Accelerating Kubernetes with In-network Caching

Stefanos Sagkriotis
University of Glasgow
Glasgow, United Kingdom
s.sagkriotis.1@research.gla.ac.uk

Dimitrios Pezaros
University of Glasgow
Glasgow, United Kingdom
dimitrios.pezaros@glasgow.ac.uk

ABSTRACT

We present a new Kubernetes architecture that leverages in-network caching to accelerate one of Kubernetes' core components, its key-value store. We also identify performance limitations of previous in-network caching platforms and propose a new platform that demonstrates better throughput and scalability by utilising a different replication method.

1 INTRODUCTION

Kubernetes has been the orchestration framework that drove the transition to the era of microservices for many system administrators [2]. By utilising a set of virtualisation tools, it provided enhanced management and a hardware-agnostic approach towards service deployment. It constitutes a trustworthy and extensible framework that enables service deployment in diverse computing environments.

One of Kubernetes' integral components is etcd [7], which maintains a consistent Key-Value Store (KVS) and provides coordination services to other control-plane components, including the API server and the Container Network Interface (CNI) (responsible for networking of the deployed services). Etcd uses a quorum approach to maintain consistency, and more specifically the Raft protocol. A trade-off to this algorithm is the lack of horizontal scalability [12, 18]. For example, the latency to a write query can be up to 80ms for a cluster of 9 nodes and up to 160ms for a cluster of 21 nodes [11].

Data plane programmability, through programming tools like P4 and hardware innovations like the Tofino ASIC, accelerated the performance of various services, including KVS [5, 8, 15]. A KVS can be completely deployed in Programmable Data Plane (PDP) and accommodate queries in line rate at sub-Round Trip Time [13]. This has been shown to improve throughput and latency up to orders of magnitude. A write query can be completed in the order of μs, an important improvement compared to the aforementioned performance of etcd. In this work, we examine the integration of in-network KV caching as a way to accelerate the performance of etcd and Kubernetes.

Our contributions: we identify the shortcomings of previous state-of-the-art (SotA) in-network replication frameworks; we implement a different replication mechanism that displays better scalability and reduced latency while maintaining strong consistency; we suggest the addition of Kubernetes components that work alongside the previous ones and decide which KV pairs should be placed in PDP based on real-time metrics; we contribute to the vision of an end-to-end programmable platform that utilises in-network computing and virtualisation to accelerate performance.

2 DESIGN

Our design can be broken down in two main domains: the PDP components that utilise P4 to enable in-network replication; and the Kubernetes framework that was extended to support offloading KV pairs to PDP based on real-time metrics.

By examining NetChain [13], the fastest in-network replication platform, we identified aspects that make its scalability bottleneck to the performance of a single node. By using Chain Replication as the underlying replication mechanism, the last node in a chain of switches is treated as reference for consistency [18]. This limits the platform's performance to the response rate of this node and requires full chain traversals to retrieve a reply, generating an unnecessary amount of traffic [16]. Moreover, the used packet structure requires all chain node IPs to be included in the packets, further increasing packet processing times alongside traffic.

To alleviate these design limitations, we implemented a different replication method – CRAQ [16]. With CRAQ, each node can reply to a read query as long as the KV pair requested is up to date (clean). This can offer great performance and scalability improvements over Chain Replication. Previous work has validated our experiments, which show that the majority of queries generated from Kubernetes to etcd are read queries. Approximately a third of the queries are writes. Therefore, this method appears promising when considering Kubernetes workloads.

We implemented CRAQ in P4 by using the PSA architecture to define Match-Action pairs based on header fields [9]. An overview of the framework's design, named NetCRAQ, can be seen on the right side of Figure 1 (all red blocks define our new suggested components, green blocks represent etcd components, yellow represents Calico CNI [17], and blue is used for Kubernetes). The packet header defines the type of operation (read, write, delete). We then proceed by identifying the state of the KV pair – clean or dirty (write

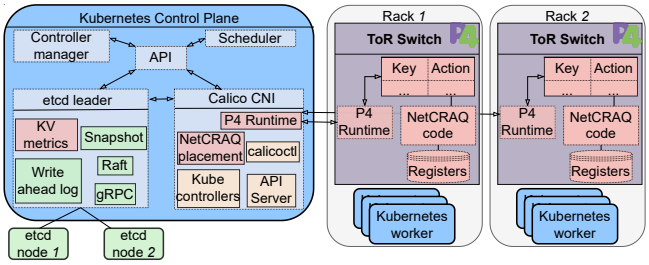

Figure 1: Design overview.

