**Background**

- In-memory object caches extensively used in public/private clouds and web installations
  - Low-latency access to data
  - Scalability
- The state-of-the-art
  - Amazon Web Service Elasticache
  - Facebook Memcache, TAO
  - Masttree [EuroSys’12]
  - MemC3 [NSDI’13]

**Motivation**

- Most systems adopt monolithic storage models and engineer optimizations on specific workload characteristics or operations such as GET
  - The main focus of most optimizations is on performance improvement on one single dimension
  - Large-scale cloud workloads exhibit temporal and spatial shifts
- They either do not or support dynamic membership but with significantly high overhead
- Cold cache warm-up causes intermittent performance degradation

![Throughput and Overall hit ratio graph](image)

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 Idea**

- The fine-grained modular design within one cache instance
  - Partition both data and metadata the independent entities called Cachelet
  - Hash table module, B+ tree module, trie module, etc.
- APis (services):
  - GET()
  - SET()
  - RANGE SCAN()
  - PREFIX MATCH()
  - etc... Flexible, customizable, extensible

**What other benefits can the system get from the fine-grained modular design?**

- Enables seamless per-instance resource re-provisioning and low-overhead dynamic membership management

![Diagram of fine-grained modular architecture](image)

**System Design**

- Each cache instance is a “fat server” comprising multiple service abstractions
  - Cachelet type abstracts the service provided to the clients
  - Services (query types, resources allocated: CPU, DRAM etc.) are configurable
- Data is stored in relatively small partitions spanning multiple cachelets

**Major module performance evaluation: Hash table**

- Experiment setup (point query, client aggregators operating mode for avoiding network overhead)
  - 6 core, 2.67 GHz, 12 GB DRAM
  - Memstore: our lock-free hash table; Mercury: Memcached hash table with fine-grained bucket locks; Memcached: original 1.4.13 version

![Graphs of GET and SET performance](image)

**Current Status**

- Integrate B+ tree and trie module into the system
- Implement client side simplistic consistent hashing + data migration scheme
- Build different case studies to demonstrate the benefits of our cache framework

---

SoCC ’13, Oct. 1-3 2013, Santa Clara, CA, USA  
http://research.cs.vt.edu/dssl