Intelligent Scheduling of Bag-of-Tasks Applications in the Cloud

Preethi Sheba Hepsiba[1], Grace Mary Kanaga E[2]
Department of Computer Science and Engineering, Karunya Institute of Technology and Sciences, Coimbatore, India[1, 2]
Department of Computer Science and Engineering, CMR Institute of Technology, Bangalore, India[1]

***Abstract*—The need of efficient provision resources in cloud computing is imperative in meeting the performance requirements. The design of any resource allocation algorithm is dependent on the type of workload. BoT (Bag-of-Tasks) which is made up of batches of independent tasks are predominant in large scale distributed systems such as the cloud and efficiently scheduling BoTs in heterogeneous resources is a known NP-Complete problem. In this work, the intelligent agent uses reinforcement learning to learn the best scheduling heuristic to use in a state. The primary objective of BISA (BoT Intelligent Scheduling Agent) is to minimize makespan. BISA is deployed as an agent in a cloud testbed and synthetic workload and different configurations of a private cloud are used to test the effectiveness of BISA. The normalized makespan is compared against 15 batch mode and immediate mode scheduling heuristics. At its best, BISA produces a 72% lower average normalized makespan than the traditional heuristics and in most cases comparable to the best traditional scheduling heuristic.**

Keywords*—Bag-of-tasks applications; intelligent agent; reinforcement learning; scheduling

I. Introduction

A formidable challenge in cloud computing is the effective allocation of resources. In traditional scheduling, the efficacy of any scheduling heuristic largely depends on the characteristics of the workload (tasks to be scheduled) and resources. For a cloud service provider, maximum resource utilization and minimum power consumption result in the most profitability. For the customer, the scheduling of tasks in the cloud is also affected by network latency and cost. In addition, resource providers also strive to meet QoS (Quality of Service) requirements [1] from the customer in addition to minimizing energy consumption and maximizing resource utilization. Several VMs are hosted in a multi-tenant architecture, migrated to different hosts with different computing capacities in a datacenter. The principles that aid in attaining these objectives often result in a dynamically changing environment on the available cloud resources. This inadvertently leads to performance unpredictability. Andreadis et al [2] when proposing a reference architecture for datacenter scheduling pointed out that because of the complex nature of the environment in a datacenter, comparing scheduling heuristics and improving performance is challenging.

Self-learning systems [3] are the future of cloud computing. Reinforcement learning has been used for energy-aware resource scheduling [4] and results show that it can enhance energy efficiency in data centers. Adaptive scheduling for resource provisioning [5] also uses reinforcement learning. This work differs from adaptive scheduling in that the heuristics or meta-heuristics are not modified. However, by incorporating reinforcement learning, the state is observed and based on the information, the agent is trained to choose the best heuristic from its repository.

The bag-of tasks applications are a popular type of workload in the cloud and challenging to schedule in heterogeneous machines as opposed to homogeneous machines[6] for which an optimal schedule can be produced. Once tasks are allotted to a private cloud or any set of resources on a public cloud, there is a need for an intelligent agent in scheduling that is adaptable to a dynamic environment prevalent in the cloud. The hypothesis is that if an agent can sense the environment, apply learned best scheduling heuristic and simultaneously explore other options, over time the agent will choose the best heuristic for a state. In this work, an intelligent agent, BISA (BoT Intelligent Scheduling Agent) is proposed that uses reinforcement learning to choose the best scheduling heuristic in a state. BISA recognizes the current state based on the characteristics of the BoT workload and the available resources. It uses reinforcement learning to choose the best-known scheduling heuristic for the given state. The learning parameters in BISA, namely α and β control the exploration vs. exploitation strategy of BISA.

In previous work [7], the BIS agent was presented and the preliminary results from the training phase of BISA were divulged and discussed. In this work, BoT workload is generated synthetically. The testing is carried out rigorously for different configurations, and comparison of the normalized makespan over a series of runs of each cycle is presented for various learning parameters.

The contributions for scheduling bag-of-tasks using agents are:

- To develop BISA with an objective to minimize makespan.
- To test and present the results of the BISA agent on synthetic workload for different sets of learning parameters
- To test the hypothesis that BISA works in a dynamic environment. (simulated by changing configurations of Hosts and VMs)

The rest of the paper is organized as follows. In Section II, an overview of the BoT Workload and several heuristics in literature and the agent-based paradigm is presented. The

framework of the intelligent agent is presented in Section III. The results and discussion are elaborated in Section IV. The conclusion and future work are described in Section V.

II. Related Work and Motivation

In cloud computing which from an architectural point of view is a large-scale distributed system, traditional scheduling algorithms are not enough to schedule tasks efficiently. The type of workload also affects the design of a scheduling algorithm.

The bag-of-tasks workload is examined and the motivation for generating a synthetic workload is presented. Traditional heuristics and meta-heuristics that have been used to solve the resource allocation problem is presented followed by the motivation for the agent-based approach.

A. BoT Workload

A type of workload prevalent in large scale distributed systems such as the grid or cloud is the bag-of-tasks (BoT) workload. BoT workload comprises of independent tasks typically submitted as part of a larger application by the same user. The scheduling of independent tasks (bag-of-tasks) in heterogeneous systems is known to be an NP-Complete problem[8]. The following rules were used to identify BoTs from a trace [9]:

- BoT Size ranges from 2 to 64 parallel tasks.
- There are no serial jobs
- The run times are not extremely small and range from a few minutes to a day.

