InnerCache: A Tactful Cache Mechanism for RDMA-Based Key-Value Store

Min Yang, Songping Yu, Rujie Yu, Nong Xiao,Fang Liu, Wei Chen
State Key Laboratory of High Performance Computing
National University of Defense Technology
Changsha, China
we.isly@163.com

***Abstract*—High-Performance network technology, Remote Direct Memory Access (RDMA), has revealed its tremendous advantage over traditional TCP/IP. With its ultra-low latency and high bandwidth, RDMA has been extensively adopted in distributed environment, especially for in-memory key-value stores. However, although RDMA does provide the ability to interact with remote user space memory directly, memory copy still exists between data memory area and communication memory area with two-sided communication semantics in in-memory key-value store. In addition, using high performance one-sided communication semantics will expose memory totally, hence an inadvertent corrupt data operation could crash system. In this paper, we propose a tactful cache mechanism for RDMA-based in-memory key-value store—InnerCache. Our design concerns two dimensions with respect to improve the system performance with two-sided communication semantics and make system less vulnerable. It merges one-sided and two-sided communication model through making communication memory cacheable. Experimental results show that InnerCache can efficiently improve the performance of RDMA-based in-memory key-value store.**

Keywords-RDMA; key-value store; one-sided communication

I. INTRODUCTION

It is well known that large-scale applications usually suffer from IO performance in the traditional disk-based data center. As DRAM price decreases, it is feasible to build a cluster of hundreds of commodity servers with main memory capacity of tens of terabytes, which serves as a cache in today's web server architecture. What's more, when the high performance network technology-Remote Direct Memory Access (RDMA) meets the RAM-based systems, further improvement of performance could be achieved.

The low latency of application requirement gives a rise to many new emerging distributed in-memory key-value systems based-on RDMA, such as RAMCloud[1], MemCached[2], MICA[7], Pilaf[3], C-Hint[4], FaRM[5], HERD[6]. As far as the implementation's concerned, these systems can be mainly classified into two paradigms. As depicted in Figure 1, the first paradigm is using RDMA to optimize the message passing. Just like the traditional two-side communication semantics of TCP/IP, server and client both take part in the procedure of comunication[1][2][7]. Server and client expose communication memory area (communication buffer) rather than the data memory area to each other, and both sides write requests or responses into the communication memory. Notwithstanding the direct interaction with remote memory in RDMA, the data movement still exists between data memory area and communication memory area in this situation. This overhead hinders performance improvement. The Second one is one-sided communication model coalescing data memory area and communication memory area. In this context, client interacts with server using one-side communication semantics of RDMA[3-6]. Server is oblivious of any request operations of clients, and clients could be aware of server's data memory area. The problem is server needs to do the complicated logic design cooperating with clients to make things right, such as preventing client from reading the invalidate data by other ones, from reclaiming the dynamic allocated memory area clients accessing and so on. What's more, one negligent data corrupt operation could make the whole system broke down.

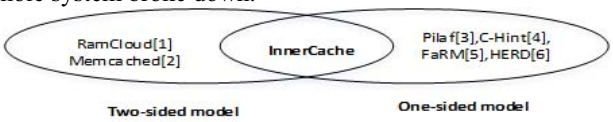

Figure 1. In-memory key-value store based-on RDMA

In this paper, we propose a tactful design of RDMA-based in-memory key-value system, called InnerCache. It takes the advantage of communication memory area to cache key-value pairs. Succinctly, InnerCache concerns two aspects of communication memory management and cache-in-cache mechanism. InnerCache could achieve the comparable performance with one-sided designed RDMA-based MemCache and 1.5 times better than two-sided implemented RDMA-based MemCache.

II. BACKGROUND AND MOTIVATION

In this section, we introduce the background of high performance network technology RDMA and the motivation of our work.

