HeteroKV: A Scalable Line-rate Key-Value Store on Heterogeneous CPU-FPGA Platforms

Haichang Yang*, Zhaoshi Li†, Jiawei Wang†, Shouyi Yin†, Shaojun Wei† and Leibo Liu†
Institute of Microelectronics, *Tsinghua University*, Beijing 100084, China
*yanghc18@mails.tsinghua.edu.cn, †{lizhaoshi, claire_wang, yinsy, wsj, liulb}@tsinghua.edu.cn

***Abstract*—In-memory key-value store (KVS) has become crucial for many large-scale Internet services providers to build high-performance data centers. While most of the state-of-the-art KVS systems are optimized for read-intensive applications, a wide range of applications have been proven to be insert-intensive or scan-intensive, which scale poorly with the current implementations. With the availability of FPGA-based smart NICs in data centers, hardware-aided and hardware-based KVS systems are gaining their popularity. In this paper, we present HeteroKV, a scalable line-rate KVS on heterogeneous CPU-FPGA platforms, aiming to provide high throughput in read-, insert- and scan-intensive scenarios. To achieve this, HeteroKV leverages a heterogeneous data structure consisting of a b+ tree, whose leaf nodes are cache-aware partitioned hash tables. Experiments demonstrate HeteroKV's high performance in all scenarios. Specifically, a single node HeteroKV is able to achieve 430M, 315M and 15M key-value operations per second in read-, insert- and scan-intensive scenarios respectively, which are more than 1.5x, 1.4x and 5x higher than state-of-the-art implementations.**

***Index Terms*—in-memory KVS, hardware-acceleration**

I. Introduction

The increasing main memory capacity has facilitated the development of the main-memory data store in data centers [13]. In-memory key-value store (KVS) is a typical NoSQL data store that keeps data in main memory to achieve high throughput and low latency. KVS systems like Memcached [7] and Redis [1] have been widely deployed by lots of enterprises in their distributed server clusters to provide key-value caching services. Recently, with the prevalence of the main-memory big data processing, the KVS works not only as a cache for in-storage database, but also as an infrastructure to store shared data in distributed systems exemplified by the parameter server for machine learning.

While most of the current KVS implementations focus on read-intensive scenarios, a large number of applications have been shown to be insert-intensive or short-range scan-intensive [2], [12], which means that KVS systems supporting high performance read, write and range queries are demanding.

Most recently, heterogeneous architectures are gaining popularity due to their potential performance and energy benefits. More and more servers equipped with programmable NICs are deployed in commodity data centers [8], [10]. Usually, a programmable smart NIC is comprised of an FPGA embedded with a network interface controller (NIC) chip to connect with the network. Applications including KVS can be accelerated by offloading some tasks onto FPGAs of programmable NICs. A number of researches have strived to design high performance KVS with such heterogeneous systems [3], [8].

In this paper, we present HeteroKV, a scalable line-rate KVS system on heterogeneous CPU-FPGA platforms, focusing on achieving high performance in different scenarios. The name HeteroKV is chosen not only because the heterogeneous CPU-FPGA platform is our underlying platform, but also because a heterogeneous data structure consisting of a b+ tree, whose leaf nodes are cache-aware partitioned hash tables, is the underlying data structure of HeteroKV.

The main contributions of this work are as follows:

- A heterogeneous data structure consisting of a b+ tree, whose leaf nodes are cache-aware partitioned hash tables, is hardware-software co-designed to maximize the collaborative performance of heterogeneous platforms.
- A fully-pipelined hardware-accelerated dynamic b+ tree is proposed, based on which a high-throughput key-value requests dispatcher is implemented on FPGA to alleviate pressure of the DRAM bandwidth and reduce inter-threads synchronization overhead on CPU.
- A cache-friendly partitioned hash table algorithm is used to enable CPU to support fast PUTs and GETs, as well as efficient SCANs.
- A prototype of HeteroKV that adopts aforementioned techniques is implemented on a CPU-FPGA platform comprised of a Xeon Gold 6138 processor and a Stratix 10 FPGA, and experiments are carried out to demonstrate the performance of the proposed design.

II. Background

Modern KVS servers offer four APIs: PUT(`key, value`), GET(`key`), DELETE(`key`) and SCAN(`head_key, tail_key`). PUT operation inserts the KV pair into the database. GET operation returns the corresponding key-value (KV) pair if it exists in database, otherwise it returns a null value. DELETE operation removes the KV pair from database if it exists. SCAN operation, also called range query, returns the existing KV pairs whose key value falls inside the interval between `head_key` and `tail_key`.

