Poster: Stateless CPU-aware Datacenter Load-Balancing

Tom Barbette, Marco Chiesa, Gerald Q. Maguire Jr. and Dejan Kostić
KTH Royal Institute of Technology

ABSTRACT

Today, datacenter operators deploy Load-balancers (LBs) to efficiently utilize server resources, but must over-provision server resources (by up to 30%) because of load imbalances and the desire to bound tail service latency. We posit one of the reasons for these imbalances is the lack of *per-core load statistics* in existing LBs. As a first step, we designed CrossRSS, a CPU core-aware LB that dynamically assigns incoming connections to the least loaded cores in the server pool. CrossRSS leverages knowledge of the dispatching by each server's Network Interface Card (NIC) to specific cores to reduce imbalances by more than an order of magnitude compared to existing LBs in a proof-of-concept datacenter environment, processing 12% more packets with the same number of cores.

CCS CONCEPTS

• **Networks** → **Packet scheduling**; **Middle boxes / network appliances**; **Network servers**; **Data center networks**; **Network resources allocation**.

1 INTRODUCTION AND BACKGROUND

Datacenter networks use LBs to spread incoming connections across a pool of servers. When a LB *uniformly* spreads the load, none of the servers is overly utilized and all requests are served *efficiently* without high tail latency. Unfortunately, today's LBs may cause imbalances in the load on servers, forcing operators to over-provision their server resources by up to 30% [5].

Limitations of today's LBs. Today's LBs cannot achieve uniform load balancing because of a lack of fine-grained information about each server's utilization of its multiple cores. Both stateless[10] and stateful[5] state-of-the-art datacenter LBs rely on *aggregated* per-server statistics, such as number of connections per server or average CPU utilization per server. However, the actual servers spread the incoming connections over their cores using stateless hash-based mechanisms (*i.e.* RSS) that map the connection's identifier to a specific core without storing any per-connection information. Such hash-based mechanisms are known to cause imbalances when elephant flows must be load balanced (see Hedera [1]). Imagine two large connections that map to the same core in an under-loaded server. These connections will experience high tail latencies even when the aggregated per-server load used by the datacenter LB is low. Therefore, others have tried to solve the intra-server per-core load balancing problem by migrating connections among cores [2]. As migration of packets among cores means the state in L1/L2 caches is lost and context switches take up to 10 microseconds, recent network stacks [6] and specific network intensive applications) [8] all advocate against migration of packets between CPU cores, and therefore revert to RSS to locally load balance packets. Moreover, intra-server LBs require modifications to the servers while a transparent solution would be easier to deploy.

The challenge. Current solutions require modification of server applications, eventually using server resources (such as CPU [2, 11] or SmartNICs [12]) for the purpose of load-balancing itself resulting in a difficult & costly adoption. At the same time, traditional LBs have no means to target specific CPU cores without relying on tagging and server modifications for dispatching [7].

The quest for a CPU-aware datacenter LB. We pose the question: *"Can we devise a load balancing mechanism that takes into account the per CPU core load – to achieve more uniform load balancing without requiring modifications to the servers?"*. Later, we discuss the main challenges in realizing such an LB, propose a scalable design, and perform a simple evaluation.

2 SYSTEM DESIGN

Incorporating per-core statistics into a LB to dispatch incoming connections to thousands of servers is non-trivial. First, fetching fine-grained information from the servers imposes additional bandwidth and other overheads on the servers. Secondly, an appropriate core-selection mechanism must be used to assign a new connection to the servers while minimizing load imbalance. Compared to selecting a single core, finding an optimal core choice is inherently more complex as the number of cores is larger than the number of servers. Third, we cannot modify the RSS hash-based mechanism on the NICs of the servers. Thus, given a new connection, we can only pick a single core at each server, as determined by the output of the RSS hash-based core selection. We leave SmartNICs to future work and note they are an expensive option and potentially introduce additional latency. Figure 1 shows our system design. An agent on each server periodically reports to an LB controller the load of each CPU core and the mapping used by RSS to map connections to