A. RDMA

In traditional network, data transfer involves packing and unpacking through user space, kernel space, and network device at source. This process results memory copy, context switch, and CPU. RDMA enables CPU in one machine to remotely read memory contents of another remote machine without involving the CPU overhead and the OS kernel of the remote machine. Therefore, RDMA is commonly known as one-sided operation (RDMA-read and RDMA-write): the CPU in the server is unaware of the memory access. Meanwhile, RDMA avoids the overhead of memory copy

978-1-5090-2675-3/16 $31.00 © 2016 IEEE
DOI 10.1109/ICWS.2016.106

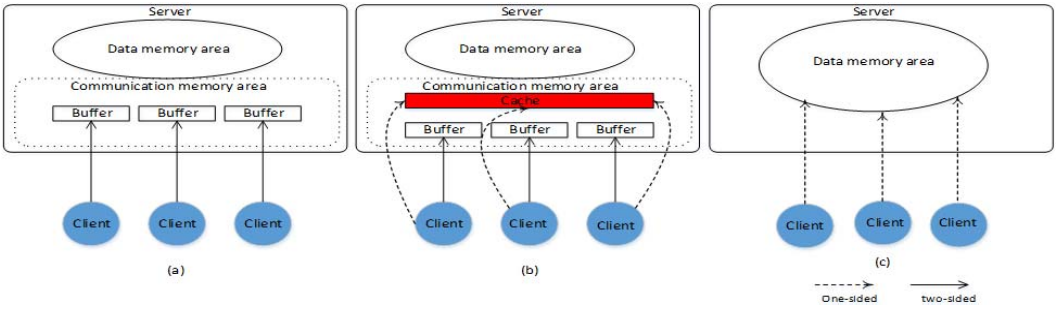

Figure 2. Three paradigms of RDMA-based in-memory key-value store: (a) two-sided communication semantics model ,(b) InnerCache: hybrid communication semantics model through shared communication cache (c) one-sided communication semantics model

between the user space and the kernel space. Besides, RNICs also provide the two-sided operation: SEND and RECV (RECEIVE). In the Send/Recv model, the source posts a Send that indicates the location of the data to be sent, the destination issues a Receive that similarly describes where the data is going to be written, and a matching capability is used to associate a posted Recv to an incoming Send. Each side has the information required for fulfilling the communication.

B. Motivation

Distributed in-memory key-value stores play an important role in current data center. These systems can be used as a medium data volume store, like RAMCloud[1] or a cache layer for huge volume data, like memcached[2]. The storage IO has little impact on systems' performance due to most of (the whole of) the data resides in main memory, whereas the network IO is the performance devil. With the capability of lowest latency, highest bandwidth, and zero CPU overhead, RDMA attracts characters' great enthusiasm to accelerate the network performance of in-memory key-value system. In general, RDMA-accelerated in memory key-value systems fall into two range: decoupling communication memory area (two-sided model, figure 2(a)) and data memory area and coupling communication memory area and data memory area (one-sided model, figure 2(c)). MPI[10] is the typical two-sided communication model. The advantage of this is easily to implement without consideration the vicious and inscrutable behavior of clients. However, the data copy overhead between data memory area and communication memory area is non-trivial, the average ratio is 18% of total latency when getting size of data from 1KB to 512KB. In contrast, much of the research in in-memory key-value system [6] in recently has shown that one-sided operations could offer better performance. One cannot make an omelet without breaking eggs. Clients totally access remote memory is dangerous, a miss operation could jeopardize the critical data in memory and break down the whole system. Protection mechanism pushes to up-level in-memory key-value store, this guarantee must be made through meticulous careful and complicated design, which is against the original intention that we could benefit the high performance from RDMA.

III. DESIGN OF INNERCACHE

To tackle the problem illustrated in section II-B, we introduce InnerCache that synthesizes a collections of ideas to design cache mechanism for RDMA-based in-memory key-value stores. The overall goal InnerCache aims to achieve is to improve the performance of implementation with two-sided communication semantics and eliminates the performance degradation without one-sided communication semantics. To clarify the presentation, we use the following terminology: server means InnerCache-server and Cache means communication cache memory.