A synthetic workload [10] can also be generated based on the characteristics of a BoT. The model used was developed to test the performance of BoT workload in large scale distributed systems[11]. A synthetic workload offers more control in varying parameters. In this work, a synthetic BoT workload is generated, and the parameters used for the distribution are elaborated in Section III.

B. Traditional Heuristics

Schedulers typically use either an immediate mode or batch mode heuristic[8], [12] to schedule tasks in the cloud. The immediate mode heuristics described are (i) MCT (Minimum Completion Time), (ii) MET (Minimum Execution Time), (iii) Switching Algorithm, (iv) K-percent best, (v) Opportunistic Load Balancing. The Batch Mode Heuristics described are Min-Min, Max-Min Heuristic, and Sufferage. Some of these heuristics are based on concepts described in prior research work [8]. Enhanced versions of the Fastest Processor to Largest Task First (FPLTF) and sufferage are also described in [13].

An extensive list of 14 heuristics derived from 3 types of ordering of tasks within the BoT (Uniform, Large to Small and Small to Large) and the mapping of the task (Random Mapping, Maximum Expected Remaining Allocation Time Mapping, Maximum Current Remaining Allocation Time Mapping, Minimum Expected Remaining Allocation Time Mapping and Minimum Current Remaining Allocation Time Mapping) is presented by Garcia & Sim [14].

In many schedulers used in datacenters, traditional scheduling heuristics are used, but the pitfall is that none of the traditional heuristics are optimal in a heterogeneous system such as the cloud.

C. Meta-Heuristics

Several metaheuristics have also been applied to resource scheduling in the cloud to reduce the makespan and increase the throughput. These heuristics aim to produce optimal schedules for a multi-objective scheduling problem.

Tabu Search (TS) and Simulated Annealing (SA) were compared against Fastest Processor Largest Task (FPLT) [9] and the results show that Tabu Search and Simulated Annealing performed consistently better than FPLT even as BoT sizes increased. SA and TS had an 8% to 9% smaller makespan as compared to FPLT for globally arriving BoTs. The only shortcoming is the performance overhead incurred by TS and SA. A variation of SA, Thermodynamic Simulated Annealing (TSA) [9] is presented that also performs considerably better than SA.

A parallel Genetic algorithm [15] has been used in cloud scheduling which also tries to improve the VM utilization rate instead of just concentrating on a good resource scheduling algorithm. An IGA (Improved Genetic Algorithm) [16] to improve the utilization rate of VM's has also been proposed and the results show that the performance of IGA is twice that of TGA (Traditional Genetic Algorithm). A hybrid algorithm that used Genetic Algorithm, round-robin scheduling in deep neural learning works effectively in minimizing the makespan and other parameters in a cloud workflow. A software framework was developed for rapid prototyping of hybrid meta-heuristic schemes [17] as hybrid meta-heuristic techniques have proved to be more effective and adaptable.

A common disadvantage of most metaheuristics is the complexity but for a large-scale scheduling problem, the performance overhead is compensated with improvement in performance.

D. Agent-Based Approach

In a large-scale system like the cloud, accurate system requirement is difficult to obtain. An estimate of the execution time of the task will not be available. Meta-heuristics may sometimes provide near-optimal schedules but the time complexity for producing the schedule may be extremely high when the magnitude of the system increases which is the case in cloud.

A concise set of challenges that result from these heuristics are that:

- Heuristics that solve this problem require accurate information about the environment.
- The execution time of each task should be known a priori.
- The complexity (although polynomial) may be unacceptably high [13] when there are many tasks or resources.

A vital observation based on applying 14 scheduling heuristics based on ordering of tasks within a BoT and the choice of mapping [14] is that due to the *NP-complete* nature of the scheduling problem, there was not a dominant scheduling heuristic from among the proposed heuristics or the benchmark scheduling heuristics . In a configuration where there are some powerful virtual machines (VMs) that are free, a batch mode heuristic that sorts a set of tasks, largest to smallest (LtoS) and assigns the largest tasks to a machine that is more powerful first would reduce the makespan. In a different configuration where all the powerful machines are already overloaded, assigning tasks sorted smallest to largest (StoL) and assigning the smallest tasks to the overloaded powerful machines and the larger tasks to a less powerful machine would result in a reduction in makespan.

E. The Model of BISA

The BISA (BoT Intelligent Scheduler Agent) is an intelligent agent [18] that senses the environment and based on the current state chooses a scheduling policy, calculates the utility of that scheduling policy using a reward function and chooses to execute the schedule produced or choose another scheduling policy that gives a better utility. The result of the BIS agent is a schedule for the set of tasks in a BoT on a set of resources with a goal to minimize the makespan. Fig. 1 shows the abstract working of BISA and the context it works in a hybrid cloud environment. BISA works within a cloud (private or public) to effectively schedule the BoTs.

The following variables are be used to describe the working of the BIS agent.

Let S be the set of states that the agent has come across where

$S = \{s_1, s_2 \ldots s_l\}$

Let BoT_t be a *BoT* submitted at time *t*.