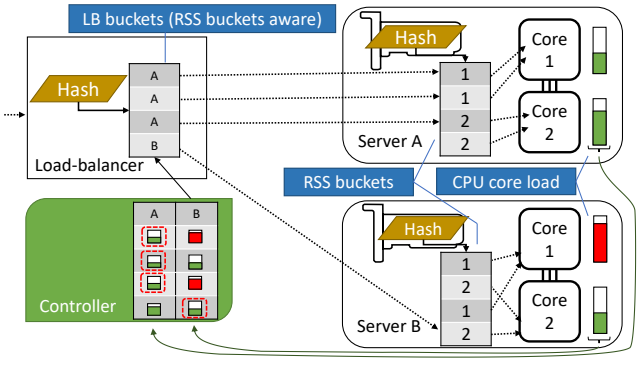


Figure 1: System design: an agent periodically rewrites a server selection table according to the RSS mapping in the servers and corresponding CPU loads

CoNEXT '20, December 1–4, 2020, Barcelona, Spain

ACM ISBN 978-1-4503-7948-9/20/12.
https://doi.org/10.1145/3386367.3431672

the cores. More specifically, RSS stores an array of buckets where each bucket maps to a specific core. The least significant bits of the hash of a connection identifier are used to select the bucket used by a connection to retrieve its core assignment. While bucket assignments may be changed on-the-fly [2] this leads to inter-core packet processing, thus changes in assignment should be avoided.

The LB controller (shown as a green box) periodically takes as input the bucket-to-core-load statistics from each server and computes a bucket-to-server mapping. For each bucket, the controller has information about the load of one specific core for each server. The controller computes the least loaded core among these cores. It then constructs a bucket-to-server mapping based on the selected cores and their corresponding servers and installs this mapping in the data-plane of the LB (see the LB buckets' mapping in the figure). The LB data-plane uses this mapping to assign connections to the servers. When the load on these cores changes, the controller updates the mapping. A bucket is selected based on the hash's least significant bits, as computed by the servers' NICs, *i.e.* RSS. Therefore, a given connection will always map to the same bucket and will be directed to the least loaded core of all servers for that specific bucket. We use a pseudo-random number generator (with server IDs as a seed) to randomize each server's indirection table, thus a pseudo-random core of each server is considered for each bucket of the LB, rather than a fixed set of cores for each server, similarly to the advanced Power-of-K-choices load balancing technique [9]. Since connections are spread across k minimally loaded CPUs, a small information delay will not cause a server to be overloaded. Changing this mapping might break existing connections, *i.e.* routing them to an incorrect server. We propose to use stateless techniques as in Cheetah [4] to avoid breaking such connections without the need to make the LB stateful. CrossRSS also enables scaling through hierarchical aggregation, as the minimal load of a group of servers is the same as the minimal load of any server.

3 EVALUATION

We simulate our proposal using FastClick[3]. A module reads a real campus network trace and pipes it to a load-balancing module that dispatches packets to 4 simulated servers. Each server has a simulated NIC, utilizing a hashing mechanism similar to RSS to dispatch packets to 4 cores, represented by packet counters, running on 16 different CPU cores of an 18-cores Xeon Gold 6140. In Figure 2, we compare multiple combinations of algorithms for load-balancing: **RR** uses a per-flow round-robin dispatcher (to avoid breaking connections) to the servers, **HASH** uses hashing to dispatch packets to servers and **Pow2** uses the Power-of-2-choices. In all cases, the NIC uses RSS. **CrossRSS** is implemented using a 10 Hz CPU load reporting frequency to a controller module that rewrites the entries in the LB buckets table as explained in Section 2. **Pow2** uses the load reported at the same frequency. In Figure 2a we measure (i) server load variance (i.e., number of packets received by each CPU divided by the average number of packets) and (ii) the per-core variance across all machines for consecutive 1 s windows. While existing methods show a variance between servers below 0.5%, they do not decrease the variance in load between cores inside each server. CrossRSS has, on average, more than an order of magnitude smaller variance as it simultaneously achieves good spreading both between servers and inside servers. Figure 2b shows the consequence of such imbalance, when each core runs a heavy network function chain. As the load increases, some cores will drop packets as they become overloaded, while others are starving. CrossRSS's even distribution more fully utilizes all cores.

Future work. A variety of overheads may hinder the deployment of a CPU-aware LB. We investigated the *potential* of such an LB compared to existing solutions and plan to evaluate the scalability of such an LB through extensive micro-benchmarks.

