Caribou: A Platform for Building Smart Storage

Zsolt István (presenter) David Sidler Gustavo Alonso
Systems Group, Dept. of Computer Science, ETH Zürich, Switzerland
{firstname.lastname}@inf.ethz.ch

1. Background

As an answer to increasing data sizes and stagnating CPU performance, novel database engines and data processing applications are designed to have disaggregated architectures [3]. This means separating compute and storage nodes with the benefit that the two layers can be scaled separately. This leads to more efficient use of resources. As a tradeoff, however, the increased data movement often becomes a bottleneck. For this reason it can be beneficial to push down parts of filtering or processing to the storage to avoid unnecessary data movement through the application stack.

In this work we look beyond logical specialization of storage nodes for data processing applications and explore how physical specialization can offer benefits in terms of throughput and latency, but not only. Thanks to hardware pipelining, complex application-specific processing can be pushed down to the storage without impacting performance.

The resulting system, that we call Caribou, provides a key-value store interface common to many data processing applications and has a modular architecture that allows plugging in different application-specific processing units (e.g., complex filtering predicates for SQL queries).

2. Design Overview

In this poster we present Caribou: a distributed key-value store with scan capabilities that exposes DRAM over a 10Gbps network, and has been designed with in-storage processing in mind, that the higher level application can take advantage of. We demonstrate how a distributed query engine can push down selection predicates and regular expression matching, but the architecture is modular and extensible.

Caribou combines the state-of-the-art with new ideas. On the one hand, we build upon our 10Gbps low-latency TCP/IP stack [1] and our work on low-latency replication in hardware [2]. On the other hand, we address the shortcomings of the state-of-the-art in FPGA key-value stores by implementing a hash table that can handle collisions well and proper memory allocation that can scale to larger capacity storage. We chose to implement a Cuckoo hash table because it offers constant time read operations, important for providing predictable access to the data. Writes are also constant time from the point of view of the layers above because kick-outs are handled in parallel to the regular operations. To make sure that memory is used efficiently we implemented a slab-based memory allocator similar to the one in Memcached. In addition to point lookups, Caribou supports scan operations as well. Scans rely on bitmaps exposed by the memory allocator to access only parts of the memory that contain values.

The key-value store has been designed to accommodate pluggable application-specific processing units, that work on data that is being retrieved from memory. For our prototype we focused on distributed databases and added two ways of filtering: a pipelined comparison-based selection unit and a regular expression-based selection unit. Both are runtime parametrized to perform complex filtering. The overhead of parameterization per request is negligible, but filtering is most effective when combined with scans.

In the poster we highlight why reconfigurable hardware is a promising way of exploring ideas around smart storage: while in software there is an inherent trade-off between functionality and performance, in that additional or more complex functionality often leads to lower throughput, in hardware the trade-off is different. Throughput can be fixed and additional functionality results in more chip area used.

3. Faster Exploration

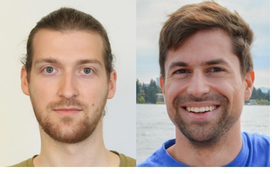
Caribou offers high performance access to DRAM at the moment, but it could serve data from flash storage as well. More importantly, the key-value store API and the ability to push down application-specific processing makes it suitable for prototyping different kinds of smart storage. Since the processing logic is plugged into the key-value store using simple streaming interfaces, the design of compute units is greatly simplified and data management is already taken care of. This reduces the "entry barrier" to exploring new ideas. In the poster we sketch how Caribou could be used to speed up domain-specific processing in two data science use-cases.

References

[1] D. Sidler, Z. Istvan, G. Alonso. Low-Latency TCP/IP Stack for Data Center Applications In *FPL'16*.

[2] Z. István, D. Sidler, G. Alonso, M. Vukolic. Consensus in a Box: Inexpensive Coordination in Hardware In *NSDI'16*.

[3] S. Loesing, M. Pilman, and others. On the design and scalability of distributed shared-data databases. In *SIGMOD'15*.



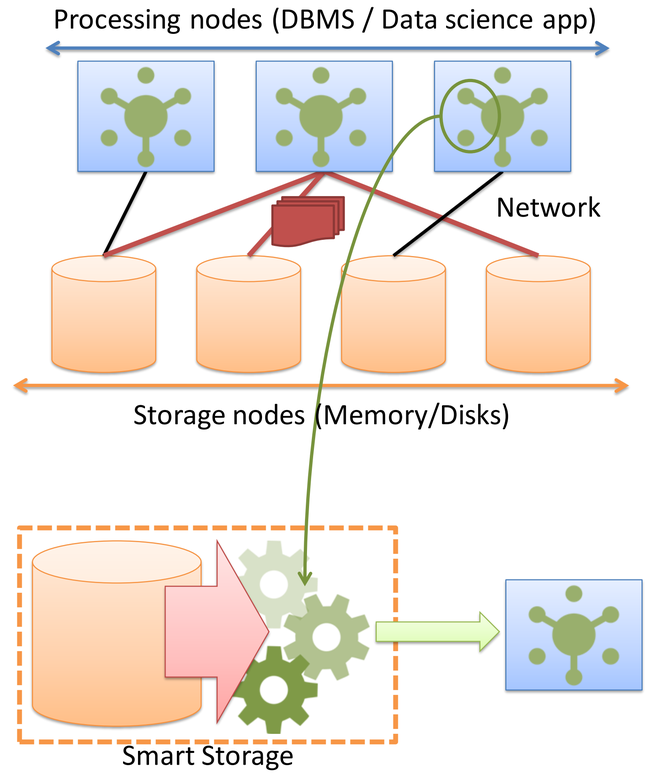
Zsolt István, David Sidler, Gustavo Alonso
Systems Group, Department of Computer Science, ETH Zürich
{firstname.lastname}@inf.ethz.ch

