Exploring Efficient ML-based Scheduler for Microservices in Heterogenous Clusters

ISCA 2022 Submission **#NaN** – Confidential Draft – Do NOT Distribute!!

***Abstract*—In the recent years, cloud computing is going though a major transformation throughout its system stack, from its application to hardware. Its services are increasingly shifting from large monolithic applications to complex graphs with many single-purpose microservices, which offer many advantages in terms of deployment and development. On the other hand, cloud datacenters are becoming increasingly heterogeneous as they host more GPUs, FPGAs, and ASICs. While this heterogenous hardware can not only accelerate but also expand the capability of microservices, they further complicate the complex action space in microservices scheduling. Importantly, the convergence of the changes in both applications and hardware brings up unique challenges in datacenter scheduling for microservices. Recent innovations has shown that data-driven Machine Learning (ML) approaches leveraging neural networks can improve both end-to-end latency of the applications and probability of QoS violations. However, these works have focused on a rather homogeneous clusters which may become prohibitive as the scheduling problem gets more complex as the datacenters become heterogeneous. This paper first analyzes the potential limitations of the previous approaches and explores a new dimension of *efficiency* in the development of schedulers for microservices by incorporating a *light-weight* ML-based model. To this end, the paper develops a prototype light-weight ML-based scheduler dubbed Octopus that harbors a decision tree to *efficiently* schedule microservices on heterogeneous clusters. Comparisons against conventional scheduling techniques including *First-Fit*, *Random*, and *Kubernetes-like* schedulers show that Octopus provides 6.35× faster end-to-end latency.**

I. Introduction

Cloud services are increasingly shifting from large monolithic applications to complex graphs with many single-purpose microservices [11, 30]. In fact, cloud providers such as Amazon, Twitter, Netflix, and Apple have adopted this model of development [1, 3]. This model of development allows the cloud providers to benefit from accelerating development, lifting language of framework restrictions, and simplifying correctness and performance debugging [11]. However, microservices complicate resource management as dependencies between them introduce backpressure effects and cascading Quality-of-Service (QoS) violations [30]. On the other hand, the cloud datacenters are becoming increasingly heterogeneous [7] with GPUs [2], FPGAs [5, 22], and ASICs [16]. While the central motivation behind traditional datacenters with general-purpose servers were its compatibility to any application, heterogeneous computing fabrics in the cloud has the potential to not only accelerate but also expand the capability of microservices. Despite the significant benefits, the heterogeneity in the datacenters further complicate the scheduling as the cluster schedulers need to be aware of the different profiles of heterogeneous machines when allocating resources to applications.

Recent innovations such as [20, 30] has shown that data-driven Machine Learning (ML) approaches such as neural networks can improve both end-to-end latency of the applications and probability of QoS violations. However, these works focus on a rather homogeneous clusters with general-purpose servers and may not be sufficient to cope with the heterogeneous clusters. For instance, actions spaces in the schedulers increase significantly as the datacenters become heterogeneous and relatively compute-intensive schedulers based on neural networks may become prohibitive. In fact, it may not only increase the scheduling time but also the end-to-end latency of the services, leading to QoS violations for interactive and latency-critical services with strict performance constraints. To this end, this paper first analyzes the potential limitations of the previous approaches by developing an infrastructure that simulates heterogeneous clusters. We first develop a random microservice workload generator based on Nightcore [15] and Tailbench [18], and a simulator to simulate a *heterogeneous, multi-node cluster* with servers, each with CPU and additional accelerator such as GPU or ASICs. Then, we add a scheduler to make decisions on the placement of the incoming microservices tasks. On top of this simulation infrastructure, we develop multiple scheduling techniques: *First-Fit*, *Random*, *Kubernetes-like*, and *light-weight ML-based scheduler* dubbed Octopus. Evaluations show that using even a light-weight ML-based schedulers, instead of neural network-based schedulers, can provide up to 6.35× end-to-end latency improvement compared to conventional schedulers, while also keeping the scheduling overhead low. Considering the significant benefits from the light-weight ML-based scheduler and the increasing complexity of scheduling microservices onto heterogeneous clusters that may make neural network-based schedulers prohibitive, the results suggest exploring the new dimension of *efficiency* in the schedulers.

