Less is More: Learning Simplicity in Datacenter Scheduling

Wenkai Guan and Cristinel Ababei
Electrical and Computer Engr., Marquette University
1515 W. Wisconsin Ave., Milwaukee, WI USA
E-mail: {wenkai.guan, cristinel.ababei}@marquette.edu

***Abstract*—In this paper, we present a new scheduling algorithm, Qin2, for heterogeneous datacenters. Its goal is to improve performance measured as jobs completion time by exploiting increased server heterogeneity using deep neural network (DNN) models. The proposed scheduling framework uses an efficient automatic feature selection technique, which significantly reduces the training data size required to train the DNN to levels that provide satisfactory prediction accuracy. Its efficiency is especially helpful when the DNN model is re-trained to adapt it to new types of application workloads arriving to the datacenter. The novelty of the proposed scheduling approach lies in this feature selection technique and the integration of simple and training-efficient DNN models into a scheduler, which is deployed on a real cluster of heterogeneous nodes. Experiments demonstrate that the Qin2 scheduler outperforms state-of-the-art schedulers in terms of jobs completion time.**

***Index Terms*—datacenters, heterogeneity, scheduling, deep neural networks**

I. Introduction

In developing artificial intelligence (AI) based scheduling approaches for heterogeneous datacenters, one face several challenges including how to identify models that are sample-efficient and easily adaptable. An AI model such as a deep neural network (DNN) is sample-efficient if it needs less data for its training in contrast to large-scale approaches that require huge amounts of training data. Easily adaptable or generalizable models are models that can be easily adapted by re-training to quickly capture inputs that may be different than those that were used during the initial training. In our context, such new inputs represent new types of application workload that arrive to the datacenter scheduler. Recent work [1], [2] focused on the generalizable challenge using complex DNN models trained with large-scale training data; but, training such comple DNNs is costly and not necessarily eco-friendly. The study in [3] tackled the sample-efficiency challenge by employing only selected input features that were fed into simple prediction techniques (linear regression); they reported that such simple models outperform complex machine learning models. However, finding features that show linear relationship with target objectives is difficult or they do not even exist, and often, one only has available features that show *non-linear* relationships with target objectives.

In this paper, we propose to learn simplicity in developing new datacenter scheduling algorithms by automatic sample-efficient feature selection techniques applied to simple or minimal training-efficient DNN. Our main contributions include: 1) We propose an automatic sample-efficient feature selection technique based on the *SuperFeatures* concept, to significantly reduce the required training data size that results in satisfactory performance levels. 2) We develop simple training-efficient DNN models used to predict MIPS (millions instructions per second) of application workloads. These DNNs can be easily and efficiently re-trained to adapt to new application workloads. 3) We incorporate these "less is more" techniques into the Qin2 scheduler that is based on the D-choices greedy scheduling for heterogeneous datacenters. 4) We evaluate the proposed Qin2 scheduler on a real small cluster with real-world application workloads. The rest of the paper is structured as follows. Section II reviews related work on scheduling. Section III presents the Qin2 scheduler. Section IV evaluates the Qin2 scheduler. Section V concludes the paper.

II. Related Work on Scheduling

There has been significant prior work on scheduling algorithms in datacenters. Here, we only briefly review the most relevant previous works, including those against we will compare our proposed scheduler. At datacenter or cluster-level, the work in [4] proposed the Mage scheduler that explored intra and inter server heterogeneity. The research in [5] proposed the Qin scheduler as a unified cross-layer cluster-node scheduling, which considered interference and heterogeneity for multi-objective optimizations in heterogeneous datacenters. The study in [6] presented a deep Q-Network for optimization of job scheduling in both information technology and cooling systems. At the server or node-level, the research in [7], [8] employed long short-term memory (LSTM) and DNN models to predict workloads in multicore processors and to develop tread-to-core scheduling strategies. The work in [9] leverages imitation learning based scheduler for optimizing the performance of domain-specific applications.

