論文 / 著書情報
Article / Book Information

Title	An Efficient Gear-shifting Power-proportional Distributed File System
Author	Hieu Hanh LE, Satoshi HIKIDA, Haruo YOKOTA
Journal/Book name	Proceeding of the 26th International Conference on Database and Expert Systems Applications (DEXA 2015), , , pp. 153-161
発行日 / Issue date	2015, 9
DOI	10.1007/978-3-319-22852-5_14
権利情報 / Copyright	The original publication is available at www.springerlink.com.
Note	このファイルは著者（最終）版です。 This file is author (final) version.

An Efficient Gear-shifting Power-proportional Distributed File System

Hieu Hanh Le[1], Satoshi Hikida[2], and Haruo Yokota[2]

[1] Center for Technology Innovation, R&D Group, Hitachi Ltd., Japan*
[2] Department of Computer Science, Tokyo Institute of Technology, Japan
{hanhlh,hikida}@de.cs.titech.ac.jp,yokota@cs.titech.ac.jp

Abstract. Recently, power-aware distributed file systems for efficient big data processing have increasingly moved toward power proportional designs. However, inefficient gear-shifting in such systems is an important issue that can seriously degrade their performance. To address this issue, we propose and evaluate an efficient gear-shifting power proportional distributed file system. The proposed system utilizes flexible data placement that reduces the amount of reflected data and has an architecture that improves the metadata management to achieve high-efficiency gear-shifting. Extensive empirical experiments using actual machines based on the HDFS demonstrated that the proposed system gains up to 22% better throughput-per-watt performance. Moreover, a suitable metadata management setting corresponding to the amount of data updated while in low gear is found from the experimental results.

1 Introduction

Commercial off-the-shelf-based distributed file systems (DFS) have been widely used for cloud applications for their fast deployment and easy scaling. Among these systems, power-aware DFS have increasingly moved toward power proportional designs [1]. To realize such systems, current data placement methods commonly divides the nodes into a set of small and separated groups [2–4]. These groups are then configured to operate in multiple "gears" where each gear contains a different number of groups, and offers a different level of parallelism and aggregate I/O throughput [2].

However, the current methods do not fully consider the effects of the reflection of updated data during gear-shifting on the performance. For example, in the morning, the system may have to update the datasets modified in a low gear while a subset of the nodes was powered off overnight. When the system moves to a higher gear to gain a better performance by reactivating inactive nodes, it must replicate the updated data to the reactivated nodes to share the load among all the active nodes for better performance. Inefficient reflection of updated data with large amounts of retransferred data is believed to degrade the performance of such power proportional systems greatly during gear-shifting.

* This work was done when the author was at Tokyo Institute of Technology

Moreover, metadata management in the DFS is believed to play an important role during gear-shifting because the metadata management will be more complex. In the low gear, the system generally creates log records specifying the locations of updated data. When changing to a higher gear, it must identify the replicated data from the log records, access their metadata, transfer the data to the appropriate nodes, and update the corresponding metadata for later references. Carrying out this process effectively with efficient distributed metadata management is vital in realizing power proportionality DFS.

To provide efficient gear-shifting for power proportional DFS, an integration of distributed metadata management and data placement is further important because they are so closely related to each other. By leveraging both actions, the carefully designed integration will greatly increase the efficiency of gear-shifting with less throughput performance degradation.

In this paper, we propose a novel DFS that efficiently combines both of our previous works, Accordion [5, 7] and NDCouplingHDFS [6], to provide high throughput performance during gear-shifting. Although the amount of retransferred data is reduced in Accordion, efficient metadata management is required for better power proportional throughput performance. In the proposed system, this is achieved with support from NDCouplingHDFS, which distributes the metadata management cost efficiently to multiple nodes with small overhead.

The contributions of this paper are as follows.

- We propose a DFS for efficient gear-shifting to maintain high power proportional results during gear-shifting.
- We evaluate the effectiveness of the proposed system through empirical experiments. The experiments show that the proposed system gained up to 22% better power proportional performance than the base system configured with Accordion and the default HDFS.
- It is observed that the proposed system gains better performance for large amounts of updated data under a heavy metadata load; and for small amounts of updated data under a light metadata load.

