Scheduling of Time-Varying Workloads Using Reinforcement Learning

Shanka Subhra Mondal1+, Nikhil Sheoran2+, Subrata Mitra2†
1Princeton University
2Adobe Research

smondal@princeton.edu, sheoran@adobe.com, subrata.mitra@adobe.com

Abstract

Resource usage of production workloads running on shared compute clusters often fluctuate significantly across time. While simultaneous spike in the resource usage between two workloads running on the same machine can create performance degradation, unused resources in a machine results in wastage and undesirable operational characteristics for a compute cluster. Prior works did not consider such temporal resource fluctuations or their alignment for scheduling decisions. Due to the variety of time-varying workloads and their complex resource usage characteristics, it is challenging to design well-defined heuristics for scheduling them optimally across different machines in a cluster. In this paper, we propose a Deep Reinforcement Learning (DRL) based approach to exploit various temporal resource usage patterns of time-varying workloads as well as a technique for creating equivalence classes among a large number of production workloads to improve scalability of our method. Validation with real production traces from Google and Alibaba show that our technique can significantly improve metrics for operational excellence (e.g., utilization, fragmentation, resource exhaustion etc.) for a cluster compared to the baselines.

Introduction

In large production clusters, multiple workloads are co-located on the same machine to achieve operational efficiency at scale. They share the underlying physical resources of the machine such as CPU, cache, memory, disk, network-bandwidth. Today, the standard practice is to deploy these workloads as microservices inside containers that are managed by various orchestration-engines (COEs) such as Mesos (Hindman et al. 2011) and Kubernetes (Burns et al. 2016). These engines either use standard bin-packing algorithms or custom heuristics (Ghodsi et al. 2011; Garefalakis et al. 2018; Verma et al. 2015) to place these workloads on the available machines in a multi-tenant cluster. However, a very significant fraction, if not all, of these services or applications show time-varying-workload (TVW) characteristics, i.e. their resource usage vary significantly over time (Tian, Zheng, and Wang 2019; Reiss et al. 2012; Mishra et al. 2010). This can happen because of algorithmic phases (Amvrosiadis et al. 2018; Reiss et al. 2012; Mitra et al. 2017; Zhang et al. 2007), variations in load or the number of users interacting with the workload. For user-facing services, such temporal resource usages can have daily and seasonal patterns due to fluctuations in user-demands (Amvrosiadis et al. 2018; Mishra et al. 2010) (e.g. some services are mostly used during working hours while some are used during holiday seasons).

In a multi-tenant cluster, services are developed and deployed by different engineering groups, where none, including the cluster manager, usually have a detailed understanding of the temporal resource-usage characteristics of each of these services. (Garefalakis et al. 2018) reported that today a substantial fraction of common clusters as well as dedicated cluster in production are devoted to only long-running workloads. When placing long-running workload containerizers, the cluster scheduler must also target operational excellence, such as improving overall cluster utilization, minimizing performance degradation due to resource contention (e.g., memory usage of two workloads on the same machine spikes at the same time), service level objective (SLO) violations due to resource over utilization or exhaustion, resource fragmentation and the number of machines used. Finding optimum initial placement for these workloads are crucial as later migration has a high overhead and causes degraded user-experience (Garefalakis et al. 2018).

Problem Statement: In this paper we explore how we can build a self-learning scheduler that can take historical resource-usage characteristics for each service1 as a time-series: \( r_d(t) \) and attempt to optimize various components of operational excellence in a cluster. Here \( r_d(t) \) is the resource usage along the measurable resource dimension \( d \) (CPU, Memory etc.) and \( t \) is a timestamp.

The absence of optimal co-location labels makes this an unsupervised problem. Since the impacts of placement decisions are not immediately observable, a cost-based heuristic would be sub-optimal. While Reinforcement Learning is a feasible solution, the large number of ways a variety of services can be co-located over different machines in the cluster, makes state-transition probability modeling infeasible.

1Equal Contribution
2Corresponding Author
Copyright © 2021, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.
Our proposed TVW-RL is a Deep Reinforcement Learning (DRL) based approach to make the cluster scheduler automatically learn temporal resource usage patterns for each service and dependencies across different services using deep neural network as function approximators. This learning helps the scheduler place the services in a shared cluster without requiring any manually provided placement heuristics in order to achieve operational excellence.