Let U be the set of users who submit BoTs where $U = \{u_1, u_2, \ldots u_n\}$

Let N_t denote the number of tasks in a BoT_t submitted at time *t*.

Let I_t be the ideal makespan a *BoT* submitted at time *t*.

Let $M(BoT_t)$ be the makespan of BoT submitted at time *t* after it has finished execution.

Let ST_t be the submission time of BoT_t.

Let $st_{i,t}$ be the execution start time of task$_i$ in BoT_t where $i=1,\ldots N_t$.

Let $wt_{i,t}$ be the waiting time of task$_i$ in BoT_t where $i=1,\ldots N_t$.

Let $ft_{i,t}$ be the finish time of task$_i$ in BoT_t where $i=1,\ldots N_t$.

Let $size\_mi_{i,t}$ be the size of task *i* in BoT_t where $i=1,\ldots N_t$ in Million Instructions(MI).

Let H be the set of hosts in a cloud where $H= \{h_1, h_2 \ldots h_n\}$.

Let *VM* be the set of virtual machine's in a cloud where $VM = \{vm_{1,1}, vm_{1,2}, \ldots vm_{r,p}\}$ where $vm_{r,p}$ denotes a *VM p* assigned to host *r*.

Let HN_r be the number of VMs assigned to host *r*.

Let $mips_{r,p}$ denotes the MIPS rating of a *VM p* assigned to host *r*.

Let *A* be the set of actions that is available to the agent. $A = \{a_1, a_2 \ldots a_m\}$. Table I shows the ordering and mapping policy that are used to produce the 15 possible heuristics as in [14]. Each action is produced by combining a task ordering with the task mapping policy. CT and ET are interpreted based on definitions in [12].

Let $Q_{private}$ and Q_{public} denote the set of tasks that are assigned to the private or public Queue.

Let $r_{l,j,k}$ be the reward for the k^{th} time an action *j* applied on state *l*.

Let $Q_{l,j,k}$ be the average of the first *k* rewards on action *j* in state *l*.

The working of the BIS intelligent agent is based on reinforcement learning [19] and the various stages are outlined below:

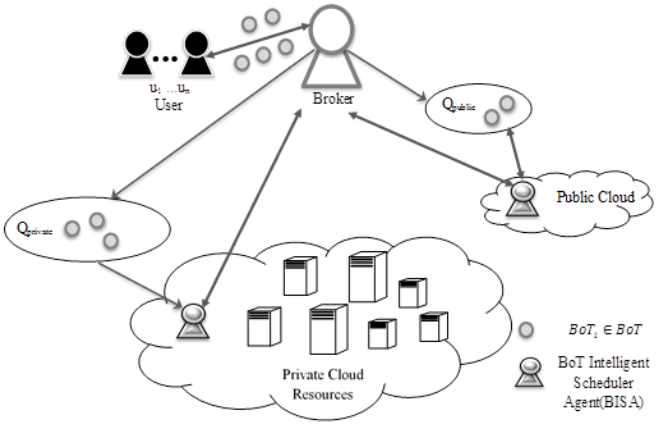

Fig. 1. The BISA Agent in Hybrid Cloud.

TABLE I. ORDERING AND MAPPING POLICY OF THE HEURISTICS

Ordering	
Unordered (U)	Random ordering of tasks within each BoT.
LtoS	Tasks are ordered Large-to-Small within the BoT.
StoL	Tasks are ordered Small-to-Large within the BoT.
Mapping Policy	
CT: Expected completion Time on a machine after completing all the tasks the machine that was previously allotted.	
ET: Expected completion time on a machine when no load is assigned to it.	
MinCT	Allocate task to the machine with minimum CT
MaxCT	Allocate task to a machine with maximum CT
MinET	Allocate task to a machine with minimum ET
MaxET	Allocate task to a machine with maximum ET
Random (R)	Allocate task to a random machine

1) Recognize the state: The size ratio of a BoT and the VM utilization for each classification of VMs form a state.

Let $geo\_mean(BoT_i) = \left(\prod_{n=1}^{N_t} size\_mi_{n,t}\right)^{1/N_t}$ be the geometric mean of BOT_t.

Let #small_tasks (BoT_t) be the count number of tasks in BoT_t whose $size\_mi_{i,t}$ is lesser than $geo\_mean(BoT_i)$

Let #large_tasks (BoT_t) be the count number of tasks in BoT_t whose $size\_mi_{i,t}$ is greater than $geo\_mean(BoT_i)$

Let $size\_ratio(BoT_t)$ be the size ratio of BoT submitted at time *t* calculated in Equation (1)

$$size\_ratio(BoT_t) = \frac{\#\text{small\_tasks}\ (BoT_t)}{\#\text{large\_tasks}\ (BoT_t)} \quad (1)$$

The size ratio is calculated as the number of small tasks divided by the number of large tasks. It determines whether the BoT is left normally distributed (a greater number of small tasks) or right normally distributed (a greater number of large tasks) or an equal number of large and small tasks. The size ratio is essential when the agent runs in a real-time environment where the parameters for the normal distribution that are typically used in a simulation are not known a priori.

In order to calculate VM utilization, VMs are first classified into three categories based on MIPS following which the average percentage of computing utilization in each of the three categories are obtained.

