Dynamic Management of Caching Tiers

SPEC Distinguished Dissertation Award 2013 (Invited Abstract)

Anshul Gandhi
IBM T.J. Watson Research Center
gandhian@us.ibm.com

1. INTRODUCTION

Application owners are typically concerned about getting the best performance at the lowest cost. When the workload demand for an application is dynamic, lowering the cost necessitates dynamic management of the hosting infrastructure (physical servers or virtual machines). An application deployment is often divided into multiple tiers. The frontend tier is usually stateless, and processes incoming workload requests using data from the backend. The backend tier is stateful, and typically consists of a persistent storage system, such as a database. To alleviate load at the backend, a stateful, but non-persistent, caching tier is often used to cache data or partial results. For such multi-tier deployments, the ultimate goal is to scale *all* of the tiers to match the varying workload demand.

Stateless tiers are easy to scale (see, for example, [7, 6]) since they do not store data that is required for request processing. Scaling the stateful data tier is more complicated. For the database tier, there needs to be at least one copy of the data, so research is limited to questions of how many replicas are needed and where to place the data (see, for example, [2, 13]). The caching tier, on the other hand, stores only a fraction of the total data, and this data is not necessarily replicated. Since the caching tier is non-persistent, we are not limited by a minimum amount of cached data, so there is significant potential for scaling down the caching tier. However, the caching tier plays a crucial role in filtering the amount of requests that go to the database tier. Thus, dynamically managing the caching tier requires careful consideration of the possible impact on end-to-end application performance. To the best of our knowledge, there has been no work on dynamically managing the caching tier.

As part of this thesis, we propose two orthogonal approaches to dynamically managing the caching tier: (i) CacheScale [14], and (ii) SOFTScale [8]. CacheScale works by dynamically scaling the caching tier in response to changes in workload demand. To avoid performance degradation due to transient cache misses, CacheScale carefully redistributes important data elements *prior* to scaling. SOFTScale, on the other hand, does not scale the caching tier, and instead leverages the spare capacity in the caching tier nodes to handle some of the stateless tier load. SOFTScale is especially useful for handling transient overload caused by a sudden increase in demand.

2. EXPERIMENTAL SETUP

We experiment with a testbed of 28 commodity servers, which are divided into multiple tiers. We employ one of these servers as the load generator running httperf [11]. Another server is used as the load-balancer running the Apache HTTP Server, which distributes PHP requests from the load generator to 20 application servers. The application servers (Intel Xeon E5520 processor-based) parse the incoming PHP requests and collect the required data from the caching tier and the data tier. The caching tier comprises 5 memcache [10] servers (Intel Xeon X5650 processor-based) and the data tier comprises a server (Intel Xeon E5520 processor-based) with 5 disks running an Oracle BerkeleyDB [12] database with a billion key-value pairs (250GB).

We design a key-value workload to model realistic multi-tier applications such as the social networking site, Facebook, or e-commerce sites like Amazon [4]. Serving a workload request involves fetching specific sets of items from the data tier based on the request (more details on our workload can be found in the thesis [5]).

Our goal in managing the caching tier is to successfully meet end-to-end response time SLAs while minimizing the total number of active servers employed, on average, by the application.

3. CACHESCALE: SCALING THE CACHING TIER

The caching tier plays a crucial role in regulating application performance by filtering the number of data access requests to the database tier. If the caching tier does not filter enough requests, the database tier can get overloaded, leading to very high response times. These observations speak against scaling the caching tier. However, the caching tier is expensive and consumes a lot of power. For comparison, consider Amazon's EC2 "High-Memory Quadruple Extra Large" instance versus their "High-CPU Extra Large" instance [1]. These have approximately the same compute power, but the high memory instance (60GB more memory) costs about 2-3 times more than the high cpu instance [1]. The high cost for memory makes sense if we consider the power consumed by the memory subsystem, which accounts for up to 40% of the power in a server [9]. Thus, the caching tier is expensive both in terms of monetary cost and power draw.

CacheScale makes the case for a vast reduction in the size of the caching tier during low loads. We show [14] that we can achieve huge savings in the size of the caching tier (up to 90%) without violating performance goals. *The key insight is that when the load drops, we do not need to filter as many* *requests to the data tier, so we can get away with a lower cache hit rate.* If the requested data items were uniformly distributed, a small decrease in the cache hit rate would not lead to much savings. However, many studies have shown that web requests follow a very skewed distribution, often modeled as a Zipf distribution [3]. This implies that a small decrease in cache hit rate can lead to a large decrease in the amount of cached data.

However, the above savings are stipulated on being able to cache the right data when scaling down/up. When removing a caching tier node, we experience an immediate loss of the "hot" data that was stored on the instance. When adding a caching tier node, its cache is "cold" and the initial requests will all be misses that go to the database tier. In order to minimize the performance degradation due to these cache misses, CacheScale proactively redistributes the cache contents to ensure that the availability of the hot data in the caching tier is not affected by the scaling actions. Importantly, CacheScale does this without requiring access to the (elusive) LRU list. Full details on CacheScale can be found in [14].

