I Can't Believe It's Not (Only) Software! Bionic Distributed Storage for Parquet Files

Lucas Kuhring
IMDEA Software Institute
Madrid, Spain
lucas.kuhring@imdea.org

Zsolt István
IMDEA Software Institute
Madrid, Spain
zsolt.istvan@imdea.org

ABSTRACT

There is a steady increase in the size of data stored and processed as part of data science applications, leading to bottlenecks and inefficiencies at various layers of the stack. One way of reducing such bottlenecks and increasing energy efficiency is by tailoring the underlying distributed storage solution to the application domain, using resources more efficiently. We explore this idea in the context of a popular column-oriented storage format used in big data workloads, namely Apache Parquet.

Our prototype uses an FPGA-based storage node that offers high bandwidth data deduplication and a companion software library that exposes an API for Parquet file access. This way the storage node remains general purpose and could be shared by applications from different domains, while, at the same time, benefiting from deduplication well suited to Apache Parquet files and from selective reads of columns in the file.

In this demonstration we show, on the one hand, that by relying on the FPGA's dataflow processing model, it is possible to implement in-line deduplication without increasing latencies significantly or reducing throughput. On the other hand, we highlight the benefits of implementing the application-specific aspects in a software library instead of FPGA circuits and how this enables, for instance, regular data science frameworks running in Python to access the data on the storage node and to offload filtering operations.

PVLDB Reference Format:
Lucas Kuhring, Zsolt István. I Can't Believe It's Not (Only) Software! Bionic Distributed Storage for Parquet Files. *PVLDB*, 12(12): 1838-1841, 2019.
DOI: https://doi.org/10.14778/3352063.3352079

1. INTRODUCTION

The popularity of data science applications in recent years has lead to the wide-spread usage of analytics systems that span several machines and access datasets stored on distributed storage. This distributed setup introduces data movement bottlenecks at all levels of the architecture and motivates the ongoing effort to build solutions that are power-efficient but, at the same time, offer low latency and high throughput access to data.

Using specialized hardware, such as Field Programmable Gate Arrays (FPGAs), shows promising results [11, 15, 20, 10]. One of the advantages of using FPGA-based nodes is that they can be designed such that the in-storage processing steps can be guaranteed to be performed without reducing read or write bandwidths to the nodes. This is done by structuring computation as pipelines that process input at a fixed rate [10]. In this work we explore deduplication as a specific example of in-storage processing that happens on the "write" side, not the "read" side where such functionality is usually provided.

Building distributed storage using FPGAs is in effect the specialization of the server hardware but, nonetheless, it is possible to offer general purpose, software-like services with such systems [9, 10]. However, if one would specialize the nodes further to support only specific file-type-based operations, their deployment might become uneconomical. Instead, in this work, we push general purpose functionality, namely data management and deduplication, to the hardware and add a software library that implements application-specific operations and interfaces with the FPGAs.

Deduplication is used in a wide variety of systems [3, 6, 21, 17] but we believe that the popularity of column-oriented file formats, such as Apache Parquet [12], offers new opportunities for in-storage deduplication. This is because many users manipulate Parquet files by adding or modifying a subset of the columns, or by adding batches of rows, resulting in duplicate data. The storage node can improve deduplication ratios in the knowledge of the file's internal structure. Our library breaks up the files into pages and stores these as separate key-value pairs. This approach achieves good deduplication ratios while also opening up further opportunities in in-storage processing, including filtering [10] and transformations of columnar data, without requiring the FPGA to implement Parquet-specific meta-data parsing.

The system we demonstrate, called $Multes{+}{+}$, is based on our earlier work [9, 10] and extends the nodes with deduplication functionality and a software library written in Golang with bindings to C and Python. In the demonstration we will focus on the following three aspects of the system:

- Thanks to the library, the FPGA can be seamlessly integrated with data processing applications written in Python. We will show an example that retrieves only the columns of interest of a Parquet file, reducing this way the data movement across the network.

Proceedings of the VLDB Endowment, Vol. 12, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3352063.3352079

- Given common scenarios for modifying Parquet files we show that our proposed method of storing these files broken up into parts based on their internal structure delivers as good deduplication ratios as state-of-the-art methods but requires less computation.
- Even though deduplication adds computation to write operations inside the storage node, the FPGA maintains high throughput and low latency.

2. BACKGROUND AND RELATED WORK

2.1 FPGA Key-Value Stores and Multes

Due to growing power consumption concerns, there is an increasing interest in offering FPGA-based or FPGA-accelerated distributed storage in the datacenter [23, 8, 10, 2, 5, 14]. These often offer an order of magnitude improvement in energy efficiency when compared to traditional server CPUs. They are also well suited to provide network-facing services because the applications they implement can be designed in ways that ensure network line-rate operation.