Let *C= {LG, ST, SM}* denote the classifications of VMs. Large VMs (*LG*) having processing capacity of MIPS > 80000, Standard VMs (*ST*) having MIPS between 40,000 and 80,000 and Small VMs (*SM*) having MIPS below 40,000.

Let $VM\_util_{c,}$ *= {H, M, L}* denote the average computing utilization of all the cloudlets/tasks running on the *VM* where *c = {LG, ST, SM}*. *H* denotes high CPU utilization (>80%). *M* denotes medium CPU utilization (40% to 80%). *L* denotes low CPU utilization (<40%).

2) Decide on the action that is matching with the goal: In this stage, the agent will choose the appropriate scheduling heuristic that minimizes the makespan of the BoT. The policy is chosen using a roulette wheel selector where all the probabilities for a set of actions in each state is on the scale of one. The values chosen for α and β allow us to control the tradeoff between exploration vs. exploitation. Exploration allows the agent to choose a new scheduling heuristic whereas exploitation acts like a greedy heuristic that choose the scheduling policy known to perform well.

3) Execute action: Schedule the BoT according to the scheduling policy decided upon in the previous step.

4) Assess chosen action: After the state is recognized, the ideal makespan is calculated for the given BoT. The ideal makespan, I_t is taken as the time taken for the largest task in a BoT to execute on the fastest available machine. It gives the agent a reference value to calculate the reward.

The reward is calculated by how well a policy performs with respect to the ideal makespan. The ideal makespan is calculated using Equation (2) which is the time is taken for the largest task i.e. the task with the longest instruction length (MI) to execute in the fastest machine (the VM with the largest MIPS rating).

$$I_t = \frac{max \sum_{i=1}^{N_t} (size\_MI_{i,t})}{max \sum_{r=1}^{H} \sum_{p=1}^{HN_r} mips_{r,p}} \quad (2)$$

The makespan of a BoT_t on action a_j is calculated using Equation (3) which is the difference between the maximum finish time of a task in BoT_t and the submission time BoT_t.

$$M(BoT_t) = max \sum_{i=0}^{N_t} (ft_{i,t}) - ST_t \quad (3)$$

This value of the makespan divided by the ideal makespan will be the reward, $r_{l,j,k}$ at the k^{th} time the policy a_j was used on a given state *l* using Equation (4). Initially, the reward, $r_{l,j,0}$ is 0.

$$r_{l,j,k} = \ln(1/(\ M(BoT_{t,} a_j) - I_t) \quad (4)$$

An incremental implementation is used to calculate the cumulative reward at step *k+1* in Equation (5) and this value is updated in the table of rewards.

$$Q_{l,j,k+1} = Q_{k+1,j} + \frac{1}{r_{l,j,k}}\left[r_{l,j,k+1} - Q_{l,j,k}\right] \quad (5)$$

In order to decide which action to choose at k^{th} time to schedule BoT_t in state *l*, the probability of choosing each action or policy is calculated. This probability is calculated by looking up a table of preferences for each action based on the reference reward.

Let $\pi_{l,\tau}(a_j)$ be the probability of choosing action a_j at play τ for state *l*, i.e. the t^{th} time the state *l* is encountered.

Let $P_{l,\tau}(a_j)$ be the preference of an action selected at play τ of state *l*. The preference for an action determines the likelihood of an action being selected for a state. The initial action preferences to 0. The initial reference reward, $\bar{r}_{l,0}$ for every state *l* where play τ is 0 is set to 0.1 meaning if the makespan of the scheduling policy is more than 10% as efficient as the ideal makespan, the reward is incremented and the preference for that action is set to a positive value. The reference reward following the first run, $\bar{r}_{l,\tau+1}$ is calculated as in Equation (6) where $0 < \alpha \leq 1$.

$$\bar{r}_{l,\tau+1} = \bar{r}_{l,\tau} + \alpha\left[r_{l,\tau} - \bar{r}_{l,\tau}\right] \quad (6)$$

During the first run, all the scheduling policies will have an equal probability of being chosen. Once a scheduling policy is chosen in play τ in state *l*, the preference for that action is updated. The preference is the difference between the reward $r_{l,\tau}$ and the reference reward $\bar{r}_t$ as given in Equation (7). The initial action preferences are set to 0.

$$P_{l,\tau+1}(a_j) = P_{l,\tau}(a_j) + \beta\left[r_{l,\tau} - \bar{r}_{l,\tau}\right] \quad (7)$$

The constant β is a positive step size parameter. A high reward will increase the probability of reselecting the action and a low reward should decrease the probability.

The probability $\pi_{l,\tau}(a_j)$ of selecting an action j at play τ in state l is calculated using Equation (8).

$$\pi_{l,\tau}(a_j) = \frac{e^{P_{l,\tau}(a_j)}}{\sum_{b=1}^{A} e^{P_{l,\tau}(a_b)}} \quad (8)$$

The performance metrics used to assess BISA are presented below:

- Overall makespan:

The overall makespan of each run is calculated by first calculating the makespan of individual BoTs and then taking the sum of the makespan. The individual makespan is calculated as shown in Eq (2) and the overall makespan is computed according to Equation (9).

$$Overall\ Makespan = \sum_{t=0}^{BoT} M(BoT_t) \quad (9)$$

- Normalized makespan:

The overall makespan can be a misleading metric because during each run of the simulation the load varies. The normalized makespan is based on the overall Makespan for all the BoTs generated within each run (48 intervals) calculated from Equation (9), the total runtime that denotes the actual runtime in the allotted VM of each task in a BoT during simulation and the number of tasks are those generated during each run (48 intervals). The normalized makespan is calculated by taking the sum of the tasks in all the BoTs generated during each run as shown in Equation (10).

$$Normalized\ Makespan = \frac{Overall\ Makespan}{Total\ Runtime * Number\ of\ Tasks} \quad (10)$$

III. Experimental Setup

The framework provided by cloudsim [20] to simulate the cloud and incorporate the intelligent agent as a thread. The VM's in the cloud run on the assigned host in the datacenter the allocation policy is time-shared. Each VM runs on a Processing Element (PE) which is 1 core of the host machine.