4. SOFTSCALE: STEALING FROM THE CACHING TIER

SOFTScale takes a very different approach to managing the caching tier. Rather than scaling the caching tier during low loads, SOFTScale aims to instead leverage the spare compute capacity at the caching tier to alleviate some of the load at the stateless tiers. This allows for more aggressive scaling of the stateless tiers, thereby lowering overall resource consumption.

SOFTScale works by overloading the functionality of the caching tier nodes to also act as frontend tiers. This requires installing the frontend tier software at the caching tier nodes. As a result, when the frontend tiers are unable to process all the incoming requests in a timely manner (for example, during load spikes or periods of transient overload), a fraction of the requests can be routed directly to the caching tier nodes which can temporarily act as frontend tiers. Of course, one has to be careful not to overload the caching tier when offloading frontend tier requests to it. Further, the interference between caching tier work and frontend tier work at the caching tier nodes must be minimized to avoid any potential performance degradation. We address these concerns in our implementation of SOFTScale in [8], and show that SOFTScale can handle a range of load spikes which would cause a normal system (without SOFTScale) to exhibit very high response times. If needed, SOFTScale can be coupled with techniques like admission control and request prioritization to further minimize the damage caused by load spikes. Full details on SOFTScale can be found in [8].

5. CONCLUSION

A popular approach to dealing with dynamic workload demand is scaling the application deployment. While there has been a lot of work [7, 6] on scaling the stateless tiers in a deployment, the scaling of stateful tiers, such as the database and caching tiers, has received much less attention, largely because of the difficulty involved in scaling the stateful tiers without negatively impacting end-to-end performance. We propose two orthogonal approaches to dynamically managing the stateful caching tier: (i) CacheScale [14] carefully scales the caching tier while ensuring availability of hot data items, and (ii) SOFTScale [8] opportunistically steals spare capacity from the (unscaled) caching tier to alleviate some of the load at the other tiers. To the best of our knowledge, these are the first approaches to dynamically managing the caching tier.

6. REFERENCES

[1] Amazon Inc. Amazon elastic compute cloud (Amazon EC2). http://aws.amazon.com/ec2.

[2] Hrishikesh Amur, James Cipar, Varun Gupta, Gregory R. Ganger, Michael A. Kozuch, and Karsten Schwan. Robust and flexible power-proportional storage. In *SOCC 2010*, pages 217–228, Indianapolis, IN, USA.

[3] Lee Breslau, Pei Cao, Li Fan, G. Phillips, and Scott Shenker. Web caching and zipf-like distributions: evidence and implications. In *INFOCOM 1999*, pages 126–134, New York, NY, USA.

[4] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. In *SOSP 2007*, pages 205–220, Stevenson, WA, USA.

[5] Anshul Gandhi. *Dynamic Server Provisioning for Data Center Power Management.* PhD thesis, School of Computer Science, Carnegie Mellon University, 2013.

[6] Anshul Gandhi, Yuan Chen, Daniel Gmach, Martin Arlitt, and Manish Marwah. Minimizing data center sla violations and power consumption via hybrid resource provisioning. In *IGCC 2011*, pages 49–56, Orlando, FL, USA.

[7] Anshul Gandhi, Mor Harchol-Balter, Ram Raghunathan, and Michael Kozuch. AutoScale: Dynamic, Robust Capacity Management for Multi-Tier Data Centers. *Transactions on Computer Systems*, 30, 2012.

[8] Anshul Gandhi, Timothy Zhu, Mor Harchol-Balter, and Michael Kozuch. SOFTScale: Scaling Opportunistically For Transient Scaling. In *Middleware 2012*, pages 142–163, Montreal, Quebec, Canada, 2012.

[9] David Meisner, Brian T. Gold, and Thomas F. Wenisch. PowerNap: Eliminating server idle power. In *ASPLOS 2009*, pages 205–216, Washington, DC, USA.

[10] Memcached. A distributed memory object caching system. http://www.danga.com/memcached.

[11] David Mosberger and Tai Jin. httperf—A Tool for Measuring Web Server Performance. *Sigmetrics: Performance Evaluation Review*, 26(3):31–37, 1998.

[12] Michael A. Olson, Keith Bostic, and Margo Seltzer. Berkeley db. In *USENIX ATC 1999, FREENIX Track*, pages 183–191, Monterey, CA, USA.

[13] Eno Thereska, Austin Donnelly, and Dushyanth Narayanan. Sierra: practical power-proportionality for data center storage. In *EuroSys 2011*, pages 169–182, Salzburg, Austria.

[14] Timothy Zhu, Anshul Gandhi, Mor Harchol-Balter, and Michael Kozuch. Saving Cash by Using Less Cache. In *HotCloud 2012*, Boston, MA, USA, 2012.