KeySched: Timeslot-based Hot Key Scheduling for Load Balancing in Key-Value Store

Heng Yu, Jun Bi, Chen Sun*
Tsinghua University

CCS CONCEPTS

• **Networks** → **Packet scheduling**; *Programmable networks*; *Network monitoring*;

KEYWORDS

Load balance; Key-value store; SmartNIC

ACM Reference format:
Heng Yu, Jun Bi, Chen Sun. 2018. KeySched: Timeslot-based Hot Key Scheduling for Load Balancing in Key-Value Store. In *Proceedings of ACM SIGCOMM 2018 Conference Posters and Demos, Budapest, Hungary, August 20–25, 2018 (SIGCOMM Posters-Demos '18),* 3 pages.
https://doi.org/10.1145/3234200.3234248

1 INTRODUCTION

Software based in-memory key-value store (KVS) has become a critical infrastructure that provides services for many systems in data center networks. The performance of KVS is considered as a key factor that directly determines the system efficiency [4].In a server with multi-core CPUs, traditional KVS performs *hashing* on the *requests* to evenly distribute workload to multiple cores. Even if multiple requests are querying the same key, request-based hashing may still distribute them to different cores. Therefore, *synchronization* mechanisms such as mutexes and version-based locking are adopted to ensure data consistency across CPU cores, which incurs serious performance overhead due to contention and frequent cache invalidity [5].

To address this challenge, some researches [5] partition KVS requests based on the *key* they are visiting. By allocating

*Heng Yu, Jun Bi, Chen Sun are with Institute for Network Sciences and Cyberspace, Tsinghua University and Beijing National Research Center for Information Science and Technology (BNRist). This poster is supported by the National Key R&D Program of China (No.2017YFB0801701) and National Science Foundation of China (No.61472213). Jun Bi (junbi@tsinghua.edu.cn) is the corresponding author.

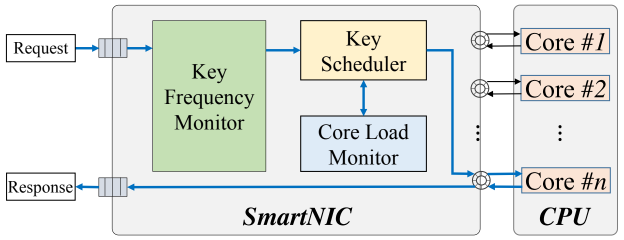


Figure 1: KeySched design overview

requests targeting at the same key to the same CPU core, the need for synchronization is eliminated and therefore the performance is significantly improved. However, real KVS workload often follows *Zipfian distribution* [1], in which a few keys are *extremely popular*. The skewed workload results in imbalanced workload among multiple cores. Hot cores in this situation suffer from serious throughput degradation (up to 30%), long tail latency and low energy efficiency [4].

Unfortunately, current researches cannot effectively address this load imbalance problem. NetCache [2] buffers hot data in switches for *read* requests to relieve the workload of servers but cannot deal with *write-intensive* workload, which is common in distributed computation such as graph computation. FlexNIC [3] supports dynamic assignment of key groups to different cores but lacks an effective mechanism to split key groups to CPU cores to ensure balancing.

To address this problem, we propose *KeySched, a timeslot-based hot key scheduling framework for KVS that targets at achieving relatively balanced core workload while maintaining key-based partition.* We first measure the hottest and average core workload of a KVS with key-based partition to motivate KeySched. Then we propose our key observation from KVS workload that *load imbalance is caused by very few extremely popular keys (i.e. hot keys).* Therefore, we design the *KeySched framework* which innovatively proposes a *timeslot-based hot key scheduling scheme* while maintaining key-based partition. We have implemented KeySched on Smart Network Interface Card (SmartNIC). Preliminary evaluation shows that KeySched could balance the core workload in KVS with little performance overhead.

2 KEYSCHED DESIGN

Motivation: Max-Ave-Ratio measurement. *Max-Ave-Ratio* is the ratio of the workload of the hottest CPU core and the average workload of all cores, which is used as a key benchmark to assess the load imbalance extent of KVS [4]. To obtain this ratio, we build a KVS platform with key-based partition according to [5]. The platform consists of two Dell R730 servers, each equipped with two Intel Xeon E5-2680 v4 CPUs (36 logical cores @2.30 GHz) and runs Ubuntu with Linux kernel 4.5.0. One server carries an Agilio CX 2x10GbE SmartNIC to perform key-based hashing to split workload across multiple cores. The other server equipped with a dual-port Intel X520 10G NIC generates KVS requests that follow the Zipfian distribution [1].We vary the number of cores used by KVS and measure the load of each core. Experimental results show that with the increase of the number of cores (from 8 to 64), the Max-Ave-Ratio increases from 1.32 to 5.61, which represents very serious load imbalance [4] that could result in significant performance degradation in KVS.

Observation: very few hot keys cause imbalance. Now we try to understand the load imbalance problem based on the measurement of KVS workload that follows the Zipfian distribution. We observe that the frequency of the hottest key takes 6.6% of the entire workload. Even if we allocate a single core for this key and evenly distribute the rest workload to the remaining 63 cores, the Max-Ave-Ratio still reaches as high as 4.45. Highly skewed workload in KVS results in the situation that very few hot keys could introduce serious load imbalance and compromise performance.

