Scalable and Distributed Key-Value Store-based Data Management Using RDMA-Memcached

Xiaoyi Lu, Dipti Shankar, Dhabaleswar K. (DK) Panda
Department of Computer Science and Engineering, The Ohio State University
{luxi, shankard, panda}@cse.ohio-state.edu

Abstract

Over the recent years, distributed key-value stores have been extensively used for designing scalable industry data management solutions. Being an essential component, the functions and performance of a distributed key-value store play a vital role in achieving high-speed data management and processing. Thus, many researchers and engineers have proposed various designs to continuously improve them. This paper first presents a survey of recent related works in the field of distributed key-value store and summarizes the research hotspots. Along these emerging research dimensions, we present a design overview of our proposed RDMA-Memcached library, which aims for achieving high throughput and low latency by taking full advantage of native InfiniBand features including RDMA and different transports (RC, UD, Hybrid), and high-performance storage such as SSDs. Though Memcached was originally used for caching online query results, we have implemented further extensions for Memcached to enrich its functions, APIs, and mechanisms, which extend its capabilities to more use cases. With our proposed designs, RDMA-Memcached can be used as a scalable data caching layer or a Burst Buffer component for both online data processing and offline data analytics. We also propose a set of benchmarks which are publicly available to help researchers to evaluate these advanced designs and extensions.

1 Introduction

In the Web 2.0/3.0 era, many social web services exist that deal with people's personal information and relationships (e.g. Facebook, Twitter), whereas a number of services also exist that store other information, such as goods (e.g. Amazon), photos (e.g. Flickr), travel (e.g., Yahoo! Travel), etc. These web-services essentially handle either traditional SQL-based or cloud NoSQL-based On-Line Data Processing (OLDP) workloads. Typically, most of the data that is accessed by these web-service applications are stored in databases and accessing these databases are often expensive in terms of latency and throughput. Since millions of users access these web-services, the servers must be equipped to deal with a large number of simultaneous access requests efficiently and fairly. Today, with cloud computing emerging as a dominant computing platform for providing scalable on-line services, there has been a tremendous increase in the number of systems developed and deployed for cloud data serving, thus making the performance and scalability of cloud OLDP applications extremely vital. Studies [1, 2] have shown that leveraging a distributed and scalable key-value store is invaluable to application servers in these scenarios.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Due to this diversified nature of social networking websites and other web services, massive amounts of data is constantly generated and is stored and processed on data centers in an offline fashion. These offline data analytics tasks have necessitated the design of Big Data middleware (e.g., Hadoop, Spark, etc.) that is capable of delivering high-performance. Traditionally, disk-centric distributed file systems such as the Hadoop Distributed File System (HDFS) have been employed as the storage layer for offline Big Data analytics. But researchers found memory-centric distributed file systems such as Alluxio (formerly Tachyon) [3] can provide higher throughput and scalability for these workloads. Inspired by the 'in-memory I/O' concept, recent research works [4, 5, 6] have started to explore the use of key-value store based burst-buffer systems to boost I/O performance of Big Data applications.

Because of these usage requirements on distributed key-value stores, the classical designs and implementations such as Memcached [7] have become an essential component of today's scalable industry data management solutions. Being an elementary component, the functions and performance of a distributed key-value store play a crucial role in achieving high-speed data management and processing. Thus, many novel designs and enhancements have been proposed to continuously improve them. Recent studies, such as [8, 3, 12, 12], have shed light on the fact that the novel designs of Memcached for clusters with Remote Direct Memory Access (RDMA) capable interconnects have demonstrated that the latency of Memcached operations can be significantly reduced. Research works, including [13, 14, 15, 16] propose several hybrid, high-performance key-value stores, which can deliver higher overall performance for data-intensive workloads, by taking advantage of high-speed flash-based storage such as SATA SSDs, PCIe-/NVMe-based SSDs.

Thus, it is essential to systematically study and understand the performance characteristics, advantages, and trade-offs of key-value stores that exist on modern multi-core, networking, and storage architectures. In the light of these facts, we need to consider the following broad questions and challenges for achieving scalable key-value store-based data management and processing: **1)** What kind of advanced designs have researchers proposed in the literature and community for improving the functions and performance of distributed key-value stores? **2)** How can existing key-value stores such as Memcached be re-designed from the ground up to utilize RDMA and different transports available on high-performance networks? What kind of new extensions can be proposed for fully exploiting the benefits of high-performance storage, such as SSDs? **3)** How to enrich the functions, APIs, and mechanisms of current-generation key-value stores, to extend its capabilities to more use cases? **4)** How to design a set of benchmarks to help researchers and engineers understand these new features for key-value stores that capture the requirements of different key-value store application scenarios? **5)** Can we show performance benefits for some concrete use cases by using these advanced designs for key-value stores?

To address the above challenges, this paper is organized as follows. In Section 2, we first present a survey of highly cited works in this field, and summarize the research hotspots, to help understand the challenges of obtaining high-performance for distributed key-value stores. In Section 3, we present a design overview of our proposed RDMA-Memcached library [17], which aims for achieving high throughput and low latency by fully taking advantage of native InfiniBand features, such as RDMA and different transports (RC, UD, Hybrid). In Section 4, we have undertaken further extensions (i.e., hybrid memory, non-blocking APIs, and burst buffer design with efficient fault-tolerance support) into Memcached to extend its capabilities to more use cases. We also propose a set of benchmarks to help researchers to evaluate these advanced designs and extensions in Section 5. With our proposed designs, RDMA-Memcached can be used as a scalable data caching layer for online data processing workloads or a Burst Buffer component for offline data analytics. We show some use cases and benefits in Section 6. Section 7 concludes the paper.