We make the following key contributions:

- We novelly use DRL approach to provide better placement for time-varying workloads (TVWs).
- We propose novel reward designs to improve operational excellence for a cluster handling long-running TVWs.
- We present resource-usage based equivalence mapping of workloads for robust learning and scalability.
- We present extensive evaluation using real workload traces from Google and Alibaba production clusters and show that our technique can improve resource utilization by 30-100%, reduce resource fragmentation by 5-13% and reduce the number of machines needed by 8-50% under variable load conditions.

Background and Related Work

A cluster administrator targets to improve the Operational Excellence on the following aspects.

Maximize Workload Performance: A scheduler should attempt to minimize the probability of performance degradation through contention in poorly isolated resources (last-level cache, memory bandwidth etc.). If the surge in the resource demand in two co-located services do not happen at the same time, it is unlikely that they would contend each other and hence their performance will be maximized.

Maximize Overall Cluster Utilization: Maximizing overall cluster utilization by efficient packing of workloads is important to reduce computing cost for any organization.

Minimize Resource Overshoot: When the number of workloads is large and/or load for the workloads is high, some machines might experience periods of resource overshoot and exhaustion (sum of resource requests from co-located applications shoots beyond the machine’s capacity). Placement algorithms should avoid such situations.

Minimize Resource Fragmentation: If amounts of unutilized resources remain spread across a large number of machines, those fragmented resources cannot be used to host new workloads leading to undesirable resource wastage.

Minimize Number of Machines used: If same workloads can effectively be run on a smaller number of machines, it can significantly reduce operating cost as idle machines can be stopped/hibernated. It is challenging to efficiently explore this trade-off space for a mix of TVWs as several of these aspects are naturally conflicting to each other.

Reinforcement Learning. In RL, at a high-level, an agent interacts with a system and tries to learn an optimized policy. At each timestep $t$, the agent observes the state of the system $s_t$, and chooses to take an action $a_t$ that changes the state to $s_{t+1}$ at timestep $t + 1$, and the agent receives a reward $r_t$. The goal of the agent is to learn the best policy to maximize its expected cumulative discounted reward: $E[\sum_{t=0}^{\infty} \gamma^t r_t]$ where $\gamma \in (0, 1)$ determines how much the future rewards contribute to the total reward (Sutton, Barto et al. 1998). In DRL, neural networks are used as agents to handle large state and action spaces (Lillicrap et al. 2015; Mnih et al. 2015). We used policy-gradient method with REINFORCE algorithm (Sutton et al. 2000) with the neural network as a function approximator.

Related works. Prior work related to scheduling and job packing in multi-tenant clusters looked at multi-dimension resource packing (Parkes et al. 2015; Joe-Wong et al. 2013; Ghodsi et al. 2011; Grandl et al. 2014), opportunistic scheduling for improving cluster utilization (Boutin et al. 2014; Verma et al. 2015; Schwarzkopf et al. 2013), performance-aware placement (Nathuji et al. 2010; Delimitrou and Kozyrakis 2013, 2014; Gog et al. 2016; Isard et al. 2009) or cost-models for template workloads (Marcus and Papaemmanouil 2016). None, handles time-varying resource usage aspects of the workloads. DeepRM (Mao et al. 2016) used DRL to schedule jobs that use constant amount of resources, on a single monolithic machine. It neither considers temporal variations in resource usage nor individual machine-specific views that is necessary to understand and optimize for job alignment to minimize resource interference and resource fragmentation. (Mitra et al. 2019) cannot handle real production workloads because of scalability challenges. Other works applied DRL in video streaming (Mao et al. 2017), routing (Mestres et al. 2017) and device-placement (Mirhoseini et al. 2017). Tetris (Grandl et al. 2014) uses heuristics to account multiple resource dimensions but does not consider the temporal resource usage changes during runtime. Tetris assumes complete knowledge of the resource requirements of tasks and resource availability at machines. Recently proposed Decima (Mao et al. 2019) used RL to learn optimized scheduling for DAG-structured analytic jobs and is complementary to our work as it does not handle TVWs.

Design of TVW-RL

TVW-RL represents the workload scheduler as an agent where the scheduling policy is encoded in a neural network. For an incoming scheduling request of a TVW, a trained policy network takes actions: where that service should be placed to optimize for the operational excellence.