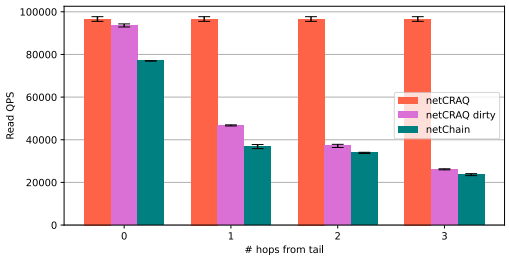

Figure 2: Max read QPS vs distance from tail.

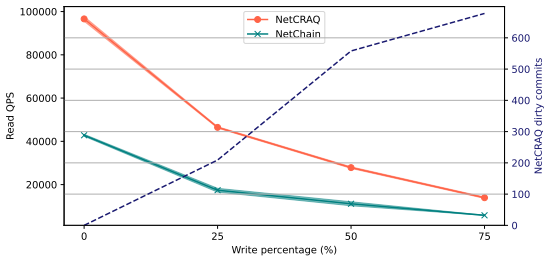

Figure 3: Performance under mixed read/write workloads.

not yet fully committed). KV pairs are stored in fast-access TCAM registers. Then, a reply is generated or the clean value is fetched from the last chain node. Otherwise, the write is locally committed and propagated to other chain nodes. A delete query works in the same manner but commits a null value.

Two Kubernetes components have been extended to support the integration of PDP: etcd and CNI [3]. A monitoring component has been added to etcd in order to identify most commonly accessed KV pairs. It uses the integrated Promitheus endpoint to read metrics [1]. The most frequent KV pairs are selected as candidates for deployment in PDP. Monitoring is also in place for the values already existing in PDP, which have counters for access frequency. These statistics are obtained through P4Runtime which is executed as part of the CNI [10]. Through the NetCRAQ placement scheduler, also located within the CNI, these metrics are compared and a decision on which values will be transferred to PDP is made. The most frequently read values are placed in data plane.

3 EVALUATION

Both NetCRAQ and NetChain have been implemented in P4 and tested using BMv2 with Mininet and P4-utils API [4, 6, 14].

We evaluate the throughput of both platforms based on the maximum attainable rate at which they can provide responses to queries. The measurements are in Queries Per Second (QPS). In figure 2, we test NetCRAQ's ability to provide responses to read queries of a clean version versus NetChain's behaviour for the same queries. We monitor the throughput each node is able to achieve given the distance it has from the reference node. NetCRAQ's throughput appears unaffected by distance when the queried object is clean. This effectively enables every participating node to reply to a query, increasing scalability significantly. The reduction of required hops benefit NetCRAQ's performance: 4.08× higher throughput for queries directed to the head of the chain. In case of dirty objects, throughput is still higher than NetChain with the difference being attributed to the smaller packet size used by NetCRAQ and its ingress control logic that minimises parsing.

We evaluate both platforms under realistic workloads containing a mix of reads and writes. The results are shown in Figure 3. Starting from a read-only workload, we gradually increase the percentage of writes with a step of 25%. The performance of the platforms is determined by their attainable response rate. NetCRAQ achieves more than double the read throughput for all write percentages. The efficiency it shows on read queries enables higher throughput regardless of the write percentage. Adequate register cells need to be budgeted to maintain all dirty versions before they can be committed in the chain. This is depicted by the increasing amount of dirty commits observed in the right y axis of Figure 3.

