

Implementing Efficient Data Compression and Encryption in a Persistent Key-Value Store for HPC

Jungwon Kim
Oak Ridge National Laboratory
kimj@ornl.gov

Jeffrey S. Vetter
Oak Ridge National Laboratory
vetter@ornl.gov

ABSTRACT

Recently, persistent data structures, like key-value stores (KVSs), which are stored in an HPC system’s nonvolatile memory, provide an attractive solution for a number of emerging challenges like limited I/O performance. This paper investigates how to efficiently integrate data compression and encryption into persistent KVSs for HPC with the ultimate goal of hiding their costs and complexity in terms of performance and ease of use. We implement the proposed techniques on top of a distributed embedded KVS to evaluate the benefits and costs of incorporating these capabilities along different points in the dataflow path, illustrating differences in effective bandwidth, latency, and additional computational expense.

ACM Reference Format:

Jungwon Kim and Jeffrey S. Vetter. 2018. Implementing Efficient Data Compression and Encryption in a Persistent Key-Value Store for HPC. In *Proceedings of SC18*. ACM, New York, NY, USA, 3 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 INTRODUCTION

The emergence of new nonvolatile memory (NVM) technologies provide the potential for truly massive amounts of memory capacity (e.g., exabytes) in future HPC systems. Researchers are exploring various options for NVM including burst buffers [6], in-situ visualization and analytics processing, and persistent data structures. In particular, one popular persistent data structure – the key-value store (KVS) – has proven effective in managing large data sets for a variety of applications, mostly in cloud computing. Recently, they have been making inroads into the areas of HPC [3, 4, 7].

Meanwhile, contemporary scientific HPC simulations and experimental facilities are producing massive amounts of data at unprecedented rate. Users are looking for alternative data management solutions that have different I/O characteristics (e.g., streaming, random access IOPs) that are at odds with existing I/O systems. Furthermore, more sensitive data (e.g., health records [5], DNA data [1]) is being used in distributed infrastructures [2], and users need practical methods to secure their data throughout its lifecycle, including while it is being processed by the system.

Data compression and encryption are two well-known techniques for improving several properties of data-oriented systems. Compression can improve effective storage capacity and bandwidth,

while encryption can provide a practical level of security for the data itself. This paper focuses on how to integrate these capabilities into a persistent KVS on NVM as transparently as possible in terms of cost and ease of use. The key contributions of this paper are the following:

- We evaluate the benefits and costs of several designs for implementing data compression and encryption capabilities along different points in the persistent KVS dataflow path, illustrating differences in effective bandwidth, latency, and additional computational expense.
- We present a data compression implementation that exploits memory hierarchy in HPC systems to achieve both storage reduction and performance improvement.
- We propose a data encryption technique that adds an extra security layer of data management in scientific workflows.
- We integrate our data compression and encryption techniques into PapyrusKV (an open source DEKVS) [4], and empirically evaluate these techniques on two HPC systems (CSCS’s Grand Tavé and NERSC’s Cori).

2 DESIGN

Figure 1 shows the data structures and their flow controls in PapyrusKV. PapyrusKV basically follows the log-structured merge-tree that stores data in a hierarchical structure. A database consists of a MemTable and a set of SSTables.

When data in the MemTable is compressed, it can reduce the DRAM memory footprint, memory transfer time, and interconnection network overhead. However, compressing/decompressing data also requires additional CPU overhead and increases MemTable access latency, potentially degrading performance. To address this issue, we introduce a new data structure, called WriteBuffer, in PapyrusKV. The WriteBuffer resides between the MPI rank and MemTable. It lies on a contiguous memory space on HBM or DRAM (if HBM is unavailable). WriteBuffer is partitioned into the same number of fixed size blocks as the number of running MPI ranks in

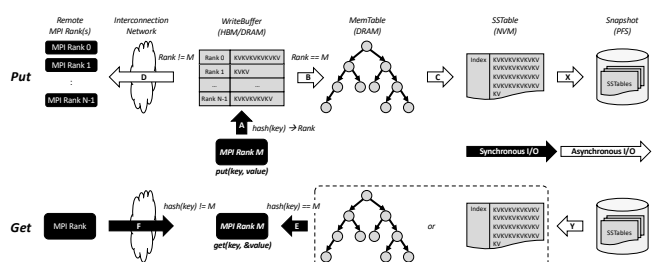


Figure 1: Data structures and flow control in PapyrusKV.