III. Proposed Cluster-level Scheduling Approach

A. Overview

The system level diagram describing the proposed cluster-level scheduling framework is presented in Fig. 1. The *Cluster Manager* is in charge with the management of all the components of the framework. It runs on a dedicated server and

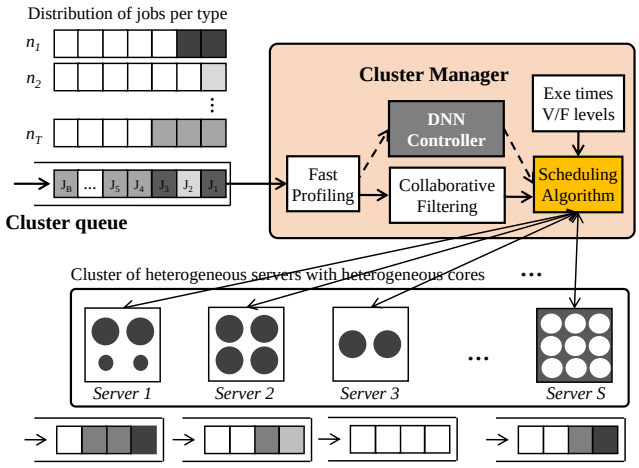

Fig. 1. System level diagram of the cluster-level scheduling framework.

Algorithm 1: Cluster-level scheduling algorithm.

Inputs: Incoming jobs/applications M to cluster
Outputs: Application-to-server scheduling at cluster-level
Function ***CLUSTER-LEVEL-SCHEDULING()***
  BenchmarkProfiling() // 20% of applications
  FeatureSelection() // sample-efficient feature selection
  TrainSimpleDNN() // training-efficient DNN
  for m *in* M **do**
    H = HeterogeneityPrediction(m) // Simple DNN based
    D-ChoicesGreedyScheduling(H)
  end
end

is comprised of several components. The role of the *Collaborative Filtering Controller* is to estimate the heterogeneity and interference scores during Phase 1 as described later on (Section III-B). Its function is replaced by the *DNN Controller* in Phase 3. The *DNN Controller* is designed to capture the job's characteristics and their relationship to the scheduling decisions. The manager is also responsible for recording and storing training data pairs into a dataset, which will be used for supervised training of the simple DNN model in Phase 2. The role of the *Scheduling Algorithm* is to perform the scheduling of jobs that arrive to the cluster queue.

Algorithm 1 describes the *cluster-level scheduling algorithm*. First, a relatively small number of the incoming application workloads (around 20% of applications) is quickly profiled. Then, we use a sample-efficient feature selection technique that automatically selects critical features from the profiled data to train a simple DNN model. For the rest of the incoming application workloads, the function *HeterogeneityPrediction(m)* uses the pre-trained simple DNN to estimate the heterogeneity score, which will be used by *D-Choices Greedy Scheduling Algorithm* (described later) for application-to-server scheduling.

In this paper, heterogeneity is captured by considering *different types of servers* and *different server configurations* (e.g., different voltage and frequency (V/F) levels). *Heterogeneity scores* for applications are calculated simply as normalized application performance scores measured in million of instructions per second (MIPS) for each type of server for each possible V/F pair setting. We assume that workloads run on systems with the same ISA. However, the heterogeneity score would capture indirectly the impact of different ISAs because different ISAs would result in different number of instructions, the number which is used in the calculation of the heterogeneity score.

B. Sample-Efficient Feature Selection

Here, we describe the proposed profiling based feature selection technique:

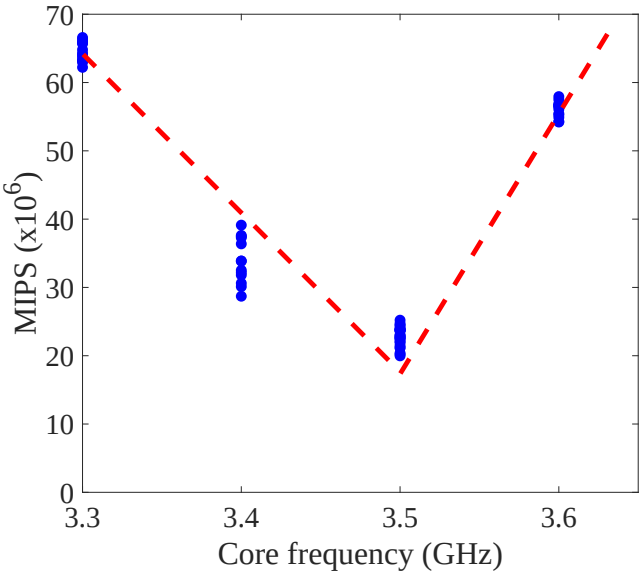

Fig. 2. Relationship of MIPS with V/F levels - identified during profiling in Step 1.

