Network-Accelerated Cluster Scheduler

Radostin Stoyanov
University of Oxford
radostin.stoyanov@eng.ox.ac.uk

Wesley Armour
University of Oxford
wes.armour@oerc.ox.ac.uk

Noa Zilberman
University of Oxford
noa.zilberman@eng.ox.ac.uk

ABSTRACT

Efficient use of computing clusters is crucial in large-scale data centers: even small gains in utilization can save millions of dollars. However, as the number of microsecond-scale tasks increases, using a CPU to schedule tasks becomes inefficient. Cluster scheduling running within the network can solve this problem, and brings additional benefits in scalability, performance and power efficiency. However, the resource constraints of programmable network devices make network-accelerated cluster scheduling hard. In this paper we propose P4-K8s-Scheduler, a network-accelerated cluster scheduler for Kubernetes implemented on a programmable network device. Preliminary results show that by scheduling Pods in the network at line-rate, P4-K8s-Scheduler can reduce the scheduling overheads by an order of magnitude compared to state-of-the-art Kubernetes schedulers.

KEYWORDS

Kubernetes; Scheduling; In-network Computing; P4

1 INTRODUCTION

Recent years have seen a growing use of real-time data processing applications. These applications facilitate popular online services such as web search [2, 27], social networks [3, 24], video streaming [6, 11], and real-time object localization [19, 26]. In addition, with the increased adoption of microservices and serverless computing [8], the number of short tasks scheduled in real-time has grown significantly [1, 9, 10].

Today, applications typically depend on low-latency tasks that require service times of a microsecond or less [14]. These tasks are inherently latency-sensitive and non-optimal scheduling placement can significantly impact application performance [4]. Therefore, efficient scheduling in large-scale clusters is becoming increasingly more important. At the same time, efficient resource utilization can significantly reduce expenditure and operational costs [5]. A standard method used today to address this problem is to deploy a centralized scheduler that can monitor all cluster nodes and be able to make high-quality placement decisions [12]. However, in practice, scheduling a large number of tasks as short as 1 μs can overwhelm centralized schedulers [13, 20].

This work develops P4-K8s-Scheduler, a network-accelerated scheduler for Kubernetes [7] implemented on a high-performance programmable network device. A Kubernetes cluster consists of a set of worker machines, called *nodes*, which run containerized applications, and a *control plane* that manages the nodes. Each node runs an agent, called *kubelet*, that ensures that all containers are running and healthy. The smallest deployable unit of compute in Kubernetes is called *Pod*. A Pod is a group of tightly coupled containers with shared storage and network resources.

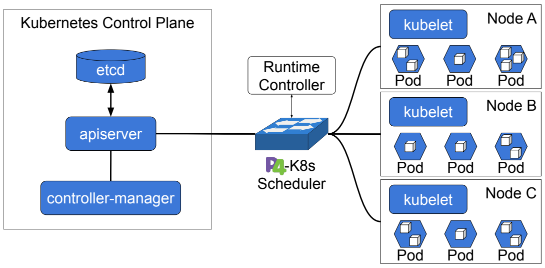

Figure 1: Network-Accelerated Kubernetes Scheduler

2 DESIGN

P4-K8s-Scheduler is designed to address two main problems for achieving high-performance: (i) reduce the scheduling overhead of short tasks, and (ii) increase scheduling throughput of Kubernetes clusters. P4-K8s-Scheduler can run as a P4 program on a programmable switch (illustrated in Fig. 1), or on Infrastructure/Data Processing Unit (IPU/DPU) attached to the Kubernetes Control Plane. This approach allows performing scheduling decisions at line rate based on Pod requirements such as number of CPUs, amount of memory, storage, as well as preferred/required affinity.

A workflow comparison of the standard Kubernetes scheduler (kube-scheduler) and P4-K8s-Scheduler is shown in Fig. 2. The kube-scheduler watches for Pods without assigned node, finds the best node for the Pod to run on, and notifies the API Server about this decision in a process called *binding*. In contrast, P4-K8s-Scheduler processes scheduling requests in the network and forwards the request to the assigned node.

Network Protocol. The communication protocol allows the API Server to (i) register nodes and (ii) send scheduling requests to the P4-K8s-Scheduler. The corresponding packet headers are shown in Fig. 3. These headers are encapsulated by a UDP packet header, which allows packets to be processed by standard network hardware.

