Achieving Higher Performance of Memcached by Caching at Network Interface

Eric S. Fukuda*, Hiroaki Inoue†, Takashi Takenaka†, Dahoo Kim*,
Tsunaki Sadahisa*, Tetsuya Asai*, and Masato Motomura*
*Graduate School of Information Science and Technology
Hokkaido University, Sapporo, Hokkaido 060–0814, Japan
Email: {fukuda@lalsie., kim@lalsie., sadahisa@lalsie., asai@, motomura@}ist.hokudai.ac.jp
†NEC Corporation,
1753 Shimonumabe, Nakahara-ku, Kawasaki, Kanagawa 211–8666, Japan
Email: {takenaka@aj, h-inoue@ce}.jp.nec.com

***Abstract*—As the volume of data that web services handle is becoming larger, many web service providers are utilizing memcached, an in-memory key-value store to improve their web server's performance. While memcached usually runs on a server with a high performance processor, various hardware platforms has been evaluated for running memcached in order to achieve higher performance. Recently, several works that use FPGAs have successfully achieved higher performance than Xeon. These works, however, struggles to utilize a large memory with FPGAs. In this paper, we propose a system that enables us to overcome this problem and enhances memcached by caching a part of software memcached's commands and data to the network interface card equipped with an FPGA and a DRAM. Our evaluation showed that the NIC cache has less than 30% of hit rate for workload with Latest key selection distribution, and 30% to 60% for Zipf distribution workloads.**

I. Introduction

Web service providers that have tremendous amounts of user and other information are eager to facilitate new technologies that enable their servers to handle more data traffic. One such technology employed by many web service providers is key-value stores (KVSs). Memcached is a kind of KVS technology that reduces the latency of data retrieval by storing KV-pairs (KVPs) in distributed servers' memories instead of fetching from the hard drives of database servers.

Memcached is used by a number of major web service providers such as Facebook, Wikipedia and YouTube. According to Facebook's research on their own memcached workloads, they use hundreds of memcached servers [1]. In view of such extensive use, improving the memcached performance would have a large impact on web services' response. In fact, researchers have investigated the suitability of various hardware platforms for running memcached, from multiple low power CPUs [2] to many-core processors [3]. Meanwhile, FPGA-based memcached systems are outperforming high performance CPUs such as Intel® Xeon® by an order of magnitude [4], [5], [6], [7].

Although these efforts have improved the performance of memcached, major challenges remain. One such challenge is to efficiently manage a large memory size with an FPGA. Memcached servers usually have a few dozen gigabytes of memory, and such a memory space is too large for an FPGA to efficiently manage [8]. Research groups trying to handle large memory size by utilizing an internal or an external CPU from the FPGA [5], [6], [7].

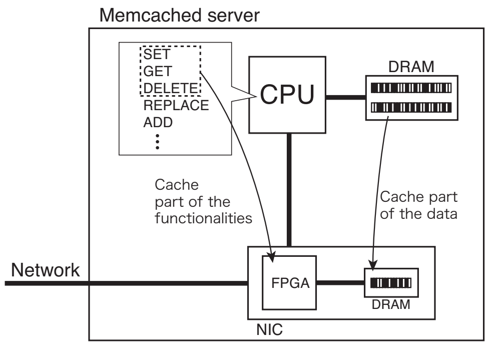

Fig. 1. The basic idea of the proposed method.

In this paper, we propose a method that makes possible a low latency hardware memcached system with less memory than others require. Our method caches the subset of data stored in software memcached running on the host CPU at the network interface card (NIC) equipped with an FPGA and a DRAM memory. When the server receives a request from a client, the NIC tries to retrieve the data within the DRAM and sends it back if the data is found. If not, the NIC passes the request to the host CPU and the CPU executes the usual memcached operation. Since memcached data has locality, the NIC requires only a fraction of the amount of memory that the host server has. Furthermore, the commands the NIC cache does not support can be delegated to the host CPU; therefore only the frequently used memcached commands have to be supported on the NIC.

II. Proposed Method and Results

The behavior of the NIC is shown in Table I. The most important thing to note here is that when a request that has a command other than SET, GET or DELETE comes in, the corresponding entry in the cache is invalidated. This is to keep the coherency between the NIC and the host: if we pass the request without invalidating the entry, there will be different data in the NIC and the host, and the unmodified data will be

TABLE I. System behavior for various commands.

SET	Cache data and pass request to host
GET	Send reply to the network if hit; pass request to host if miss
DELETE & other	Invalidate data and pass request to host

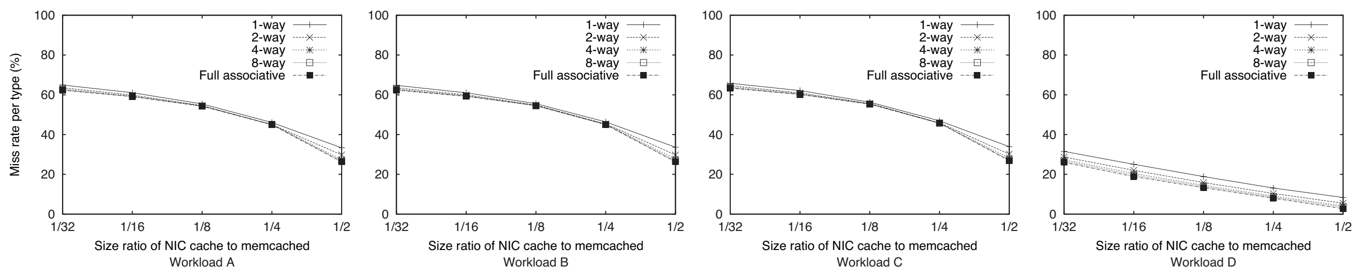

Fig. 2. Miss rates for GET requests with LRU replacement policy.

