Load Balancing of Heterogeneous Workloads in Memcached Clusters

Jinho Hwang (IBM Research)
Wei Zhang, Timothy Wood, H. Howie Huang (George Washington Univ.)
K.K. Ramakrishnan (Rutgers University)
Background: Memory Caching

- Two orders of magnitude more reads than writes
- Solution: Deploy memcached hosts to handle the read capacity
Memcached at Scale

- Databases are hard to scale… Memcached is easy
  - Facebook has 10,000+ memcached servers

- Partition data and divide key space among all nodes
  - Simple data model. Stupid nodes.

- Web application must track where each object is stored
  - Or use a proxy (load-balancer) like moxi
Scales easily, but loads are imbalanced

- Random placement…
- Skewed popularity distributions…

Load on Wikipedia’s memcached servers
Motivation

- Cache clusters are typically provisioned to support peak load, in terms of request processing capabilities and cache storage size
  - This worst-case provisioning can be very expensive
  - This does not take advantage of dynamic resource allocation and VM provisioning capabilities

- There can be great diversity in both the workloads and types of nodes
  - This makes cluster management difficult

- Solution: large-scale self-managing caches
Contributions

- A high speed memcached load balancer
  - can forward millions of requests per second
- A hot spot detection algorithm
  - uses stream data mining techniques to efficiently determine the hottest keys
- A two-level key mapping system
  - combines consistent hashing with a lookup table to flexibly control the placement and replication level of hot keys
- An automated server management system
  - determines the number of (types of) servers in the caching cluster
Outline

- Background and Motivation
- Workload & Server Heterogeneity
- Memcached Load Balancer Design
- Hotspot Detection Algorithm
- Conclusions
Workload Heterogeneity

- Varied key popularity depending on applications
- Read/write rates, churn rates, costs for cache misses, quality-of-service demands

![Diagram showing hash space and time with popular and unpopular regions.]

Based on Wikipedia 2008 database dump and access trace
Memcached Cluster Grouping

- Traditionally, many applications share memcached servers
- Today, to handle different applications, memcacheds are clustered *manually*

Need smarter way to manage cache clusters
Server Heterogeneity

- Software and hardware can be diverse
- Different memory cache softwares out there
- Many researches prototyped memcached on different HW architectures
  - GP-GPU [Hetherington:ISPASS12]
  - RDMA (remote direct memory access) [Stuedi:ATC12]
  - FPGA [Maysam:Computer_Architecture_Letter13]
  - Intel DPDK [Lim:NSDI14]

Dynamically adapt to workload heterogeneity and server heterogeneity
Outline

- Background and Motivation
- Workload & Server Heterogeneity
- Memcached Load Balancer Design
- Hotspot Detection Algorithm
- Conclusions
Self-Managing Cache Clustering System

- Cache clustering system considers heterogeneous workloads and servers
  - Identifies server capabilities to adjust key space
  - Finds hot items based on the item frequency to handle them differently
- Dynamically increase/decrease the number of memcached servers
- Uses two-level key mapping system
  - Consistent hashing: *applies to warm/cold items (LRU)*
  - Forward table: *applies to hot items*
Automated Load-Balancer Architecture

- Use UDP protocol to redirect the packets directly from memcached servers to clients (lower load in load-balancer)
- **Hot Spot Detector** runs a streaming algorithm to find hot items
- **Server Manager** manages the memcached servers
- **Key Replicator** manages the key replication
- **Key Director** manages forwarding table and consistent hashing ring
Outline

- Background and Motivation
- Workload & Server Heterogeneity
- Memcached Load Balancer Design
- Hotspot Detection Algorithm
- Conclusions
Lossy Counting based Hot Spot Detection

- Lossy counting algorithm is a one-pass deterministic streaming algorithm with user defined parameters
  - Support threshold $s$
    - Given the stream length $N$, return at least $sN$
  - Error rate $\varepsilon$ ($\varepsilon << s$)
  - No item with true frequency $< (s - \varepsilon)N$ is returned
    - (no false negative)
  - Memory size is not monotonically increasing
Find Hot Items and Groups of Hot Items

- Estimated frequency (i.e., request rate) seen by the top keys
- The goal is two-fold
  - Find hot items
  - Find groups of items that can balance the loads across servers
Adaptively Sizing Number of Hot Items

- Given the number of servers, and found hot spot groups, the load-balancer decides to increase/decrease the number of keys.
- In a scheduling window, we adjust the support parameter $s$ to adjust the number of hot items.

![Graph showing number of hot items vs. stream length](image)
Conclusion

- **Summary**
  - Self-managing cache clustering system
    - Workload and server heterogeneity
  - Adaptive hot item groups
    - Number of keys and groups based on servers
  - High speed load-balancer (~10 million requests per sec / single core)
    - Leverage high-performing NICs and processors

- **Ongoing work**
  - Optimize the number of servers and replicas of hot items
    - (cloud environment)