II. Exploring Efficient ML-based Scheduler for Microservices in Heterogeneous Clusters

We consider a scheduling scenario where there are multiple servers in the cloud datacenter and the tasks arrive at arbitrary time t_n. Figure 1 illustrates the microservice scheduling on the aforementioned *heterogeneous cluster* where each server in the cluster has different capability: some servers only with CPUs and some equipped with GPUs or ASICs in addition to the CPUs. As the microservice tasks arrive, the Scheduler make an *online* decision regarding which machine to assign the

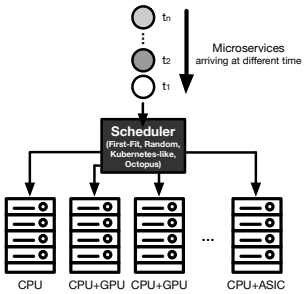

Fig. 1: System overview of microservices scheduling.

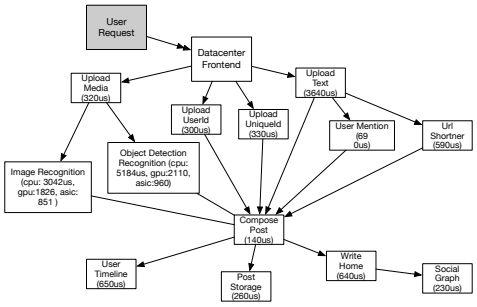

Fig. 2: Microservice architecture of Social Network from DeathStarBench [11].

task to. To analyze the implications of different microservice scheduling techniques for heterogeneous cluster, we develop a simulation infrastructure that lets us experiment with various scheduling techniques: both conventional (Section II-A) and light-weight ML-based (Section II-B) algorithms.

A. Conventional Scheduling Techniques

First-Fit scheduler iterates through each server and checks if the server meets the task's requirement. If the requirement is met, the job is scheduled. Otherwise, the algorithm goes on to check the next server. This is sequential and scales linearly with the number of servers in the cluster.

Random scheduler randomly selects a server to check if the server meets the task's requirement. If the requirement is met, the task is scheduled on it. Otherwise, the algorithm continues to randomly select another server until it finds one that meets the task's requirement.

Kubernetes-like scheduler simulates the default Kube scheduler [4]. First, the scheduler selects a subset of feasible servers that meet the tasks' requirement such as the hard constraint on which device (e.g., GPU) the task can be executed on. Then, the scheduler ranks all feasible servers based on an empirical scoring rule to determine the server to schedule. This ranking is sequential and a potential bottleneck as its running time scales linearly with the number of servers in the cluster.

B. Octopus: Light-weight ML-based Scheduler

Contrary to the above-mentioned conventional scheduling techniques, recent innovations [20, 30] take advantage of neural networks to cope with the complex search space of task scheduling. We note, however, that the heterogeneity in our setup may further complicate the scheduling problem that would require more complex neural architectures, hence too long an inference time to cope with the strict QoS requirements imposed on the cloud services. To cope with this micro-second scale, strict requirements, we explore *light-weight* ML-based microservice scheduling algorithm for heterogeneous clusters.

Model architecture. The main driving factors in designing the scheduler was the *scheduling performance* and *efficiency*. We consider *decision trees* to predict the latency of the microservices as it meets both requirements. In fact, the decision trees are known for its logarithmic complexity which leads to orders of magnitude faster speed compared to other ML approaches such as neural networks. Also, it not only requires little data to train to a good accuracy but also requires little time to train. Therefore, it also suggests potential for further offline training to continually improve the performance of the scheduler.