A. Data Structure Design

B+ Tree and Hash Table are two data structures that are most widely used by in-memory KVS.

Hash Table is widely deployed because it supports fast read and write operations with the algorithmic complexity of $\mathcal{O}(1)$. However, the significant drawback of the Hash Table lies in its inefficiency for SCAN operations. For SCAN operations, Hash Table needs to search hash bucket of every key between

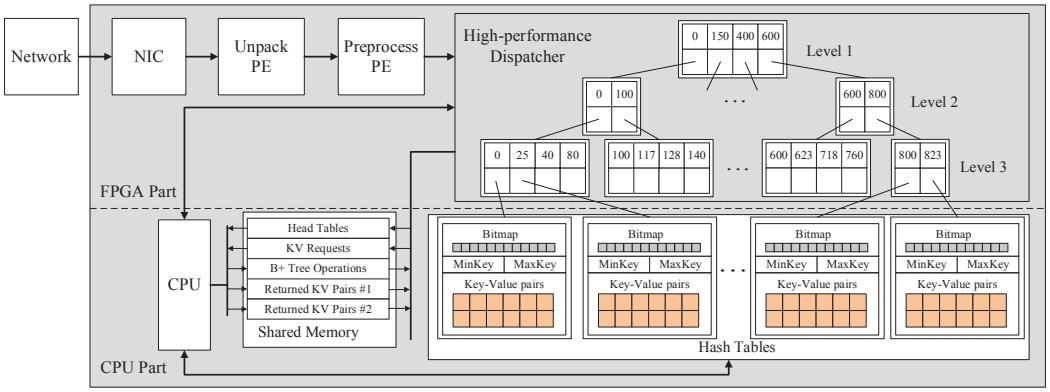

Fig. 1: Overall architecture of HeteroKV

`head_key` and `tail_key` range even though it may not exist. Since hash functions tend to scatter adjacent keys, SCANs in Hash Table suffer from deteriorated performance by violating the locality of caches.

As a variant of B Tree, B+ Tree [4] is widely used by many in-memory KVS systems for its support for efficient SCAN operations. Inside the nodes of B+ Tree, KV pairs with adjacent keys are stored in contiguous memory space, which require an algorithmic complexity of $\mathcal{O}(LogN)$ for SCANs to access, where N is the number of KV pairs in the system. High throughput would be guaranteed by high spatial locality of data. However, for PUT and GET operations on single keys, B+ Tree also features an algorithmic complexity of $\mathcal{O}(LogN)$.

B. Software Acceleration KVS

As a re-design of Memcached, MemC3 [6] is a highly concurrent in-memory KVS optimized for read-intensive systems, by adopting *optimistic cuckoo hashing*. But it is slow under write-intensive workloads for not supporting concurrent writes. MICA [9] partitions the KV pairs among different cores to support concurrent reads and writes. But the partition strategy may lead to load imbalance among different cores. Moreover, both MemC3 and MICA perform poorly for SCANs as they use Hash Tables.

Meanwhile, a large portion of works have been proposed to improve the concurrency of B+ Tree-based KVS. PALM [11], a main-memory B+ Tree design, takes advantage of batch processing to boost data-level parallelism. Before being served, requests in a batch are sorted firstly and then partitioned across cores such that these partitions can be served concurrently. Inspired by PALM, *HeteroKV also adopts the strategy of batch processing to achieve high performance*.

Though many works have been proposed to improve the performance of B+ Tree-like data structures, all these implementations can hardly avoid $\mathcal{O}(LogN)$ times higher algorithmic complexity for PUT and GET operations than Hash Tables.

C. Hardware Acceleration KVS

Recently, the rise of FPGA-based smart NICs stipulates interests in hardware-accelerated KVS. An FPGA-based Memcached accelerator [3] is proposed to offer accelerated KV caching service without supporting KVS semantics. KV-Direct [8] serves all requests solely on FPGAs by optimizing the PCIe traffic to host main memory. But it shows a huge performance gap between read-intensive and write-intensive workloads. Analysis shows that KV-Direct is still bottlenecked by PCIe bandwidth. Besides, current FPGA-based KVS systems are built upon Hash Tables and can hardly support SCANs.