Unfortunately, FPGAs also have drawbacks: All "program code" occupies chip space, which means that not all types of applications are a good fit for FPGA-based acceleration (e.g., ones with branching logic). Furthermore, in a cloud setting, it can be challenging to provide multi-tenant applications on an FPGA with custom functionality per tenant (e.g., file-type-specific operations) because partitioning the device across tenants leads to smaller available areas for each [13]. Even though it is possible to dynamically change the set of operations supported on the FPGA through partial reconfiguration, this can be done only at coarse granularity. We make the case that it is a better approach to implement general multi-tenant functionality in hardware and move application-specific functionality into software.

In this work we extend *Multes* [9] which is an open source, replicated, multi-tenant key-value store (KVS). It uses a high throughput hash table based on the Cuckoo hashing algorithm [19] combined with a slab-based memory allocator. In addition, *Multes* implements performance and data isolation for multiple tenants. Replication and multi-tenancy are orthogonal to our deduplication extension and their behavior remains unchanged. However, the ideas presented in this work are not limited to *Multes* and could be used in other FPGA-based key-value stores as well – mostly because most of the specialization happens not in hardware but in the software library.

2.2 Deduplication

Deduplication has been studied in extensive related work and is part of numerous storage and cloud systems and operating systems. One differentiating factor is the way that files are subdivided (chunked) for their fingerprints to be computed: this can be done either by using fixed chunk size [3, 22] that requires no additional compute to determine the chunk boundaries, or variable chunk size (VSC) where a hash function, such as Rabin fingerprinting [3, 18] is used to determine chunk boundaries.

Deduplication can be done either on the client [4, 7] or in the storage node [6, 21, 16, 7]. In the case of the former, the network bandwidth is preserved by sending less data, but latency is increased because extra communication is required to identify duplicate chunks. Performing deduplication entirely in the storage node removes the compute burden from the client but requires that the underlying hardware guarantees high throughput for deduplication operations (hence our interest in FPGA-based solutions).

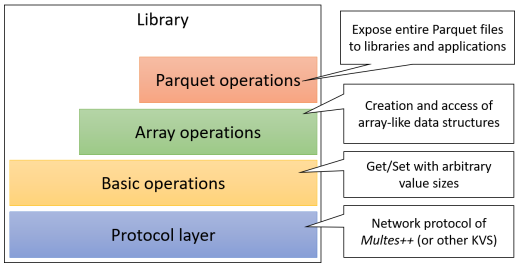

Figure 1: The client library of *Multes++* has been designed to expose operations of increasing complexity to the clients.

3. SYSTEM OVERVIEW

When compared to the system it extends, *Multes++* brings two improvements: First, it implements line-rate deduplication logic inside the storage node that relies on a modified hash table data structure to manage duplicates. Second, it provides a companion software library that exposes several layers of abstraction when accessing the FPGA. At the highest level of abstraction are the Parquet-file operations.

Furthermore, we devise a hybrid chunking scheme for Parquet files. The chunk boundaries are determined by the file's internal structure, each "page" resulting in an initial chunk. Since the FPGA has internal limitations on the value size and a page could be significantly larger than the maximum value size, large pages are split once again, resulting in smaller chunks. This method requires less compute resources (no Rabin fingerprinting) but still results in deduplication ratios as good as VCS.

3.1 Software Library

The client library is organized into layers as shown in Figure 1, starting with the basic get/put operations over TCP and increase in level of abstraction until Parquet file operations (e.g., accessing a specific column).

Since the FPGA has an internal limit on the maximum size of a value, in the library we split up a large key-value pair into several parts which are then stored under "helper keys". These keys are derived from the initial user-defined one by appending additional "sequence bytes" to it but the hardware has no knowledge of the meaning of these bytes, and treats all of them as independent keys. When writing large values to the FPGA, the overhead of storing multiple key-value pairs can be fully hidden by writing them as a batch. Read operations will require retrieving the user key first, and if present, additional parts of the value in a batched manner. This adds one RTT overhead, but for large values transmission time will dominate anyhow.

We implement array operations similarly to the large values, by relying on a special "header key-value pair" that encodes the keys that compose the array. This is opaque to the clients, that see API functions requiring only a key and an array data structure. Arrays and large values can be combined, allowing the software to store 100s of megabytes under a single logical key, even if the FPGAs internal maximum values size is, for instance, 1KB.