2 A Survey on Related Work

Through thorough studies of highly cited works in the field of high-performance key-value store design, five major research hotspots have been identified, including network I/O, data management, fault tolerance, hardware acceleration, and usage scenario. A brief overview of these five dimensions along with works that have focused on each of these individual aspects is presented below.

Network I/O: Key-value store operations are network-intensive by nature. With the emergence of high performance interconnects (e.g., InfiniBand, RDMA over Converged Ethernet/RoCE), several research works have focused on the network perspective. These works are represented in the first five rows in Table 1. Research works including HERD [8], Pilaf [12], FaRM [3], and HydraDB [19] propose different ways to design a low-latency communication engine that leverages zero-copy and CPU-bypass features of RDMA, while exploiting multi-core architectures. Similarly, MICA [12] designs a key-value store that maps client requests directly to specific CPU cores at the server NIC level, using Intel DPDK along with the UDP protocol.

Key-Value Data Management: Most key-value stores contain two important data structures: (1) A hash table or tree to lookup the key-value pairs stored, and, (2) a list or log that stores the actual key-value pairs. These data structures are co-designed with the underlying hardware, network protocol, and available storage for the best performance. Some works such as MICA [12], FaRM [3] take a holistic approach to design the key-value data structures with their communication engines. The data management can be divided into the following categories:

- Hashing: The hashing scheme employed is vital for inserting and looking up data quickly and consistently in a multi-client and multi-threaded server environment that leverages the large number of CPU cores available on the nodes in today's data centers and scientific compute clusters. One such notable research work, MemC3 [20], proposed a new hashing scheme for read-heavy workloads. Several general studies on hashing like optimistic hashing, cuckoo hashing have been experimented with and used to implement key-value stores.

- Cache Management: An efficient cache must be able to store and retrieve data quickly, while conserving the available memory. Most volatile in-memory key-value stores have intelligent eviction techniques that aim to minimize the miss ratio and memory usage, and in turn improve the average response time. While Memcached uses LRU-based eviction, Twitter memcached [21] proposed configurable eviction policies. Research works such as LAMA [22] propose more adaptive techniques for automatically repartitioning available memory at the Memcached servers by leveraging locality inherent in the Memcached requests to reduce the miss ratio.

- Storage Architecture: While the works mentioned above (first ten rows of Table 1) are meant for volatile in-memory (i.e., DRAM) stores, several works such as NVMM [23] in Table 1 have focused on leveraging non-volatile byte-addressable main memory. Along similar lines, fatcache [16] exploits high-speed SSDs to extend available storage capacities of the server.

Fault-Tolerance and Data Availability: Though traditionally in-memory key-value stores were volatile in nature, with their increased use, fault-tolerance and data availability are desirable features. In-memory replication is a popular technique for enabling fault-tolerance that is implemented in FaRM [3] and HydraDB [19]. On the other hand, research works such as Cocytus [24] are leveraging an alternative and more memory-efficient fault technique, namely Erasure Coding (EC), to design an efficient and resilient version of Memcached.

Hardware Acceleration: Hardware-based acceleration that leverages InfiniBand HCAs, GPGPUs, Intel KNL, etc., plays a prominent role in speeding up applications on modern HPC clusters; they are paving their way into data center environments. While key-value store operations are not primarily compute-intensive, offloading indexing and communication to the underlying accelerators can help define better overlap of concurrent data operations to increase overall throughput and client's performance. Mega-KV [25] is a fast in-memory key-value store that maintains a hash table on the GPGPU and utilizes its high memory bandwidth and latency hiding capability to achieve high throughput. On the other hand, FPGA-Memcached [27] is a Memcached implementation that employs a hybrid CPU+FPGA architecture. Similarly, MemcachedGPU [26] presents a

GPU-accelerated networking framework to enhance the throughput of Memcached to exploit massively parallel GPUs for batched request processing at the servers.

Usage Scenario: Key-value based solutions are vital for accelerating data-intensive applications. The two major categories of Big Data workloads that currently leverage key-value stores include: (1) Online workloads, such as database query caching, and, (2) Offline workloads, such as I/O-intensive Hadoop or Spark workloads over parallel file systems, graph processing, etc. FaRM [3] can be used as a query cache and has also been used to implement offline graph processing stores. Similarly, HydraDB [19] can be used to accelerate cloud-based NoSQL key-value store workloads and offline Hadoop/Spark workloads such as Sort.

Our proposed work, RDMA-Memcached [17, 8, 13, 28], attempts address various factors in each of the five identified hotspots. To improve network I/O performance, RDMA-Memcached leverages RDMA semantics on modern high-performance interconnects. For improving data management, it leverages a log-structured SSD-assisted extension to increase hit rate without compromising performance, while also proposing non-blocking API extensions. RDMA-Memcached can support both replication and online erasure coding schemes for achieving fault tolerance support [35]. Through careful designs, RDMA-Memcached can be used to accelerate several online and offline Big Data workloads.

