DFS on a Diet: Enabling Reduction Schemes on Distributed File Systems

Ryan Nathanael Soenjoto Widodo†, Hirotake Abe*, Kazuhiko Kato

Department of Computer Science, University of Tsukuba, Tsukuba, Japan
ryannsw(at)osss.cs.tsukuba.ac.jp, {habe, kato}(at)cs.tsukuba.ac.jp

I. Background and motivation

Most distributed file systems do not directly support data reduction schemes like compression or deduplication, which are crucial for reduced data footprints and transferring big data. For example, Lustre [1] and Hadoop DFS [2] by default does not support any reduction schemes, thus the schemes must be applied at the application layer or file system layer as shown in Figure 1 with their inherent disadvantages. At the application layer, the use of reduction schemes is optional, thus some applications may not utilize the available schemes. At the file system layer, those schemes can be applied to all applications by enabling reduction schemes at the underlying file system. For example, Lustre and HDFS can use ZFS as the base file system and use ZFS's compression or deduplication on the DFS's data. However, the data at DFS layer still consume the original size of the data because the reduction only occurs at the file system layer, thus any DFS operation will require more bandwidth to move the dataset than application layer applied schemes.

Additionally, both approaches usually have limited selection of reduction schemes supported by the underlying platforms. For example, in the case of HDFS, it relies on the available compression codecs of Hadoop. If the source code of the platform is available, it might be possible to add user-favorite reduction schemes, but it requires an expensive implementation cost or is virtually impossible. In such cases, adding a domain-based algorithms like Logzip [3], which perform well on log type datasets, can be challenging and expensive in both time and development cost.

In this study, we propose a system design that works with all applications and simplifies reduction schemes implementation in DFSs, which usually only capable of using reduction schemes when combined with client, application, or the underlying file system side reduction schemes [4], [1]. The design moves the reduction schemes from the application-level to the file system-level without relying on the underlying file system, thus it is beneficial to all applications without any configuration on the application side. Additionally, the design allows generic approaches that are not tied to a specific platform for the reduction scheme implementation.

II. Implementation

As a proof of concept, we created a framework that enables reduction schemes to work at a file system-level by directly providing HDFS blocks to the reduction scheme named Hadoop data reduction framework (HDRF) [5] through HDFS source code modification. HDRF handles all communication between HDFS and a reduction scheme and allows the user to implement their data reduction scheme in Java or C++ with minimal knowledge on Hadoop Libraries. As illustrated in Figure 2, HDRF operates at DFS level, thus all applications can benefit from the reduction schemes without using a file system that supports the schemes.

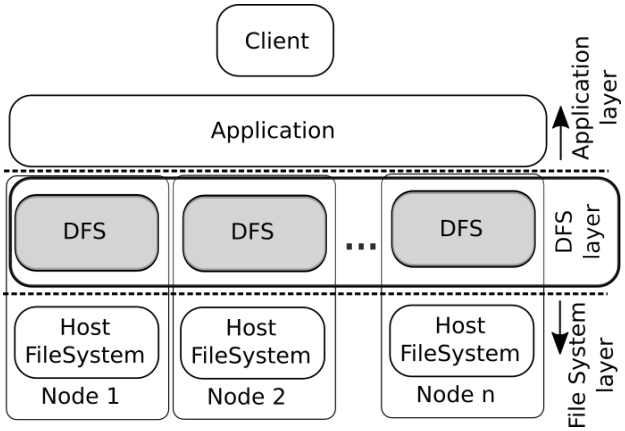

Fig. 1: Structure of a common distributed file systems

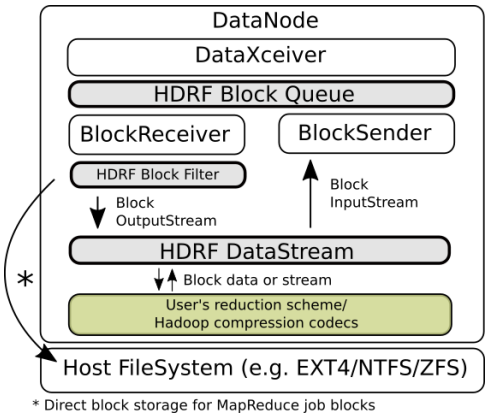

Fig. 2: The design on HDRF. The grayed parts are HDRF's components.

HDRF features three optimizations to improve its compati-

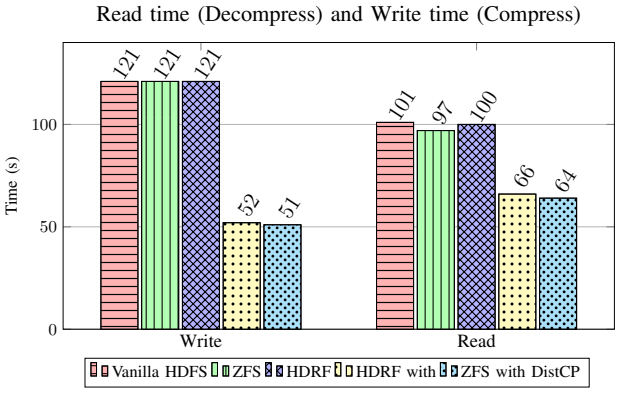

Fig. 3: The time required to write or read the dataset to the tested systems, lower is better.