The remainder of this paper is organized as follows. The related work is reviewed in Sections 2. The proposed system is described and evaluated in 3 and Section 4. The conclusions of this paper are discussed in Section 5.

2 Related work

Rabbit [4] was the first method to provide power proportionality to an Hadoop Distributed File System (HDFS) by focusing on the read performance by utilizing an equal-work data layout policy based on data replication on organized nodes. Sierra [3] also organizes the replicas of the dataset such that each replica is stored in a group of nodes. However, Sierra differs from Rabbit in that each replica of the dataset is evenly distributed to all the nodes in each group.

We also proposed Accordion [5], a flexible data placement method based on data replication to reduce the amount of retransferred data during gear-shifting by differentially considering the locations of primary data. As the primary data are located at all nodes, when the modified dataset is updated (or appended)

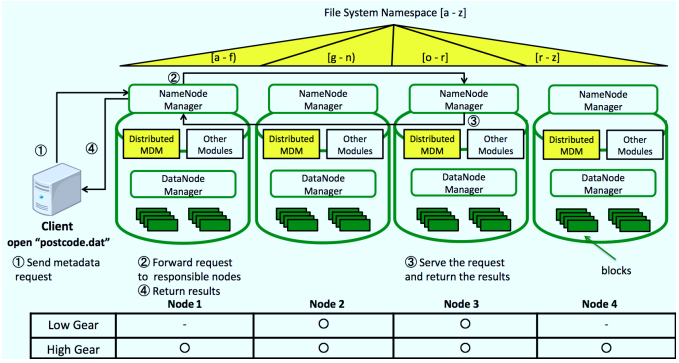


Fig. 1. The NDCoupling HDFS architecture and data flow.

in low gear, part of the primary data in the updated dataset is already stored on the active nodes. Hence, only the remainder of the updated dataset, which should have been written to the deactivated nodes, must be retransferred when the system shifts to a higher gear. Although Accordion improves the power proportional performance by 30% compared with Rabbit and Sierra [7], the metadata management in this study is still constrained by the single metadata node in the system.

We also previously presented an architecture known as NDCouplingHDFS [6] to facilitate the efficient reflection of updated data in a power proportional HDFS. NDCouplingHDFS focuses on coupled metadata management and data management on each HDFS node, which reduces the cost of managing the metadata generated during changes in the system configuration. However, the effect of NDCouplingHDFS on the throughput performance during gear-shifting was not considered in the earlier study.

3 System description

We confidently expect that utilizing distributed metadata management, NDCouplingHDFS can improve further the effectiveness of Accordion, because of the very close relationship between metadata management and data placement. In this section, we describes the NDCouplingHDFS architecture, the Accordion data placement then presents the updated data reflection process.

3.1 NDCouplingHDFS architecture

In this paper, because we focus on the locality of metadata management for improving the efficiency of reflecting the updated data in gear-shifting, we have applied equivalent coupling as each node contains both NameNode Manager and DataNode Manager. In the NDCouplingHDFS, NameNode Manager includes the distributed metadata management (Distributed MDM) and other modules such as Data Placement and Block Mapping, as at the NameNode in a default HDFS. The difference from a default HDFS is that the namespace of the file system is divided among all the nodes in the cluster while taking locality into consideration. The local Distributed MDM and the Block Mapping only manage the

metadata for local files and blocks. In this system, we utilize the Fat-Btree [8] method, which is an update-conscious parallel B-Tree structure to maintain the metadata of the system whose efficiency was verified in [6]. The DataNode Manager module at each node is the DataNode Manager at DataNode in the default HDFS. Figure 1 shows an example of the architecture and the data flow of ND-CouplingHDFS in the four-node system. This system operates in two gears; the Low Gear requires two active nodes Node 2 and Node 3 and the High Gear requires all four active nodes.