Research Work	Network I/O	Key-Value Data Management	Fault-Tolerance	Hardware Acceleration	Usage Scenario
MICA [12]	Intel DPDK + UDP	Keyhash-based partition + Lossy concurrent hash+ Append-only log	-NA-	Zero-Copy NIC-to-NIC	Online
HERD [8]	RDMA/RoCE	Based on MICA's hashing design	-NA-	IB/40GigE HCA	Online
FaRM [3]	RDMA/RoCE	Hopscotch hashing	Replication	IB/40GigE HCA	Online/ Offline
Pilaf [12]	RDMA/RoCE	2-4 cuckoo hashing + Self-verifying data structs + async logging	-NA-	IB/40GigE HCA	Online
HydraDB [19]	RDMA/RoCE	NUMA-aware + compact hash + remote pointer sharing	Replication	IB/40GigE HCA	Online/ Offline
MegaKV [25]	Based on MICA	Based on MICA's caching design	-NA-	GPU-offload for indexing	Online
Memcached-GPU [26]	TCP/UDP	Set-associative hash + Local LRU per hash set	-NA-	GPU Network offload mgmt.	Online
MemC3 [20]	TCP/UDP	Optimistic cuckoo hash + Concurrent reads w/o locks + CLOCK cache management	-NA-	-NA-	Online
LAMA [22]	TCP/UDP	Locality-aware memory repartitioning based on footprints	-NA-	-NA-	Online
FPGA-Memcached [27]	TCP/UDP	Based on Default Memcached	-NA-	FPGA acceleration	Online
NVMM [23]	TCP/UDP	Memcached w/ non-volatile main memory support	Durable writes w/ NVMM	-NA-	Online
Cocytus [24]	TCP/UDP	Based on Default Memcached	Hybrid EC and replication	-NA-	Online
FatCache [16]	TCP/UDP	Default Memcached w/ SSD eviction	-NA-	-NA-	Online
RDMA-Memcached [17]	RDMA/RoCE [8, 28]	Hybrid Memory extension w/ SSD [13]	Replication/ EC [35]	IB/40GigE HCA	Online [37]/ Offline [6]

Table 1: Research Hotspots Addressed in Recent Distributed Key-Value Store Solutions

3 Design Overview of RDMA-Memcached

Figure 1 depicts the design overview of our proposed RDMA-Memcached. To enable RDMA-based communication for Memcached, we have to redesign the communication substrate in Memcached server and Libmemcached client components. In the server side, we also propose a hybrid memory scheme with SSD-based slab management to enlarge the caching capacity. For the client side, to achieve better overlapping among communication, computation, and SSD accessing, we propose a new set of non-blocking APIs for Libmemcached. We also extend the current fault-tolerance mechanism in Memcached from replication-only to 'replication & erasure coding.' Advanced designs with accelerators (such as Nvidia GPGPU, Intel Xeon Phi) will be available soon. Our designs can support all the existing Memcached applications with no changes needed. For offline data analytics, our proposed non-blocking APIs and Burst-Buffer mode will deliver the best performance.

3.1 RDMA-based Communication Substrate

The current-generation Memcached library is designed with the traditional BSD Sockets interface. While the Sockets interface provides a great degree of portability, the byte-stream model within Sockets interface mismatches with Memcached's memory object model. In addition, Sockets-based implementations internally need to copy messages, resulting in further loss of performance. High-performance interconnects and their software APIs, such as OpenFabrics Verbs [29], provide RDMA capability with memory-based semantics that fits very well with the Memcached model. Scientific and parallel computing domains use the Message Passing Interface (MPI) as the underlying basis for most applications. MPI libraries, such as MVAPICH2 [30], can achieve very low one-way latencies in the range of 1-2 us. Whereas, even the best implementation of Sockets on InfiniBand is still much slower.

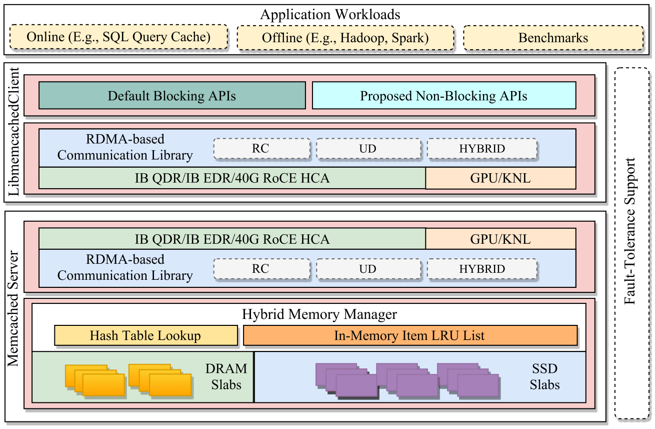


Figure 1: Design Overview of RDMA-Memcached

Inspired by these observations and experiences, we propose an RDMA-based communication substrate [8] over native Verbs APIs for Memcached. Our communication substrate exposes easy-to-use Active Message based APIs that can be used by Memcached, without re-implementing performance critical logic (like buffer management, flow control, etc.). Our design is optimized for both small and large messages. If the amount of data being transferred fits into one network buffer (e.g., ≤8 KB), it is packaged within one transaction and the communication engine will do one eager send for it. With this, we can save handshaking overhead, which is needed for RDMA operations. For large messages, we use RDMA Read based approach to transfer data with high-performance and overlapping. These designs can work directly with both InfiniBand and RoCE networks. We have done extensive performance comparisons of our Memcached design with unmodified Memcached using Sockets over RDMA, 10 Gigabit Ethernet network with hardware-accelerated TCP/IP, and IP-over-IB protocols on different generation InfiniBand HCAs. Our performance evaluation reveals that the latencies of RDMA-Memcached are better than those schemes by up to a factor of 20. Further, the throughput of small Get operations can be improved by a factor of six when compared to Sockets over 10 Gigabit Ethernet network. A similar factor of six improvement in throughput is observed over Sockets Direct Protocol.