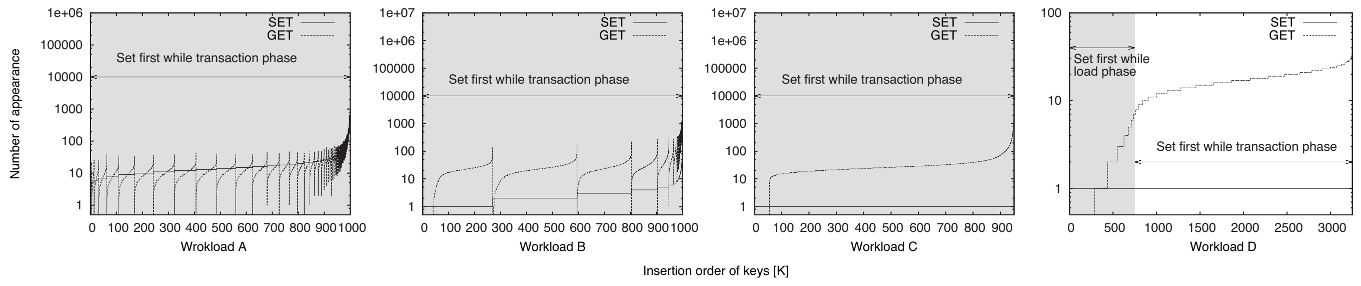

Fig. 3. Key access characteristic of the workloads.

returned from the NIC for the following GET request for the same key.

We evaluated the miss rates of our system with a software simulation. Test workload was generated by Yahoo Cloud Serving Benchmark (YCSB) [9], which is a widely used KVS benchmarking tool. Among the five workload scenarios that YCSB provides, we used Workload A, B, C and D, which uses only commands that memcached supports. We examined various cache associativity with LRU eviction algorithm. Fig. 2 shows the miss rate with varying sizes of cached capacity, from 1/32 to 1/2 of the memcached capacity that runs on the host computer. The miss rates for Workload A, B and C was approximately 30% to 60%, while it was less than 30% for Workload D.

III. Workload Analysis

The differences of hit rates come from their key selection distribution. Workload A, B and C follow the Zipf distribution, which certain data are popular regardless of when they were recently accessed, and Workload D follows the Latest distribution, which data that are recently accessed are the most popular ones. Fig. 3 illustrates the characteristics of the workloads. The x-axis denotes the keys in the order of their first appearance, and the y-axis denotes their numbers of appearance. The shaded area signifies that the keys in the area were set at the warm-up phase, and appeared for the first time in the evaluation phase otherwise. The number of appearance during the warm-up phase was not counted.

According to these figures, the workloads that follow the Zipf distribution are more skewed than the one which follows Latest. What is more, Workload D has much more keys than the others. These two characteristics indicate that Workload D is more likely to result in higher miss rate, although it has a lower miss rate. In order to improve the performance for Zipfian workloads, we need to look into the characteristics of the workloads in more details.

IV. Future work

Our next step is to implement a practical system based on the method we proposed in this paper. Although many difficulties are expected such as efficient memory allocation and scalability issues, the hit rate at the NIC cache remains a major factor of improving the performance. We think that the current hit rate for Zipfian workloads is still too low. In order to improve the hit rate, we will investigate how the workloads behaves and look for a method to efficiently cache the frequently accessed key-value pairs.

References

[1] Y. Xu, E. Frachtenberg, S. Jiang, and M. Palecezny, “Characterizing facebook’s memcached workload,” *IEEE Internet Computing*, vol. 99, pp. 41–49, 2014.

[2] W. Lang, J. M. Patel, and S. Shankar, “Wimpy node clusters: What about non-wimpy workloads?” in *Proceedings of the 6th International Workshop on Data Management on New Hardware*, 2010, pp. 47–55.

[3] M. Berezecki, E. Frachtenberg, M. Paleczny, and K. Steele, “Many-core key-value store,” in *Proceedings of the 2nd International Green Computing Conference and Workshops*, 2011, pp. 1–8.

[4] S. R. Chalamalasetti, K. Lim, M. Wright, A. AuYoung, P. Ranganathan, and M. Margala, “An fpga memcached appliance,” in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2013, pp. 245–254.

[5] M. Blott, K. Karras, L. Liu, K. Vissers, J. Bär, and Z. István, “Achieving 10gbps line-rate key-value stores with fpgas,” in *Proceedings of the 5th USENIX Workshop on Hot Topics in Cloud Computing*, 2013, pp. 1–6.

[6] K. Lim, D. Meisner, A. G. Saidi, P. Ranganathan, and T. F. Wenisch, “Thin servers with smart pipes: Designing soc accelerators for memcached,” in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, 2013, pp. 36–47.

[7] M. Lavasani, H. Angepat, and D. Chiou, “An fpga-based in-line accelerator for memcached,” *IEEE Computer Architecture Letters*, vol. 99, pp. 1–4, 2013.

[8] A. Wiggins and J. Langston, “Enhancing the scalability of memcached,” http://software.intel.com/en-us/articles/enhancing-the-scalability-of-memcached-0.

[9] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, “Benchmarking cloud serving systems with ycsb,” in *Proceedings of the 1st ACM Symposium on Cloud Computing*, 2010, pp. 143–154.