A. Overview

Basically, in RDMA-based in-memory key-value store, server initiatively exposes a communication memory area to clients to directly operate on. When a client stores (sets) data, data first arrives to the communication memory area, then server moves data to data memory area, and read procedure is just the opposite. The memory copy is unnecessary when data is already in memory, as depicted in figure 2(a). In order to eliminate data movement, figure 2(c) shows that one-sided semantics design allows clients access memory without participation of server. The whole system needs collaboration of clients and servers. Compare with the other two paradigms, as is shown in the figure 2(b), InnerCache devides the communication memory area into two parts: buffer area and cache area. Section B discusses the management of communication memory. Buffer area is only enable under the two-sided communication semantics. Cache area is used for read acceleration. The idea stems from the classic three cache layer architecture in computer to minimize the performance gap between register and DRAM. Similar to the dataflow between cache and memory, when clients stores data, server combine the data in the buffer and cache, then move into cache area. Clients load data first from cache area of server, if not exists, then server goes to data memory area to retrieve data back to cache area. Section C is going to elaborate the cache-in-cache mechanism.

B. Communication memory management

Communication memory area is the memory which server registers at start used for receiving request from clients and sending responds to clients. In InnerCache, communication memory contain two areas: buffer and cache.

Every client has its own buffer area on server side. Data write requests from client will always arrive to buffer area first, which elegantly avoids perilous operation to other validate data. And this buffer can be reclaimed and reused without any side effects. However, buffer (memory) registration and deregistration are not without any costs. They involve context switches and map virtual memory to make user space DMA-able. This registration cost is actually similar to and sometimes exceeds the overhead of a small size memory copy (<16K) [9]. Hence, InnerCache delays the buffer memory reclaiming operation. When every communication between client and InnerCache, it attaches a timestamp to this buffer. This buffer can't be released until the frozen time is due. During the frozen time, if there is connection request from clients, InnerCache reuses the frozen buffer and set the timestamp invalidate.

Cache area in InnerCache is shared by all clients. The communication cache memory management is slab-like way except that the chunk size in all slabs is multiples of 64 bytes. Intuitively, clients could also have their own cache area in InnerCache. Unfortunately, costly dynamic memory registration for every client will lead to abysmal performance in the circumstances of many connections. And also, memory footprint of temporal connections is wasted. There is one more point, I should touch on, that when a client accesses data in other cache area, and data is replicated. Last but not least, the modification synchronization between data area and cache area is also complicated. Alternatively, a positive consequence of shared cache is that every client can virtually get a large size cache area. For clients, cache is only for read. And performance distraction is mainly from two facets: data eviction policy and coordination between clients' concurrence access and data eviction within InnerCache. The former is up to data access characteristics of user applications, it is not discussed here due to the limited space. As for the latter, we hold the view that RDMA is too fast to make the traditional access contention (lock mechanism) a big deal to performance.

C. Cache-in-Cache Mechanism

Cache-in-Cache is refer to the data in in-memory key-value store usually used as cache layer for storage which is cached by its own communication memory. As describe in section 3.2, InnerCache makes a shared cache area out of communication memory area. Clients could directly read data from the shared cache using one-sided operation of RDMA.

In order to illuminate the cache-in-cache mechanism, we unfold the dataflow of set and get operations in InnerCache. For set operation, to begin with, clients set its data to the communication buffer through two-sided verbs. Then, InnerCache copy data to cache area and data area and build index for it. Success message will be replied to clients if any of two data copy operations succeeds. In detail, for the data modification, InnerCache only first modifies the data existing in cache area, or do as normal. Copy-on-Eviction will make the data area modification happen. It is similar to the CPU cache "write-back" mechanism. When the cache area is full, InnerCache will evict data item to data memory area using LRU or FIFO policy.