3.2 Multi-Transport (RC/UD/Hybrid) Support

The basic design with RDMA mentioned above has shown that the use of RDMA can significantly improve the performance of Memcached. This design is based on InfiniBand as connection-oriented Reliable Connection (RC) transport. However, exclusive use of RC transport hinders scalability due to high memory consumption. The reason is that a typical RC connection with standard configuration parameters requires around 68 KB [31] of memory and RC requires a distinct QP per communicating peer. Since a Memcached server may need to serve thousands of clients simultaneously, this may cause really high memory consumption which limits the scalability of RDMA-based designs for Memcached. On the other hand, the Unreliable Datagram (UD) transport of InfiniBand addresses this scalability issue because UD is a connectionless transport, and a single UD QP can communicate with any number of other UD QPs. This can significantly reduce the memory consumption for a large number of connections. However, UD does not offer RDMA, reliability, and message ordering. Moreover, messages larger than a Maximum Transmission Unit (MTU) size (4 KB in current Mellanox hardware) have to be segmented and sent in MTU size chunks.

Due to these, we find none of these transports can achieve both high-performance and scalability as they are currently designed. Pure RC-based Memcached designs may deliver good latency and throughput, but may not have the best scalability. Pure UD-based Memcached designs may deliver good scalability, but may not deliver the same performance as Memcached with RC. Thus, we introduce a hybrid transport model [9] which leverages the best features of RC and UD to deliver both high-performance and scalability for Memcached. The idea behind this is we can limit the maximum number of RC connections to a specified threshold. Further connections can be made over UD transport. This ensures that the memory requirement does not increase above a particular limit. Another feature is to dynamically change the transport mode from UD-to-RC or RC-to-UD, based on the communication frequency between the server and a specific client, and we can also impose a limit on the maximum number of RC connections. These hybrid modes can be selected depending on platform characteristics. For UD transport, we proposed novel designs to handle the reliability, flow control, and large message transfers. With all of these, our performance evaluations reveal that the hybrid transport delivers performance comparable to that of RC while maintaining a steady memory footprint as that of UD. The scalability analysis with 4,096 client connections reveals that the hybrid transport achieves good memory scalability.

4 Advanced Extensions for RDMA-Memcached

4.1 Hybrid Memory Extensions

Internet-scale distributed key-value store-based caches have become a fundamental building block of several web architectures in the recent years, for scaling throughput and reducing overall application server access latency. However, the issues of cost and power requirements limit the possibilities for increasing the DRAM capacities available at these caching servers. With the advent of high-performance storage (e.g. SATA SSD, PCIe-/NVMe-SSD) that offer superior random I/O performance in comparison to conventional disk storage, several efforts have been directed towards employing 'RAM+SSD' hybrid storage architectures for extending in-memory key-value stores, with the goal of achieving high data retention while continuing to support millions of operations per second. Such solutions to increasing caching layer capacities without sacrificing performance have been explored by Twitter [16], Facebook [15], and many others.

We proposed a hybrid memory scheme for RDMA-enhanced Memcached [17, 13], which has an SSD-assisted design that extends the in-memory slabs with SSD-based slabs to enlarge the data capacity available to the Memcached servers. It employs the existing in-memory lookup table to store the key i.e., the index of the data object, and a log-structured SSD storage to store the value of evicted data objects. For Memcached Set requests, a newly inserted object is cached in the DRAM or in-memory slabs, and the index points to the in-memory location. When the in-memory slabs are full, the cached objects are compacted and evicted to the

SSD from the LRU list on a per-slab basis using batched writes; the corresponding indices in the lookup table are updated to point to the appropriate file offset. Similarly, Memcached Get requests are served either from the in-memory slabs or the SSD using random reads. Garbage collection features are provided via lazy reclamation, and it keeps the data in the SSD up-to-date. With higher data retention enabled through the use of high-speed SSDs, the Memcached caching layer can provide better hit rate, and in turn increased performance.

4.2 Non-Blocking API Extensions

Blocking key-value request semantics are inherent to most online Big Data applications; data needs to be available to the client upon request. Additionally, these tend to be read-mostly workloads. Recently, several Offline Big Data analytical workloads have started to explore the benefits of leveraging high-performance key-value stores. Prominent examples include a Memcached-based Burst Buffer layer for the Lustre parallel file systems [6, 4], a Memcached-based Burst-buffer for HDFS [5], and intermediate data caching with Memcached in Hadoop MapReduce [32]. These write-heavy workloads incorporate different I/O and communication characteristics compared to the traditional online workloads. On the other hand, while the SSD-assisted hybrid memory design for RDMA-based Memcached can guarantee a very high success rate due to its property of high data retention, studies have shown that these designs have two major bottlenecks: (1) The current hybrid design with RDMA-based Memcached provides support for blocking Memcached Set/Get APIs only, i.e., memcached_set and memcached_get, which mandates that the client waits until the operations complete at the Memcached server, and, (2) the server-side SSD I/O overhead is in the critical path of the Memcached Set/Get request.