To test the reinforcement learning agent, 7 Virtual Machines (VMs) are set up in various configurations. Configuration 1 & 2 are given in Table II. In configuration 1, the ratio of LG:ST:SM is 1:3:3 with a total capacity of 30, 7000 MIPS. In configuration 2, the ratio of LG:ST:SM is 3:2:2 with a total capacity of 39, 6000 MIPS.

The Bag-of-tasks workload is generated synthetically based on characteristics outlined in [11]. Table III gives the parameters used to generate a synthetic workload. In each run of the simulation, a daily cycle of 48 intervals is simulated. The user submitting the BoT is generated using a Zipf distribution. The inter-arrival time between each BoT in a given interval is modeled using a Weibull Distribution. The size of BoTs is assigned to be powers of 2 between 4 to 512 also modeled using a Weibull Distribution. The average task runtime within a BoT is given using a Normal distribution and the task runtime variability of the tasks is given using a Weibull Distribution. The runtime is specified in MI (Million Instructions. The arrival rate is used to control the system load, and for this simulation, it is set to 1 The VMs operate on a time-shared basis on the hosts. The tasks (cloudlets) allotted to the VMs are also time-shared and hence, context switching takes place when multiple tasks take turns in using the processing element.

TABLE II. Configuration of the Cloud

Configuration 1		**Configuration 2**	
	MIPS		**MIPS**
VM #0	82000	VM #0	82000
VM #1	49000	VM #1	82000
VM #2	49000	VM #2	82000
VM #3	49000	VM #3	49000
VM #4	26000	VM #4	49000
VM #5	26000	VM #5	26000
VM #6	26000	VM #6	26000
Total Capacity	**307000**	**Total Capacity**	**396000**
LG:ST:SM = 1:3:3		**LG:ST:SM = 3:2:2**	

TABLE III. Workload Characteristics of BoT Workload

Number of Intervals in each Run	**48**
User (Zipf Distribution)	(368,1.31)
Interarrival Time (IAT) Weibull Distribution	(4.25,7.86)
Size of BoT (Weibull Distribution)	pow ((1.76,2.11),2)
Class of Bots	{4, 8, 16, 32, 64, 128, 256, 512}
Average Task Runtime within BoT (Normal Distribution)	(2.73,6.1)
Task Runtime Variability (Weibull Distribution)	(2.05,12.05)
Arrival rate	1
Runtime factor	1000,000

IV. Results and Discussion

The private cloud is set up in cloudsim and the BIS agent runs like a thread. BISA recognizes the state, decides on the action and updates the rewards and number of plays on a state. Fig. 2 shows a sample set of states observed after 10 runs comprising of a daily cycle of 48 cycles per run. The size ratio, along with the utilization of processing capacity in the various categories of VMs forms a state. State 30 with size_ratio as 1 and all the categories of VM utilization being "High" has the highest number of plays followed by State 1 and 18 that also have size ratio as 1 but with varying VM utilization.

Fig. 3 presents the normalized makespan after every run for a succession of 10 runs with different values for α and β. Each run consists of 48 intervals with BoTs generated using characteristics in Table II. Every marker represents the normalized makespan after a run. The normalized makespan is plotted against the number of tasks executed after ever run rather than run 1 to 10 so that the trend can be observed accurately.

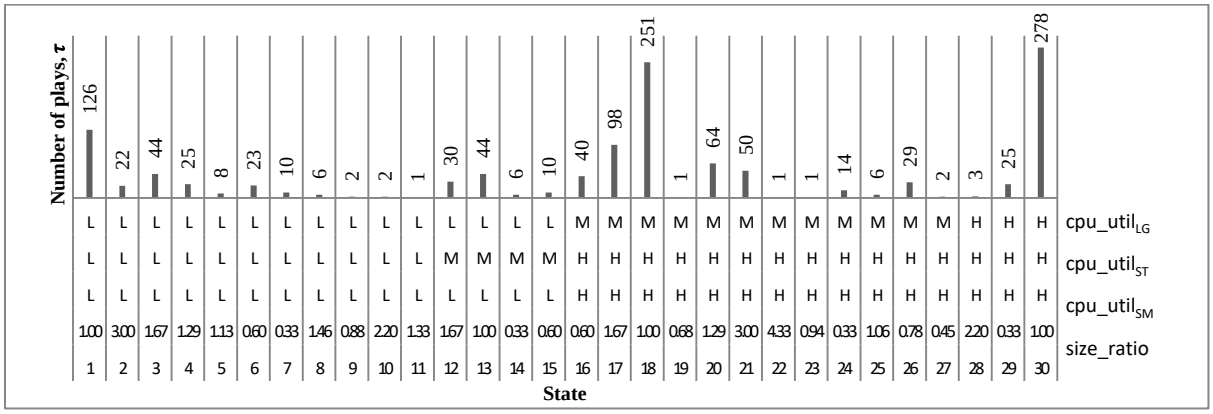

Fig. 2. The Number of Plays, τ on Various States at the end of 10 Runs (Daily Cycle of 48 Intervals Each).

