Reinforcing Cloud Environments via Index Policy for Bursty Workloads

Venkatraman Balasubramanian*†, Moayad Aloqaily‡, Olufogorehan Tunde-Onadele†, Zhengyu Yang†, and Martin Reisslein*

*Arizona State University, Tempe, AZ, USA
‡Al Ain University, UAE and xAnalytics Inc., Ottawa, ON, Canada
†Samsung Semiconductor Inc., Memory Solutions Lab, San Diego, CA, USA
Emails: †\{v.balas, fogo.onadele, yang.zhe\}@samsung.com, ‡maloqaily@ieee.org, *mreiss@asu.edu

Abstract—**In recent years, the amounts of network traffic targeted towards cloud data centers have fluctuated based on user requests. This traffic is bursty and requires a high degree of attention. Due to the variable nature of this traffic, some requests need to be re-allocated on-the-fly. Such circumstances result in performance degradations due to resource management. As appropriate solutions can be proposed only based on understanding the workload and the environment, Reinforcement Learning (RL) is a strategy that is predominantly used. Further, it has been shown that the *Poisson arrival rates* do not capture real-world burstiness. Thus, we mainly have a two-fold problem to address: (i) the traffic requires a new modelling approach that can characterize the burstiness, and (ii) balancing the load that can maximize the reward to the provider in such circumstances. In this paper, we propose a novel, yet simple traffic modelling that enables burst detection based on an index policy. We show that the throughput constraints play a crucial role in scheduling and our proposed RL technique produces reliable results in such a scenario. Our RL algorithm decides what instance of the request traffic needs to be processed so that the cloud provider can maximize its profit and the decisions made in hindsight are non-biased. We compare the proposed policy with two state-of-the-art approaches and draw key inferences as to why an index policy performs better in scenarios that demand RL. We observe over five times shorter average wait times while bursty workload crosses a saturation limit of 150% compared to conventional policies.**

Index Terms—**Bursty workload, Index policy, Mobile Cloud, Reinforcement learning (RL).**

I. Introduction

The arrival rates in traditional cloud data centres have become explosive in the past decade. In the US alone, the data surge has resulted in creation of a "hyperscale" market as seen in [1][2]. Different user requests ranging from multimedia and gaming applications to web apps have shown a drastic need to manage the Quality of Service (QoS) in such environments so as to achieve high throughput as well as low-latency service [3], [4]. Typically, when the network load goes beyond the usual threshold, a barometer is needed to manage the change. The sustainability and management of cloud data centers infrastructure is a critical challenge [5].

The data arrivals at the cloud location are connected to scheduling delays that go by the rule that the higher the variability in arrivals, the longer are the scheduling delays. This means that if resources are committed, changing schedules will result in unsatisfied service level agreements (SLAs). In cloud environments, balancing the requests from customers is the first step towards resource allocation. The next step is application specific, if one particular application is in higher demand, computational resources needs to be reassigned to that particular application instance. These requests are often incoming in short periods that are responsible for the spikes with higher arrival rates. Hence, we need a decision making strategy to address this challenge.

To this end, RL advancements, such as [6], make online decision while understanding the environment. For instance, some applications that have different execution requirements, such as serially loadable or parallely executable, frequently create changing resource needs. Such use cases demand novel mechanisms to maintain client satisfaction and QoS provisioning. Typically, ON/OFF source aggregation methods have been designed to tune to specific traffic classes. These methods have too many parameters to compute that makes traffic modelling tedious. Additionally, these models do not give importance to throughput computation. Thus, we propose a novel yet simple model that takes assistance from the RL paradigm. We design an agent that takes inspiration from the restless bandit index policy [7]. In this model, we show how throughput constraints are key towards addressing the scheduling problem. We show how this index enables traffic characterization. Lastly, we show how this modelling can benefit the network operator in maximizing their rewards by balancing load in a cloud environment.

A. Problem and Contributions

The changing bursty workloads degrade QoS and affect service provider profits. The RL paradigm specifically deals with such challenges that makes experience based decisions. This means that the agent is at first unaware about the task, but learns with time whereby a reward is specified based on how well the task is accomplished. In order to do so, the job arrival and completion rate of the workloads need to be optimistically considered without dropping requests. To this end, this paper contributes the following:

- A burst indication parameter based on an index policy calculation is proposed. The index policy calculation is derived from the restless multiarm bandit problem that allows us to predict the burst value which gets assigned to a cluster.