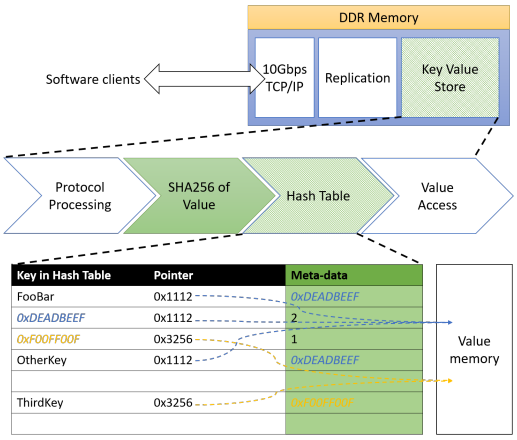

Figure 2: *Multes++*extends the hash table pipeline of *Multes* with a value fingerprinting step and the memory used for storing keys now also stores the fingerprints (all parts of the system that were changed are shown in green). Each key's entry contains both a pointer to the value and its fingerprint.

Parquet files are decomposed into a structure similar to an array, where each column and page within the column can be accessed in a random-access manner through the array header. In our implementation we focus on single-threaded clients, representative, e.g., of Python-based data science applications, but for arrays we offer the possibility of writing and reading multiple entries in parallel on separate goroutines to increase throughput.

3.2 Hardware Modules

The internal architecture of *Multes* is pipelined, which allowed us to introduce additional logic for deduplication-related computations without slowing down the system. As Figure 2 shows, most of the changes (marked in green) are limited to the hash table implementation inside the FPGA. We introduced a SHA256 [1] hash step before the hash table that processes the values and we modified the hash table to manage the "fingerprints" of value chunks. This required changes to the write and delete operations in the KVS, but no changes to read operations, resulting in identical "get" behavior to *Multes*.

In order to achieve line-rate behavior even in the presence of the compute intensive hashing step, we use data parallel execution on the FPGA and deploy 11 SHA256 cores in parallel. As we demonstrate, this results in increased write latency (around 3μs added per 512B data chunk to the RTT when using deduplication) but does not impact write or read bandwidths. Furthermore, the deduplication functionality does not impact read latency, since each key's value pointer is accessible directly without requiring additional lookups.

Multes++ inherits the multi-tenant behavior of *Multes* where each tenant has access only to its own hash table partition, and as a result, deduplication will only happen across values belonging to the same tenant. This design decision can be easily changed, by storing the fingerprints in a memory area that can be accessed by all tenants – this, however, might introduce security concerns and break privacy assumptions across tenants.

Figure 3: The demonstration is controlled and visualized through an interactive Jupyter notebook.

4. DEMONSTRATION OVERVIEW

4.1 Setup

The demonstration is controlled and visualized through a Jupyter notebook that executes benchmarking scripts and commands in the background, shown in Figure 3. The notebook resides on a server at IMDEA Software. *Multes++* runs on Xilinx VCU1525 boards with Virtex Ultrascale+ FPGAs and 64GBs of DDR4, from which we use 32GB (two 16GB SODIMMS) split between the hash table and the value storage. The FPGAs are in the same cluster as the server machines and they communicate over a regular 10Gbps switch and TCP/IP sockets.

4.2 Performance

To show that deduplication has no significant impact on the throughput seen by clients, we will compare *Multes++* to *Multes* (no deduplication), and to memcached, for a software baseline. Clients issue set commands with large Parquet files to measure the write bandwidth. Visitors can choose between a set of Parquet files of varying sizes.

To explore the cost of deduplication in terms of latency, the visitors will have the opportunity to pick various value sizes for which the CDF of SET response times will be plotted. This allows a more in-depth comparison and discussion on trade-offs.

4.3 Space Saving

In the second part of the demonstration we show that our proposed file-type-aware chunking scheme delivers as good deduplication ratios as state of the art methods. For this, we store different versions of the Police dataset, either with additional rows or columns. The experiment is set up such that the visitors can choose how many times to store the file and with what modifications. The results are plotted after the experiment has run as a bar chart showing the total amount of storage space in use on the node.

4.4 Access from User Applications

The last part of the demonstration comprises of a small Python application that the visitors can interact with. The client library of *Multes++* exposes bindings to Python and can be used to read specific columns of a Parquet file without having to retrieve all data (i.e., to perform projection). In the provided example we access two columns of a Parquet file containing the Flight History dataset from DataSF (`https://datasf.org/opendata/`). The file resides on the FPGA, and the Python application loads the columns into a Pandas DataFrame for analysis to determine which companies had the heaviest cargo landing at SFO.

The additional importance of this integration with Python is that it enables future in-storage processing of columnar data formats (e.g., push-down of various filtering expressions), controlled directly from high level applications.