Caribou: A Platform for Building Smart Storage

Motivation

Growing Data Sizes → Distributed Systems

- Data sizes increasing, more complex problems ↔ CPU speeds stagnating
- Data processing systems designed to be distributed
- Scalability achieved through separation of processing and storage nodes
- Networking is an important limiting factor

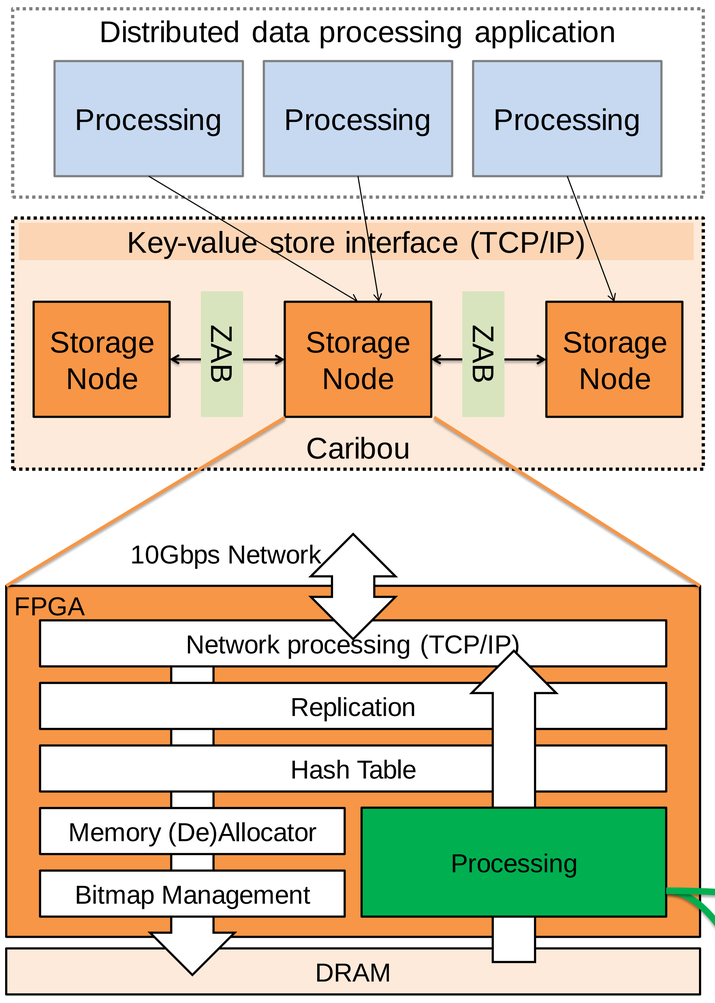


Reducing the Data Movement Bottleneck

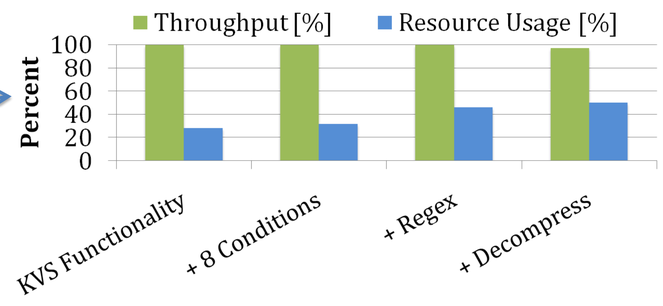
- (Pre-)Processing data in storage reduces transfer sizes
- Design challenge: Complex processing without impacting throughput or latency?