REFERENCES

[1] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *7th USENIX Conference on Networked Systems Design and Implementation*, NSDI. USENIX, 2010.

[2] Tom Barbette, Georgios P. Katsikas, G. Q. Maguire Jr., and Dejan Kostić. RSS++: Load and state-aware receive side scaling. In *15th International Conference on Emerging Networking Experiments And Technologies*, CoNEXT '19, 2019.

[3] Tom Barbette, Cyril Soldani, and Laurent Mathy. Fast userspace packet processing. In *Eleventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS, pages 5–16. IEEE, 2015.

[4] Tom Barbette, Chen Tang, Haoran Yao, Dejan Kostić, G. Q. Maguire Jr., Panagiotis Papadimitratos, and Marco Chiesa. A high-speed load-balancer design with guaranteed per-connection-consistency. NSDI, pages 667–683. USENIX, 2020.

[5] Daniel E. Eisenbud, Cheng Yi, Carlo Contavalli, Cody Smith, Roman Kononov, Eric Mann-Hielscher, Ardas Cilingiroglu, Bin Cheyney, Wentao Shang, and Jinnah Dylan Hosein. Maglev: A fast and reliable software network load balancer. NSDI, pages 523–535. USENIX, 2016.

[6] EunYoung Jeong, Shinae Woo, Muhammad Asim Jamshed, Haewon Jeong, Sunghwan Ihm, Dongsu Han, and KyoungSoo Park. mTCP: a highly scalable user-level TCP stack for multicore systems. NSDI, pages 489–502. USENIX, 2014.

[7] Georgios P. Katsikas, Tom Barbette, Dejan Kostić, Rebecca Steinert, and Gerald Q. Maguire Jr. Metron: NFV Service Chains at the True Speed of the Underlying Hardware. NSDI'18, pages 171–186. USENIX, 2018.

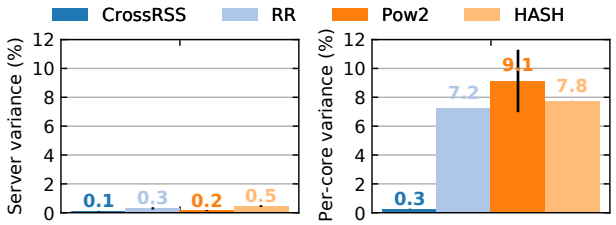
[8] Hyeontaek Lim, Donsu Han, David G Andersen, and Michael Kaminsky. MICA: A holistic approach to fast in-memory key-value storage. NSDI. USENIX, 2014.

[9] Michael David Mitzenmacher. *The Power of Two Choices in Randomized Load Balancing*. PhD thesis, University of California, Berkeley, 1996.

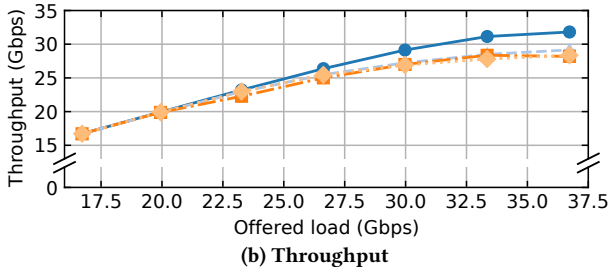
[10] Vladimir Olteanu, Alexandru Agache, Andrei Voinescu, and Costin Raiciu. Stateless datacenter load-balancing with Beamer. NSDI. USENIX, 2018.

[11] Amy Ousterhout, Joshua Fried, Jonathan Behrens, Adam Belay, and Hari Balakrishnan. Shenango: Achieving high CPU efficiency for latency-sensitive datacenter workloads. NSDI, pages 361–378. USENIX, 2019.

[12] Alexander Rucker, Muhammad Shahbaz, Tushar Swamy, and Kunle Olukotun. Elastic rss: Co-scheduling packets and cores using programmable nics. In *Proceedings of the 3rd Asia-Pacific Workshop on Networking 2019*, pages 71–77, 2019.



(a) Variance of the number of packets (i) between servers (ii) between all cores of all machines



(b) Throughput

Figure 2: Comparison of four load-balancing techniques. CrossRSS exhibits a near-perfect variance (a), allowing it to handle more packets without overloading any core (b).

REVISION 1

The scripts to reproduce the experiments are available at https://github.com/tbarbette/crossrss.