We model the cluster environment as composed of $N$ machines on which the TVWs are to be scheduled. Each machine has $C_d$ amount of total physical resource capacity for resource dimension $d$ (e.g., CPU, memory). For a service $j$, the agent observes the resource usages’ time-series denoted as $r_d(t)$, where $r_d$ is the resource usage along the resource dimension $d$ and $t$ is a timestamp. Along with the current placement map of which services are running on which machines, the agent keeps track of the incoming scheduling requests in the queue. The complete workflow of the TVW-RL technique is shown in Fig. 1 and Algorithm 1 describes the high-level online learning and placement logic.
State Space Representation for RL

We design the state-space such that it can capture the temporal variations of resource usages as well as the degree of competition among co-located services sharing the same underlying resources within a machine.

Fig. 2 illustrates our input-space representation. State of each machine in the cluster is represented as a 2D matrix with $h \times C_d$ pixels for each of the resource dimension $d$, where $h$ is the number of previous logical timesteps. Within each machine, one dimension of the matrix represents the time axis and captures the utilization of TVWs for up to $h$ previous logical timesteps. Length of the history, i.e. $h$ helps the agent to learn the temporal characteristics of each TVW. The value of $h$ should be reasonably large w.r.t. the scheduling time-scale so that it helps the agent to capture a significant overlap among services. $h$ can be configured for different cluster deployments, depending on the typical periodicity of their corresponding workloads.

For each machine, the number of pixels in the horizontal direction ($C_d$) represents the quantized resource capacity of that machine for resource dimension $d$. We use quantization to map both resource usage by TVWs and machine capacities to integer units of resources to aid fixed state-space representation for DNN. $C_d$ can be calibrated to adjust trade-off between reducing resource wastage by reducing quantization error vs. increase in of state-space size.

The pixels in this matrix representing machine-states are labeled differently to denote different workloads. Empty label denotes unused resources as shown with different colored pixel labels in Fig. 2. Our state-space representation supports up to $G$ labels. To make TVW-RL scalable, we can map a large number of workloads to this $G$ set of equivalence-classes (EC).

Labels help TVW-RL to distinguish and learn resource usage characteristics between different equivalence-class of services. A waiting-queue captures the workloads waiting to be scheduled. TVW-RL combines the state of individual machines and the waiting-queue into a cluster-level state representation that is given as input to the policy-network.

Rewards Design for RL

We use negative rewards or penalty to teach the RL-agent with the following components:

**Resource contention penalty.** To help the agent learn a policy that avoids placement of TVWs whose resource usages are likely to spike at the same time, we use a penalty proportional to the contention-score, which is a timeseries inner product between two TVWs running on the same machine.

$$P_C = - \sum_d \sum_{m \in M} K_c \cdot Cr(m, d)$$

Contention-score ($Cr$) for a machine $m$, along resource dimension $d$ is calculated as:

$$Cr(m, d) = \sum_{W_j \in W_m} \sum_{j \neq i} \langle R(W_i, d), R(W_j, d) \rangle$$

where $W_m$ is the set of TVWs running in the machine $m$ in the increasing order of their starting time and $R(W_i, d)$ is a vector denoting the resource usage timeseries of workload $W_i$ along resource dimension $d$, throughout its life time on the machine. While taking the inner product of two resource usage vectors relative time shifting and appropriate padding with zeros are done to ensure each vector is of the same length and the overlap time is taken into account. $M$ is the total number of machines in the cluster. $K_c$ is a constant that determines the weight of resource contention penalty. The contention-score formula amplifies the effect of simultaneous spike (see Fig. 3) in resource usage among TVWs and thus would avoid resource contentions among co-located services. We observed that such penalty also helps the agent to learn better alignment w.r.t. resource usage (or rather complementary of resource usage) among different TVWs during placement – one of our target goals.