III. Proposed Design

A. Overview of HeteroKV Architecture

Since commercial CPUs feature higher main-memory bandwidth than FPGAs by leveraging on-chip caches, HeteroKV aims to partition requests with B+ Tree-based indices on FPGAs and access the partitioned main-memory KVS on CPUs. Fig. 1 depicts the block level diagram of the HeteroKV architecture. The proposed architecture can be divided into two main parts: FPGA part and CPU part. The FPGA part of the proposed system is shown on top of the Fig. 1. Packets containing KV requests are firstly received by the NIC, and are unpacked by *unpack processing element* (PE). The *preprocess PE* buffers KV requests from *unpack PE* firstly. When the number of KV requests reaches a threshold, *preprocess PE* will sort them according to their key value to improve the efficiency of the *dispatcher*. Once sorted, KV requests are then sent to a b+ tree-based high-performance KV requests *dispatcher*. Each leaf node of b+ tree corresponds to a hash table stored in CPU's main memory. The bottom of Fig. 1 shows the CPU part of HeteroKV, the core of which is *cache-aware partitioned hash tables* stored in CPU's memory. CPU and FPGA communicate and exchange data with each other through shared memory.

B. Preprocess PE: Merge Sorter

Preprocess PE is comprised of a *KV requests buffer* and a *sorter*. Requests from the *unpack PE* are buffered until a batch size is reached. A larger batch size improves the average throughput, but also results in higher latency. A batch size of 8K is selected to make a trade-off between throughput and latency (§Section IV-C). To maximize the throughput of the *dispatcher*, KV requests are sorted before sent to *dispatcher*. A line-rate merge sorter customized for FPGAs is designed to make the processing rate keep up with the network bandwidth.

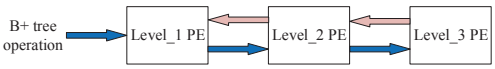

Fig. 2: Leveled architecture of hardware-oriented B+ Tree

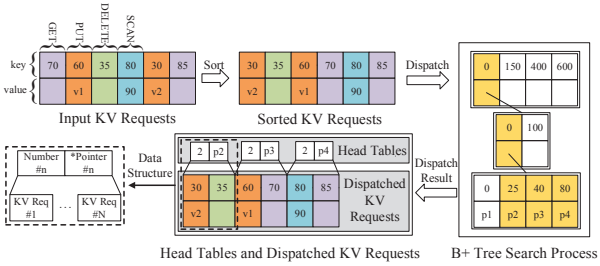

Fig. 3: Flow of a dispatch process

C. Dispatcher: Indexing, Dispatching and Updating

1) Indexing and Updating: To improve KV service throughput, HeteroKV decouples index and storage of KVS. A b+ tree-base data index is stored in FPGA BRAM and maintained by FPGA, based on which a high-performance KV requests *dispatcher* is built to help to improve efficiency of CPU cores. A 3-level b+ tree index is depicted on top of Fig. 1. KV requests are partitioned and dispatched to the CPU according to the leaf nodes they belong to. After update operations (i.e., PUTs and DELETEs) in the batched KV requests are processed by the CPU, the b+ tree-based index on the FPGA has to be maintained accordingly.

So as to improve the update efficiency, b+ tree update operations are sorted by CPU threads before being served. And by adopting a reservation station strategy, a fully pipelined *hardware-oriented B+ Tree* (HOBT) is proposed in this work, which is able to perform one search or update operation every two clock cycles. Fig. 2 shows the overall architecture as well as the search and update process of a 3-level HOBT. Different levels of the tree are maintained by different PEs. These PEs only differ from each other in their BRAM size. Their inner structure and pipeline scheduling are similar. Update operations of HOBT go through a downstream (left-to-right) and an upstream (right-to-left) process as shown , whereas the lookup operations only go through a downstream one.

2) Dispatching: To simplify the synchronization between the CPU and the FPGA, two message queues, named *KV requests queue* and *head tables queue*, are instantiated. The *head tables queue* and *KV requests queue* are written-only and read-only for FPGA and CPU. Data structures of them are shown in left bottom of Fig. 3. A head table is comprised of a **Pointer* and a *Number* entry, representing the pointer of a leaf node (i.e., hash table) and the number of KV requests falling into this leaf node respectively. FPGA will inform CPU to serve KV requests according to metadata in head tables once dispatch process is over. Fig. 3 demonstrates a dispatch process for a small number of KV requests. KV requests falling into the same leaf node will be served only by one thread to reduce memory footprint and inter-thread synchronization overhead.