Node Registration. Nodes are registered with the scheduler using unique names. However, the node name can have a length of up to 253 characters (2024 bits) and Kubernetes supports up to 5000 nodes [17]. The memory and compute constraints of programmable network devices limit the operations that can be performed with such values [15]. To address this challenge, the API Server has been extended to map node names to 16-bit node ID values. These values are used as register array index on the programmable network device to keep track of the available resources of each node in the cluster. The API Server sends registration packets with node ID, number of CPUs, amount of memory and storage. These packets also specify operation (OP): *ADD*, *UPDATE* or *REMOVE*. The scheduler parses the packet and updates the registers accordingly. It then sends the packet to the runtime controller to update the match-action tables used for filtering and scoring.

Pod Scheduling. Scheduling is the process of mapping Pods to nodes in the cluster. However, implementing complex scheduling

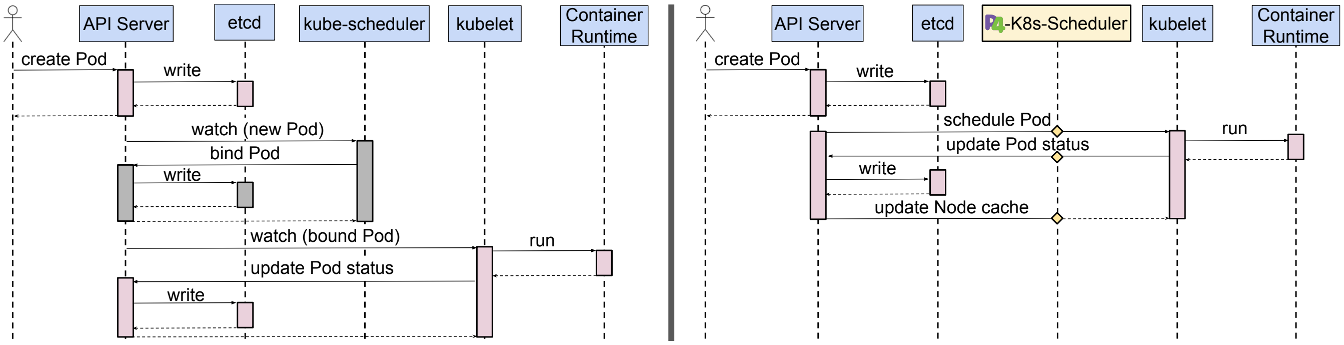

Figure 2: Workflow comparison between kube-scheduler and P4-K8s-Scheduler.

Node Registration

Node ID	CPU	Memory	Storage	OP
16b	16b	16b	16b	8b

Pod Scheduling Request

Pod ID	Node ID	CPU	Storage	Memory	RAK	RAV	PAK	PAV	RAO	PAO	SS	PL
32b	16b	16b	16b	16b	32b	32b	32b	32b	8b	8b	8b	

Figure 3: Packet Headers used by P4-K8s-Scheduler.

policies on a programmable network device is not straightforward. For example, it is not feasible to iterate over an array with the number of CPUs for each node at terabits per second speed. We address this problem by pre-computing the results of node filtering and scoring in the runtime controller and updating the data plane with appropriate set of match-action entries. Similar to node registration, the API Server encodes Pod scheduling requests as packets. Each packet contains scheduling requirements for CPU, memory, storage, required and preferred affinity key/value/operation (RAK/RAV/RAO/PAK/PAV/PAO), scoring strategy (SS). In addition, the packet also contains specification payload (PL) that is used by the kubelet to create the Pod. If the Pod is unschedulable, the packet is sent back to the API Server.

3 PROTOTYPE IMPLEMENTATION

P4-K8s-Scheduler was developed using Kubernetes v1.23.5 running on Ubuntu 20.04 server with AMD EPYC 7302P 16 core CPU @ 3GHz, 256GB RAM, Mellanox ConnectX-5, and 64 × 100*GE* Intel Tofino platform.

Node Filtering. Filtering a set of nodes that match a scheduling request is performed using match-action tables. The table entries are bitmaps corresponding to node IDs, where a set bit indicates satisfying the Pod requirements. A logical conjunction of all filter bitmaps is calculated with *node filter* action using *binary AND* operation and the result is stored in P4 user metadata.

Node Scoring. All nodes are scored based on available resources. A match-action table using the user metadata bitmap and scoring strategy as keys assigns the node with highest score to the Pod request. The scheduling response is then forwarded to the destination Ethernet, IP address, and output port matching the node ID. Fig 4 depicts the mechanism for filtering and scoring.