Step 1: Initialization of *SuperFeatures*. We fast profile a small number of the incoming application workloads to identify the features that show strong correlation [3] (e.g., Pearson coefficient) to the target objective (MIPS in our case). For instance, Fig. 2. shows an example where the core frequency (always used in a pair with a voltage setting) has a strong correlation to MIPS according to the profiled data for the *barnes* application from the Splash-2 benchmark suite [14]. The output of Step 1 is the list of *SuperFeatures* that includes all critical features identified during profiling. For example, in the case of MIPS, the *SuperFeatures* list contains the configuration-level feature (V/F levels), server-level features (num. of cores, profiled instruction count, and memory), and application-level features (num. of threads, application size, and input size).

Step 2: Ranking the *SuperFeatures* list. We use the forward

stepwise feature selection algorithm from [10] to rank all features in the list *SuperFeatures*. We first sort the features according to the correlation (i.e., Pearson coefficient [3]) to the target objective in descending order; then, we choose the first feature as the *pivot* feature from the *SuperFeatures* list to train the simple DNN and record n-fold cross-validation ρ. We then add one-by-one the rest of the *SuperFeatures* to the *pivot* feature to re-train the simple DNN and keep only those features that improve the n-fold cross-validation ρ. We finally rank the features in *SuperFeatures* list according to the improvement they provide to the n-fold cross-validation ρ. The output of Step 2 is the ranked list of *SuperFeatures*.

Step 3: Selecting sample-efficient features. We choose the top 1 feature in each of the three categories (configuration-level features, server-level features, and application-level features) as the final sample-efficient features. These features significantly reduce the training data size, but at the same time they represent the most critical features for the target objective, i.e., MIPS. The output of Step 3 is the list of sample-efficient features. For example, in the case of MIPS, the sample-efficient features turned out to be: V/F levels (as configuration-level feature), profiled instructions count (as server-level feature), and application size (as application-level feature).

Further discussion: A challenge in the proposed approach is related to how to identify all the candidate features in the *SuperFeatures*? In addressing this challenge, datacenter operators can leverage their expert knowledge of the application to provide a list of candidate features. Also, one can leverage tracing statements embedded into workloads by the application developers. For example, applications from Splash-2 and Parsec 3.0 suites can identify the number of threads and input size as candidate features based on the data collected by the application tracing statements or instruments.

C. Training-Efficient Simple DNN

We now introduce the training-efficient simple DNN model that uses the sample-efficient features identified using the strategy presented in Section III-B. The latest previous work from [1], [2] used large-scale complex DNN models that are generalizable to different jobs; however, training such DNN is costly and not eco-friendly. Therefore, in this paper, we adopt the philosophy of "less is more" and propose to use only training-efficient simple DNN models (a few layers with a dozen neurons only) that can be easily and efficiently re-trained at runtime to capture the behavior of any new types for application workloads. Please note that the key novelty is the idea of using training-efficient simple DNN models, and not using DNN models, which has been done before.

The DNN is implemented as a series of stacked restricted Boltzmann machines and provides the main function to the *DNN Controller* from Fig. 1. Its construction and training are done following already well established techniques. The novelty consists in the types and number of input sample-efficient features as well as of outputs that are employed. In addition, to address the lack of training data, we propose

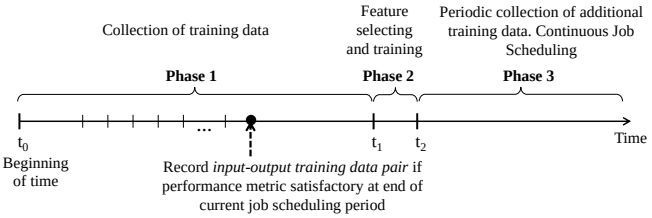

Fig. 3. Timeline of operation of proposed framework from Fig. 1 to illustrate the generation of training data pairs for the DNN model.