3.2 Accordion data placement

Accordion [7] is designed to provide power proportionality in distributed file systems that use commodity computer servers such as the HDFS or the Google File System. In Accordion, the files are divided into a large number of blocks and a number of replicas of each data block are distributed among the nodes of the cluster. Like other approaches, Accordion aims to control the power consumption of the system by dividing the nodes into several separate groups. An Accordion-based system can then operate in a multiple-gear mode where higher gears have more groups of nodes. In Accordion, the nodes are arranged geometrically in a horizontal array because the nodes that belong to lower groups are bounded by the nodes of higher groups.

At first, the primary data in the dataset are distributed to all the nodes in the system. This means that each node stores the same amount of primary data. Then, starting with the highest group, the data stored in this group are replicated to the next lower group. To guarantee the data reliability in the lowest gear, the chained declustering policy is applied to the smallest group. Each node replicates its data to its neighbor node, which guarantees that all of the data in the dataset are replicated in the two neighbor nodes. In the example in Figure 1, all the data from Node 1 and Node 4 are replicated to Node 2 and Node 3 accordingly. Then, the data of Node 2 are replicated to Node 3 and vice versa.

3.3 Gear controller

For easy implementation, there is one master Gear controller at a node, which is assumed to be always active and is responsible for any request related to controlling the gear of the system from the administrator such as down gear or up gear. Here, the master Gear controller will communicate with other Gear controllers to fulfill requests. Other approaches such as allowing any Gear controller among the nodes of the lowest group to be the master Gear controller are possible.

3.4 Updated data reflection process

In this section, we refer to Figure 2 and describe the behavior of the proposed system in serving data update requests in low gear and reflecting the updated data when the system changes to a higher gear by reactivating a subset of nodes. In the default HDFS, basically all the operations are similar; however, because there is only a single NameNode that is in charge of metadata management, all the metadata operations are processed at the NameNode.

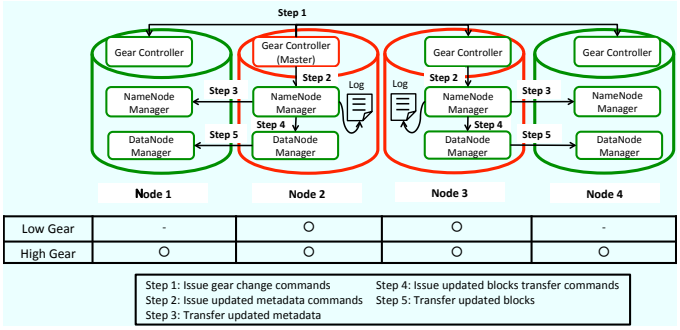


Fig. 2. Flow of the updated data reflection process.

Step 1: Issue a gear change command When receiving the gear-shifting command from the administrator, the Gear controller at the master node (Node 2) sends the command to the Gear controllers at all other active nodes.

Step 2: Issue update metadata commands After receiving the commands, active nodes will respond according to their roles. Nonoffload nodes that are not affected by the gear-shifting simply delay I/O requests from the clients. Offload nodes (node Node 2 and Node 3) that store the updated data issue the update metadata command to the Metadata Management.

Step 3: Transfer updated metadata The Metadata Management modules that receive the command check the log files and transfer only the changed metadata to the intended nodes specified in the log files. When the updated metadata transfers have finished, both the offload nodes and the intended nodes are ready to process the I/O requests from the clients, including any requests queued during the data reflection process, and send a "finished" indication to the Gear controller at the master node. After gathering all the finished indications from offload nodes, the Gear controllers forward this indication to all the nonoffload nodes. When the nonoffload nodes receive the indication, they are ready to process I/O requests from clients. Concurrently, in the background, the updated data reflection process continues with Step 4.

Step 4: Issue block transfer commands Next, the Metadata Management searches the log records for updated file blocks and issues block transfer commands with pairs of blocks and intended node identifiers to the local DataNode Manager. After each **heartbeat_interval**, the DataNode Manager receives a command and transfers the blocks to the intended nodes.

