charge of the NameNode. Because we address MDM in this paper, the number of appended files when the system operates at Low Gear is fixed at 16000 dividing equally to 16 nodes. Here, we use low-power-consuming ASUS Eeebox EB1007 machines, whose specifications are given in Tab. 3. The *max.rep-stream*, which specifies the maximum number of blocks that can be replicated by a SM at the same time, is set to 100. To efficiently perform the updated data reflection, the communication frequency between NM and SMs is maximized by setting *heartbeat_interval* to one (Tab. 4).

Experimental Results. Figure 3(a) shows the execution time for reflecting the updated data with different configurations. The left vertical axis shows the execution time from the time that the system begins to change from low gear to high gear until all the just-activated nodes catch up with the most current status of the updated data set. The right vertical axis shows the maximum number of transfer block command issuances, which is the number of times that the SM has to make a connection with the MDM to drain the block transfer queue.

Performance of NDCouplingHDFS. To confirm the NDCouplingHDFS's performance, we focus on the experimental results of **NormalHDFS** and **SSS**, the simplest configuration of NDCouplingHDFS, in Fig. 3(a). We see that NDCouplingHDFS has significantly reduced cost (nearly 41%) in reflecting updated data. In the HDFS, because of the high load at the NameNode with the processing of 8000 files that should be replicated to eight nodes, it requires about 40 connections between the NM and SM to drain the block transfer queue of the SM (about 58 seconds). Meanwhile, the process is distributed to eight nodes in NDCouplingHDFS, hence overall is completed in only about 34 seconds.

Performance of the command issuance. From the results of **SSS** and **SBS**, we see that the batch command issuance provided a slightly worse result than did sequential command issuance. The reason is that the SMs in **SBS** wasted several first connections to the NM before it had finished retrieving all 1000



(a) Updated data reflection

(b) Read throughput

(c) Write throughput

Fig. 3. Experiment results

updated files' data. On the other hand, the SM in **SSS** can perform the block replication process immediately from the very first communication.

Performance of the block transfer method. Figure 3(a) shows that **SBB** reduces the execution time of the process to 31 seconds compared with **SBS**. This means that batch block transfer was able to reduce the cost of opening a new network connection for sending blocks. In total, SDP-based NDCouplingHDFS was able to reduce the execution time required for reflecting the updated data by 46% relative to **NormalHDFS**.

Fat-Btree-based method. There was little difference between the performance of **FBB** and **SBB**. The cost of the latter is slightly less by 0.5 seconds owing to the lower cost of MDM operations. This is due to the process of transferring incremental metadata, as the Fat-Btree-based method has to transfer more information than SDP because of the complex structure.

4.2 Distributed MDM Performance

In this part, we report the performance evaluation relating to the scalability of metadata operations to confirm the effect of SDP and Fat-Btree-based methods. The configurations of this experiment are shown in Tab. 5.

Experimental Results. Figure 3(b) and 3(c) show the read and write throughput of two evaluated methods. Here, the operation includes searching/creating for the metadata and reading/writing the physical data of the query file. Figure 3(b) shows that the read performance of the Fat-Btree method significantly scales out. The good balance of the parallel B-tree structure means that the read requests are effectively distributed to all the nodes; hence, the overall throughput increased as the number of nodes increased. In contrast, in the SDP method, the throughput slightly decreased as the number of nodes increased from one to two. The reason is that the cost of opening a new connection to other responsible

Table 5. Workload used in distributed MDM performance evaluation experiment

Fat-Btree leaf fanout	16
Data size ($\#files$)	3000
Number of nodes	1, 2, 4, 8
File size	1KB
$\#write\ accesses\ per\ node$	$\frac{\#files}{\#nodes}$
$\#read\ accesses\ per\ node$	$\#files$

nodes is much larger than the cost of searching for the responsible metadata. From Fig. 3(c) which describes the overall throughput for write requests, the Fat-Btree method is seen not to provide such a considerable efficiency compare with the SDP method because of the high synchronization cost inside the B-tree structures during an update. Overall, the Fat-Btree is believed more suitable for the read-mostly workloads in MapReduce applications.

5 Conclusion and Future Work

In this paper, we first described the problem of inefficient reflection of updated data in power-proportional distributed file system and then proposed the NDCouplingHDFS architecture, which couples metadata management and data management at each node to solve it. Empirical experiments verified that our solution was able to shorten the execution time required to reflect updated data by 46% relative to the time required by the default HDFS. Moreover, NDCouplingHDFS was able to increase the throughput of the system supporting MapReduce by applying an index in metadata management. In the future, we would like to carry out more experiments with different workloads and a larger scale of nodes. Moreover, we would like to develop a system that integrates NDCouplingHDFS with suitable data placement to provide power proportionality.

Acknowledgements

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

References

1. Hrishikesh, A., James, C., Varun, G., Gregory R., G., Michael A., K., Karsten, S.: Robust and Flexible Power-proportional Storage. In: Proc. the 1st ACM Symposium on Cloud Computing. SoCC '10 (2010) 217–228
2. Kim, J., Rotem, D.: Energy Proportionality for Disk Storage using Replication. In: Proc. the 14th Int'l Conference on Extending Database Technology. (2011) 81–92
3. Apache Hadoop: HDFS Hadoop Wiki. http://wiki.apache.org/hadoop/HDFS
4. Yokota, H., Kanemasa, Y., Miyazaki, J.: Fat-Btree: An Update Conscious Parallel Directory Structure. In: Proc. the 15th Int'l Conference on Data Engineering, IEEE Computer Society (1999) 448–457
5. Narayanan, D., Donnelly, A., Rowstron, A.: Write Off-loading: Practical Power Management for Enterprise Storage. In: Proc. 6th USENIX Conference on File and Storage Technologies. (2008) 253–267
6. Le, H.H., Hikida, S., Yokota, H.: An Evaluation of Power-proportional Data Placement for Hadoop Distributed File Systems. In: Proc. Cloud and Green Computing, IEEE Computer Society (2011) 752–759
7. Yoshihara, T., Kobayashi, D., Yokota, H.: A Concurrency Control Protocol for Parallel B-tree Structures Without Latch-coupling for Explosively Growing Digital Content. In: Proc. the 11th Int'l Conference on Extending Database Technology: Advances in Database Technology, ACM (2008) 133–144
8. Rodeh, O., Teperman, A.: zFS-a Scalable Distributed File System using Object Disks. In: Proc. 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST 2003), IEEE (2003) 207–218
9. Braam, P.: The Lustre Storage Architecture