in our framework to develop the ability to generate training data *automatically* in three phases as illustrated in Fig. 3. In Phase 1, training input-output data pairs are collected as those data points which have resulted in satisfactory outcome in terms of performance. We collect only those data points, rather than all that are generated, because we want for the DNN to be refined during the supervised training in Phase 2 only with the best training data. In this way, the DNN model will capture only desired behavior and will thus be able to provide later on, in Phase 3, the best recommendations to the scheduling algorithm. The input features to the DNN model include selected sample-efficient features that use information collected during the fast profiling (e.g., Step 1 discussed earlier) of any newly arrived job. This information includes performance counters sampled, V/F levels, and other features discussed before during the fast profiling step, which represent a signature trace for the application. Please note that the fast profiling time is a *user defined parameter* that is passed to the *perf* tool, and the user can change it based on the different types of workloads. The outputs of the DNN model represent the heterogeneity scores (discussed in Section III-A).

Prediction Model: We propose to use the training-efficient simple DNN model thus developed for prediction of MIPS. We hand-tune a simple 4-layer DNN with 29 neurons (baseline DNN) and compare it to linear regression (commonly used in previous work) and to collaborative filtering (used by Qin scheduler [5]). The comparison is done using workloads generated as explained in Section IV. Note that one can use Adatune tool [11] to generate advanced simple DNN with fine-tuned neurons, but here, we show the case of a hand-tuned *baseline* DNN. To assess predictions, we use R^2 (coefficient of determination, the higher, the better) [12], which generally range from 0 to 1. We collect the training and predicting overhead (measured in seconds) because they are indicative of how generalizable or adaptable to new workloads at runtime the models are. The results of the comparison are summarized in Table I. We can conclude from Table I that using simple DNN for predicting MIPS is reasonable and achieves a good trade-off in accuracy and training/predicting overhead.

Further discussion: What happens when new application workloads arrive? The proposed idea of using training-efficient simple DNN is adaptable (generalizable) to new workloads because it can be easily re-trained using profiled training data collected from Phase 1 at runtime as illustrated in Fig. 3.

TABLE I
COMPARISON OF HAND-TUNED DNN MODEL TO LINEAR REGRESSION AND COLLABORATIVE FILTERING APPROACHES.

Prediction Model	Training and Predicting (s)	Accuracy (R^2)
Linear Regression	0.0015	0.12
Collaborative Filtering	20.4579	0.76
Simple DNN	3.8427	0.81

D. D-Choices Greedy Scheduling

The scheduling algorithm distributes the jobs (e.g., applications) from the cluster queue shown in Fig. 1 to the servers in the cluster. The proposed implementation builds on the scheduling ideas from [5], where we introduced and described details of the function *D-ChoicesGreedyScheduling(H)* from Algorithm 1. Here, we modify those scheduling ideas by employing predictions of MIPS done with the developed training-efficient simple DNN model. Unlike traditional scheduling approachs, where only *one* optimal candidate server is picked, here, we pick the *top D* ($D \geq 2$) candidates and then randomly select one from among these D candidates [5].

Further discussion: Why D choices instead of the best one? The work in [5] mathematically proved that D-choices greedy scheduling method achieves significantly better load balancing over traditional best choice scheduling methods - which is the case of Kubernetes [13] scheduler. It was also shown that reduced load balancing translates into minimal applications queuing time, which in turns results into improved performance.

IV. EXPERIMENTS

A. Experimental Setup

Custom cluster: We implemented the cluster-level scheduling framework from Fig. 1, called Qin2, as a "plug-in" custom scheduler (will be released publicly) managed by the Kubernetes v1.14.0 on a real small cluster containing six heterogeneous computers. Table II lists the specific characteristics of these six computer-nodes. We use *acpi-cpufreq* to get the current frequency of the CPU on the real server, and the Linux *perf* tool v5.4.148 for fast profiling.

TABLE II
CHARACTERISTICS OF THE SIX COMPUTER NODES OF THE KUBERNETES CLUSTER.