- An RL architecture is proposed that calculates operator profits based on successful service provisioning in bursty environments. In this model, we employ the standard *Q-learning* based strategy for learning.
- Our simulation results show that there is over five times reduction in average (normalized) waiting under throughput constraints with over 150% bursty load saturating the link.

The remainder of this paper elabprates the state-of-the-art approaches in Section II followed by the model of our design in Section III and its flow of control Section IV. Finally, in Section V, we take some measurements and discuss the proof of concept experiments, followed by concluding remarks in Section VI.

II. Related Work

Bursty workload management is a well-researched topic. A major limiting factor in most studies is the choice of Poisson or uniform arrivals that generally do not appropriately reflect real-world scenarios. However, RL based techniques have grown to capture and enhance the performance of variety of such dynamic decision-making systems. Importantly, resource allocation is a challenge that can predominantly be satisfied by RL.

The authors in [8] have evaluated an incoming request (bursty) process that is predictably following a uniform distribution. In contrast, our study is more realistic as the incoming jobs are arbitrary. Noticeably, in [9], the packet routing was observed where the size of the network was not dynamically changing. Furthermore, new learning techniques have been designed for alleviating network congestion, such as [10], albeit without considering any reward based issues for the cloud operator. Additionally, various scheduling based algorithms have been designed in [11] and [12] that are specific to clusters. However, neither of these studies follow a learning design that makes online decisions.

Mao et al. in [13] have proposed a deepRM model that performs comparably to our study, however, Mao et al. do not consider a cloud data-center environment that is focus of our framework. The key difference is that, a data-center cluster works with a traffic model that is highly varying based on changing user request, unlike the neural network based complex strategy that Mao et al. have proposed. In [6], Wang et al. propose a prediction mechanism that is trained in a greedy layer-wise fashion used for learning generic internet traffic flow features. However, [6] does not provide any strategy for resource allocation based on an online decision making mechanism. Likewise, in [14], Jacobs et al. propose an affinity based neural network which provides a plan for deployment of network functions at the infrastructure provider. On the contrary, our study is rooted in realistic data center environments with changing workloads that applies RL in cloud clusters. In summary, our study is one of the first research explorations to define a new network traffic model for a high reward/high profit based cloud environments.

III. The Proposed Model

The key advantage of the proposed model is that burstiness is described based on an indexable parameter. This problem can be transformed into a restless multi-arm bandit problem [7], where every node in the cloud environment behaves as a restless bandit. Essentially this parameter allows learning traffic behaviors and provides feedback to the operator on-the-fly. More specifically, the model fitting and trace generation is efficient as it scales linearly with the size of the data.

A. Entropy and Hurst Parameter

Entropy is defined in [15] as the "uniformity of a discrete probability function P". The entropy value $E(P)$ for an event E_i with probability p_i is given as

$$E(P) = \sum_{i=1}^{n} p_i \log_2 1/p_i. \tag{1}$$

When all the probability values p_i are the same, then the entropy reaches its maximum value. On the other hand, when one event dominates, then the entropy approaches zero. Thus, the entropy characterizes the burstiness.

A global value of burstiness per se is judged by the Hurst parameter [16]. It is actually a notion of self-similarity. An important point here to notice is that self-similar processes do not always generate bursty sequences. Further, the hurst parameter pertains to usage over long time scales. Hence, we draw inspiration from a statistical index in this paper as elaborated below.

B. Indexability of Traffic and Reinforcement Learning

In [7], an optimal strategy for the multi-arm bandit problem was proposed called the Gittins Index. Eventually, a number of similar studies were conducted, see e.g., [17]. In our study, we use the index as a parameter that demarcates requests within a time interval. For example, an index of 0.7 indicates that 70% of the requests arrive in one half of the time interval and the remaining 30% in the other half. During this arrival process, two types of actions can be taken, one is the overall optimal strategy for pre-empting traffic, the other is the computation of the index. In line with this concept, we compute the optimal strategy of allocating compute resources based on how the traffic construction grows over time. For instance, the initial construction begins with two time divisions and gradually recurses over the number of requests generated on each half of the time axis according to the index computation. Our intention here is not to show how fast this index can be computed, but to explore the insights gained by using the index as an input to learn the traffic characteristics. As the value of the index approaches one, the traffic irregularity increases; while for uniform traffic, the index values are 0.5.