D. Cache-aware Partitioned Hash Table

As discussed in Section II, Hash Tables support fast PUT and GET operations with the complexity of $\mathcal{O}(1)$, while falling back to brute force searches for SCAN operation. The inefficiency of such kind of brute force searches is rooted in large amount of memory accesses caused by cache misses. With the aid of high-performance FPGA *dispatcher*, block-grained hash tables are maintained by CPU threads. The size of the hash tables needs to be tuned according to the size of L2 Cache. Data structure of the *cache-aware partitioned hash tables* is shown in Fig. 1. A bitmap is used to indicate if a key is possible to exist in this hash table, which works as a bloom filter to reduce unnecessary memory accesses. A two-way Cuckoo hash algorithm is used to reduce hash collisions. KV pairs with key bigger than $(MinKey+MaxKey)/2$ will be moved into a new node when split happens. As KV requests falling into the same leaf node are served only by one thread in thread pool, the leaf node data only needs to be read once from memory and will reside in cache until all KV requests falling into it are served. After serving all KV requests of one leaf node, thread writes the corresponding returned KV pairs and b+ tree update operations into *returned KV pairs queue* and *b+ tree operations queue*, and master thread will inform FPGA to fetch data. For b+ tree operations caused by hash tables splits or deletions, slave threads push them into the sorted queue by mutual exclusion mechanism. For large-size leaf nodes, the number of b+ tree update operations caused by one batch of KV requests is very limited even for insert-intensive scenarios.

IV. Evaluation

In this section, we present the evaluation results of HeteroKV with regard to hardware resource usage and performance. HeteroKV was prototyped on a CPU-FPGA platform comprised of a Xeon Gold 6138 processor and a Stratix 10 FPGA interconnected through one PCIe 3.0 x16. The measured peak bandwidth of our system can reach 12 GB/s.

A. Methodology

We use YCSB workloads [5] as system benchmarks, with a default configuration that is zipfian distribution with 99% skewness. Key size and value size of KV pairs are both 32 bits, and SACN range is fixed with 100. Different workloads with different ratio of GET, PUT and SCAN operations are generated to test the performance.

On CPU side, we activate 19 slave threads and 1 master thread, and pin one thread per physical core to avoid cache pollution caused by hyperthreading. On FPGA side, HeteroKV runs at a clock frequency of 250MHz.

TABLE I: Resource utilization

	ALMs	BRAM Bits	DSP Blocks
Preprocess PE	16%	2%	0%
Dispatcher	19%	42%	0%
Other modules	7%	4%	0%
Total	42%	48%	0%

B. Resource Usage

TABLE I shows the resource usage of the different modules of HeteroKV. The design uses less than 50% of total resources. With the increase of network bandwidth and CPU compute power, HeteroKV is able to scale up by implementing larger-scale line-rate merge sorter and making more copies of b+ tree index.

TABLE II: Performance comparison with other KVS systems

KVS systems	Platform characteristics	Throughput (M queries/s)			Normalized throughput (M queries/s)		
		GET-heavy	PUT-heavy	SCAN-heavy	GET-heavy	PUT-heavy	SCAN-heavy
HeteroKV	FPGA, 250MHz; CPU, 20 cores	430	315	15.3	430	315	15.3
KV-Direct [8]	FPGA, 180MHz	180	114	1.8	250	158	2.5
PALM [11]	CPU, 12 cores	170	130	1.7	283	217	2.8
MICA [9]	CPU, 24 cores	137	135	1.4	114	113	1.1
HiKV [12]	CPU, 16 cores	33	14	1.5	41	18	1.9