Training. We train our decision tree-based scheduler offline using workload traces. These traces are generated by a random sampling algorithm which takes in inputs the microservices and latency critical applications from [15, 18]. We provide more details about the trace generation in the Section III-A. In addition to the above microservice-architecture applications, we consider an additional application that includes DNNs. We consider a SocialNetwork application shown in Figure 2 with image recognition and object detection [13, 27]. We measure the inference latency of these DNNs on CPU, GPU, and DNN accelerator. We train our model by feeding it microservices with its meta-data: {CPU requirement, memory requirement, disk requirement, dependent microservice, accelerator type} as the input feature and use the measured latency as the target. THe model achieves above 90% accuracy on a randomly generated test set. We note that the test set contains randomly generate traces which are not part of the training set.

Scheduling. Octopus first performs inference on all available devices then rank the devices in the increasing order of latency. Unlike the *Kubernetes-like* scheduler, Octopus overrides the hard constraints of the tasks. This allows Octopus to take full advantage of all the acceleration opportunities. However, naively selecting the fastest device while ignoring the hard constraints may lead to a serious contention problem, where a task α (can be executed anywhere) arrived in t_α placed on device A (e.g., GPU) may preclude task β (can be executed only on device A) that arrived in $t_\alpha < t_\beta$ from being placed on device A. In this case, if task β is on the critical path of some application, this scheduling may lead to backpressure effects and cascading QoS violations. To prevent such issue, we include an optional constraint to the Octopus called `FractionOfHeldOutDevices` where some fraction p of a device are always held out for the tasks that can only be executed on that device. Importantly, the scheduling overhead of this approach can be minimal as we can perform the inference in parallel using multiple threads. On the other hand, the above-mentioned *First-Fit* and *Random* schedulers are not amenable to such parallelization.

III. Evaluation

A. Methodology

Microservices workload. To analyze the performance of different scheduling techniques, we use randomly sampled

subgraph from the microservices dependency graph of the Social Network application form DeathStarBench [11]. Each sample comes with a launch time, the microservices' hardware requirements. The number of microservices per sample ranges from one to four. In addition to the above sampled microservices, we inject some batch processing applications with long execution time to increase the fidelity of our simulation in a datacenter setup. We randomly generate 10,000 samples of the application for the job scaling and node scaling analysis.

Simulation infrastructure for heterogeneous cluster. We develop a custom simulation infrastructure coupled with a random workload generator to analyze the implications of different microservice scheduling onto heterogeneous clusters. The simulator estimates the end-to-end latencies using the latency and server utilization statistics taken from Nightcore [15] and Tailbench [18]. For DNNs, the simulator uses ONNX Runtime [9] to get the DNN model latency on GPUs and a cycle-accurate simulator for ASIC DNN accelerator.

B. Experimental Results

Job Scaling Analysis. We perform a job scaling analysis to see how the end-to-end latency changes as we increase the number of jobs (each job might have multiple microservices) on a fixed sized cluster. Figure 3 summarizes the job scaling on a cluster with one with 20 CPU, 5 GPU, and 5 ASIC, while varying the number of jobs from 100 through 2000. We observe a near linear increase in the speed-up of our ML based scheduler as compared to the conventional schedulers. This can be attributed to two factors: decision trees have quick inference time and ML based scheduler takes full advantage of the acceleration capabilities provided by the GPUs and ASICs.

Figure 4 summarizes the job scaling on a cluster with 40 CPU, 10 GPU and 10 ASIC, while varying the number of jobs from 100 to 7000 to see the effect when the cluster is oversubscribed. We observe the same increase in speed-up until the cluster is fully subscribed. After the cluster is fully subscribed, end-to-end latencies gradually converge towards other conventional schedulers. Since all servers are busy, the scheduler is not able to find servers with acceleration capabilities.