In our training model, we enumerate a state space that maintains the resource profiles and requests as tabular entries. The resource profiles are data-centre computation and memory images and the requests are the jobs waiting to be scheduled. This is fed as input to our simulation environment. Our action space is very large as it grows exponentially based on the admitted jobs. The action space is defined as a method where the scheduler sends out a message, such as *schedule request r in slot k*. Further, a valid action is considered to satisfy a *BestQos* value or falls in the range from *LeastQos* to *BestQos*; an invalid action is below this range. As each valid action is chosen, the cluster entries in the resource table move up by one slot k and new jobs can be viewed by the agent.

IV. Policy Design

We observed in [7] and [18] that the usage of Gittins indices is only defined for indexable problems. However, as our primary constraints include the throughput of the links, our problem is not easily indexable. Thus, we need to relax the throughput constraints. As we intend to manage workloads based on the system burstiness windows (K), the throughput within a time slot acts as a constraint. Primarily, an incoming job can be satisfied in a number of available computing sites. These computing sites are the nodes selected while forming a composition cluster given as input for our training algorithm (from the tabular entries mentioned above). Although our algorithm finds the best available computing sites, the efficiency of our algorithm resides in detecting burstiness and maximizing the throughput. For example, consider a job from a user goes to location N_1 at a time t_1, as this job distribution is completely arbitrary over time, we do not know which job to schedule first that will maximize the reward to the provider. This problem can be reduced to a typical multiarm bandit problem [7]. The difference in our case is that the incoming jobs are part of a bandit process that evolves in an uncontrolled manner. In other words, the nodes behave as restless bandits. To this end, we design our system in a state space X_t such that a higher index means that a particular workload corresponds to the bursty traffic workload.

Consider playing a bandit process in state α. Clearly, by continuing to play this process, it would prove optimal for instances in states represented by the continuation set $C(\alpha)$ as for any $b \in C(\alpha), \lambda(b) > \lambda(\alpha)$ [19]. Thus, the stopping time that maps to the beginning of the bandit process in state α is the time $T(S(\alpha))$ of set $S(\alpha)$, that is, the beginning of the process $S(\alpha)$.

We define ζ as the optimal stopping time set. Notice that the index of a bandit process depends only on that process, so we can drop other labels and use the bandit process representation by a set. Accordingly, we label the maximum index value as ζ. To find the highest index value, the $argmax$ of ζ should be computed. The key idea is to order the set ζ in decreasing order.

Based on this reasoning, we deduce: if we have a stopping time for a traffic given as $T(S(\alpha))$, where $S(\alpha), S(\alpha) \subset \zeta$, represents the bursty workload at instance α, then we can define the index with the symbols definitions in Table I as

$$\lambda(\alpha) = \max \frac{E[\sum_{t=0}^{T(S(\alpha))} \beta^t r^t(X_t)|X_0 = \alpha]}{E[\sum_{t=0}^{T(S(\alpha))} \beta^t |X_0 = \alpha]}. \quad (2)$$

TABLE I: Symbols and Notations

Symbols	Definition
β^t	Beta distribution at time t
$Q(s,a)$	Learning parameter with state s and action a
d^k,b^k	$m \times 1$ vectors
$Q_{a,b}$	Recursion step, where a, b are intermediate steps
r^t	Reward parameter at time t
o	Represents a Bandit process o
$C(a)$	Continuation set
$S(a)$	Stopping Set

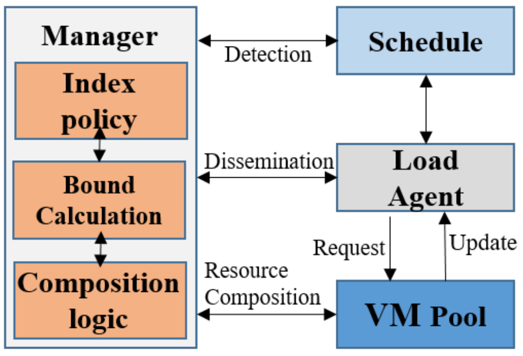

Fig. 1: The Proposed Framework

Equation (2) shows the index computation for offline scenarios. The rationale behind using the index for traffic modelling is that the probability state x_i in state space X_t can be substituted with p_i in the entropy equation. Due to space limitations we can only summarize this formulation.

