High Performance In-memory Caching through Flexible Fine-grained Services

Yue Cheng†, Aayush Gupta‡, Anna Povzner‡, Ali R. Butt†
†Virginia Tech, ‡IBM Almaden Research Center
{yuec,butta}@cs.vt.edu, {guptaaa, apovzne}@us.ibm.com

1 Introduction and Motivation

In-memory object caches are extensively used in today's web installations [1, 6]. Most existing systems adopt monolithic storage models and engineer optimizations on specific workload characteristics [3, 6] or operations [4, 5]. Such optimizations are insufficient as large-scale cloud workloads typically exhibit both temporal and spatial shifts - requirements vary within the same deployment over time and different parts of the same workload demonstrate different access patterns. To this end, we propose a caching tier that supports differentiated services in multiple dimensions. Since there is no best "one-size-fits-all" solution for all workload requirements, we argue that a fine-grained modular design will provide both high performance and flexibility in supporting multiple services.

2 Design Overview

There are two possible ways of providing multiple services from a caching tier - multi-instance approach (different instances provide different services) and fine-grained control within a single instance (each instance provides a variety of services). The former approach, while simpler to implement, can result in inefficient utilization of resources, e.g., duplication of data across instances, and can be slow to adapt to workload shifts. The latter method, which we propose, makes effective utilization of available resources, provides great flexibility and near-seamless resource re-allocation and load balancing support. Our system enables multiple services by instantiating different data layouts within a single cache instance. We abstract the basic management unit termed *cachelet*. A cachelet is an independent entity for managing the data objects and the metadata. Each cachelet acts like a resource container encapsulating a fixed data layout, e.g., hash table, sorted list, etc., and exposes configurable interfaces (`get`, `set`, `scan`) to the applications. Cloud service users [1] can benefit from partitioning their applications into cachelets and specifying different data layouts and performance requirements for each cachelet, thus meeting the application demands.

3 Case Studies

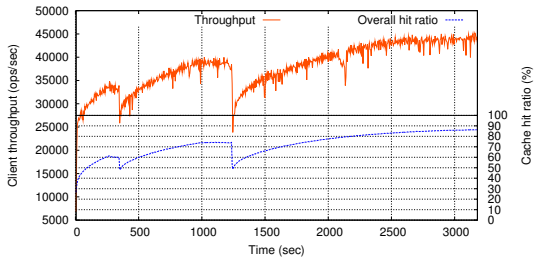

Figure 1: YCSB benchmarking with 10 GB data and caching tier enabled. Systems start up with 4 cache nodes. At sec 340 and 1240, 4 new cache nodes are added in respectively. While warming up, overall throughput reduces up to 41% and performance recovery takes up to 10 min.

The scale-out nature of distributed caches like Memcached [2] can cause intermittent performance degradation while the newly added nodes warm-up, as shown in Fig. 1. Our system can enable selective migration of cachelets, e.g., migrate cachelets which have low load/lower priority and use the resources of these cachelets to serve the changed workloads, thus providing a near-seamless transition.

SoCC'13, 1–3 Oct. 2013, Santa Clara, California, USA.
ACM 978-1-4503-2428-1. http://dx.doi.org/**10.1145/2523616.2525964**

References

[1] Amazon web service elasticache. `http://aws.amazon.com/elasticache/`. accessed on Aug. 23, 2013.

[2] Memcached. `http://memcached.org/`. accessed on Sept. 15, 2013.

[3] N. Bronson, Z. Amsden, G. Cabrera, P. Chakka, P. Dimov, H. Ding, J. Ferris, A. Giardullo, S. Kulkarni, H. Li, et al. Tao: Facebook's distributed data store for the social graph. In *USENIX ATC'13*, 2013.

[4] B. Fan, D. G. Andersen, and M. Kaminsky. Memc3: Compact and concurrent memcache with dumber caching and smarter hashing. In *USENIX NSDI'13*, 2013.

[5] Y. Mao, E. Kohler, and R. T. Morris. Cache craftiness for fast multicore key-value storage. In *ACM EuroSys'12*, 2012.

[6] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, et al. Scaling memcache at facebook. In *USENIX NSDI'13*, 2013.