Node Scaling Analysis. Figure 5 and Figure 6 demonstrates the case when we increase the number of nodes while the jobs are fixed. In this case, we observe a constant end-to-end latency for each scheduler because for a fixed set of jobs we keep increasing the number of available nodes and hence there is lesser contention of physical resources. The reason we observe a slight dip in speedup for Octopus is because when the number of nodes increase, the inference time also increases.

IV. Related Works

Microservices gained traction with friendliness to fast deployment cycles and better scalability. They typically are run on CPUs within the datacenter of cloud service providers. However, with new heterogeneous hardware and emerging accelerators being available on the cloud (e.g. GPU, FPGA,

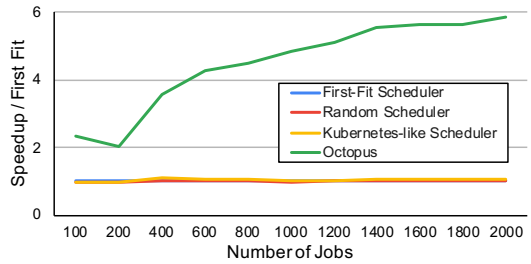

Fig. 3: Speedup when the number of jobs (each job may have multiple microservices) are scaled on a 20 CPU, 5 GPU, 5 ASIC cluster.

Fig. 4: Speedup when the number of jobs (each job may have multiple microservices) are scaled on a 40 CPU, 10 GPU, 10 ASIC cluster.

Google's TPU, and AWS's Inferentia and Gaudi for machine learning inference and training), the existing schedulers might not be a good option since they do not consider the rich and growing portfolio of heterogeneous hardware when making scheduling decisions. There has been limited previous efforts on the scheduling of microservices on heterogeneous clusters including accelerators. Our work investigates a simple yet effective machine learning based scheduling mechanism for microservice workloads. We discuss the related works:

Microservices DeathStarBench [11] presents a set of comprehensive benchmarks for microservice. DeathStarBench discusses the implications of microservices on architectural, system and networking, and cluster management. In the aspect of cluster management, microservices complicate cluster management as its complex dependencies might trick cluster manager into making inefficient scaling and scheduling decisions. Wisp [26] applies rate limiting and back-pressuring to microservices such that the microservices are able to meet their SLA. GrandSLAm [17] estimates the completion time of individual requests to batch or reorder microservices; thereby making the microservices meet their SLA requirement with higher throughput. SoftSKU [25] and Accelerometer [24] characterize representative Facebook/Meta's production microservices on system and architectural level. SoftSKU is focused on deriving the hardware and system configurations across a limited set of server CPU models for optimized microserivces performance. Accelerometer profiles system overhead, such as I/O processing, compression, serialization and encryption/decryption. Accelerometer derives an analytic model to predict potential speedup of these system overhead if they are built into accelerators. Sage [10] uses unsupervised machine learning models to capture the root cause of performance unpredictability and further act upon it. Nightcore [15] is

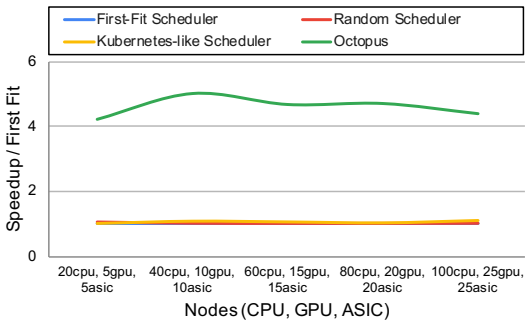

Fig. 5: Speedup when the number of nodes are scaled for 1000 jobs (each job may have multiple microservices)

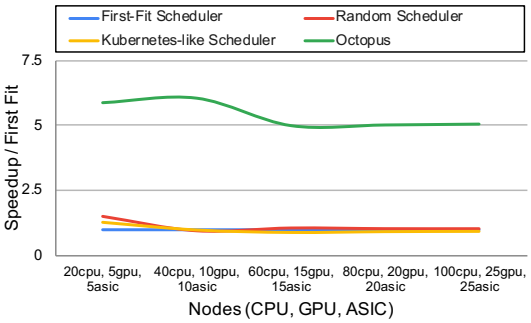

Fig. 6: Speedup when the number of nodes are scaled for 2000 jobs (each job may have multiple microservices)

a serverless runtime optimized for interactive microservices especially with sub-millisecond latency requirements.