**Under-utilization penalty.** Since our goal is to improve overall utilization of the cluster by helping the scheduler learn how to achieve tighter packing and pack on less number of machines whenever possible, we add a penalty proportional to the sum of unused resources in the used machines. A used machine is one with at least one workload running. Empty pixels in our state-representation denote the number of unused resources at any given time.

$$P_U = - \sum_d \sum_{m \in M_u} |U_m(t, d)|^{K_u}$$

$U_m(t, d)$ denotes the unused resource for machine $m$ at time $t$ across resource dimension $d$. The constant term $K_u$ which also acts as a weighting factor, is an exponent here so that the error gradient dimension has a direct component denoting amount of unused resources. $M_u$ denotes the set of used machines.
We design high CPU time-slices, memory thrashing etc. leading to undesir-
ables. But the
Resource contention
and
The agent receives the
Wait-time penalty.
Ru
Dynamic Time Warping (DTW).

**Workload Equivalence-Class (EC) Mapping**
Real production clusters need to handle a large number of
different workloads. It is not scalable to treat each TVW
differently. Too many distinct TVW-labels would make the
task more difficult, and less generalizable. We explore how
these numerous real production traces of TVWs
can be mapped to a much smaller number of equivalence-
classes (EC). Then we can assign a unique type-label to each
EC (rather to each individual TVW) helping the agent in its
policy learning. Note, the TVWs belonging to the same EC
will have **similar shape** in terms of temporal resource usage
characteristics and thus can create higher contention (if placed
together), as compared to TVWs in different ECs. We consider
two different **distance-metrics** to capture the
temporal aspects described as follows. **Temporal Features.**
We extract temporal features such as auto-correlation, linear
trend, agg. linear trend as well as aggregate features like mean,
mean absolute change and std. of the input time-series. These features capture the aspects of shape, trend and
diurnal patterns. We normalize the obtained values of each
feature to [0,1] allowing for comparability between different
features. On the obtained feature vector, we define euclidean
as the distance metric.

**Overshoot penalty.** We teach the agent to prevent scheduling
of more tasks than that can be handled by a single
machine, as this would lead to TVWs not getting enough
CPU time-slices, memory thrashing etc. leading to undesir-
able performance or even SLA violations. We design high
penalty to the cases where any of the machines were not able to meet the aggregate resource requirement of TVWs
scheduled in that machine, during any period of time. It is
calculated by adding a high value \(K_o\) each time a machine
is unable to provide appropriate resources to the running
services for the first time. An indicator function \(I\) keeps track
of whether a TVW running in machine \(m\) overshooted.

\[
P_O = - \sum_d \sum_{m \in M} K_o \cdot I_{m,d} \text{[First overshoot for the TVW]}
\]

**Wait-time penalty.** To prevent the scheduler from holding
scheduling requests for a long time in search of a better
placement, in each timestep we add a penalty that is equal
to the number of waiting requests in the queue \(|Q_t|\) multiplied by a constant \(K_w\).

\[
P_W = -K_w \cdot |Q_t|
\]

The calibration of weight \(K_w\) in the penalty function also
helps in teaching the scheduler to switch between highly op-
timum placement mode (when less incoming TVW scheduling
requests are expected) to a less optimum placement
mode (when burst of scheduling requests are expected to hit
the cluster). TVW-RL can have different penalty coefficients
for different resource dimensions.

The agent receives the **Wait-time** and **Under-utilization**
instantly. But the **Resource contention** and **Overshoot** penal-
ties are delayed because resource contention can only be de-
tected at a later time, and hence the actions that caused it
should be penalized in future, but scheduling delay (Wait-
time) can be detected immediately.

**Temporal Features.**
Temporal features such as auto-correlation, linear

diurnal patterns. We normalize the obtained values of each
feature to [0,1] allowing for comparability between different
features. On the obtained feature vector, we define euclidean
as the distance metric.

**Dynamic Time Warping (DTW).** DTW (Ding et al. 2008)
gives a shape specific distance measure between two time-
series adjusting for any time shifts. We compute the distance
metric by averaging the normalized DTW distances
during different resource dimensions. We use K-Means (K-
Medoids for DTW) clustering (MacQueen 1967) on the ob-
tained distance metrics. We perform selection of suitable
value of \(k\), by plotting the Silhouette Score (Rousseeuw 1987) for different values of \(k\) and selecting \(k\) correspond-
ing to the maximum silhouette score. Characteristics of pro-
duction cluster workloads vary across different organiza-
tions (Amvrosiadis et al. 2018). Potentially the optimum
distance-metric and associated number of ECs can be dif-
f erent for different organizations. Training a policy-network
using RL is a computationally expensive task. Hence, it is
not practical to evaluate the goodness of the EC mapping by
training the RL agent for all different choices. We propose a
lower overhead proxy for evaluating the distance-metric and
goodness of EC creation by training a resource usage predic-
tion model (a multi-variate time-series prediction model (Lai
et al. 2018)) per EC. We choose most suitable combination
of distance-metric and EC numbers by comparing the
prediction accuracy of this model. A reasonably good time-
series prediction model would also get similar benefit during
RL training from the chosen best combination.