4 FUTURE STEPS

Our future steps include transferring our implementations to the Tofino ASIC in order to evaluate performance differences in hardware. Because the obtained performance improvements stem from optimisations not tied to hardware, i.e., the reduced number of hops and reduced traffic/parsing, we expect the results of our emulation environment to be indicative of real-world behaviour. We also want to conclude work in Kubernetes components that redirect queries to PDP and evaluate the performance difference between the default setup and a setup that utilises in-network replication.

REFERENCES

[1] Prometheus Authors. 2022. Prometheus. (2022). Retrieved May 30, 2022 from https://prometheus.io/

[2] The Kubernetes Authors. 2022. Kubernetes - Production-Grade Container Orchestration. (2022). Retrieved May 30, 2022 from https://kubernetes.io/

[3] The Kubernetes Authors. 2022. Kubernetes Components. (2022). Retrieved May 30, 2022 from https://kubernetes.io/docs/concepts/overview/components/

[4] Mininet Project Contributors. 2022. Mininet. (2022). Retrieved May 30, 2022 from http://mininet.org/

[5] Intel Corporation. 2022. Intel Tofino. (2022). Retrieved May 30, 2022 from https://www.intel.co.uk/content/www/uk/en/products/network-io/programmable-ethernet-switch/tofino-series.html

[6] The BMv2 Developers. 2022. BEHAVIORAL MODEL (bmv2). (2022). Retrieved May 30, 2022 from https://github.com/p4lang/behavioral-model

[7] etcd Authors. 2022. etcd - A distributed, reliable key-value store for the most critical data of a distributed system. (2022). Retrieved May 30, 2022 from https://etcd.io/

[8] Open Networking Foundation. 2022. P4 Open Source Programming Language. (2022). Retrieved May 30, 2022 from https://p4.org/

[9] The P4.org Architecture Working Group. 2022. P416 Portable Switch Architecture (PSA). (2022). Retrieved May 30, 2022 from https://p4.org/p4-spec/docs/PSA-v1.1.0.html

[10] The P4.org API Working Group. 2022. P4Runtime Specification. (2022). Retrieved May 30, 2022 from https://p4.org/p4-spec/p4runtime/main/P4Runtime-Spec.html

[11] Andrew Jeffery, Heidi Howard, and Richard Mortier. 2021. Rearchitecting Kubernetes for the Edge. In *Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking (EdgeSys '21)*. Association for Computing Machinery, New York, NY, USA, 7–12. https://doi.org/10.1145/3434770.3459730

[12] Ricardo Jiménez-Peris, M. Patiño Martínez, Gustavo Alonso, and Bettina Kemme. 2003. Are Quorums an Alternative for Data Replication? *ACM Trans. Database Syst.* 28, 3 (sep 2003), 257–294. https://doi.org/10.1145/937598.937601

[13] Xin Jin, Xiaozhou Li, Haoyu Zhang, Nate Foster, Jeongkeun Lee, Robert Soulé, Changhoon Kim, and Ion Stoica. 2018. NetChain: Scale-Free Sub-RTT Coordination. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 35–49. https://www.usenix.org/conference/nsdi18/presentation/jin

[14] Networked Systems Group (NSG@ETH). 2022. P4-Utils. (2022). Retrieved May 30, 2022 from https://nsg-ethz.github.io/p4-utils/index.html

[15] Amedeo Sapio, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan R. K. Ports, and Peter Richtárik. 2019. Scaling Distributed Machine Learning with In-Network Aggregation. *CoRR* abs/1903.06701 (2019). arXiv:1903.06701 http://arxiv.org/abs/1903.06701

[16] Jeff Terrace and Michael J. Freedman. 2009. Object Storage on CRAQ: High-Throughput Chain Replication for Read-Mostly Workloads. In *2009 USENIX Annual Technical Conference (USENIX ATC 09)*. USENIX Association, San Diego, CA.

[17] Inc. Tigera. 2022. Project Calico. (2022). Retrieved May 30, 2022 from https://www.tigera.io/project-calico/

[18] Robbert van Renesse and Fred B. Schneider. 2004. Chain Replication for Supporting High Throughput and Availability. In *6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8, 2004*, Eric A. Brewer and Peter Chen (Eds.). USENIX Association, 91–104. http://www.usenix.org/events/osdi04/tech/renesse.html