A. *Functional Interaction*

As shown in Figure 1, the framework has two main phases. Algorithm 1 represents the algorithm which is explained below:

1) Phase 1: **Detection**
The Load Agent lies at the center of the system, and supplies information about the cluster resources along with the requirements for a particular request, closely connected resource nodes and node/VM pool which we have modeled as a resource allocation table. This information is used by the Bound Calculation entity so that the system can compute the measure of the actual workload involved and not just use the number of jobs enqueued to figure out the spike degree (burstiness). The Manager labels a time window as a Strong, Medium or Weak bursty period which it communicates to the rest of the system. Although, this difference in the type of burst does not affect the overall scheme, it is important to note that the change in bursty load affects the average wait times. Typically, a weighted first come first serve policy produces a higher wait time as observed in Section V.
2) Phase 2: **Load Balancing**
The spike detection result is supplied to the Load Agent and used to decide the appropriate learning strategy. A low degree of bursty job arrivals allows room for more accurate decisions. This is the initial training period in

our model. During intermediate traffic periods, a join the shortest queue strategy is used, using information about the actual workload state of the queues. Finally, in the presence of a strong degree of incoming traffic, the status information could likely be delayed. Thus, a round robin strategy is chosen. In all the cases the throughput constraints are maintained such that $T_q^j \leq T_q^\pi$ where $j \in$ Set of load and q is the minimum throughput, π is the max set of achievable throughput.

B. Algorithm 1(a)- Detection

Algorithm 1 begins with an index assignment in Line 2, that is based on the burst index definition. The two sets, namely the continuation set and the stopping set, are defined based on the chosen time slots. The Markovian values a, b are the action updates each time the resources are reallocated for the jobs which enter the cluster. Lines 3 to 13 execute the following steps:

1) The Best QoS values are chosen, these are the ones that are defined primarily at the time of execution (when the system is initialized), this can be considered as the initial SLA agreement with the user.
2) As the burst traffic would require a new allocation, inevitably, the QoS falls. However, based on the first learning instance, this is fed back into the system which in turn generates the lowest value of QoS for that instance. We use the throughput constraints to judge the next best/least QoS values.
3) Based on these inputs, the load agent calculates the confidence bounds B_j. We assign the new incremental resource value "I", which eventually becomes the new resource assignment for the next burst cycle. Each iteration gives a new resource allocation which would in turn satisfy the generated burst; however, as the number of iterations increases, the produced rewards saturate.
4) The reward signals are those which provide better output for the objective. For instance, lower the waiting/slowdown of the requests, higher would be the rewards.

Specifically, this leads to the following steps:

1) In the initialization phase, we find the highest value of index λ. The stopping phase for a state α can be regarded as $S(\alpha)$, if α_1 has the highest Burst Index Policy (BIP) metric, which we also refer to as Burst Index (BI) for brevity, then we have $\lambda = S(\alpha_1)$.
2) In this step, iterating over the α_1 state such that if $C(\alpha_k) = \alpha_1, \ldots, \alpha_{k-1}$ represents the next k^{th} largest BIP. We have, $Q''_{a,b} = P_{a,b}$, if $b \in C(\alpha_k)$, and $Q''_{a,b} = 0$ otherwise, whereby $P_{a,b}$ represents the updated Markovian values using an $m \times 1$ matrix. Further, I is the $m \times m$ identity matrix. Then, $d^{(k)} = [I - \beta Q''^{(k)}]^{-1}\zeta$ (alternatively, considering that α_1 has the highest Gittins index value, then $S(\alpha_1) = \zeta$ and we can replace ζ by λ) and $b^{(k)} = [I - \beta Q''^{(k)}]^{-1}1$, giving the BIP value

$$BIP = \frac{d^k_{\alpha_k}}{b^k_{\alpha_k}}. \tag{3}$$

3) For each burst we recognize what is the best completion rate and decide on a time T, which is the stopping time. Until T is reached, the bursts continue. We call on the Load Agent as the burst time slot comes to a close.

C. Algorithm 1(b)- Load Balancing

1) As the degree of variability is very high, we determine a base resource parameter B_j that is subjected to be assigned to loads j before the peak detection.
2) From the time the burst began till a stopping time, we have a total of N allocations with n_j being the incremental updates done for assignments. This is given by an estimated incremental load managing value I'

$$I' = \sqrt{\frac{2 \ln N}{n_j}}. \tag{4}$$

For instance, when the load agent is called for the first time, an initial expected B_j payoff plus the load managing value I' is computed. Essentially, we have the new updated value of B_j= B_j+I'
3) It is important to drive resource assignment through a learning process. Hence, we follow a RL strategy called Q-learning [20]. According to this technique we demarcate a mean reward that an agent could get from the environment. For our algorithm we approximate the expectation by using an exponentially weighted moving average (EWMA). We iterate over the state-action cycle as follows, here s, s'indicates BIP measurements, past and present states, respectively, a indicates different action parameter and r is the immediate reward parameter

$$\hat{Q}(s, a) := r(s, a) + \gamma \max_{a_i} Q(s', a_i). \tag{5}$$

Using EWMA:

$$\hat{Q}(s, a) := \alpha \hat{Q}(s, a) + (1 - \alpha) Q(s, a). \tag{6}$$

This is further elaborated in Section V.