Acknowledgments

We would like to thank Xilinx for their generous donation of software tools and IP. The VCU1525 boards have been purchased as part of the Xilinx Accelerator Program.

5. REFERENCES

[1] Fips 180-4 secure hash standard (shs), national institute of standards and technology (nist). https://csrc.nist.gov/publications/detail/fips/180/4/final.

[2] M. Blott, L. Liu, K. Karras, and K. A. Vissers. Scaling out to a single-node 80Gbps memcached server with 40Terabytes of memory. In *HotStorage'15*, 2015.

[3] D. R. Bobbarjung, S. Jagannathan, and C. Dubnicki. Improving duplicate elimination in storage systems. *ACM TOS*, 2(4):424–448, 2006.

[4] D. Cannon. Data deduplication and tivoli storage manager. *Tivoli Storage, IBM Software Group (September 2007)*, 2009.

[5] S. R. Chalamalasetti, K. Lim, M. Wright, A. AuYoung, P. Ranganathan, and M. Margala. An FPGA memcached appliance. In *FPGA'13*, pages 245–254. ACM, 2013.

[6] B. K. Debnath, S. Sengupta, and J. Li. Chunkstash: Speeding up inline storage deduplication using flash memory. In *USENIX ATC'10*, 2010.

[7] L. DuBois and R. Amatruda. Backup and recovery: Accelerating efficiency and driving down it costs using data deduplication. *EMC Corporation*, 2010.

[8] E. S. Fukuda, H. Inoue, T. Takenaka, D. Kim, T. Sadahisa, T. Asai, and M. Motomura. Caching memcached at reconfigurable network interface. In *FPL'14*, pages 1–6. IEEE, 2014.

[9] Z. István, G. Alonso, and A. Singla. Providing multi-tenant services with FPGAs: Case study on a key-value store. In *FPL'18*, pages 119–124, 2018.

[10] Z. István, D. Sidler, and G. Alonso. Caribou: intelligent distributed storage. *PVLDB*, 10(11):1202–1213, 2017.

[11] S. Jun, M. Liu, S. Lee, J. Hicks, J. Ankcorn, M. King, and S. X. and. Bluedbm: An appliance for big data analytics. In *ISCA'15*, pages 1–13, 2015.

[12] J. Kestelyn. Introducing Parquet: Efficient columnar storage for Apache Hadoop. *Cloudera Blog*, 3, 2013.

[13] A. Khawaja, J. Landgraf, R. Prakash, M. Wei, E. Schkufza, and C. J. Rossbach. Sharing, protection, and compatibility for reconfigurable fabric with amorphos. In *OSDI'18*, pages 107–127, 2018.

[14] M. Lavasani, H. Angepat, and D. Chiou. An fpga-based in-line accelerator for memcached. *IEEE Computer Architecture Letters*, 13(2):57–60, 2014.

[15] B. Li, Z. Ruan, W. Xiao, Y. Lu, Y. Xiong, A. Putnam, E. Chen, and L. Zhang. Kv-direct: high-performance in-memory key-value store with programmable nic. In *SOSP'17*, pages 137–152, 2017.

[16] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezis, and P. Camble. Sparse indexing: Large scale, inline deduplication using sampling and locality. In *Fast'09*, volume 9, pages 111–123, 2009.

[17] X. Lin, F. Douglis, J. Li, X. Li, R. Ricci, S. Smaldone, and G. Wallace. Metadata considered harmful... to deduplication. In *HotStorage'15*, 2015.

[18] A. Muthitacharoen, B. Chen, and D. Mazieres. A low-bandwidth network file system. In *ACM SIGOPS Operating Systems Review*, volume 35. ACM, 2001.

[19] R. Pagh and F. F. Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2), 2004.

[20] B. Salami, G. A. Malazgirt, O. Arcas-Abella, A. Yurdakul, and N. Sonmez. AxleDB: A novel programmable query processing platform on FPGA. *Microprocessors and Microsystems*, 51:142–164, 2017.

[21] K. Srinivasan, T. Bisson, G. R. Goodson, and K. Voruganti. idedup: latency-aware, inline data deduplication for primary storage. In *FAST'12*, volume 12, 2012.

[22] N. Tolia, M. Kozuch, M. Satyanarayanan, B. Karp, T. C. Bressoud, and A. Perrig. Opportunistic use of content addressable storage for distributed file systems. In *USENIX ATC'03*, volume 3, pages 127–140, 2003.

[23] S. Xu, S. Lee, S.-W. Jun, M. Liu, J. Hicks, et al. BlueCache: A scalable distributed flash-based key-value store. *PVLDB*, 10(4):301–312, 2016.