Contribution

Caribou: Distributed Storage + Pluggable Processing

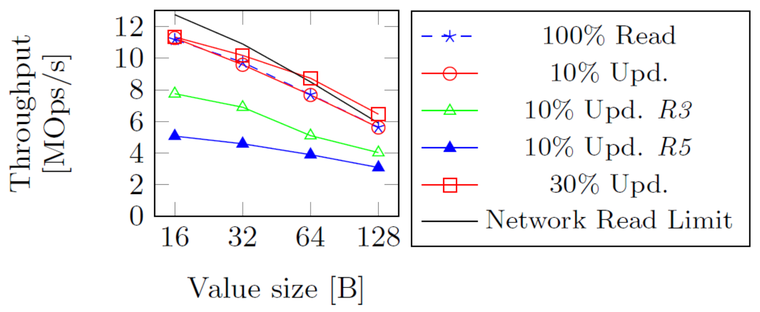


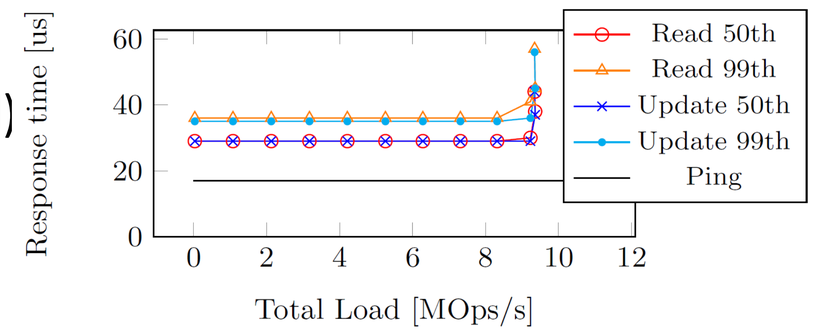
~~**Functionality vs. Throughput**~~
Functionality vs. Chip Area



- KVS Interface + Scans
- 10Gbps TCP/IP networking
- Cuckoo hash table
- Slab-based memory allocation
- Replication (Zookeeper Atomic Broadcast)

- High throughput (network bound)
- Latency dominated by clients





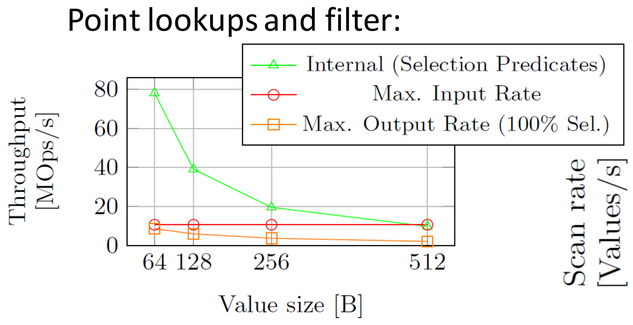
Applications

Intelligent Storage for DBs

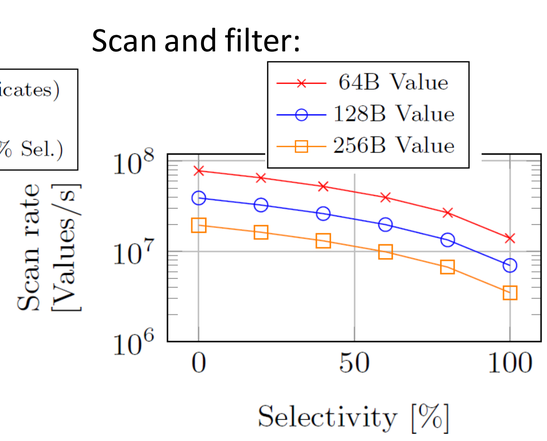
- Filtering tuples during a scan or lookup

1) Byte-offsets and predicates (int, byte columns)
2) Regular expressions (less structured columns)

Point lookups and filter:



Scan and filter:



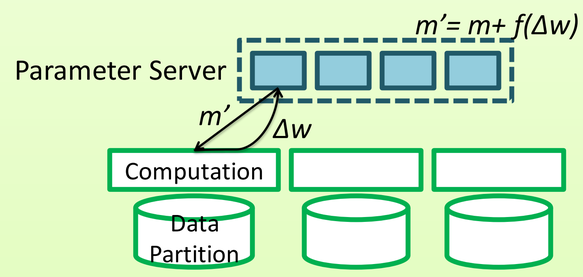
- Pipeline rate might be lower than SW – but can add functionality without overhead

Processing for Data Sciences

- KVS manages data **+ custom processing block**

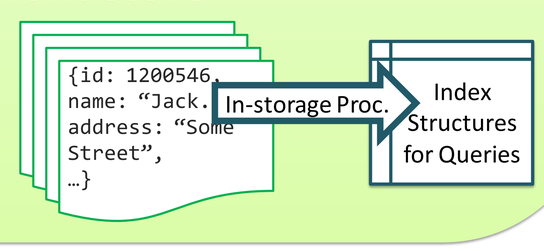
- Example1: Machine learning – Parameter Server

  - Clients compute updates
  - Server stores updates and computes new model
  - Clients get new model



- Example2: Search – Document store

  - Store document (JSON) data
  - Processing on schema, Regex to index



Acknowledgements: This work is funded in part through the Microsoft Joint Research Center MSR-ETHZ-EPFL and an equipment donation from Xilinx.