Review of NetCache: Balancing Key-Value Stores with Fast In-Network Caching

Jake Pitkin
University of Utah

November 18, 2018

Introduction

NetCache is a novel approach to a caching layer for a key-value store distributed across a storage cluster. The innovation this work provides comes in where they place the cache: inside a network switch. Network switches are becoming increasingly programmable and NetCache can run at line rate: processing 2+ billion queries per second for 65k items with 16-byte keys and 129-byte values on a single switch. A dedicated network switch is programmed to contain NetCache and is placed on the path between clients and the storage cluster.

Rather than replicate hot items, NetCache instead keeps a single copy of each key-value pair. Without a caching layer, this highly-skewed workload will leave the servers with the key-value extremely over worked while leaving the other servers under utilized. NetCache's goal is to statistically detect these hot items and place them in the cache. For example, prior studies have shown that 10% of items account for $60 - 90\%$ of queries in the Memcached deployment at Facebook [1]. Only 10% of the items being hot at a given time works in favor of NetCache as network switches have limited on-chip memory. As such, NetCache uses switches as a load-balancing cache with medium cache hit ratio ($< 50\%$) with a goal of caching these very hot items. The authors show that NetCache improves the throughput by $3 - 10x$ and reduces the latency of up to 40% of queries by 50%, for high-performance, in-memory key-value stores.

Key Aspects

NetCache relies on a key theoretical analysis. That is, a cache only needs to store O(N log N) items to balance the load for a hash-partitioned key-value cluster with N store nodes [2]. Given N nodes and N*T total load, we don't want a single node to experience more than T load with a high probability. Given N isn't ever huge, the on-chip memory of a NetCache switch has the capacity for O(N log N) key-value pairs.

NetCache keeps caching simple by never replicating hot key-value pairs. This removes the need for cache consistency and query routing which keeps the system simple and removes overhead. To avoid loss of data in the event of network switch failure, NetCache uses write-through to stable storage for all write requests from clients.

Cache hits don't need to visit the node that contains the key-value pair on its stable storage. Instead, as the queries are routed through a NetCache switch a hit is detected and the query is answered by the switch. This novel approach provides an incredible speedup in RTT for highly-skewed workloads.

The recent advent of highly programmable network switches brings about a new way to think about distributed systems. Switches have high I/O but limited programmability and resources. On the other hand servers have low I/O but are highly programmable and have a large amount of compute and storage power. By combining the two, we could strive for a high I/O, highly programmable system with ample storage.

Future work

This paper explored the novel idea of caching at the switch level but only considered a single rack. Consider a datacenter with multiple racks (perhaps now with duplicated key-value pairs) and a NetCache switch situated in front of each rack. This would require a layer of routing and cache consistency would now need to be considered. The authors indicate they will explore this in future work as this is a practical scenario.

Research into using network switches as caches is a hot area. NetChain [3], which just won best paper at NSDI 18, quickly built on the work presented by NetCache. NetChain provides scale-free sub-RTT coordination in a storage cluster with replicated data.

I find this work interesting and see it as the future of how caching will be handled for large storage clusters. The concept of something going "viral" online is common across social media and news platforms. As such, pieces of data becoming hot is also common. As in the Facebook study [2] we only need to concern ourselves with caching a small to medium amount of the data. Network switches really shine at solving this problem given there incredible I/O but limited on-chip memory. This paper is groundwork, but I anticipate a boom of research in this area as it has strong practical implications.

Limits

The largest limitation of NetCache is its ability to handle write requests. The cache uses a write-through policy and as such each write access will invalidate the cache inside the switch and access the storage node. In an examined real-world scenario [2], the GET/SET ratio is 30:1 which works well with NetCache. However, under a highly-skewed write-intensive workload NetCache doesn't provide any increase in performance but rather hurts performance. On average, once the workload passes 20% write operations the overhead of NetCache begins to hurt performance. One way to improve write operations of course would be to utilize the cache. This creates a new problem of potentially losing data in the event of network switch failure as new writes aren't immediately being forwarded to the backing stable storage.

The other limitation I observed is monetary cost and availability of these cutting-edge network switches. Barefoot doesn't have prices listed on their site for these switches which is always a sign of expensive. You would also need one for each rack in your datacenter which would be a huge investment if you aren't a multi-billion dollar company. If the work in caching at the switch level continues to grow past an area of research this limitation of availability should handle itself.

Conclusion

I found this work to be both concise and impactful. The authors have presented a novel approach to caching at the storage cluster level. Switches possess high I/O but limited programmability and resources. Where servers have low I/O but are highly programmable with a large amount of compute and storage power. Developing systems that exploit the strengths of both feels like strong area for new breakthroughs in distributed computing. By handling client GET requests on a network switch we circumvent the overhead of the operating system on the storage node as well as reduce the RTT of the request.

Future work can build on this foundation and tackle the weaknesses of NetCache. These include developing ways to improve the performance of write-heavy workloads. As well as expanding caching to across multiple racks without incurring a performance penalty for the additional routing and cache consistency. As is though, the authors provided impressive speedup for read-heavy workloads which seem to be more common than write-heavy workloads.

References

[1] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. 2012. Workload Analysis of a Large-scale Key-value Store. In ACM SIGMETRICS.

[2] Bin Fan, Hyeontaek Lim, David G. Andersen, and Michael Kaminsky. 2011. Small Cache, Big E$ect: Provable Load Balancing for Randomly Partitioned Cluster Services. In ACM SOCC.

[3] NetChain: Scale-Free Sub-RTT CoordinationNSDI2018 Xin JinXiaozhou Li-Haoyu ZhangNate FosterJeongkeun LeeRobert SoulChanghoon KimIon Stoica.