For get operation, since clients use one-side operation to directly access cache area, it is indispensable for clients to know where the data is. There is one simple and concise way that server broadcasts the whole indexes to every client on every set operation. However, when the data volume increases, the size of indexes is also very big. Network bandwidth is wasted due to clients do not need the whole index to access all data in server. Actually, server only can one-sided access its own set data and the indexes of data sent at period by server. For reading the data not set by itself, the client has to explicitly send request to server at first time, then gets the data through one-sided operation after server sends its index back.

It is crucial to note some additional remarks. For one thing, the cache role in server is to minimize the logical gap performance between two-sided operations and one-sided operations. For another, prefetching data into cache area may accelerate clients' read operation. Nevertheless, it could be more efficient if clients provide data access history [4] to server.

IV. EVALUATION

In this section, we evaluate InnerCache by comparing it with RDMA-based MemCache with two-sided and one-sided communication model in terms of latency of get and set operations.

Cluster setup: All of our experiments are conducted on a small InnerCache cluster of 3 x86-64 machines. Every machine is equipped with 2.10GHz Intel Xeon E5-2620L 6-core processors, featuring 32KB for L1 instruction and data caches, 256KB L2 and 15MB L3 cache. 16GB memory is installed on each node. All nodes are connected through InfiniBand FDR using Mellanox MT27500 ConnectX-3 HCA. On each machine, we run Red Hat Enterprise Linux Server 6 with kernel 2.6.32. We use two machines as InnerCache-sever and one as InnerCache-client to run one client application. We have allocated each InnerCache-server 1 InnerCache with 4GB data memory area and 255MB shared communication cache to cache hot key-value pairs. All the communication cache are registered to the NIC at the beginning to allow one-sided Read. While the communication buffers are registered to the NIC dynamically during the runtime of receiving clients' requests.

Workloads: We use Yahoo! Cloud Serving Benchmark (YCSB) [11] to examine the performance of InnerCache. We have generated 9 workloads of uniform distribution, latest distribution and Zipf distribution, and each distribution has three different GET/SET ratios, which are 50% GET + 50% SET, 90% GET + 10% SET, 10% GET + 90% SET, respectively. Each workload consists of 10 thousand operations over about 500MB of key-value pairs (100B Key with range from 1KB to 512KB value) in total. In Figure 3

and 4, uniform (latest or zipf)_A_B, A stands for read ratio and B stands for set ratio.

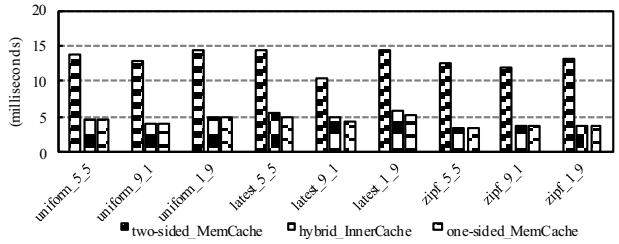

Figure 3. Performance of get and set operations with 100B key with average 1KB value

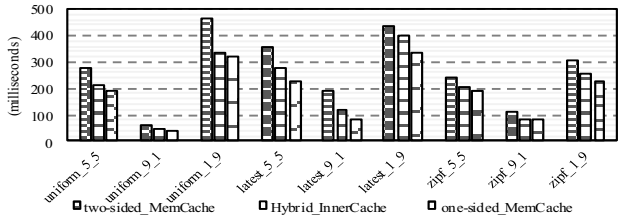

Figure 4. Performance of get and set operations with 100B key with average 512KB value

Figure 3 shows that the latency of set and get operation with small size 1KB key-value pairs. Figure 4 illustrates the latency of get and set operations with average 512KB key-value pairs. Overall, the performance of InnerCache is almost the same with one-sided RDMA-based MemCache. In comparison with two sided RDMA-based MemCache, InnerCache is 1.5 times better. Because of the large size of memory copy overhead increasing, InnerCache is 12 percent slower than one-sided RDMA-based MemCache under the set-intensive workloads.