Server Type	Role	GHz	Cores	L1 (KB)	Mem (GB)
Xeon E5-1620	master	3.60	8	32	16
Intel i5-6600	worker	3.30	4	32	16
Intel i7-4790	worker	3.60	8	32	16
Intel i5-7600	worker	3.50	4	32	8
Intel i5-4690	worker	3.50	4	32	8
Intel i5-4670	worker	3.40	4	32	8

Schedulers: We compare the implemented Qin2 scheduler against the Kubernetes default scheduler [13] (deployed on Amazon AWS, Google Cloud Platform, Microsoft Azure, and IBM Cloud, assumed to be the best scheduler from industry), and against the Qin scheduler [5] (tested on real cluster, assumed to be the best scheduler from academia). Table III lists the main characteristics of the cmpared schedulers.

TABLE III
SUMMARY OF THE COMPARED SCHEDULERS.

Scheduler	Method	Metrics
Kubernetes	Best Node	Multiple (resource, constraints, ...)
Qin	Collaborative Filtering and D-choices Greedy	Performance, Energy, EDP
Qin2	Simple DNN and D-choices Greedy	Performance

Workloads: We evaluate the proposed Qin2 scheduler using 100 *batch jobs* generated by Splash-2 benchmarks [14]. Modern datacenter workloads contain throughput-bound jobs. To represent modern datacenter workloads better, we use selected throughput-bound applications from Parsec 3.0 [15] (*blackscholes, bodytrack, facesim, ferret, fluidanimate, raytrace, swaptions, canneal*), randomly replicated with equal likelihood and randomized interleaving to generate 60 *throughput-bound jobs*.

B. Results

We first fast profile the first 20% of the incoming application workloads for sample-efficient feature selection. We do this under the assumption that the rest of upcoming applications will not have significant differences from the first 20% applications. If such an assumption does not hold, then, this percentage must be increased. The identified critical features are then used to train the proposed hand-tuned simple DNN with 4 layers and 29 neurons - which will be used to predict MIPS. We set the fast profiling time to be 0.05s for each application. We use the same DNN architecture for both 100 *batch jobs* and 60 *throughput-bound jobs* workloads, but with different weights due to fast re-training (less than 4s). We selected $d = 2$ for the D-choices greedy scheduling algorithm since it results in better load balancing.

Performance: We estimate performance as normalized jobs completion time. Fig. 4 compares the proposed Qin2 scheduler vs. Kubernetes and Qin schedulers for 60 *throughput-bound jobs* workloads on six-node heterogeneous cluster. The x-axis represents the number of workloads and the y-axis denotes the normalized jobs completion time. We observe that on average the proposed Qin2 scheduler outperforms Kubernetes (by 23.75%) and Qin (by 14.57%), respectively.

Server utilization: Fig. 5 shows the heat map of the server utilization (calculated as average CPU utilization and collected by *Metrics API*) with respect to time for Kubernetes and Qin2 schedulers for 60 *throughput-bound jobs* on five workers. Fig.5.a shows that the Kubernetes scheduler leads to a *long tail* phenomenon during the ending period, which is caused by the *best choice* scheduling approach that Kubernetes employs. In contrast, the Qin2 scheduler uses D-choices greedy scheduling method and avoids such a long tail phenomenon. However, we notice that there is still room for improvement in server utilization because the *throughput-bound jobs* contain some applications that have longer execution times (long jobs), and the short jobs scheduled to server ID 1 and 2 are completed

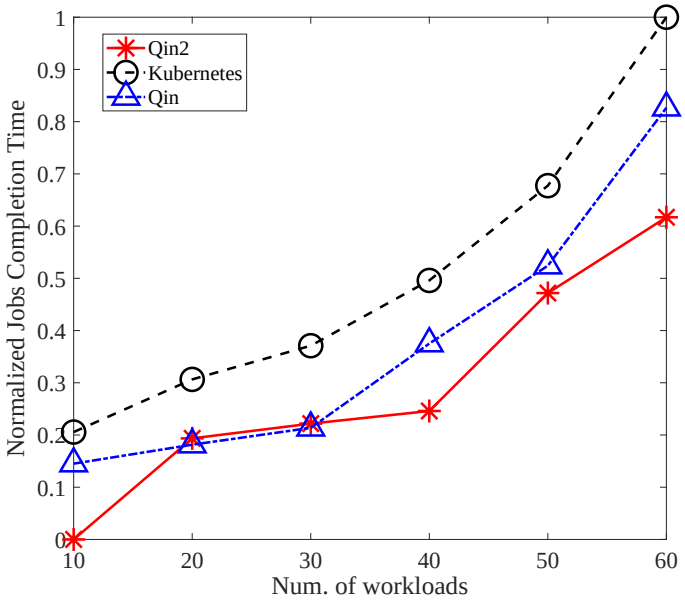

Fig. 4. Comparison in terms of jobs completion time of the proposed Qin2 scheduler against state-of-the-art schedulers.

faster than the long jobs scheduled to server ID 3, 4, 5, which results in spare utilization during the ending period of server ID 1 and 2. In our future work, we plan to consider short/long jobs in the scheduling process.