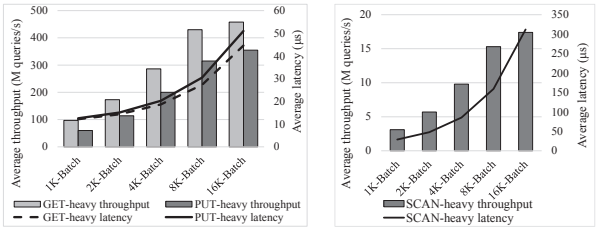

Fig. 4: Average throughput and latency under different batch sizes

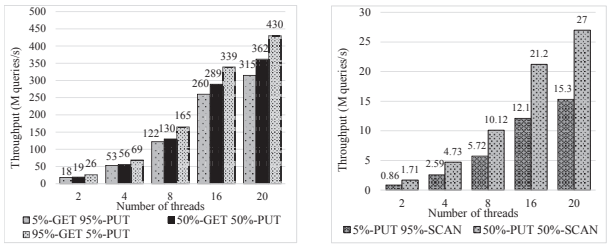

Fig. 5: Scalability of HeteroKV

C. Performance

1) Scalability and Latency: Fig. 4 presents the throughput and latency under different batch sizes. GET-heavy workloads are composed of 95% GETs and 5% PUTs. PUT-heavy workloads are composed of 95% PUTs and 5% GETs. SCAN-heavy workloads are composed of 95% SCANs and 5% PUTs. As shown, throughput of all workloads increases with the batch size. This is because with a larger batch size, more KV requests within a batch will fall into the same leaf node, which improves spatial locality thus CPU threads efficiency. However, a sharp increase in average latency is observed when batch size reaches 16K. According to the results above, a batch size of 8K is selected for HeteroKV. For PUTs and GETs, an average latency less than 35μs is acceptable [9], [11]. For SCANs, the average latency is less than 160μs.

Fig. 5 exhibits the scalability of the HeteroKV under different workloads. The number of threads includes a master thread and slave threads in thread pool. As depicted, the throughput scales nearly linearly with the number of threads. For GET-, PUT- and SCAN-heavy workloads, HeteroKV is able to achieve 430M, 315M and 15.3M queries per second (qps) respectively.

2) Comparison with Other Works: Table II shows the performance as well as normalized performance comparison with other works. For fairness consideration, throughput of other works is normalized according to their FPGA frequency and number of CPU cores. Because KV-Direct, MICA and PALM do not exhibit their SCAN performance in papers, we estimate their SCAN throughput as GET throughput divided by 100 as a SCAN with a fixed range of 100 must be split into 100 GETs. HiKV was designed to provide persistent KVS services on DRAM and NVM. The throughput of HiKV is bottlenecked by the lower bandwidth of NVM as well as persistency overhead. As demonstrated, HeteroKV outperforms existing works in all scenarios by leveraging both the parallel processing capability of FPGA and compute power of all CPU cores.

V. Conclusion

In this paper, we describe the design and evaluation of HeteroKV, a scalable line-rate KVS on heterogeneous CPU-FPGA platforms. By implementing a scalable line-rate merge sorter and a HOBT-based high-throughput KV requests dispatcher on FPGA, HeteroKV is able to achieve high scalability. Leveraging a cache-aware Hash Table, HeteroKV enables CPU to support fast search, insert and efficient scan operations simultaneously. Experiment results demonstrate the superior performance of HeteroKV.

References

[1] Redis. https://redis.io/.

[2] Berk Atikoglu, Yuehai Xu, et al. Workload analysis of a large-scale key-value store. In *ACM SIGMETRICS Performance Evaluation Review*, volume 40, pages 53–64. ACM, 2012.

[3] Sai Rahul Chalamalasetti, Kevin Lim, et al. An fpga memcached appliance. In *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, pages 245–254. ACM, 2013.

[4] Douglas Comer. Ubiquitous b-tree. *ACM Computing Surveys (CSUR)*, 11(2):121–137, 1979.

[5] Brian F Cooper, Adam Silberstein, et al. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154. ACM, 2010.

[6] Bin Fan, David G Andersen, and Michael Kaminsky. Memc3: Compact and concurrent memcache with dumber caching and smarter hashing. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 371–384, 2013.

[7] Brad Fitzpatrick. Distributed caching with memcached. *Linux journal*, 2004(124):5, 2004.

[8] Bojie Li, Zhenyuan Ruan, et al. Kv-direct: High-performance in-memory key-value store with programmable nic. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 137–152. ACM, 2017.

[9] Hyeontaek Lim, Dongsu Han, et al. Mica: A holistic approach to fast in-memory key-value storage. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 429–444, 2014.

[10] Andrew Putnam, Adrian M Caulfield, et al. A reconfigurable fabric for accelerating large-scale datacenter services. *ACM SIGARCH Computer Architecture News*, 42(3):13–24, 2014.

[11] Jason Sewall, Jatin Chhugani, Changkyu Kim, et al. Palm: Parallel architecture-friendly latch-free modifications to b+ trees on many-core processors. *Proc. VLDB Endowment*, 4(11):795–806, 2011.

[12] Fei Xia, Dejun Jiang, Jin Xiong, and Ninghui Sun. Hikv: a hybrid index key-value store for dram-nvm memory systems. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 349–362, 2017.

[13] Hao Zhang, Gang Chen, et al. In-memory big data management and processing: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 27(7):1920–1948, 2015.