To alleviate these bottlenecks, new non-blocking extensions to the existing Libmemcached APIs have been proposed for RDMA-based Hybrid Memcached. The proposed APIs, i.e., `memcached_iset/ iget` and `memcached_bset/bget` [17, 28], introduce mechanisms to fetch and store data into Memcached in a non-blocking manner. The proposed extensions allow the user to separate the request issue and completion phases. This enables the application to proceed with other tasks, such as computations or communication with other servers while waiting for the completion of the issued requests. The RDMA-based Libmemcached client library benefits from the inherent one-sided characteristics of the underlying RDMA communication engine and ensures the completion of every Memcached Set/Get request. In this way, we can hide significant overheads due to blocking waits for server response at the client side and SSD I/O operations at the server side. In addition to bridging the performance gap between the hybrid and pure in-memory RDMA designs, the server throughput of the hybrid Memcached design can be increased by about overlapping concurrent Set/Get requests.

4.3 Burst-Buffer Design Extensions

Key-Value Store-based Burst-Buffer: Distributed key-value store-based caching solutions are being increasingly used to accelerate Big Data middleware such as Hadoop and Spark on modern HPC clusters. However, the limitation of local storage space in these environments, due to the employment of the Beowulf architecture, has placed an unprecedented demand on the performance of the underlying shared parallel storage (e.g., Lustre, GPFS). Memory-centric distributed file systems such as Alluxio [3] have enabled users to unify heterogeneous storage across nodes for high throughput and scalability. While this approach is being explored to develop a two-tier approach for Big Data on HPC clusters [33], it was not designed specifically for traditional HPC infrastructure, as it still depends on data locality for best performance. Such an I/O contention to shared storage can be alleviated using a burst-buffer approach. Research works [6, 4] are actively exploring employing generic key-value stores such as Memcached to build burst-buffer systems.

We propose a hybrid and resilient key-value store-based Burst-Buffer system (i.e., Boldio [6]) over Lustre for accelerating I/O-phase that can leverage RDMA on high-performance interconnects and storage technologies. Boldio provides efficient burst-buffer client/server designs that enable optimal I/O request overlap via non-blocking API semantics and pipelined stages. It employs a client-initiated replication scheme, backed by a shared-nothing asynchronous persistence protocol, to ensure resilient flushing of the buffered I/O to Lustre. To enable leveraging this light-weight, high-performance, and resilient remote I/O staging layer, we design a file system class abstraction, `BoldioFileSystem`, as an extension of the Hadoop `LocalFileSystem`. Thus, many Hadoop and Spark based Big Data applications can leverage our design transparently.

Resilience Support with Erasure Coding in Key-value Stores: To enable the resilience that is critical to Big Data applications, it is necessary to incorporate efficient fault-tolerance mechanisms into high-performance key-value stores such as RDMA-Memcached that are otherwise volatile in nature. In-memory replication is being used as the primary mechanism to ensure resilient data operations. The replication can be initiated by the client, as done by Libmemcached and its RDMA-based variant, or the Memcached server can be enhanced to replicate data to other servers in the cluster (however, this requires modifying the underlying shared-nothing architecture of Memcached). The replication scheme incurs increased network I/O with high remote memory requirements.

On the other hand, erasure coding [34] is being extensively explored for enabling data resilience, while achieving better storage efficiency. Several works [24, 35] are exploring the possibilities of employing Online Erasure Coding for enabling resilience in high-performance key-value stores. With erasure coding, each data entity is divided into K fragments and encoded to generate M additional fragments. These K + M fragments are then distributed to N unique nodes. Any K fragments can be used to recover the original data object, which means erasure coding can tolerate up to M simultaneous node failures. Since erasure coding requires a storage overhead of N/K where (K<N), as compared to the high overhead of M+1 (i.e., replication factor) that is incurred by replication, it provides an attractive alternative to enable resilience in key-value stores with better memory efficiency. While this provides a memory-efficient resilience, it also introduces a new compute phase in the critical path of each Set/Get request. This necessitates us to design efficient methods to overlap and hide these overheads to enable high performance. The new APIs proposed in Section 4.2 enable us to design a non-blocking request-response engine that can perform efficient Set/Get operations by overlapping the encoding/decoding involved in enabling Erasure Coding-based resilience with the request/response phases, by leveraging RDMA on high performance interconnects.

5 Benchmarks

Since key-value stores like Memcached are mostly used in an integrated manner, it is vital to measure and model performance based on the characteristics and access patterns of these applications. On the other hand, the emergence of high-performance key-value stores that can operate well with 'RAM+SSD' hybrid storage architecture and non-blocking RDMA-Libmemcached API semantics have made it essential for us to design micro-benchmarks that are tailored to evaluate these upcoming novel designs.

5.1 Micro-Benchmarks

Micro-benchmarks suites such as Mutilate [36] are designed with load generation capabilities to model real-work Memcached workloads (e.g., Facebook), with high request rates, good tail-latency measurements, and realistic request stream generation. Our proposed OSU HiBD (OHB) Memcached micro-benchmarks [17] model the basic Set/Get latencies that can be used to measure the performance of RDMA-based Memcached design. This micro-benchmark suite provides benchmarks to measure the request latencies using both blocking and non-blocking RDMA-Libmemcached APIs. These benchmarks are mainly designed to model the performance of RDMA-Memcached on different clusters with high-performance interconnects such as InfiniBand, RoCE, etc.