Runtime Controller. The P4 controller processes node registration packets from the API Server as well as status updates from kubelets. It generates score and filter match-action table entries, and groups table updates into batches and transactions to avoid race conditions.

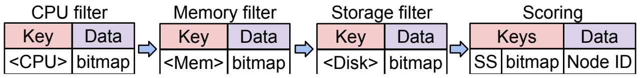

Figure 4: Node filtering and scoring mechanism.

4 PRELIMINARY EVALUATION

P4-K8s-Scheduler achieves median task placement latency of ~50 μs with 1,000-machine switch support, and ~170 ms total delay for 1,000 scheduling requests. The scheduling overhead has been reduced by an order of magnitude compared to state-of-the-art Kubernetes schedulers [13], and by up to 50% compared to other network-accelerated schedulers [16].

5 CONCLUSION & FUTURE WORK

P4-K8s-Scheduler is a network-accelerated cluster scheduler for Kubernetes that runs on a programmable network device. P4-K8s-Scheduler increases the efficiency and scalability of Kubernetes by performing scheduling decisions at line-rate on packets exchanged between nodes in the cluster. Preliminary results demonstrate that P4-K8s-Scheduler can reduce task scheduling overheads by an order of magnitude compared to state-of-the-art Kubernetes schedulers. Future work will explore advanced scheduling policies (e.g., data locality [21], network-aware [23]), flow-based scheduling [12], preemption [22] and multi-profile schedulers [18, 25].

6 ACKNOWLEDGMENTS

The authors would like to thank Dr. Adrian Reber for helpful discussions regarding Kubernetes. This work was supported by the EPSRC Doctoral Training Partnership (DTP) [grant number EP/N509711/1]. We acknowledge support from Intel.

REFERENCES

[1] Lixiang Ao, Liz Izhikevich, Geoffrey M. Voelker, and George Porter. 2018. Sprocket: A Serverless Video Processing Framework. In *Proceedings of the ACM Symposium on Cloud Computing*. Association for Computing Machinery, New York, NY, USA, 263–274. https://doi.org/10.1145/3267809.3267815

[2] Dhruv Arya and Ganesh Venkataraman. 2017. Search Without a Query: Powering Job Recommendations via Search Index at LinkedIn *(SIGIR '17)*. Association for Computing Machinery, New York, NY, USA, 1347. https://doi.org/10.1145/3077136.3096470

[3] Dhruv Arya, Ganesh Venkataraman, Aman Grover, and Krishnaram Kenthapadi. 2017. Candidate Selection for Large Scale Personalized Search and Recommender Systems. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Shinjuku, Tokyo, Japan) *(SIGIR '17)*.

Association for Computing Machinery, New York, NY, USA, 1391–1393. https://doi.org/10.1145/3077136.3082066

[4] Luiz Barroso, Mike Marty, David Patterson, and Parthasarathy Ranganathan. 2017. Attack of the killer microseconds. *Commun. ACM* 60, 4 (2017), 48–54.

[5] Luiz André Barroso and Urs Hölzle. 2009. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*.

[6] Niels Bouten, Steven Latré, Jeroen Famaey, Werner Van Leekwijck, and Filip De Turck. 2014. In-Network Quality Optimization for Adaptive Video Streaming Services. *IEEE Transactions on Multimedia* (2014).

[7] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. 2016. Borg, Omega, and Kubernetes: Lessons Learned from Three Container-Management Systems over a Decade. *Queue* 14, 1 (jan 2016), 70–93. https://doi.org/10.1145/2898442.2898444

[8] Paul Castro, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. 2019. The Rise of Serverless Computing. *Commun. ACM* 62, 12 (nov 2019), 44–54. https://doi.org/10.1145/3368454

[9] Yanpei Chen, Sara Alspaugh, and Randy H. Katz. 2012. Interactive Analytical Processing in Big Data Systems: A Cross-Industry Study of MapReduce Workloads. *CoRR* abs/1208.4174 (2012).

[10] Yanpei Chen, Archana Ganapathi, Rean Griffith, and Randy Katz. 2011. The Case for Evaluating MapReduce Performance Using Workload Suites. In *2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*. 390–399. https://doi.org/10.1109/MASCOTS.2011.12

[11] Sadjad Fouladi, Riad S Wahby, Brennan Shacklett, Karthikeyan Vasuki Balasubramaniam, William Zeng, Rahul Bhalerao, Anirudh Sivaraman, George Porter, and Keith Winstein. 2017. Encoding, Fast and Slow: Low-Latency Video Processing Using Thousands of Tiny Threads. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 363–376.

