Be Fast, Cheap and in Control with SwitchKV

Xiaozhou Li
Raghav Sethi
Michael Kaminsky
David G. Andersen
Michael J. Freedman
Goal: fast and cost-effective key-value store

- Target: cluster-level storage for modern cloud services
  - Massive number of small key-value objects
  - Highly skewed and dynamic workloads
  - Aggressive latency and throughput performance goals

- This talk: **scale-out flash-based storage** for this setting
Key challenge: dynamic load balancing

- How to handle the **highly skewed and dynamic workloads**?
- Today’s solution: data migration / replication
  - system overhead
  - consistency challenge
Fast, small cache can ensure load balancing

Need only cache $O(n \log n)$ items to provide good load balance, where $n$ is the number of backend nodes. [Fan, SOCC’11]

- How to efficiently serve queries with cache and backend nodes?
- How to efficiently update the cache under dynamic workloads?

E.g., 100 backends with hundreds of billions of items + cache with 10,000 entries
High overheads with traditional caching architectures

- Cache must process all queries and handle misses
- In our case, cache is small and hit ratio could be low
  - Throughput is bounded by the cache I/O
  - High latency for queries for uncached keys

Look-aside

Look-through
Switches route requests directly to the appropriate nodes

- Latency can be minimized for all queries
- Throughput can scale out with # of backends
- Availability would not be affected by cache node failures
Clients encode key information in packet headers
  ▪ Encode **key hash in MAC** for read queries
  ▪ Encode destination **backend ID in IP** for all queries

Switches maintain forwarding rules and route query packets

---

**Exploit SDN and switch hardware**

- Packet In
- L2 table: **exact match rule per cached key**
  - hit: Packet Out to the cache
  - miss: TCAM table: **match rule per physical machine**
  - Packet Out
Keep cache and switch rules updated

- New challenges for cache updates
  - Only cache the hottest $O(n \log n)$ items
  - Limited switch rule update rate
- Goal: **react quickly** to workload changes with **minimal updates**
Evaluation

• How well does a fast small cache improve the system load balance and throughput?

• Does SwitchKV improve system performance compared to traditional architectures?

• Can SwitchKV react quickly to workload changes?
Evaluation Platform

Reference backend

- 1 Gb link
- Intel Atom C2750 processor
- Intel DC P3600 PCIe-based SSD
- RocksDB with 120 million 1KB objects
- 99.4K queries per second
Evaluation Platform

- Client
  - 40 GbE
- Cache
  - Pica8 P-3922 (OVS 2.3)
  - Ryu
  - 40 GbE
- Backends
  - 40 GbE
  - Xeon Server 3
  - Xeon Server 4

# of backends: 128
backend throughput: 100 KQPS
keyspace size: 10 billion
key size: 16 bytes
value size: 128 bytes
query skewness: Zipf 0.99
cache size: 10,000 entries

Default settings in this talk

- Use Intel DPDK to efficiently transfer packets and modify headers
- Client adjusts its sending rate, keep loss rate between 0.5% and 1%
Throughput with and without caching

- Cache (10,000 entries)
- Backends aggregate (with cache)
- Backends aggregate (without cache)

Throughput (MQPS)

Workload Distribution

uniform
zipf-0.9
zipf-0.95
zipf-0.99
Throughput vs. Number of backends

backend rate limit: 50KQPS, cache rate limit: 5MQPS
End-to-end latency vs. Throughput

Throughput limit (beyond which >1% of packets are dropped)
Throughput with workload changes

Make 200 cold keys become the hottest keys every 10 seconds
Conclusion

SwitchKV: high-performance and cost-efficient KV store

• Fast, small cache guarantees backend load balancing

• Efficient content-aware OpenFlow switching
  ▪ Low (tail) latency
  ▪ Scalable throughput
  ▪ High availability

• Keep high performance under highly dynamic workloads