Cluster management and scheduling: Conventional cluster managers, e.g. Borg [29], Mesos [14], YARN [28], do not reason about neither the accelerator performance model nor inter-changeable between general purpose CPU and accelerators. Omega [23] and Hydra [6] are able to reason about limited heterogeneity that comes from different CPU models and memory capacities, but they don't take accelerators into account while making the scheduling decision. Paragon [8] uses analytical methods together with collaborative filtering to classify and schedule latency critical workloads considering co-location interference and server heterogeneity. Gavel [21] is a scheduler designed specifically for deep learning training workload on a GPU/CPU cluster. Gavel considers the hardware heterogeneity but is limited to interchangeable computation on training workload between CPU and GPU only but no other accelerator. ML-based cluster managers emerge as an attractive alternative because they are driven by the underlying workload and is more likely to closely optimize for it. Decima [20] schedules data processing jobs on Spark clusters by using neural network based reinforcement learning. Sinan [30] uses a convolution neural network to handle dependencies and interactions among microsevices and a boosted trees model to prevent possible SLO latency violation caused by queued requests. Both Decima and Sinan operate on a decision interval around a second. Though ML-based cluster managers make good decision using well-trained neural networks, this second-level granularity of decision interval will not work well with latency-sensitive, interactive microservices.

Machine learning for systems: The research community has utilized machine learning techniques to optimize systems beyond cluster management and scheduling. Researchers have use ML in storage system to improve the predictability of storage access latency [12]. Learned index [19] is applied onto database to replace tree-based index to map a key to its data location. Zhang and Huang [31] argue that we can take the learning based approach even further to replace heuristics based policy and mechanisms in the entire operating system.

Conclusion

Cloud has been going through a transformation over the past decade: services shifting from monolithic application to microservices and cloud servers becoming increasingly heterogeneous. This paper explores an efficient ML-based scheduling algorithm to cope with the complex resource management problem that arises from the convergence of microservices and heterogeneous clusters. Evaluation with variegated microservices on a simulated heterogeneous cluster shows significant gains in end-to-end latency.

References

[1] "Decomposing twitter: Adventures in service-oriented architecture." [Online]. Available: https://www.slideshare.net/InfoQ/decomposing-twitter-adventures-in-serviceoriented-architecture

[2] "Deep learning on gpu instances." [Online]. Available: https://aws.amazon.com/machine-learning/accelerate-machine-learning-P3/

[3] "The evolution of microservices." [Online]. Available: http://www.slideshare.net/adriancockcroft/microservices-workshop-craft-conference.

[4] "Kubernetes," https://kubernetes.io.

[5] E. Chung, J. Fowers, K. Ovtcharov, M. Papamichael, A. Caulfield, T. Massengil, M. Liu, D. Lo, S. Alkalay, M. Haselman, C. Boehn, O. Firestein, A. Forin, K. S. Gatlin, M. Ghandi, S. Heil, K. Holohan, T. Juhasz, R. K. Kovvuri, S. Lanka, F. van Megen, D. Mukhortov, P. Patel, S. Reinhardt, A. Sapek, R. Seera, B. Sridharan, L. Woods, P. Yi-Xiao, R. Zhao, and D. Burger, "Accelerating persistent neural networks at datacenter scale," in *HotChips*, 2017.

[6] C. Curino, S. Krishnan, K. Karanasos, S. Rao, G. M. Fumarola, B. Huang, K. Chaliparambil, A. Suresh, Y. Chen, S. Heddaya, R. Burd, S. Sakalanaga, C. Douglas, B. Ramsey, and R. Ramakrishnan, "Hydra: a federated resource manager for data-center scale analytics," in *NSDI*, 2019.