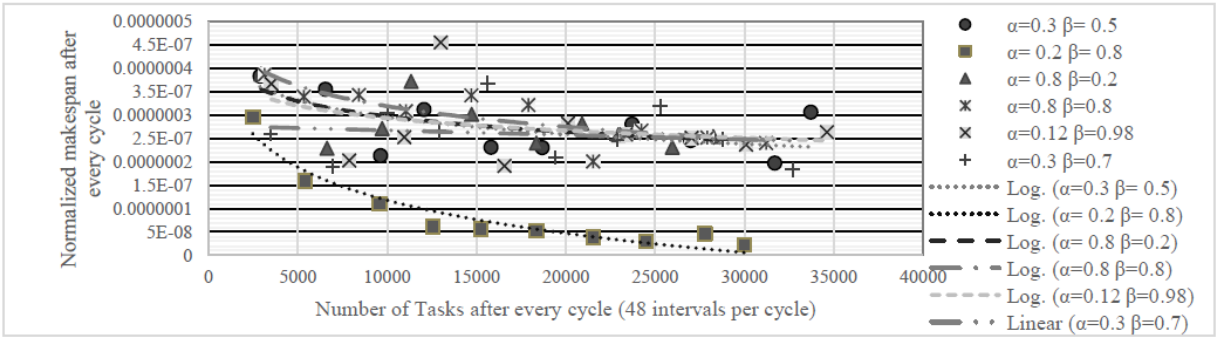

Fig. 3. Normalized Makespan Over 10 Runs for 6 Sets of Learning Parameters (Configuration 1-Tasks Time-Shared in VM).

For all learning parameters, a logarithmic decrease in the makespan can be observed. In some cases, the decrease is steeper. For α=0.3 and β=0.5 the likelihood that the agent will rely on a previously learned policy that produces a better schedule is slightly more than its behavior to explore other possibilities. The normalized makespan fluctuates with every run and even after 33720 tasks at the end of the 10th run. If a high value is set for both the exploration and exploitation parameters at α=0.8 and β=0.8, this results in the slightly higher normalized makespan in most runs. If an exploration value is comparatively much higher than exploitation value in the case of α=0.8 and β=0.2, this prevents the agent from getting stuck in any local optima and ensures all possibilities are explored. It can be observed that there is a steep learning curve in the initial 3 runs, but because of the high exploration value, the agent keeps choosing different heuristics and the makespan could not be reduced further. The agent produces surprisingly good results for a comparatively high exploitation value as compared to exploration value with α=0.2 and β=0.8. The normalized makespan in the first run itself is 23% better than the lowest recorded value when α=0.3 and β=0.3. Likewise, the learning curve is also steepest among all the sets. The lowest recorded makespan is 7 times lower than all other normalized makespan produced for any other set. Lastly, a very high exploitation value was set with α=0.12 and β=0.98. The agent stabilized after 4 runs at a suboptimal normalized makespan but produces the largest recorded value for normalized makespan in the second run.

The average normalized makespan of the last 5 runs in a total of 10 runs on BISA with the different set of learning parameters is presented in Fig. 4. The average of the last 5 runs is taken as the agent is expected to stabilize. It can be observed that BISA with α=0.2 and β=0.8 produced the best results. All other sets of learning parameters produce suboptimal results. For this configuration, U_MinCT produces the best results among the 15 scheduling heuristics. BISA with α=0.2 and β=0.8 has an average normalized makespan that is 72% smaller than U_MinCT.

BISA with α=0.8 and β=0.8 which has the largest average normalized makespan has an average normalized makespan which is 1.8 times larger than U_MinCT and performs better than 9 out of the 15 scheduling heuristics.

BISA was also tested in a second configuration as outlined in Table II. As can be seen in Fig. 5, the steepest learning curve is observed for α=0.2 and β=0.8. The best scheduling heuristic is U_MinCT and BISA (α=0.3 and β=0.5) produces a 39% larger average normalized makespan as compared to U_MinCT. On average, BISA produces suboptimal results and results in average normalized makespan that is less than 50% higher than the best (Fig. 6).

To test the hypothesis that BISA will be able to adapt to dynamic changes to the resources, BISA is run with the best performance learned from configuration 1 (α=0.2 and β=0.8) to configuration 2 (Fig. 7).