D. Resource Management

Our objective is to maximize provider profit under throughput constraints. As estimating the changing cloud environment conditions is NP-hard, we model it as an ILP, with x_j and g_j being the cost of accessing the cloud node and cost of the link in \$/hr/month, respectively. Thus,

$$\max \sum_{i \in I} \sum_{j \in J} (x_j^e p_i^e - r_i^c g_j^e) \ ; \ \forall c \in C; \forall e \in E \tag{7}$$

$$r_i^c = \begin{cases} 1, & \text{if VM } c \text{ serves burst } i \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

A binary variable p_i^e, such that

$$p_i^e = \begin{cases} 1, & \text{if burst } i \text{ takes the extra resource } e \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

Algorithm 1 solves this problem by making cluster assignments following bin-packing heuristics, whereby bins are the virtual machines that execute the computations.

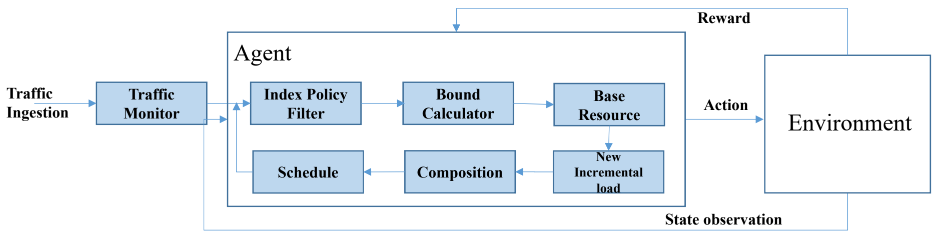

Fig. 2: Reinforcement Learning via Index Policy for Bursty Load Management

V. Preliminary Evaluation and Experiment

We deploy our simulator in python with an N node network. The results shown are an average of 10 simulations. Based on the j, the load on the node goes on increasing likewise the incremental value I' will be changing. The higher the value of n_j, the harder is the throughput constraint. We compare our model with a weighted first-come-first-serve policy (FCFS) and random weight policy (RW). We show through our experiments the maximum throughput achieved as follows:

$$T_{\max} = \max_j \{z_j(K+1)/Kq_j\} \tag{10}$$

We have set of time slot $K \in \{10^4, 5\cdot10^4, 10^5, 5\cdot10^5, 10^6, 5\cdot 10^6\}$. We vary $z_j, q_j \in (0,1]$ Our objective through these experiments is to measure the maximum throughput $T_{\max}$. We compare our model based on key metrics, such as reward earned, throughput, average wait due to differing loads, and variation in nodes. Further, we evaluate how the increase in iterations results in improved performance. Figure 2 presents the workload interactions with the agent. The arrivals are passed through the Index Policy Filter, where each burst window is time stamped. During the arrival process two types of actions can be taken, one is the overall optimal strategy for preempting traffic and the other is the index computation. The output of this goes to the Bound Calculator which calculates the incremental load value based on the predicted spike requirement. This leads to a reallocation of resources that is done in the next phase, as illustrated in Figure 2.

In this next phase, the Orchestrator measures the required resource increment (I') needed for the change in the initial allocation and performs re-computation of the desired resource requirement. This is given as input to the Resource Composition Logic. Hence, this phase works in cooperation with the infrastructure that plays an integral role in cloud resource provisioning. This module performs the requested resource VM chaining tailored to the SLA service demands. The VM composition logic design is left for future work.

Table II shows the variable definitions used in the simulations.