**Run-time Logic.** Line #5-8 in Algorithm 1 describes the steps for deciding the placement for an incoming workload.
Before probing the policy-network for the optimum ma-
chine (line #10), the state-representation is updated by re-
predicting the EC-labels for the already running workloads
(line #8). EC prediction is a multi-class classification prob-
lem. TVW-RL uses a fully connected network to classify it
to one of the ECs. It takes two inputs (1) featurized represen-
tation of meta-information (requested CPU/memory limits,
 scheduling priority, etc.) and (2) a vectorized representation
of the partial resource usage.
Evaluations

Real Production Workload Trails

Google traces (Wilkes 2011) contain production workload scheduling requests for a period of 29 days. Alibaba traces (Alibaba 2018) contain production traces from 4k machines over 8 days. Both contains CPU/memory numbers used by each workload at a granularity of 5 minutes, along with scheduling details, e.g., priority, class and original resource request. We filtered the workload traces by discarding the ones that were evicted, killed or failed. We also removed the partially recorded traces. Since our focus is on optimal placement of long running jobs, we filtered-in long-running jobs that had at least 6000 continuous timestamped resource usage records for Google traces and, at least 2000 timestamped resource usage records for Alibaba traces.

Implementation

EC Creation. For temporal features, we used tsfresh (Christ et al. 2016) for feature extraction and K-Means clustering algorithm (Pedregosa et al. 2011) with ‘k-means++’ initialization for EC creation. For DTW, we used K-Medoid algorithm (Kaufmann et al. 1987) with random medoid initialization. The value of k ranged from 3 to 15 and k corresponding to maximum average silhouette over 50 different initializations was selected.

EC Prediction. This model has two dense layers with ReLU activation and output softmax layer. It was trained with Adam optimizer and a categorical cross-entropy loss.

Policy Network. The policy network is implemented using Theano. It consists of a single hidden layer of 20 neurons followed by output neurons equal to the number of actions (= number of machines in the cluster) and ReLU activation function for the hidden-layer. For the output layer we use softmax activation. We use Adam optimizer and a learning rate (η) of 0.001. We train using REINFORCE algorithm (Sutton et al. 2000) with the number of trajectories (N) set to 20 and in an episodic manner (Mnih et al. 2013) for a total of 2000 iterations, with maximum episode length (L) 200. In a given episode, a fixed number of jobs arrive and are scheduled by the agent. The parameters for the state-space are M = 10, h = 20, d = 2, C1 = C2 = 8. The weights of the penalty parameters are chosen as K_c = 0.1, K_u = 3, K_o = 30000, K_w = 50. Training and testing used a batch size of 20 examples run in parallel on a 32 core Intel Xeon CPU E5-2686 v4.

<table>
<thead>
<tr>
<th>Distance</th>
<th>Google Traces</th>
<th>Alibaba Traces</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>𝐾</td>
<td>𝐽</td>
</tr>
<tr>
<td>Aggregate</td>
<td>5</td>
<td>0.65</td>
</tr>
<tr>
<td>Temporal</td>
<td>6</td>
<td>0.58</td>
</tr>
<tr>
<td>DTW</td>
<td>3</td>
<td>0.55</td>
</tr>
</tbody>
</table>

Table 1: Different distance methods for Usage Prediction

Equivalence-class (EC) Analysis

For EC analysis, we compare the proposed Temporal Features and DTW against an aggregate-features based baseline inspired by (Zhang et al. 2011). The baseline is defined as the euclidean distance on the job characteristic vector of total and avg. CPU, total and avg. memory, job duration and avg., std. and normalized std. of the memory-CPU ratio. Table 1 shows the optimum k that maximizes the Silhouette score for the distance-metric chosen. Optimum k can be different for other production environment traces.