[7] C. Delimitrou, "The increasing heterogeneity of cloud hardware and what it means for systems," 2020. [Online]. Available: https://www.sigops.org/2020/the-increasing-heterogeneity-of-cloud-hardware-and-what-it-means-for-systems/

[8] C. Delimitrou and C. Kozyrakis, "Paragon: Qos-aware scheduling for heterogeneous datacenters," in *ASPLOS*, 2013.

[9] O. R. developers, "Onnx runtime," https://onnxruntime.ai/, 2021.

[10] Y. Gan, M. Liang, S. Dev, D. Lo, and C. Delimitrou, "Sage: Practical and Scalable ML-Driven Performance Debugging in Microservices," in *ASPLOS*, 2021.

[11] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson, K. Hu, M. Pancholi, B. Clancy, C. Colen, F. Wen, C. Leung, S. Wang, L. Zaruvinsky, M. Espinosa, Y. He, and C. Delimitrou, "An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud and Edge Systems," in *ASPLOS*, 2019.

[12] M. Hao, L. Toksoz, N. Li, E. E. Halim, H. Hoffmann, and H. S. Gunawi, "LinnOS: Predictability on unpredictable flash storage with a light neural network," in *OSDI*, 2020.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[14] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," in *NSDI*, 2011.

[15] Z. Jia and E. Witchel, "Nightcore: Efficient and scalable serverless computing for latency-sensitive, interactive microservices," in *ASPLOS*, 2021.

[16] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *ISCA*, 2017.

[17] R. S. Kannan, L. Subramanian, A. Raju, J. Ahn, J. Mars, and L. Tang, "Grandslam: Guaranteeing slas for jobs in microservices execution frameworks," in *EuroSys*, 2019.

[18] H. Kasture and D. Sanchez, "Tailbench: a benchmark suite and evaluation methodology for latency-critical applications," in *IISWC*, 2016.

[19] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, "The case for learned index structures," in *SIGMOD*, 2018.

[20] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *SIGCOMM*, 2019.

[21] D. Narayanan, K. Santhanam, F. Kazhamiaka, A. Phanishayee, and M. Zaharia, "{Heterogeneity-Aware} cluster scheduling policies for deep learning workloads," in *OSDI*, 2020.

[22] A. Putnam, A. Caulfield, E. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. Prashanth, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. R. Larus, E. Peterson, A. Smith, J. Thong, P. Y. Xiao, and D. Burger, "A reconfigurable fabric for accelerating large-scale datacenter services," in *ISCA*, 2014.

[23] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: Flexible, scalable schedulers for large compute clusters," in *EuroSys*, 2013.

[24] A. Sriraman and A. Dhanotia, "Accelerometer: Understanding acceleration opportunities for data center overheads at hyperscale," in *ASPLOS*, 2020.

[25] A. Sriraman, A. Dhanotia, and T. F. Wenisch, "Softsku: Optimizing server architectures for microservice diversity @scale," in *ISCA*, 2019.

[26] L. Suresh, P. Bodik, I. Menache, M. Canini, and F. Ciucu, "Distributed resource management across process boundaries," in *SoCC*, 2017.

[27] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 6105–6114. [Online]. Available: https://proceedings.mlr.press/v97/tan19a.html

[28] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache hadoop yarn: Yet another resource negotiator," in *SoCC*, 2013.

[29] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *EuroSys*, 2015.

[30] Y. Zhang, W. Hua, Z. Zhou, G. E. Suh, and C. Delimitrou, "Sinan: Ml-based and qos-aware resource management for cloud microservices," in *ASPLOS*, 2021.

[31] Y. Zhang and Y. Huang, ""learned": Operating systems," *SIGOPS Oper. Syst. Rev.*, vol. 53, no. 1.