Further discussion: Why does the proposed Qin2 scheduler outperform state-of-the-art schedulers? The proposed Qin2 scheduler outperforms the Kubernetes scheduler because of the D-choices greedy scheduling method (mathematically proved to achieve significantly better load balancing [5])- which minimizes load imbalance and leads to reduced applications queuing time, and in turn to shorter jobs completion time. The proposed Qin2 scheduler outperforms the Qin scheduler because of the better prediction capability of the developed DNN model, which also requires shorter training and prediction overheads.

Summary: A summary of the comparison of the proposed Qin2 scheduler against state-of-the-art schedulers is listed in Table IV for 100 *batch jobs* workloads and for 60 *throughput-bound jobs* workloads.

TABLE IV
IMPROVEMENT ACHIEVED BY QIN2 SCHEDULER OVER KUBERNETES AND QIN.

Cluster-level Scheduler	Performance Improv. Batch jobs	Performance Improv. Throughput-bound jobs
Kubernetes	13.21%	23.75%
Qin	6.12%	14.57%

V. CONCLUSION

We developed a simple DNN model based scheduling approach for heterogeneous datacenters. The main contribution of this paper includes a technique to identify the most important input features to train the DNN model - which translate into smaller datasets for training and shorter training and re-training times - and the integration of simple and training-efficient DNN models into a scheduler deployed on a real cluster of heterogeneous nodes. Experimental results show that the proposed scheduler improves the jobs completion time by 6.12%-23.75% over state-of-the-art schedulers.