Further, we evaluate the effect of different distance-metrics and k on the goodness of EC mapping by training and evaluating a resource-usage prediction model for each EC. We compute a measure called effective prediction gain (EG):$

EG = \frac{1}{n} \sum_{e \in EC} \max(0, G(e)) \ast |e|

$where n is the total number of jobs, e is an equivalence class from the set of equivalence classes (ECs) and G(e) is the percentage gain in the correlation between the predicted and actual resource usages of a model trained on EC e with respect to an EC-agnostic model that takes all the examples into account irrespective of the EC. Table 1 shows the fraction of the data (f) and the effective prediction gain for both Google and Alibaba data. We see that distance-metrics incorporating time-varying features (Temporal Features and DTW) perform better than the aggregate features baseline. Thus, the temporal features based method efficiently captures the temporal aspects of resource usage making it the suitable choice for EC mapping for our RL-agent.

End-to-End Evaluation

Methodology. We simulate incoming workload placement requests to the cluster as a Poisson process. The interarrival-rate is calibrated to create three average cluster load scenarios: 30%, 50% and 80%. Our evaluation runs with a cluster setting of 10 machines. Each workload has 2 resource dimensions: CPU and memory. From both Google and Alibaba traces, we select a mix of production workloads with 50% of as long-running (>30 hrs), with approximately equal number of them being CPU intensive and memory intensive. For each trace, the training and test set consists of 100 and 30 such distinct job sequences, respectively. The resource utilization values from the traces were mapped to our state-space dimensions. The offline training time ranged from approximately 28-108 hrs depending on the load, state space dimensions and the number of iterations.

Baselines. We compare TVW-RL with Tetris (Grandl et al. 2014), DeepRM (Mao et al. 2016), and Best-Fit heuristic (Berkey and Wang 1987). Tetris has a setting for controlling fairness in scheduling. Since our solution does not use a notion of fairness, we turn off the fairness knob which improves job performance in Tetris. To have a fair comparison, we made the following modifications to DeepRM: (1) Inputs: As DeepRM does not support temporal variations in resource usage, we provide the peak resource used by the job along CPU and memory resource dimensions as the resource-limit. (2) State Space: As required by DeepRM, we calculate the capacity of the compute cluster as monolithic resource capacity by multiplying the capacity of each machine by the number of machines. (3) Rewards: To help optimize for cluster utilization, we added an additional penalty.
proportional to the number of total unused resources. We feed DeepRM’s action (“which job to schedule”) at a particular timestamp, through a Best-Fit heuristic to identify a machine for placement. Among the machines that have enough resources to host the job, the Best-Fit heuristic chooses the machine having the least units of the dominant resource of the job available. We also compare against vanilla Best Fit heuristic that selects the jobs in a FIFO (first-in-first-out) manner and places in the machine with the least units of the dominant resource of the job available at the current-time. We use DeepRM to denote {DeepRM+Best-Fit} combination and Best-Fit to denote vanilla Best-Fit.

**Metrics for Operational Excellence.** We use the following metrics (Garefalakis et al. 2018; Grandl et al. 2014) to capture improvements in operational excellence of the cluster. T denotes the length of the observation period until all jobs have finished running.

### [Resource Utilization]
Average utilization of the cluster along each resource type:

\[
\text{Avg Util}(d) = \frac{\sum_{t=0}^{T} \sum_{m} R(m, t, d)}{T \times \max A(t) \times C_d}
\]

Since the number of machines that are actively used varies over time (A(t)), maximum # of machines used at any point, shows up in the denominator as a normalizer. \( R(m, t, d) \) denotes the resource usage of machine \( m \) at time \( t \) across dimension \( d \). Higher utilization is better.

### [Resource Fragmentation]
It measures what fraction of all the unused resources in a cluster are concentrated.

\[
\text{Avg Frag}(d) = 1 - \frac{\sum_{t=0}^{T} \max_{m \in A(t)} U_m(t, d)}{T \times \sum_{m \in A(t)} U_m(t, d)}
\]

The lower the fragmentation, higher the ability of the cluster to schedule unanticipated large jobs, which is desirable.