Step 5: Transfer updated blocks When the DataNode Manager receives the command issued by the Metadata Management, for better efficiency it sends the blocks to the intended nodes in a batch manner, which is called **batch transfer method**. When the DataNode Manager knows all the blocks it must transfer, the cost of opening a new network connection can be reduced by sending all the relevant blocks through a single network connection. The current implementation of the HDFS requires opening a new connection for each block.

Table 1. Node specification

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

Table 2. Sizes of datasets [MB]

Configuration	Without Update	Small	Medium	Large
Updated dataset	0	4480	8960	13440
Initial dataset	26880	22400	17920	13440

After receiving the updated data, the DataNode Managers at reactivated nodes (Node 1 and Node 4) notify the newly arrived data information to the responsible Metadata Management as in the default HDFS.

4 Experimental evaluation

We conducted an empirical experiment using actual machines to verify the efficiency of the proposed system described in Section 3 during gear-shifting. For the evaluation, we chose a system that deploys Accordion with the default HDFS architecture as the base system.

4.1 Experimental method

The workloads generated were close to the actual operation of multiple-gear DFS like the HDFS. We assumed that initially the file system was operated in a High gear and stored an initial dataset. Then, the system shifted to a Low gear for a specified power proportional service agreement. During this period, this dataset was updated as new files were appended from the clients. Here, the dataset that contains all these new files is called the update dataset. Next, the system was shifted to the High gear to satisfy the higher throughput performance on reading the whole dataset from the clients. At this time, the system must serve read requests from the client while performing updated data reflection in the background. As we focused on the applications on the DFS like the HDFS, we chose the method of updating the dataset as appending new files to the dataset and the method for reading the dataset as scanning all the files in the dataset. The sizes of the reading dataset, which includes both the initial and the update dataset, is fixed to 26880 [MB]. The sizes of the initial and the updated dataset, which are used in the evaluation, are varied as in Table 2.

4.2 Framework of the experiments

Our test-bed for the experiments comprised dozens of commodity nodes based on the HDFS. We were focused on energy-aware commodity systems so we used low power consumption ASUS Eeebox EB1007 machines, the specifications for which are provided in Table 1. In the base system, there is one NameNode besides the cluster of DataNodes in each gear (2, 8 and 20 nodes). However, in the proposed system, the numbers of nodes in each gear are limited to 2, 8 and 20.

4.3 Experimental results

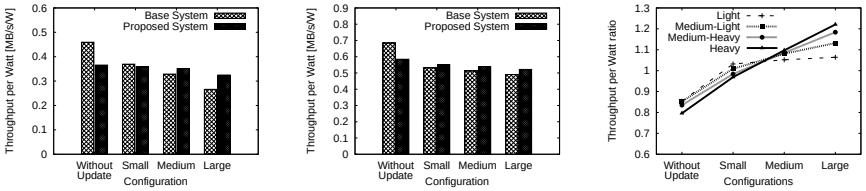
In this section, experimental results are reported for four cases relating to the load of the metadata, in which the size of the files is set to 64 MB, 16 MB, 4 MB, and 1 MB. In this experiment, because of the same data placements in both

Table 3. Settings

Case	File size [MB]	#blocks
Light	64	420
Medium–Light	16	1680
Medium–Heavy	4	6720
Heavy	1	26880

Table 4. Numbers of updated blocks

Case	Without Update	Small	Medium	Large
Light	0	42	84	126
Medium–Light	0	168	336	504
Medium–Heavy	0	672	1344	2016
Heavy	0	2688	5376	8064



(a) Average throughput-per-watt results (Heavy case)

(b) Average throughput-per-watt results (Low case)

(c) Throughput-per-watt ratio of the proposed system and the base system

Fig. 3. Experiment results

the base system and the proposed system, the sizes of the reflected data are the same. However, because of the file size, the number of blocks varies between the systems, hence the cost for metadata management changes. Table 3 describes the file size and the number of blocks for the four cases Light, Medium–Light, Medium–Heavy, and Heavy.

