Storing Parquet Tile by Tile: Application-aware Storage with Deduplication

Lucas Kuhring, Zsolt István
IMDEA Software Institute, Madrid, Spain
{firstname.lastname}@imdea.org

***Abstract*—Distributed storage in the cloud needs to offer both low latency and high bandwidth access to data and efficient use of storage capacity in order to keep up with emerging big data workloads. Deduplication has been successfully used to help with the latter requirement but it is often at odds with low latency data access. Deduplication ratios can be significantly increased if the storage nodes are aware of the file format and the ways clients interact with it – but implementing different file-type specific parsing on FPGAs for multiple tenants can be unfeasible due to area constraints.**

We show the benefits of making the storage system aware of the application through the example of Parquet files, a columnar format used in machine learning and big data frameworks to store and transfer datasets. We achieve high deduplication ratios by using a companion software library that allows Parquet files to be stored in a "divided" way. This makes deduplication more efficient and enables clients to access individual columns or meta-data fields selectively. At the same time, the storage nodes remain general purpose and can store and deduplicate arbitrary data.

This work paves the way for in-storage processing for Parquet files and other columnar formats because the different columns can be accessed in a streaming fashion and their processing requires no specialized logic on the FPGA.

***Index Terms*—FPGA, distributed storage, deduplication, column stores, application-aware storage**

I. System Overview

We designed Multes++ [6] such that it fulfills the following goals without compromising line-rate behavior for reads:

1) Deduplicate the data stored in the key-value store, aiming for line-rate for values in the kilobyte range.
2) Allow the clients to store up to megabytes in a single logical key-value pair, so that Parquet files can be transparently handled in FPGA-based storage.
3) Ensure high deduplication ratios for Parquet files (that are often modified by adding and removing columns or appending large batches of tuples to the data) without requiring file-type-specific specialization of the hardware.

A. Hardware Architecture

Due to growing power consumption concerns, there is an increasing interest in offering FPGA-based or FPGA-accelerated distributed storage in the datacenter [1], [3], [4], [7], [8]. We achieve the above stated goals by extending one of these state of the art systems, namely Multes [5].

Multes implements a multi-tenant key-value store on FPGAs that provides replication for fault tolerance, pluggable processing on the "read path" and regular 10Gbps TCP/IP connectivity to clients. This work is orthogonal to the multi-tenancy and replication aspects, and inherits them from Multes.

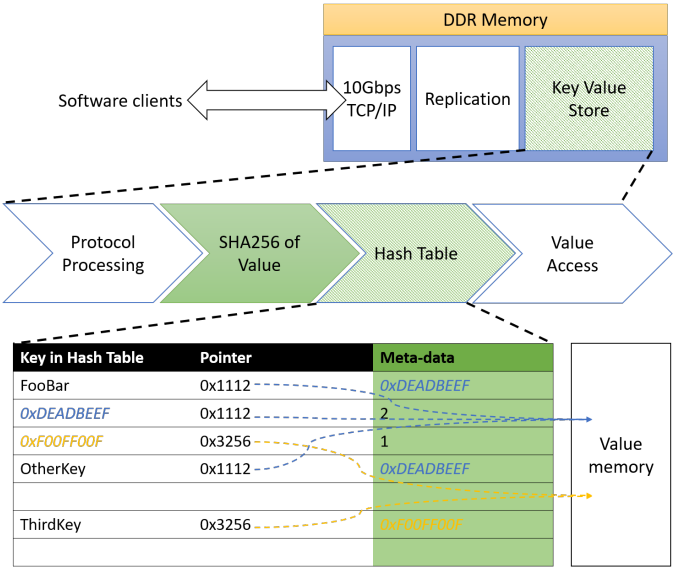

Fig. 1. Multes is extended with deduplication logic that consists of hashing cores, as well as, additional hash table operations to manage value fingerprints.

We extend the internal pipeline of Multes with logic for deduplication-related computations. As shown in Figure 1, the most important addition to the pipeline is a SHA256 hash step before the hash table that processes the values. To achieve line-rate throughput, we deploy several SHA256 cores in a data parallel fashion, to work on different input key-value pairs.

We modify the hash table to manage the fingerprints of value chunks, used to detect duplicates. As shown in Figure 1, the hash table stores regular entries and fingerprints side by side, and each regular key entry has a reference to its value's fingerprint. The implementation of deduplication required changes to the write and delete operations, but no changes to read operations, resulting in identical "get" behavior to Multes.

B. Software Library

In addition to the hardware changes, Multes++ has a software library that abstracts away FPGA idiosyncrasies (e.g., that there is an upper bound on value sizes) and provides general purpose high level operations. Building on top of the "protocol" layer, that encodes requests on the wire, the library implements so called "large values". It can store arbitrarily sized key-value pairs by transparently breaking them up into

several physical pairs and storing them as a linked list. The library offers the abstraction of arrays as well, that allows retrieving either specific items or the entire array. This feature can be combined with large values to store potentially hundreds of MBs under a single logical key.

Parquet files are stored not as BLOBs but are, instead, broken up into pieces by the software library. This allows for as good, or better, deduplication ratios of modified Parquet files than the state of the art Variable Sized Chunking (VCS) methods. These are successful in inferring "cut points" in the data structure based on common patterns across files but require costly computations of running hashes [2]. In contrast, our approach relies on the knowledge of the internal structure of the file to determine chunk boundaries and requires no additional computation. The Parquet functionality is built using the array abstraction, where each page of the file corresponds to an array entry. We store the meta-data and headers under special keys that allows the client to selectively access columns or pages of the potentially very large Parquet files.

II. Demo Setup and Experiments

The demonstration is controlled and visualized through a Jupyter notebook. Multes++ runs on stand-alone Xilinx VCU1525 boards with Virtex Ultrascale+ FPGAs that are in the same 10Gbps cluster as the client machines.

As seen in the top part of Figure 2, we show that deduplication has no significant impact on the throughput or latency by comparing 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.

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 three versions of a Parquet file using different deduplication methods and we measure the total amount of storage used. 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 (Figure 2).

The last part of the demonstration comprises of a small Python application that the visitors can interact with. The client library exposes bindings to Python and can be used to read specific columns of a Parquet file without having to retrieve all data. 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.

Fig. 2. The demonstration is controlled and visualized through an interactive Jupyter notebook.

References

[1] 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.

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

[3] S. R. Chalamalasetti, K. Lim, M. Wright, et al. An FPGA memcached appliance. In *FPGA'13*. ACM, 2013.

[4] 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*. IEEE, 2014.

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

[6] L. Kuhring, E. Garcia, and Z. István. Specialize in moderation—building application-aware storage services using FPGAs in the datacenter. In *USENIX HotStorage'19*.

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

[8] S. Xu, S. Lee, S.-W. Jun, M. Liu, J. Hicks, et al. Bluecache: A scalable distributed flash-based key-value store. *Proceedings of the VLDB Endowment*, 10(4), 2016.