### [Resource Overshoot %]
It measures the total amount of resource shortage in each machine across time \( t \) across dimension \( d \), denoted as \( S(m, t, d) \), over the total resource capacity of the cluster.

\[
\text{Avg Overshoot} \% = 100 \times \frac{\sum_{t=0}^{T} \sum_{m} \sum_{d} S(m, t, d)}{T \times m \times C_d}
\]

### [# of Machine Used]
Total number of machines where at least one TVW was placed at some point of time.

**Improvements in Operational Excellence.** Fig. 4 and 5 illustrate using Google and Alibaba traces respectively, how TVW-RL can improve the metrics for operational excellence. TVW-RL provides a 30-300\% increase in average CPU and memory utilization compared to DeepRM. The benefit is more apparent under low-load conditions as shown in Fig. 4a/5a (and Fig. 4b/5b). We also achieved 68-100\% increase in average utilization compared to Tetris across different cluster-load conditions. This is primarily due to efficient packing that requires significantly less number of machines than Tetris as shown in Fig. 4e/5e. Further, the gap between TVW-RL and Tetris in terms of the number of machines required to accommodate the jobs increases with the increase in cluster load. Notice, Best-Fit provides higher utilization because it just packs the jobs into the machines without any knowledge of peak or future resource usages. As a consequence, Best-Fit suffers from huge over-utilization of the resources as shown in Fig. 4f/5f. On the other hand, over-utilization due to TVW-RL’s placement decisions are almost negligible. Tetris and DeepRM already include peak resource usage information in their placement decisions and thus avoid over-utilization. We noticed DeepRM’s inability to schedule all the jobs in the available machines at high cluster loads because of excessive resource fragmentation (Fig. 4c/4d and 5c/5d). TVW-RL provides 5-50\% reduction in resource fragmentation compared to DeepRM and 6-13\% reduction compared to Tetris on Google and Alibaba data.

**Robustness Against Noisy Environment.** There can be some unknown background process activities (e.g. maintenance process, system noise (Hoefler et al. 2010)) or some TVWs can be noisy. We evaluate TVW-RL’s robustness.
against such noisy environments using Google traces, keeping the cluster under 50% load. First, we compare TVW-RL’s scheduling capability in the presence of a random workload (to simulate random background processes) under two situations: (1) TVW-RL is trained in the presence of another random workload vs. (2) trained without any random workload (we call not-trained). The resource values for the random workload are uniformly chosen between [0-1], [0-2], [0-3] units. Recall, capacity of each machine is 8 units. Table 2 shows the comparisons of these two situations. The numbers are averaged over 5 runs (seeds). For [0-1] and [0-2] units of noise, we observe that both the trained and the not-trained versions are similar in most metrics, in fact the not-trained version does slightly better considering the fact that the % overshoot is less. Our hypothesis is that inherent noise introduced by the initial EC creation workflow made TVW-RL’s learning robust against some degree of noise. However, as shown in Table 2, not-trained TVW-RL could not anticipate the larger amount of noise (in case of [0-3]) while the trained version was able to spread scheduling over more number of machines. Hence, the percentage of overshoot for the not-trained version is 240% larger than the trained version. This suggests, TVW-RL needs to be trained over more number of machines. Hence, the percentage of noise. However, as shown in Table 2, not-trained TVW-RL could not anticipate the larger amount of noise (in case of [0-3]) while the trained version was able to spread scheduling over more number of machines. Hence, the percentage of overshoot for the not-trained version is 240% larger than the trained version. This suggests, TVW-RL needs to be trained over more number of machines.

We varied $K_u$ from 0, 0.1, 1.0, 10.0. We found TVW-RL can bring down overshoot level to 2.20, 0.20, 0.15, 0.09% respectively. This is achieved by progressively increasing the $\theta$ of machines used from 2 to 6.2, on average. For $K_u = 0$, i.e. without the penalty, overall (CPU,memory) utilization metrics are very high (0.455,0.752) and %overshoot is also high as the scheduler packs more aggressively.

**Conclusion**

We propose a deep reinforcement learning based approach for scheduling time-varying workloads in a shared cluster for optimal alignment of workloads based on their temporal resource-usage characteristics. Our evaluations with two real production cluster traces show that our novel state-space design along with reward formulation help our approach achieve significant improvement in operational metrics for shared clusters, compared to the state-of-the-art baselines.