Algorithm 1 RL Algorithm based on Throughput Constraints for Burst Detection and Load Balancing

```
1: procedure INDEX-ASSIGNMENT()
2: Input: $S_i = \{a_i, a_{i+1}, \ldots, a_n\}$
3:   ▷ $a_n$ list of index values assigned to the workload, with $S_i$ as the stopping set
4: Input: $C_a = \{a_u^i; \forall i \in$ Continuation set$\}$
5:
6:     for $u$ in $L_n$ do
7:       $BestQoS_{A_u}, LeastQoS_{A_u}$ = $QoSe(A_u)$
8:       for $cluster_i$ in $N$ do
9:         if $BestQoS_{A_u} \leq cluster_i^{aR}$ & $cluster_i$ =="Initial Load (I')" then
10:            $cluster_i \leftarrow u$
11:            ▷ $l_n$ Available Network Resources with QoS values in $cluster_i$
12:
13:         end if
14:         if $LeastQoS_{A_u} \leq cluster_i^{aR}$ & $cluster_i$ ="I'" then
15:            $cluster_i \leftarrow u$
16:        ▷ mapping user application $A_u$ to network $cluster_i$
17:         else
18:          Go to next $cluster_i$ in the list $N$
19:         end if
20:       end for
21:       if $A_u$ not mapped to any $cluster \in N$ then
22:            ▷ Try to find a resource in available pool
23:        ▷ Repeat the Index-Assignment() for new VM pool after bound addition (B+I')
24:       end if
25:     end for
26: end procedure
```

TABLE II: Definition of Variables used for the simulation model

Variables	Definitions
$T_{\max}$	Overall Throughput achieved
j	Load variation
K	Time slots
N	Number of nodes in a network
I'	Incremental Load
z_j	Burst control parameter
q_j	Minimum throughput per load

A. Observations

We observe from Fig. 3 that as the bursty load increases, the average wait time increases. The weighted-FCFS performs better than the random-weight policy as the jobs are scheduled

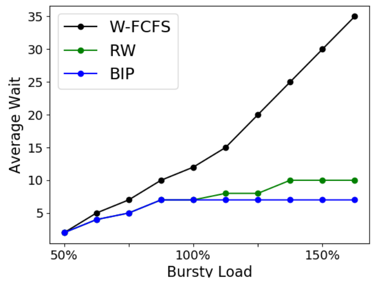

Fig. 3: Average wait

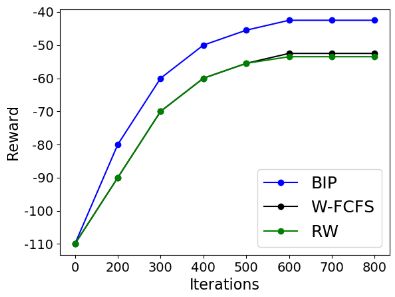

Fig. 4: Improving rewards

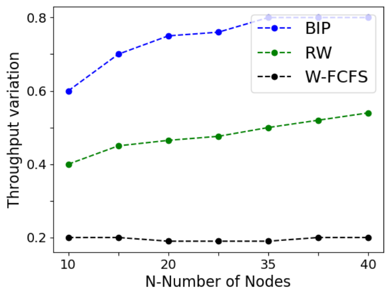

Fig. 5: Throughput variation with nodes

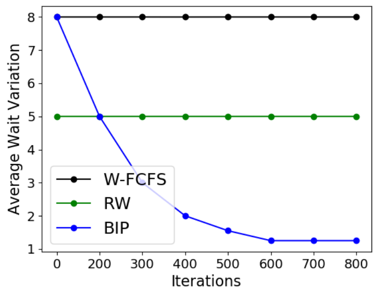

Fig. 6: Average wait variation as fct. of # of iteration

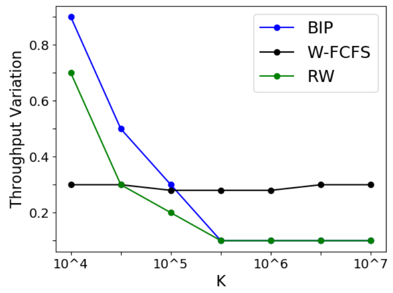

Fig. 7: Throughput variation vs. time slot

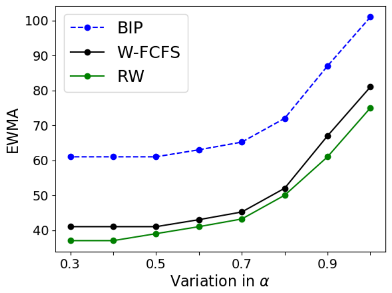

Fig. 8: Weighted moving average

based on simple weighted heuristics, whereas our Burst Index Policy (BIP) performs better than both the policies. This can be primarily attributed to the iterative training. In each training iteration multiple examples of request sequences were considered. Each load is initially assigned to a resource after which the load is kept on increasing and over-saturating the resource (upto 150%). As the main scope of this paper revolves around evaluating bursty workload management via RL we leave the communication delay comparison with job length for future work.