5.2 Hybrid Memory Benchmarks

Higher data retention is a desirable feature of the Memcached server cluster. The failure to retrieve data from key-value store-based caching layer due to the insufficient space in the caching tier can be caused by the consistency requirements or the caching tier's eviction policy; in this case, data needs to be fetched from the back-end database. However, most benchmarks are used to evaluate performance in a stand-alone manner. To model the effect of the back-end database into the stand-alone micro-benchmarks [37], we introduce hybrid Memcached micro-benchmarks with a pluggable module for analysis and prediction of an estimate of the caching tier miss-penalty. This module is designed to get an estimate on the database access latency (i.e., miss-penalty) that a request must incur if it can not get the data in the caching tier and have to access the back-end database. This is just a one-time process for the database used. The miss-penalty estimated by the prediction module is added to the overall execution time for every query that incurs a caching tier miss. In this manner, the overhead of using the database is minimal, and it needs not be involved while running the actual tests; giving us the opportunity to tune the parameters of the hybrid design for best performance.

5.3 YCSB with Memcached

Cloud serving workloads are key-value store-based workloads that give higher priority to scalability, availability, and performance over consistency of the data. YCSB [39] is a typical benchmark for these cloud data processing applications. These services are often deployed using NoSQL-like distributed key-value stores. YCSB consists of an extensible workload generator that models a variety of data serving systems into five core workloads. The workload executor, i.e., the YCSB clients loads the generated data into the data store, generates an access pattern based on the chosen workload or input parameter selection, and evaluates the underlying store to benchmark the performance and scalability of the cloud serving system. The YCSB distribution has a plugin for Spymemcached Java client [38]. For benchmarking RDMA-based Memcached, we leverage the flexible YCSB framework to develop a new client interface using JNI to design an RDMA-aware YCSB client for benchmarking hybrid RDMA-Memcached design. Memcached-over-MySQL with YCSB is also presented in [37].

5.4 Bursty I/O Benchmarks

The OHB micro-benchmarks presented in Section 5.1 can be extended to represent a block-based pattern to mimic bursty I/O workloads [28], that are inherent in various HPC workloads including (1) Checkpoint-Restart for scientific applications (e.g., MPI), and, (2) Hadoop or Spark over parallel file systems such as Lustre (described in Section 4.3). Data blocks are read and written as smaller chunks that can fit into key-value pairs within Memcached. Bursty I/O micro-benchmarks can model the chunk or key-value pair size, the overall workload size, data access pattern, workload operation mix (read:write operations per client) to study different dimensions of the performance of a key-value store-based burst buffer layer. Specifically, to study the advantages of I/O request overlap, a large iteration of non-blocking Set/Get requests can be issued and monitored in an asynchronous fashion to ensure data requests have successfully completed.

6 Use Cases and Benefits

Leveraging a distributed key-value store based caching layer has proven to be invaluable for scalable data-intensive applications. To illustrate the probable performance benefits, we present two use cases:

6.1 Use Case: Online Caching

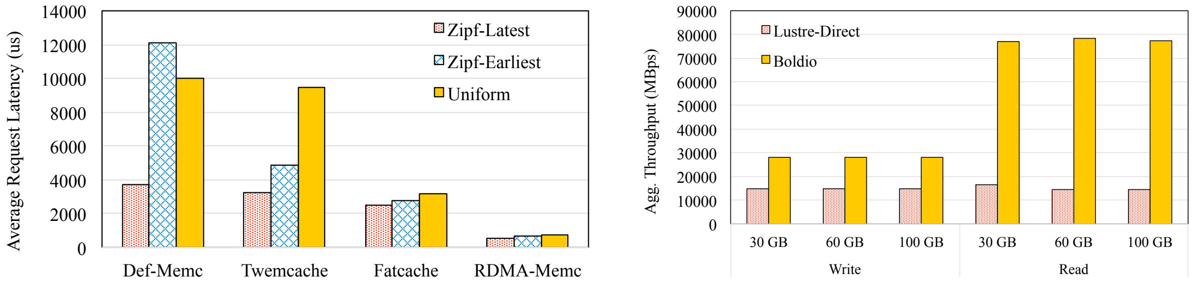
To illustrate the benefits of leveraging RDMA-Memcached to accelerate MySQL-like workloads, we employ the hybrid micro-benchmarks presented in Section 5.2 on an Intel Westmere cluster; each node is equipped with 12 GB RAM, 160 GB HDD, and MT26428 InfiniBand QDR ConnectX interconnects (32 Gbps data rate) with PCI-Ex Gen2 interface running RHEL 6.1. We use 8-16 compute nodes to emulate clients, running for both the socket-based (Libmemcached v1.0.18) and RDMA-enhanced implementation (RDMA-Libmemcached 0.9.3).

We run these experiments with 64 clients, 100% reads, an aggregated workload size of 5 GB, a key-value size of 32 KB, and an estimated average miss penalty of 95 ms.

We compare various Memcached implementations including default Memcached [7], twemcache [21], fatcache [16] and RDMA-Memcached [17]. From Figure 2a, for the uniform access pattern, where each key-value pair stored in the Memcached server cluster is equally likely to be requested, we observe that SSD-assisted designs such as fatcache and RDMA-Memcached can give the best performance due to their high data retention. Specifically, RDMA-Memcached can improve performance by up to 94% over in-memory default Memcached designs (including twemcache). We study the performance with two skewed access patterns, namely, Zipf-Latest (Latest-Is-Popular) and Zipf-Earliest (Earliest-Is-Popular), which model key-value pair popularity based on the most recently and least recently updated key-value pairs, respectively. As compared to the other solutions in Figure 2a, we observe that RDMA-Memcached can improve performance by about 68%–93%.