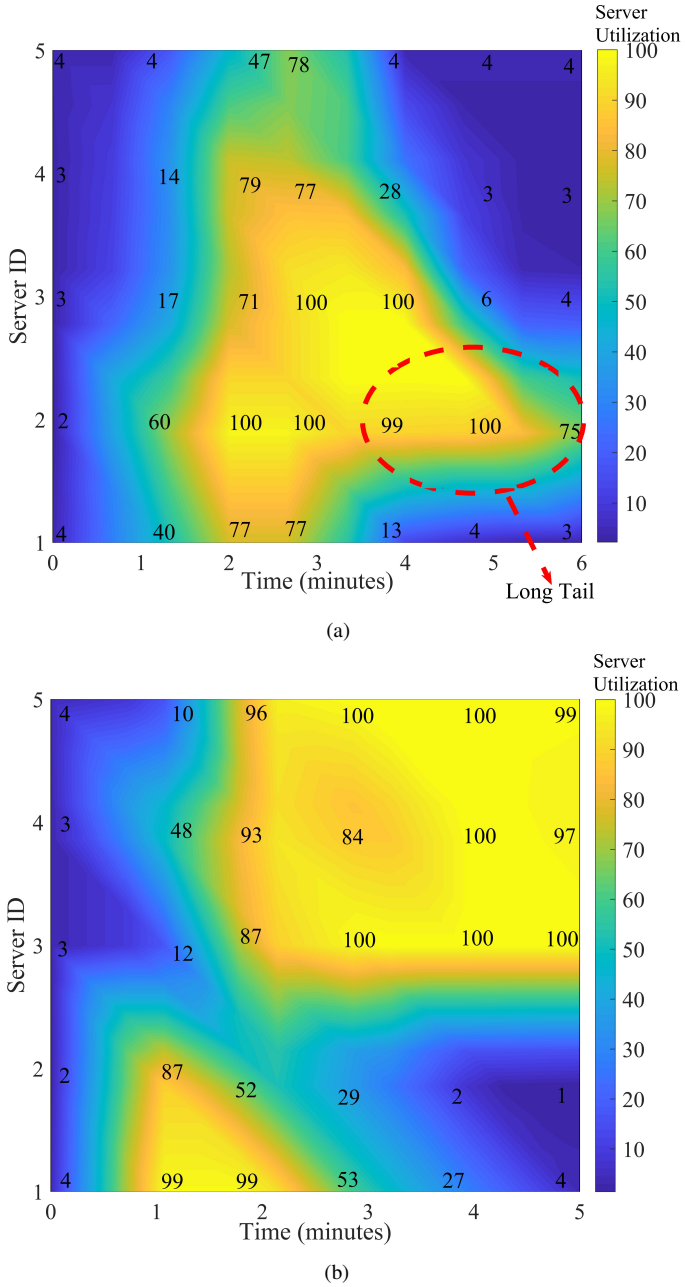

Fig. 5. Server utilization versus time: (a) Kubernetes scheduler and (b) Qin2 scheduler for 60 *throughput-bound jobs* workloads at the cluster-level.

REFERENCES

[1] K.H. Lee, O. Nachum, M. Yang, L. Lee, D. Freeman, W. Xu, S. Guadarrama, I. Fischer, E. Jang, H. Michalewki, and I. Mordatch, "Multi-Game Decision Transformers," *arXiv*, 2022.

[2] S. Reed, K. Zolna, E. Parisotto, S.G. Colmenarejo, A. Novikov, G.B. Maron, M. Gimenez, Y. Sulsky, J. Kay, J.T. Springenberg, T. Eccles, J. Bruce, A. Razavi, A. Edwards, N. Heess, Y. Chen, R. Hadsell, O. Vinyals, M. Bordbar, and N. Freitas, "A Generalist Agent," *arXiv*, 2022.

[3] S. Chen, A. Jin, C. Delimitrou, and J.F. Martinez, "ReTail: Opting for Learning Simplicity to Enable QoS-Aware Power Management in the Cloud," *Int. Symposium on High-Performance Computer Architecture (HPCA)*, 2022.

[4] F. Romero and C. Delimitrou, "Mage: Online and Interference-Aware Scheduling for Multi-Scale Heterogeneous Systems," *Int. Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2018.

[5] W. Guan and C. Ababei, "Two-level Hierarchical Cluster-Node Scheduling for Heterogeneous Datacenters," *IEEE/ACM Design Automation Conference (DAC)*, 2022 (WIP-Poster).

[6] Y. Ran and et al., "DeepEE: Joint Optimization of Job Scheduling and Cooling Control for Data Center Energy Efficiency Using Deep Reinforcement Learning," *Int. Conference on Distributed Computing Systems (ICDCS)*, 2019.

[7] M.G. Moghaddam, W. Guan, and C. Ababei, "Investigation of LSTM based prediction for dynamic energy management in chip multiprocessors," *Int. Green and Sustainable Computing Conference (IGSC)*, 2017.

[8] M.G. Moghaddam, W. Guan, and C. Ababei, "Dynamic Energy Optimization in Chip Multiprocessors Using Deep Neural Networks," *IEEE Trans. on Multi-Scale Computing Systems (TMSCS)*, vol. 4, no. 4, 2018.

[9] A. Krishnakumar, S.E. Arda, A.A. Goksoy, S.K. Mandal, U.Y. Ogras, A.L. Sartor, R. Marculescu, "Runtime task scheduling using imitation learning for heterogeneous many-core systems," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 39, no. 11, 2020.

[10] E. Ipek, O. Mutlu, J.F. Martinez, and R. Caruana, "Self-Optimzing Memory Controllers: A Reinforcement Learning Approach," *Int. Symposium on Computer Architecture (ISCA)*, 2008.

[11] M. Donini, L. Franceschi, M. Pontil, O. Majumder, P. Frasconi, "MARTHE: Scheduling the Learning Rate Via Online Hypergradients," *Int. Joint Conference on Artificial Intelligence (IJCAI)*, 2020.

[12] Coefficient of determination, 2022. [Online]. Available: https://en.wikipedia.org/wiki/Coefficient_of_determination

[13] Kubernetes, 2021. [Online]. Available: http://k8s.io

[14] S. C. Woo and et al., "The SPLASH-2 programs: characterization and methodological considerations," *ISCA*, 1995.

[15] PARSEC 3.0-Beta, 2015, [Online]. Available: http://parsec.cs.princeton.edu