Key novelty: timeslot-based hot key scheduling. Based on our observation, keeping the hot keys at the same cores all the time is the root cause of load imbalance. A naive solution is to evenly split the workload of hot keys to multiple cores. However, doing this violates *key-based* workload partition and requires costly synchronization mechanisms across cores. To address this challenge, we innovatively propose *timeslot-based hot key scheduling* for load balancing with little overhead. We slice time into timeslots (1ms in our implementation). In each timeslot, we detect hot keys and schedule them to the core with the least load in a few recent timeslots. Therefore, hot keys are dynamically placed on different cores. Only a small portion of requests for hot keys will be processed in each core, which largely mitigates imbalance. However, timeslot-based scheduling suffers from a cache miss when scheduling a hot key. Nevertheless, experiments in §3 show that cache misses happen extremely rarely with at most 32 hot keys scheduled.

KeySched Design. Finally, as shown in Figure 1, we present the design of KeySched including three functional modules.

(1) Key Frequency Monitor: To measure key frequency with little overhead, we adopt the *Count-min Sketch* [6] that is easy to implement in SmartNIC with high resource efficiency. Count-min sketch utilizes multiple hash functions to map a request to different counters, and picks the smallest counter value as the estimated key frequency. We then attach the key frequency to the request and deliver it to the Key Scheduler.

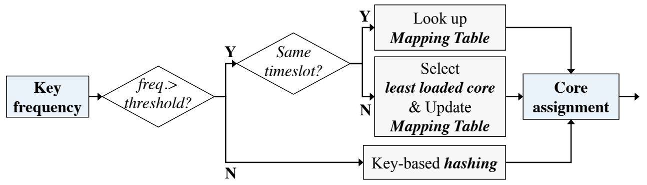
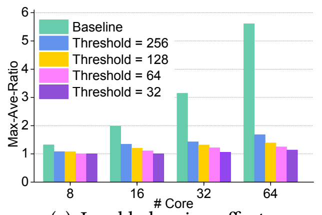
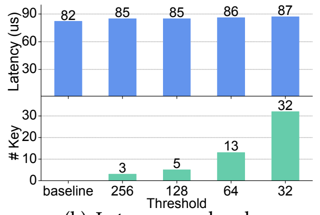


Figure 2: *Key Scheduler* workflow



(a) Load balancing effect



(b) Latency overhead

Figure 3: Scheduling effect and overhead of KeySched

We clear counters at the end of each timeslot to prepare the key frequency counting for the next timeslot.

(2) Key Scheduler: We present the workflow of Key Scheduler in Figure 2. This module first detects hot keys based on the key frequency attached in the request. If the frequency is smaller than threshold, we consider the key as a small key and follows traditional key-based hashing for core assignment. Otherwise, the key is considered as a hot key and will be scheduled to the least loaded core. We store this key-core mapping in a *Mapping Table*. For subsequent requests for this key in this timeslot, the Key Scheduler will simply look up the Mapping Table and steer the request to the same core.

(3) Core Load Monitor: This module maintains the load of each core in a few (100 in our implementation) recent timeslots. The Key Scheduler will query the Core Load Monitor and select the least loaded core to carry the detected hot key.

3 IMPLEMENTATION AND EVALUATION

We have implemented KeySched on the platform introduced in §2. We evaluate KeySched and compare it with traditional key-based partition. We vary the number of CPU cores from 8 to 64 for the KVS, and the threshold in Key Scheduler from 256 to 32. We use the server that generates KVS requests to receive responses and measure the Round-Trip-Time (RTT). As shown in Figure 3(a), with the increase of the number of cores, the load imbalance in traditional key-based partition becomes more serious. However, KeySched can achieve relatively balanced load regardless of the core number. Moreover, if we decrease the threshold, KeySched could result in a more balanced load as more hot keys will be detected and scheduled, which is illustrated in the bottom half of Figure 3(b). However, with the decrease of the threshold, key scheduling becomes more frequent and introduces larger latency overhead. Nevertheless, as shown in the top half of Figure 3(b), the latency penalty is 3 μs to 5μs, which is acceptable compared with the baseline.

REFERENCES

[1] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In *ACM symposium on Cloud computing*. 143–154.

[2] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. 2017. NetCache: Balancing Key-Value Stores with Fast In-Network Caching. In *SOSP*. 121–136.

[3] Antoine Kaufmann, SImon Peter, Naveen Kr Sharma, Thomas Anderson, and Arvind Krishnamurthy. 2016. High performance packet processing with FlexNIC. In *ACM SIGARCH Computer Architecture News*.

[4] Sheng Li, Hyeontaek Lim, and Lee Victor, et al. 2015. Architecting to achieve a billion requests per second throughput on a single key-value store server platform. In *ACM SIGARCH Computer Architecture News*, Vol. 43. 476–488.

[5] Hyeontaek Lim, Donsu Han, David G Andersen, and Michael Kaminsky. 2014. MICA: A holistic approach to fast in-memory key-value storage.

[6] Minlan Yu, Lavanya Jose, and Rui Miao. 2013. Software Defined Traffic Measurement with OpenSketch.. In *NSDI*. 29–42.