6.2 Use Case: Offline Burst Buffer

To illustrate the performance of RDMA-Memcached for accelerating Big Data I/O workloads, we evaluate the proposed Boldio design, presented in Section 4.3, with Hadoop running directly over Lustre parallel file system. We use the production-scale cluster, namely, SDSC Gordon [40], that is made up of 1,024 dual-socket compute nodes with two eight-core Intel Sandy Bridge processors and 64 GB DRAM, connected via InfiniBand QDR links. This cluster has sixteen 300GB Intel 710 SSDs distributed among the compute nodes. It consists of a Lustre setup with 1 PB capacity. We use 20 nodes on SDSC Gordon for our experiments. For Boldio, we use 16 compute nodes on Cluster A as YARN NodeManagers to launch Hadoop map/reduce tasks and a 4-node Boldio burst-buffer cluster over Lustre (replication = 2). For fair resource distribution, we use a 20-node Hadoop cluster for the Lustre-Direct mode. From Figure 2b, we observe that by leveraging non-blocking RDMA-enabled key-value store semantics over SSD-assisted Memcached server design, Boldio can sustain the 3x and 6.7x improvements in read and write throughputs over default Hadoop running directly over Lustre.



(a) Memcached Set/Get over simulated-MySQL

(b) Hadoop TestDFSIO Throughput with Boldio

Figure 2: Performance Benefits with RDMA-Memcached based Workloads

7 Conclusion

In this paper, we have described our approach to enhance and extend the design of Memcached in following five dimensions: 1) We have described a novel design of Memcached over RDMA capable networks. The RDMA-based design significantly improves the performance of Memcached software. 2) We introduced a hybrid transport model for RDMA-Memcached which takes advantage of the best features of RC and UD to deliver better scalability and performance than that of a single transport. 3) We proposed a hybrid memory extension for RDMA-Memcached, which enlarges aggregated cache capacity with SSD-based slab designs. 4) A new set of non-blocking APIs were proposed to extend the existing Libmemcached APIs for achieving higher communication, processing, and I/O overlapping. 5) To expand the usage scope of Memcached to accelerate I/O in Big Data workloads such as Hadoop, Spark, etc., we proposed an RDMA-Memcached based Burst-Buffer system, which obtained the performance benefits from all above-mentioned designs as well as fault-tolerance support with replication and erasure coding.

To evaluate these advanced designs and extensions, we also proposed a set of benchmarks. To show the benefits from RDMA-Memcached designs, we have shown the use cases and the associated benefits of using RDMA-Memcached as a scalable data caching layer for online data processing workloads or a Burst Buffer component for offline data analytics. Memcached libraries with some of these designs and benchmarks are publicly available from the High-Performance Big Data (HiBD) [17] project. In the future, we intend to look at extending our designs to run in a heterogeneous environment for further taking advantage of accelerators, such as GPGPU and Intel KNL. We also plan to explore more use cases and applications with our designs.

Acknowledgments. This research was funded in part by National Science Foundation grants #CNS-1419123 and #IIS-1447804. It used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number OCI-1053575.

References

[1] Oracle Corp. Designing and Implementing Scalable Applications with Memcached and MySQL, A MySQL White Paper. *Tech. Report*, 2010.

[2] D. Shankar, X. Lu, J. Jose, M. Wasi-ur-Rahman, N. Islam, and D. K. Panda. Can RDMA Benefit Online Data Processing Workloads on Memcached and MySQL? *ISPASS*, 159-160, 2015.

[3] H. Li, A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica. Tachyon: Reliable, Memory Speed Storage for Cluster Computing Frameworks. *SoCC*, 6:1-15, 2014.

[4] T. Wang, S. Oral, Y. Wang, B. Settlemyer, S. Atchley, and W. Yu. BurstMem: A High-Performance Burst Buffer System for Scientific Applications. *IEEE BigData*, 2014.

[5] N. Islam, D. Shankar, X. Lu, M. Wasi-ur-Rahman, and D. K. Panda. Accelerating I/O Performance of Big Data Analytics on HPC Clusters through RDMA-based Key-Value Store. *ICPP*, 2015.

[6] D. Shankar, X. Lu, and D. K. Panda. Boldio: A Hybrid and Resilient Burst-Buffer Over Lustre for Accelerating Big Data I/O. *IEEE BigData*, 2016.

[7] Memcached: High-Performance, Distributed Memory Object Caching System. *http://memcached.org/*, 2017.

[8] J. Jose, H. Subramoni, M. Luo, M. Zhang, J. Huang, M. Wasi-ur-Rahman, N. Islam, X. Ouyang, H. Wang, S. Sur, and D. K. Panda. Memcached Design on High Performance RDMA Capable Interconnects. *ICPP*, 2011.

[9] J. Jose, H. Subramoni, K. Kandalla, M. Wasi-ur-Rahman, H. Wang, S. Narravula, and D. K. Panda. Scalable Memcached Design for InfiniBand Clusters Using Hybrid Transports. *CCGrid*, 236-243, 2012.

[10] A. Dragojević, D. Narayanan, O. Hodson, and M. Castro. FaRM: Fast Remote Memory *NSDI*, 401-414, 2014.

[11] H. Lim, D. Han, D. G. Andersen, and M. Kaminsky. MICA: A Holistic Approach to Fast In-Memory Key-Value Storage. *NSDI*, 2014.