We observe in Figure 4 that BIP improves with the increase in the number of iterations, as we see initially the three policies are no better than one another, however, within the last few iterations it is clear that BIP outperforms the random allocation policy. This is exactly what the learning algorithm is explicitly optimizing. To draw further insight from this approach, running the learning algorithm for longer may produce better rewards. For instance, if some action instance is better than the one chosen it may lead to a better reward. As observed near the 600th iteration the graphs levels out, indicating convergence to a saturation state.

Clearly, Figure 4 suggests a progressively improving pattern as BIP learns to keep nodes free to satisfy jobs arriving in the near future which can be scheduled faster. In case of weighted FCFS such pre-emption is not possible as the new arrivals have to wait until the old arrivals are completely serviced. Further, both the W-FCFS and RW techniques rely on work-conservation, contrary to this BIP learns from experience and without prior knowledge.

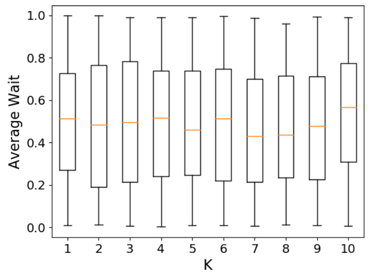

Fig. 9: Box plot of average wait

In Figure 5, the variation in throughput with the number of nodes in the network suggests a performance enhancement as opposed to the state-of-the-art W-FCFS and RW approaches. This can be attributed to the fact that as the number of nodes in the network increases, we have more options for re-allocation of jobs which in turn results in positive growth in throughput. Further, Figure 6 shows the average wait variation with respect to the iterations. As the iterations increase, the BIP wait times reduce due to the experience gained by the learning process. The boxplots in Figure 9 show that for all the 10 simulations, when we vary K, the average wait times remain consistently within the 90–95% quartile ranges, irrespective of the burstyness, showcasing the balancing act accomplished by the BIP algorithm.

Figure 7 shows the change in throughput with the variation in slot cycle. As expected, the throughput variation using BIP falls with increase in slot values but does not reach zero. This is highly dependent on the test failures when the feature inputs are not completely understood which results in drop in throughputs.

Figure 8 shows the expected moving average for workload balancing in cloud data-centres based on the state-action cycle. For BIP, we observe consistent improvements as opposed to the lower changes (indicating slower adaptation) to the changes in the state-of-the-art policies.

B. Discussion and Steps-forward

RL models highly depend on the iterations of the experiment and the action space. The scheduler may admit a specific subset of jobs. However, as the action space size increases exponentially with the number of jobs (in the order of 2^n) it is preferable to keep the action space small. In our work, we provision taking multiple actions in one step, that leads to more actions taken in each step but leads to very minimal inaccuracy. Although this inaccuracy is minimal or can be attributed to certain invalid actions, our focus in this experiment is only on scheduling a job so that the agent observes each state transition.

The formulation of our problem is based on a single pool of resources, this means that the overall machine level isolation has been removed. Practically, cloud clusters make scheduling decisions per machine, so each agent bases its allocation to a collection of cloud servers. Furthermore, all the jobs considered in this model are parallely executable as noted in Section I. These jobs may have different stages of execution. Additionally, the resource requirements of each job may not be known in advance. In such circumstances, the current model of resource assignments cannot provide an accurate prediction which would affect the provider profit adversely. In the future, we intend to explore the uncertainty of profit growth with different job models. This would further showcase the robustness of RL approaches.

VI. Conclusion

In this paper, a reinforcement learning (RL) technique to detect spikes and balance the load in cloud systems has been proposed. The preliminary evaluation shows that the agent which is controlled by the burst index policy (BIP) produces comparable or better results than the state-of-the-art techniques. Consistent drops in *average wait times* and over five times shorter wait times for bursty workloads over 150% have been observed. It is evident from the experiments that the "learning" strategies are more inclined towards gaining experience proportional to the failures observed by the agent. In this model, we employ the standard Q-learning based strategy for learning. As our framework is firmly rooted on the direct experience rather than on gaining prior knowledge, it is imperative that our framework can be used in different scenarios just by changing the reward metrics. Such models could provide practical solutions to cloud data center workload management. In the future it will be a worthwhile endeavour to analyse how such techniques can be used in vehicular environments such as [21] for improving congestion related challenges in networks.