Figures 3(a) shows the experimental results of the average throughput-per-watt when the system changes from Gear 2 to Gear 3 with scanning the dataset in four configurations: Without Update, Small, Medium, and Large configurations. Note that in the Without Update configuration, the performance of the scanning dataset workload was not affected by the update data reflection process. The effectiveness of NDCouplingHDFS is confirmed as the throughput-per-watt performance of the proposed system was better than the base system in the Medium and Large configurations, by approximately 10% and 22%, respectively. This is explained by the advantages of the coupling architecture in NDCouplingHDFS employed in the proposed system compared with the normal HDFS in the base system. However, we can also see from Figure 3(a) that NDCouplingHDFS was not effective in Without Update and Small configuration. We suggest that the default HDFS showed better results in such situations because the cost of reflecting updated data is small. Table 4 shows the number of updated blocks of four cases in all configurations.

Figure 3(b) show the results for the average throughput-per-watt results in Light case. In contrast with the Heavy case, the effectiveness of the coupling architecture NDCouplingHDFS in the proposed system was difficult to observe as the throughput-per-watt performance of the proposed system was at most 6%

better than the performance of the base system (Large configuration). The main reason was the light load in metadata management as the numbers of updated blocks in the Light cases were extremely small showing in Table 4.

The effect of the metadata load on the effectiveness of the proposed system is evaluated by comparing the throughput-per-watt results of the proposed system with the base system for our four cases (Light, Medium–Light, Medium–Heavy, Heavy). Figure 3(c) presents the throughput-per-watt results of the proposed system divided by those for the base system. We observe that the effect of the metadata load (number of blocks) depends on the amount of updated data. In the Small configuration, the smaller number of blocks is better for the proposed system as the Light case gave the best result. However, in the Medium and Large configurations in Heavy case where the amount of updated data is greater, the heavier metadata load cases delivered the better result.

5 Conclusion and future work

We have demonstrated that the distributed metadata management in NDCouplingHDFS is effective for smooth gear-shifting in systems applying the Accordion data placement. Our experiments showed that the proposed system integrating Accordion and NDCouplingHDFS could achieve up to 22% better power proportionality than the base system configured with Accordion and the default HDFS. The efficiency of the proposed system is expected to be increasing when the metadata load is higher and the amount of updated data is larger. We would like to further evaluate the proposed system with other data placement methods.

References

1. André, B.L., Urs, H.: The Case for Energy-proportional Computing. Computer 40, 33–37 (2007)
2. Charles, W., Mathew, O., Jin, Q., Andy, W.A.I., Peter, R., Geoff, K.: PARAID: A Gear-Shifting Power-Aware RAID. Transaction on Storage 3(3), 13:1–13:33 (2007)
3. Eno, T., Austin, D., Dushyanth, N.: Sierra: Practical Power-proportionality for Data Center Storage. In: Proc. 6th European Conference on Computer Systems. pp. 169–182. EuroSys '11, ACM (2011)
4. Hrishikesh, A., James, C., Varun, G., Gregory R., G., Michael A., K., Karsten, S.: Robust and Flexible Power-Proportional Storage. In: Proceeding of the 1st ACM Symposium on Cloud Computing. pp. 217–228. SoCC '10 (2010)
5. Le, H.H., Hikida, S., Yokota, H.: Efficient Gear-shifting for a Power-proportional Distributed Data-placement Method. In: Proc. 2013 IEEE International Conference on Big Data. pp. 76–84. IEEE (2013)
6. Le, H.H., Hikida, S., Yokota, H.: NDCouplingHDFS: A Coupling Architecture for a Power-proportional Hadoop Distributed File System. IEICE Transactions on Information and Systems E97-D(2), 213–222 (2014)
7. Le, H.H., Hikida, S., Yokota, H.: Accordion: An Efficient Gear-shifting for a Power-proportional Distributed Data-placement Method. IEICE Transactions on Information and Systems E98-D(5), 1013–1026 (2015)
8. Yokota, H., Kanemasa, Y., Miyazaki, J.: Fat-Btree: An Update Conscious Parallel Directory Structure. In: Proc. the 15th International Conference on Data Engineering. pp. 448–457. ICDE '99, IEEE Computer Society (1999)