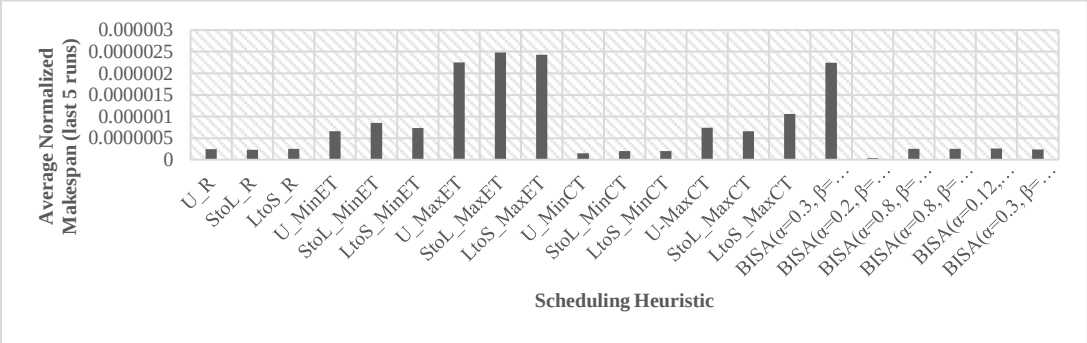

Fig. 4. Comparison of Average Normalized Makespan of Last 5 Runs (Configuration 1–Tasks Time-Shared in VM).

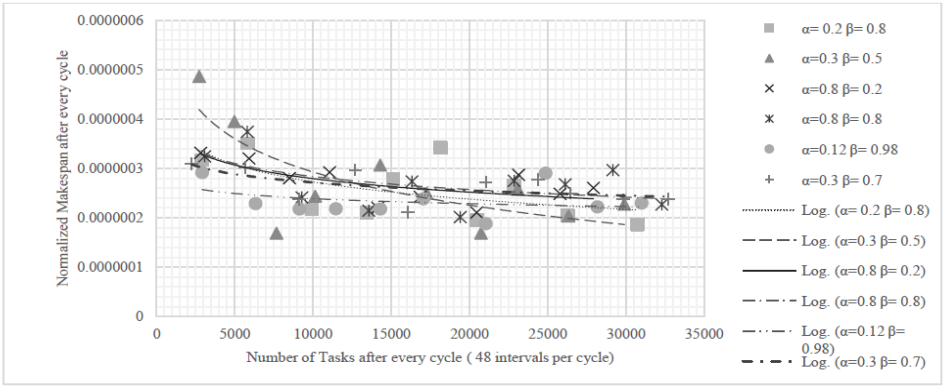

Fig. 5. Normalized Makespan Over 10 Runs for 6 Sets of Learning Parameters (Configuration 2-Tasks Time-Shared in VM).

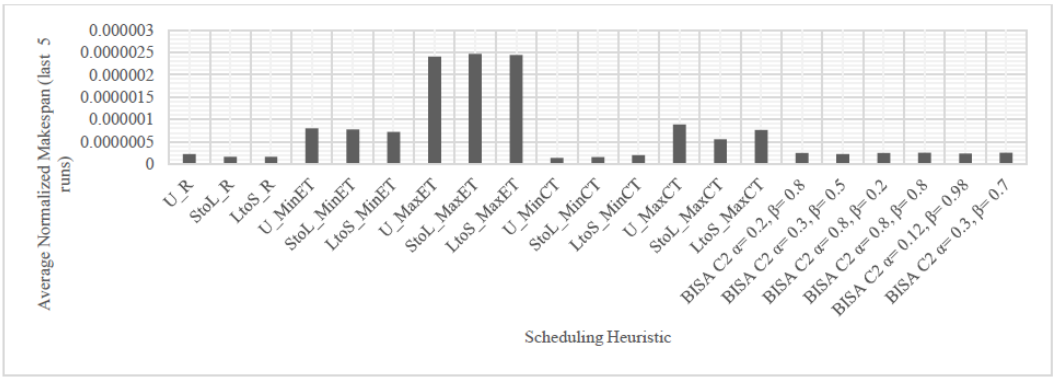

Fig 6: Comparison of Average Normalized Makespan of last 5 runs (Configuration 2 –tasks time-shared in VM)

Fig. 6. Comparison of Average Normalized Makespan of Last 5 Runs (Configuration 2-Tasks Time-Shared in VM).

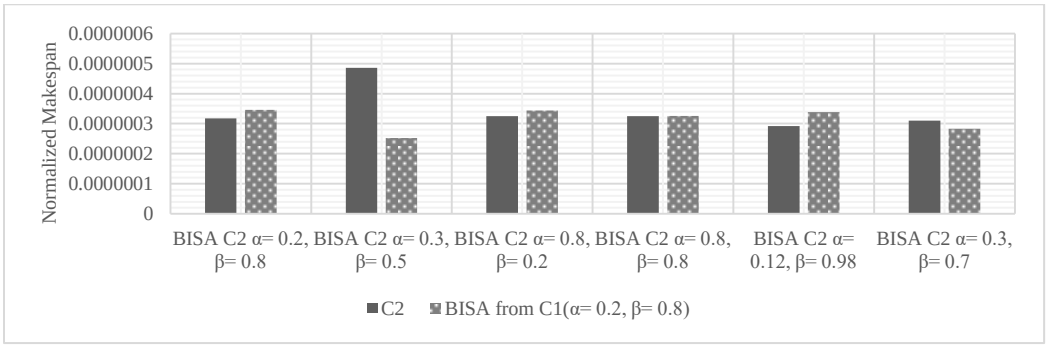

Fig. 7. Comparison of Normalized Makespan of the First Run of BISA Run aFresh on Configuration 2 (C2) with Learned Rewards and Preferences from Configuration 1(C1) (α=0.2 and β=0.8) on Configuration 2.