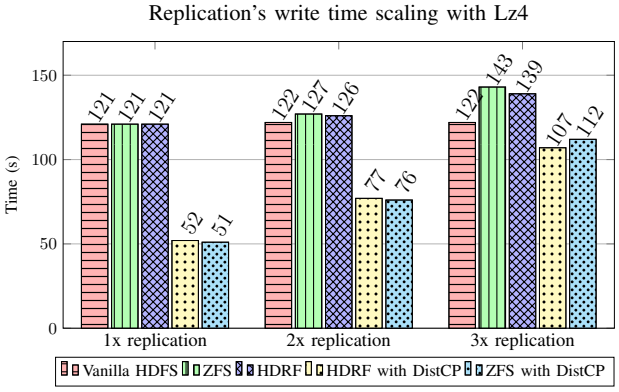

Fig. 4: Replication impact on the DFS write time, lower is better.

bility with reduction schemes and reduce its runtime overhead. First, HDRF has a block queue to accommodate reduction schemes that only operate in a single stream. Second, it can filter HDFS blocks, which can be used to exclude blocks that are proven as non redundant like MapReduce job blocks from being processed by reduction schemes, thus decreasing the runtime overhead by only working on the data blocks. Third, HDRF has block mirroring that transfers reduced blocks instead of the original HDFS blocks with its full size to reduce the network traffic. Currently, we have implemented the block queue and filter. The block mirroring is still a work in progress.

III. Experimental results, conclusion, and future work

To show the performance overhead of HDRF, we ran experiments that use read and write operation on HDFS and compare HDRF with vanilla HDFS and ZFS. The test cluster is composed of 6 data nodes and a single name node, which all equipped with Xeon E3-1220L, 8 GB DDR3, 240 GB SSD drives, and 1 Gbit ethernet. The dataset is the Wikipedia dump [6]. For the testing, we used Hadoop streaming and Distributed Copy (DistCP) to compare the tested systems.

As shown in Figure 3 and 4 Our experimental results shows that HDRF has less than 3% runtime and 1% storage over-

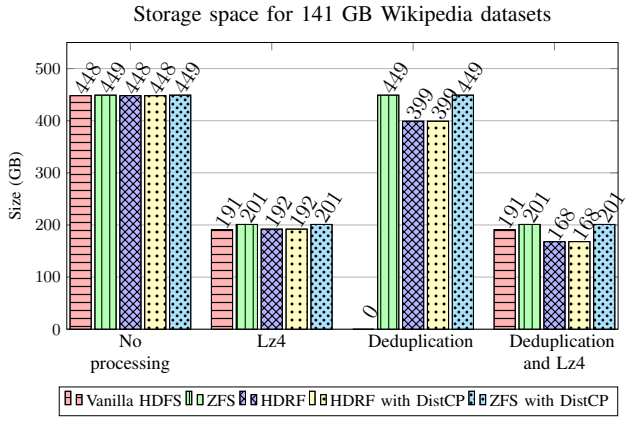

Fig. 5: Storage space usage for each tested systems, lower is better

heads. Because HDRF works with all Hadoop applications, HDRF can speed up the data transfer time by up to two times with the same compression algorithm without compromising the storage consumption by using DistCP, which is a more efficient application that cannot use compression codecs. Similar to HDRF, DistCP can also accelerate ZFS data transfer without affecting compression. However, vanilla HDFS cannot use DistCP with compression codecs.

HDRF performed 14% slower for data write jobs with 3x replication similar to ZFS because of network limitation during replication as illustrated in Figure 5. The block replication of both HDRF and ZFS occurs on full-sized blocks, unlike compression codecs, which reduces the block sizes at the application layer. In such cases, these systems rely on the network bandwidth more than vanilla HDFS with compression codecs. We expect block mirroring to solve the high network usage during replication by sending the reduced block instead of the original block.

For further improvement of the proposed method, we plan to implement the HDRF block mirroring to reduce the network overhead during replication or block transfer between nodes. Other than reduction schemes, HDRF can also support other methods of block storage which is not supported by HDFS. For example, HDRF can improve HDFS performances by storing its blocks in a multi-tier storage system, which is composed of devices with different throughputs like RAMDisk, non-volatile main memory (NVMM), and SSDs.

References

[1] Lustre. Lustre. [Online]. Available: http://lustre.org/
[2] D. Borthakur *et al.*, "Hdfs architecture guide," *Hadoop Apache Project*, vol. 53, no. 1-13, p. 2, 2008.
[3] J. Liu, J. Zhu, S. He, P. He, Z. Zheng, and M. R. Lyu, "Logzip: Extracting hidden structures via iterative clustering for log compression," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 863–873.
[4] Apache. Apache hadoop. [Online]. Available: https://hadoop.apache.org/
[5] R. Nathanael, S. Widodo, H. Abe, and K. Kato, "Hdrf: Hadoop data reduction framework for hadoop distributed file system," in *Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems*, 2020, pp. 122–129.

[6] Wikimedia. (2020) enwiki dump progress on 20200201. [Online]. Available: https://dumps.wikimedia.your.org/enwiki/20200201/enwiki-20200201-pages-articles-multistream.xml.bz2