References

[1] "CBRE Research Reports," https://www.cbre.us/research-and-reports/North-America-Data-Center-Trends-H2-2018.

[2] "Data centre reports," https://datacenterfrontier.com/the-eight-trends-that-will-shape-the-data-center-industry-in-2019/.

[3] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. ElBakoury, "Ultra-low latency (ULL) networks: The IEEE TSN and IETF DetNet standards and related 5G ULL research," *IEEE Comm. S. & T.*, vol. 21, no. 1, pp. 88–145, 2019.

[4] Z. Xiang, F. Gabriel, E. Urbano, G. T. Nguyen, M. Reisslein, and F. H. Fitzek, "Reducing latency in virtual machines: Enabling tactile Internet for human-machine co-working," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 1098–1116, 2019.

[5] F. Larumbe and B. Sanso, "Elastic, on-line and network aware virtual machine placement within a data center," in *Proc. IFIP/IEEE Symp. on Integr. Network and Service Managm. (IM)*, May 2017, pp. 28–36.

[6] W. Wang, Y. Bai, C. Yu, Y. Gu, P. Feng, X. Wang, and R. Wang, "A network traffic flow prediction with deep learning approach for large-scale metropolitan area network," in *Proc. IEEE/IFIP Netw. Operations and Management Symp.*, April 2018, pp. 1–9.

[7] J. C. Gittins, "Bandit processes and dynamic allocation indices," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 41, no. 2, pp. 148–164, 1979.

[8] J. Tai, J. Zhang, J. Li, W. Meleis, and N. Mi, "ArA: Adaptive resource allocation for cloud computing environments under bursty workloads," in *Proc. IEEE Int. Perf. Comp. and Comm. Conf.*, Nov 2011, pp. 1–8.

[9] J. A. Boyan and M. L. Littman, "Packet routing in dynamically changing networks: A reinforcement learning approach," in *Proc. Int. Conf. on Neural Information Proc. Sys.*, 1993.

[10] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "PCC: Re-architecting congestion control for consistent high performance," in *Proc. USENIX NSDI*, May 2015, pp. 395–408.

[11] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," in *Proc. ACM SIGCOMM*, 2014, pp. 455–466.

[12] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in *Proc. Eu. Conf. on Computer Sys.*, 2010, pp. 265–278.

[13] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. ACM Workshop on Hot Topics in Networks*, 2016, pp. 50–56.

[14] A. S. Jacobs, R. J. Pfitscher, R. L. d. Santos, M. F. Franco, E. J. Scheid, and L. Z. Granville, "Artificial neural network model to predict affinity for virtual network functions," in *Proc. IEEE/IFIP Network Operations and Management Symp.*, April 2018, pp. 1–9.

[15] M. Wang, T. Madhyastha, N. H. Chan, S. Papadimitriou, and C. Faloutsos, "Data mining meets performance evaluation: fast algorithms for modeling bursty traffic," in *Proc. Int. Conf. on Data Engineering*, Feb 2002, pp. 507–516.

[16] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the self-similar nature of ethernet traffic," *ACM SIGCOMM Computer Commun. Rev.*, vol. 23, no. 4, pp. 183–193, 1993.

[17] Z. Scully, M. Harchol-Balter, and A. Scheller-Wolf, "Optimal scheduling and exact response time analysis for multistage jobs," *arXiv preprint arXiv:1805.06865*, 2018.

[18] R. Meshram, D. Manjunath, and A. Gopalan, "On the Whittle index for restless multiarmed hidden markov bandits," *IEEE Transactions on Automatic Control*, vol. 63, no. 9, pp. 3046–3053, Sep. 2018.

[19] P. Varaiya, J. Walrand, and C. Buyukkoc, "Extensions of the multiarmed bandit problem: The discounted case," *IEEE Transactions on Automatic Control*, vol. 30, no. 5, pp. 426–439, May 1985.

[20] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[21] M. Aloqaily, V. Balasubramanian, F. Zaman, I. Al Ridhawi, and Y. Jararweh, "Congestion mitigation in densely crowded environments for augmenting qos in vehicular clouds," in *Proc. ACM Symp. on Design and Analysis of Intelligent Veh. Netw. and Appl.*, 2018, pp. 49–56.