Publication rights licensed to ACM. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of the United States government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.

SC18, November 11–16, 2018, Dallas, TX, USA

© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...\$15.00

<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

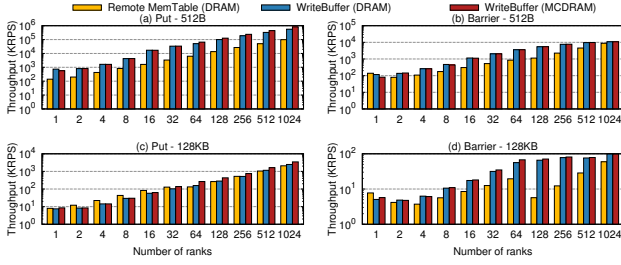


Figure 2: Put and barrier operations performance on Grand Tavé. Up to 64 MPI ranks run on a single node. For 512B, each rank performs 100K put operations with 32B keys and 512B values. For 128KB, each rank performs 1K put operations with 32B keys and 128KB values. We use KRPS (kilo requests per second) metric of performance.

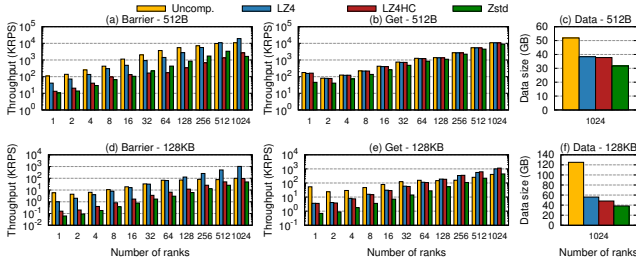


Figure 3: Barrier and get operations performance on Grand Tavé with various compression algorithms. Up to 64 MPI ranks run on a single node. For 512B, each rank performs 100K put operations with 32B keys and 512B values. For 128KB, each rank performs 1K put operations with 32B keys and 128KB values.

the application. When a block reaches its capacity limit, PapyrusKV compresses the block, and the block is flushed to the MemTable if the owner rank is the same as the caller (phase B) or migrated to the remote target owner rank via interconnection network (phase D). Both flushing and migration with compression are executed by the background threads asynchronously with the caller MPI rank.

Our data encryption uses the Advanced Encryption Standard (AES) and encrypts data at rest, that is, SSTables. SSTable encryption is executed when a new SSTable is created (phase C). In a get operation, when the KVS fails to find the matched key-value pair in the MemTable for a given key, it searches the SSTables in the database. When the KVS finds the matched key in the SSTable, it reads the data from the SSTable, decrypts the value, and returns the decrypted value.

3 PRELIMINARY EVALUATION

We evaluate the viability of our approach using Grand Tavé from Swiss National Supercomputing Centre and Cori from NERSC. Grand Tavé is a KNL-based system and NERSC’s Cori is equipped with NVM-based remote burst buffer architecture. To show different behaviors depending on the data size, we vary the size of values to 512B and 128KB.

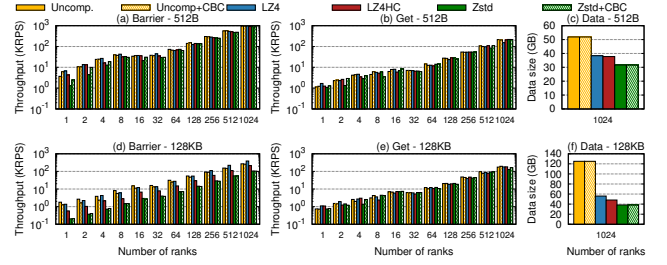


Figure 4: Barrier and get operations performance on Cori with various compression algorithms and AES-256 with CBC data encryption. Up to 32 MPI ranks run on a single node. For 512B, each rank performs 100K put operations with 32B keys and 512B values. For 128KB, each rank performs 1K put operations with 32B keys and 128KB values.