[12] C. Mitchell, Y. Geng, and J. Li. Using One-Sided RDMA Reads to Build a Fast, CPU-Efficient Key-Value Store. *USENIX ATC*, 2013.

[13] X. Ouyang, N. Islam, R. Rajachandrasekar, J. Jose, M. Luo, H. Wang, and D. K. Panda. SSD-Assisted Hybrid Memory to Accelerate Memcached over High Performance Networks. *ICPP*, 470-479, 2012.

[14] H. Lim, B. Fan, D.G. Andersen, and M. Kaminsky. SILT: A Memory-Efficient, High-Performance Key-Value Store. *SOSP*, 1-13, 2011.

[15] Facebook. McDipper: A Key-Value Cache for Flash Storage. *https://www.facebook.com/notes/facebook-engineering/mcdipper-a-key-value-cache-for-flash-storage/10151347090423920*, 2013.

[16] Twitter. fatcache: Memcached on SSD. *https://github.com/twitter/fatcache*.

[17] OSU NBC Lab. High-Performance Big Data (HiBD). *http://hibd.cse.ohio-state.edu*.

[18] A. Kalia, M. Kaminsky, and D. G. Andersen. Using RDMA Efficiently for Key-Value Services. *SIGCOMM*, 2014.

[19] Y. Wang, L. Zhang, J. Tan, M. Li, Y. Gao, X. Guerin, X. Meng, and S. Meng. HydraDB: A Resilient RDMA-driven Key-Value Middleware for In-Memory Cluster Computing. *SC*, 22:1-11, 2015.

[20] B. Fan, D. G. Andersen, and M. Kaminsky. MemC3: Compact and Concurrent MemCache with Dumber Caching and Smarter Hashing. *NSDI*, 371-384, 2013.

[21] Twitter. twemcache: Twitter Memcached. *https://github.com/twitter/twemcache*.

[22] X. Hu, X. Wang, Y. Li, L. Zhou, Y. Luo, C. Ding, S. Jiang, and Z. Wang. LAMA: Optimized Locality-Aware Memory Allocation for Key-Value Cache. *USENIX ATC*, 57-59, 2015.

[23] Y. Zhang, and S. Swanson. A Study of Application Performance with Non-Volatile Main Memory. *MSST*, 1-10, 2015.

[24] H. Zhang, M. Dong, and H. Chen. Efficient and Available In-Memory KV-Store with Hybrid Erasure Coding and Replication. *FAST*, 167-180, 2016.

[25] K. Zhang, K. Wang, Y. Yuan, L. Guo, R. Lee, and X. Zhang. Mega-KV: A Case for GPUs to Maximize the Throughput of In-Memory Key-Value Stores. *VLDB*, 1226-1237, 2015.

[26] T. H. Hetherington, M. O'Connor, and T. M. Aamodt. MemcachedGPU: Scaling-up Scale-out Key-value Stores. *SoCC*, 43-57, 2015.

[27] S. R. Chalamalasetti, K. Lim, M. Wright, A. AuYoung, P. Ranganathan, and M. Margala. An FPGA Memcached Appliance. *FPGA*, 245-254, 2013.

[28] D. Shankar, X. Lu, N. Islam, M. Wasi-Ur-Rahman, and D. K. Panda. High-Performance Hybrid Key-Value Store on Modern Clusters with RDMA Interconnects and SSDs: Non-blocking Extensions, Designs, and Benefits. *IPDPS*, 393-402, 2016.

[29] OpenFabrics Alliance. *https://www.openfabrics.org*.

[30] OSU NBC Lab. MVAPICH: MPI over InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE. *http://mvapich.cse.ohio-state.edu/*.

[31] M. J. Koop, S. Sur, Q. Gao, and D. K. Panda. High Performance MPI Design Using Unreliable Datagram for Ultra-scale InfiniBand Clusters. *ICS*, 180-189, 2007.

[32] S. Zhang, H. Jizhong, L. Zhiyong, K. Wang and S. Feng. Accelerating MapReduce with Distributed Memory Cache. *ICPADS*, 472-478, 2009.

[33] P. Xuan, W. B. Ligon, P. K. Srimani, R. Ge and F. Luo. Accelerating Big Data Analytics on HPC Clusters using Two-Level Storage. *DISCS*, 2015.

[34] J. S. Plank, J. Luo, C. D. Schuman, L. Xu, and Z. Wilcox-O'Hearn. A Performance Evaluation and Examination of Open-Source Erasure Coding Libraries for Storage. *FAST*, 253-265, 2009.

[35] D. Shankar, X. Lu, and D. K. Panda. High-Performance and Resilient Key-Value Store with Online Erasure Coding for Big Data Workloads. *ICDCS*, 2017.

[36] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny. Workload Analysis of a Large-Scale Key-Value Store. *SIGMETRICS*, 53-64, 2012.

[37] D. Shankar, X. Lu, M. Wasi-Ur-Rahman, N. Islam, and D. K. Panda. Benchmarking Key-Value Stores on High-Performance Storage and Interconnects for Web-Scale Workloads. *IEEE BigData*, 2015.

[38] Spymemcached. *http://code.google.com/p/spymemcached/*.

[39] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking Cloud Serving Systems with YCSB. *SoCC*, 2010.

[40] SDSC Gordon. *http://www.sdsc.edu/services/hpc/hpc_systems.html*.