[12] Ionel Gog, Malte Schwarzkopf, Adam Gleave, Robert NM Watson, and Steven Hand. 2016. Firmament: Fast, centralized cluster scheduling at scale. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 99–115.

[13] Ionel Gog, Malte Schwarzkopf, Adam Gleave, Robert N. M. Watson, and Steven Hand. 2016. Firmament: Fast, Centralized Cluster Scheduling at Scale. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*.

[14] Stephen Ibanez, Alex Mallery, Serhat Arslan, Theo Jepsen, Muhammad Shahbaz, Changhoon Kim, and Nick McKeown. 2021. The nanoPU: A Nanosecond Network Stack for Datacenters. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*.

[15] Theo Jepsen, Daniel Alvarez, Nate Foster, Changhoon Kim, Jeongkeun Lee, Masoud Moshref, and Robert Soulé. 2019. Fast String Searching on PISA. In *Proceedings of the 2019 ACM Symposium on SDN Research* (San Jose, CA, USA) *(SOSR '19)*. Association for Computing Machinery, New York, NY, USA, 21–28. https://doi.org/10.1145/3314148.3314356

[16] Ibrahim Kettaneh, Sreeharsha Udayashankar, Ashraf Abdel-hadi, Robin Grosman, and Samer Al-Kiswany. 2020. *Falcon: Low Latency, Network-Accelerated Scheduling*. Association for Computing Machinery, New York, NY, USA, 7–12. https://doi.org/10.1145/3426744.3431322

[17] Kubernetes Documentation. 2022. Considerations for Large Clusters. https://kubernetes.io/docs/setup/best-practices/cluster-large/. Online; accessed 23 May 2022.

[18] Kubernetes Documentation. 2022. Scheduler Configuration. https://kubernetes.io/docs/reference/scheduling/config/. Online; accessed 23 May 2022.

[19] Luyang Liu and Marco Gruteser. 2021. EdgeSharing: Edge Assisted Real-time Localization and Object Sharing in Urban Streets. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*. 1–10. https://doi.org/10.1109/INFOCOM42981.2021.9488830

[20] Sarah McClure, Amy Ousterhout, Scott Shenker, and Sylvia Ratnasamy. 2022. Efficient Scheduling Policies for Microsecond-Scale Tasks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 1–18.

[21] Thomas Rausch, Alexander Rashed, and Schahram Dustdar. 2021. Optimized container scheduling for data-intensive serverless edge computing. *Future Generation Computer Systems* (2021).

[22] Adrian Reber. 2021. Kubernetes and Checkpoint Restore. KubeCon + CloudNativeCon North America.

[23] José Santos, Tim Wauters, Bruno Volckaert, and Filip De Turck. 2019. Towards Network-Aware Resource Provisioning in Kubernetes for Fog Computing Applications. In *2019 IEEE Conference on Network Softwarization (NetSoft)*. 351–359. https://doi.org/10.1109/NETSOFT.2019.8806671

[24] Aneesh Sharma, Jerry Jiang, Praveen Bommannavar, Brian Larson, and Jimmy Lin. 2016. GraphJet: Real-Time Content Recommendations at Twitter. *Proc. VLDB Endow.* 9, 13 (sep 2016), 1281–1292. https://doi.org/10.14778/3007263.3007267

[25] Radostin Stoyanov and Noa Zilberman. 2020. MTPSA: Multi-Tenant Programmable Switches. In *Proceedings of the 3rd P4 Workshop in Europe* (Barcelona, Spain) *(EuroP4'20)*. Association for Computing Machinery, New York, NY, USA, 43–48. https://doi.org/10.1145/3426744.3431329

[26] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. 2015. Efficient Object Localization Using Convolutional Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[27] James Wong, Brendan Collins, and Ganesh Venkataraman. 2018. Large Scale Search Engine Marketing (SEM) at Airbnb. In *The 41st International ACM SIGIR Conference on Research Development in Information Retrieval* (Ann Arbor, MI, USA) *(SIGIR '18)*. Association for Computing Machinery, New York, NY, USA, 1357–1358. https://doi.org/10.1145/3209978.3210207