As shown in Figure 2a and Figure 2c, put operations with WriteBuffers show better performance than those with Remote MemTables in the original PapyrusKV. For barrier operations as shown in Figure 2b and Figure 2d, WriteBuffer alleviates the network congestion by aggregating key-value pairs and packetizing them. We measure the barrier and get performance and storage reduction with various data compression algorithms (LZ4, LZ4HC, and Zstd) as shown in Figure 3. LZ4HC shows the best get performance with 2.8x performance improvement with 0.4x data size on 1,024 ranks. Zstd shows the best storage reduction with 0.3x data size with 1.07x performance improvement on 1,024 ranks.

In Cori as shown in Figure 4, for 512B values, the data compression overhead and benefits from reduced data communication overhead are small. On the other hand, in 128KB values, LZ4 shows 1.4x performance improvement with 0.4x data size. As shown in Figure 4b and Figure 4e, all get operations show similar performance regardless of data size due to I/O overhead from the remote burst buffer.

We measure the data encryption overhead on Cori. AES-256 with CBC mode are used for uncompressed data (Uncomp+CBC) and compressed data using Zstd (Zstd+CBC) as shown in Figure 4. Data encryption produces nearly imperceptible cost as shown in Figure 4b and Figure 4e.

ACKNOWLEDGMENTS

This work was supported by a grant from the Swiss National Supercomputing Centre (CSCS) under the Accelerated Data Analytics and Computing (ADAC) collaboration. This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231.

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under contract number DE-AC05-00OR22725.

REFERENCES

- [1] Jarrod A. Chapman, Isaac Ho, Sirisha Sunkara, Shujun Luo, Gary P. Schroth, and Daniel S. Rokhsar. 2011. Meraculous: De Novo Genome Assembly with Short Paired-End Reads. *PLOS ONE* 6 (08 2011), 1–13.
- [2] Artem Chebotko, Shiyong Lu, Seunghan Chang, Farshad Fotouhi, and Ping Yang. 2010. Secure Abstraction Views for Scientific Workflow Provenance Querying. *IEEE Trans. Serv. Comput.* 3, 4 (Oct. 2010), 322–337.
- [3] S. Eilemann, F. Delalondre, J. Bernard, J. Planas, F. Schuermann, J. Biddiscombe, C. Bekas, A. Curioni, B. Metzler, P. Kaltstein, P. Morjan, J. Fenkes, R. Bellofatto, L. Schneidenbach, T. J. C. Ward, and B. G. Fitch. 2016. Key/Value-Enabled Flash Memory for Complex Scientific Workflows with On-Line Analysis and Visualization. In *Proceedings of the 30th IEEE International Parallel and Distributed Processing Symposium (IPDPS '16)*. 608–617.
- [4] Jungwon Kim, Seyong Lee, and Jeffrey S. Vetter. 2017. PapyrusKV: A High-performance Parallel Key-value Store for Distributed NVM Architectures. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '17)*. Article 57, 14 pages.
- [5] Jianghua Liu, Xinyi Huang, and Joseph K. Liu. 2015. Secure Sharing of Personal Health Records in Cloud Computing. *Future Gener. Comput. Syst.* 52, C (Nov. 2015), 67–76.
- [6] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn. 2012. On the Role of Burst Buffers in Leadership-Class Storage Systems. In *IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST '12)*.
- [7] Ke Wang, Abhishek Kulkarni, Michael Lang, Dorian Arnold, and Ioan Raicu. 2013. Using Simulation to Explore Distributed Key-value Stores for Extreme-scale System Services. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '13)*. 9:1–9:12.