V. Related Work

Much of research in how to leverage high performance network technology (RDMA) with two-sided and one-sided communication semantics to accelerate in-memory key-value store in the past few years. RAMCloud[1], a distributed in-memory key-value store, providing dozens of terabytes memory capacity with hundreds of commercial servers connected by InfiniBand. Memcached[2] integrated Unified Communication Runtime to transparently benefit from high-performance networks. MICA[7],optimizing for multi-core architectures by enabling parallel access to partitioned data, speeding up data access with RDMA. At one-sided communication model, Pilaf [3] allowed its clients to directly read data from the server's memory through RDMA-read for GET requests. C-Hint [4] focused on cache efficiency with storage such as tracking data access history, making eviction decisions, which might be orthogonal to our Cache-in-Cache mechanism. FaRM [5], a distributed share memory system, used RDMA-read to perform its lock-free reads detecting inconsistent RDMA reads with concurrent CPU memory modifications with a self-verifying data structures, which is orthogonal to our work. HERD[6] takes combined Unreliable-Connection-based RDMA Write with Unreliable-Datagram-based Send for better performance at the cost of reliable data transmission. RFP[8] explored the asymmetrical performance of out-bound RDMA-write and in-bound RDMA-read and allowed clients to use in-bound RDMA-read to remotely read. It is different from all these work that InnerCache merges low-performance two-sided and high-performance one-sided communication model by making communication memory shared-cacheable in consideration of the frangibility of totally memory exposure and performance of in-memory key-value store.

VI. Conclusion

As the high performance network push forward the boundary of performance of in-memory key-value stores. It is equally paramount to make trade-off between performance and vulnerability of systems to enhance feasibility in practice. Consequently, in this paper, we compares the merits and demerits of two-sided and one-sided communication semantics of Remote Data Memory Access extensively utilized in high performance environments. We harbor the idea that the benefits of both two worlds can be obtainable through making the communication memory of RDMA cacheable. Our experiments indicates that InnerCache could achieve almost the same performance with one-sided designed RDMA-based MemCache and 1.5 times better than two-sided implemented RDMA-based MemCache.

References

[1] Ousterhout, John, et al. "The case for RAMClouds: scalable high-performance storage entirely in DRAM." ACM SIGOPS Operating Systems Review 43.4 (2010): 92-105.

[2] Jose, Jithin, et al. "Memcached design on high performance rdma capable interconnects." Parallel Processing (ICPP), 2011 International Conference on. IEEE, 2011.

[3] Mitchell, Christopher, Yifeng Geng, and Jinyang Li. "Using One-Sided RDMA Reads to Build a Fast, CPU-Efficient Key-Value Store." USENIX Annual Technical Conference. 2013.

[4] Wang, Yandong, et al. "C-Hint: An Effective and Reliable Cache Management for RDMA-Accelerated Key-Value Stores." Proceedings of the ACM Symposium on Cloud Computing. ACM, 2014.

[5] Dragojević, Aleksandar, et al. "Farm: Fast remote memory." Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation, NSDI. Vol. 14. 2014.

[6] Kalia, Anuj, Michael Kaminsky, and David G. Andersen. "Using RDMA efficiently for key-value services." Proceedings of the 2014 ACM conference on SIGCOMM. ACM, 2014.

[7] Lim, Hyeontaek, et al. "MICA: A holistic approach to fast in-memory key-value storage." management 15.32 (2014): 36.

[8] Su, Maomeng, et al. "RFP: A Remote Fetching Paradigm for RDMA-Accelerated Systems." arXiv preprint arXiv:1512.07805 (2015).

[9] "A critique of RDMA", http://www.hpcwire.com/2006/08/18/a_critique_of_rdma-1

[10] Doerfler, Douglas, and Ron Brightwell. "Measuring MPI send and receive overhead and application availability in high performance network interfaces." Recent Advances in Parallel Virtual Machine and Message Passing Interface. Springer Berlin Heidelberg, 2006. 331-338.

[11] Cooper, Brian F., et al. "Benchmarking cloud serving systems with YCSB."Proceedings of the 1st ACM symposium on Cloud computing. ACM, 2010.