The first normalized makespan learned afresh in configuration 2 is compared with the first normalized makespan in configuration 2 learned from configuration 1 on α=0.2, β=0.8. The normalized makespan for the set (α=0.3 and β=0.5) operating on learned heuristics is half of the first normalized makespan when BISA is run on Configuration 2 afresh. The other sets do not show a comparable difference. It can also be observed that when using the same learning parameters α=0.2, β=0.8 when transitioning from configuration 1 to configuration 2 a negative result of increased makespan is observed. This indicates the state variables used in this simulation is not enough to differentiate. Hence in further work, states will be defined in further details so that learned best heuristics on a configuration can seamlessly be applied to other underlying configurations.

V. Conclusions and Future Work

An intelligent agent, BISA (BoT Intelligent Scheduling Agent) is proposed that learns the best scheduling heuristic to use in a state. The goal of BISA is to schedule each BoT such that the makespan is minimal. It is tested in two underlying configurations of cloud by allotting both tasks to VMs in a time-shared manner. It produces sub-optimal results comparable to the best scheduling heuristic for a given configuration. It can adapt to different underlying configurations and re-learn the best heuristic to use in a configuration. In configuration 1, for one set of learning parameter (α=0.2 and β=0.8) BISA outperforms all the traditional scheduling heuristics. In configuration 2 all the sets of learning parameters result in sub-optimal results with the best results on the set (α=0.3 and β=0.5). A 50% decrease in normalized makespan can be observed when transitioning from the best run on configuration 1 to configuration 2 on (α=0.3 and β=0.5) as compared to running BISA afresh in configuration 2.

It can be inferred that the choice of learning parameters determines the effectiveness of BISA. A good tradeoff between exploration and exploitation produces a lower makespan and converges to a near-best heuristic for a state rapidly. With BISA there is always the possibility of agents settling for a sub-optimal heuristic and it is crucial to choose learning parameters to minimize this occurrence.

BISA could be improved by re-examining the parameters that comprise the state and specifying them in a fine-grained manner. Additional objectives of minimizing cost, latency, and energy consumption, if factored into the reward obtained in BISA, solves a multi-objective problem. BISA trained to learn the best learning parameters that result in optimal results would also improve the performance of BISA in converging to and choosing the optimal scheduling heuristic in a state.

References

[1] R. Buyya, S. K. Garg, and R. N. Calheiros, “SLA-oriented resource provisioning for cloud computing: Challenges, architecture, and solutions,” Proc. - 2011 Int. Conf. Cloud Serv. Comput. CSC 2011, no. Figure 1, pp. 1–10, 2011.

[2] G. Andreadis, L. Versluis, F. Mastenbroek, and A. Iosup, “A Reference Architecture for Datacenter Scheduling: Design, Validation, and Experiments,” in Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, 2018, p. 37.

[3] B. Varghese and R. Buyya, “Next generation cloud computing: New trends and research directions,” Futur. Gener. Comput. Syst., vol. 79, pp. 849–861, 2018.

[4] T. Thein, M. M. Myo, S. Parvin, and A. Gawanmeh, “Reinforcement learning based methodology for energy-efficient resource allocation in cloud data centers,” J. King Saud Univ. - Comput. Inf. Sci., 2018.

[5] W. Iqbal, M. N. Dailey, D. Carrera, and P. Janecek, “Adaptive resource provisioning for read intensive multi-tier applications in the cloud,” Futur. Gener. Comput. Syst., vol. 27, no. 6, pp. 871–879, 2011.

[6] A. Benoit, L. Marchal, J. F. Pineau, Y. Robert, and F. Vivien, “Scheduling concurrent bag-of-tasks applications on heterogeneous platforms,” IEEE Trans. Comput., vol. 59, no. 2, pp. 202–217, 2010.

[7] P. S. H. Darius and E. G. M. Kanaga, “Bag-of-Tasks Intelligent Scheduling Agent (BISA) in Cloud Computing,” Adv. Comput. Commun. Paradig., pp. 239–246, 2018.

[8] O. H. Ibarra and C. E. Kim, “Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors,” J. ACM, vol. 24, no. 2, pp. 280–289, 1977.

[9] I. A. Moschakis and H. D. Karatza, “A meta-heuristic optimization approach to the scheduling of bag-of-tasks applications on heterogeneous clouds with multi-level arrivals and critical jobs,” Simul. Model. Pract. Theory, vol. 57, pp. 1–25, 2015.

[10] R. N. Calheiros and R. Buyya, “Energy-efficient scheduling of urgent bag-of-tasks applications in clouds through DVFS,” Proc. Int. Conf. Cloud Comput. Technol. Sci. CloudCom, vol. 2015-Febru, no. February, pp. 342–349, 2015.

[11] A. Iosup, O. Ozan Sonmez, S. Anoep, and D. Epema, “The performance of bags-of-tasks in large-scale distributed systems,” in Telecommunications Policy - TELECOMMUN POLICY, 2008, pp. 97–108.

[12] M. Maheswaran, A. Shoukat, H. J. Siegel, D. Hensgen, and R. F. Freund, “Dynamic Matching and Scheduling of a Class of Independent Tasks Onto Heterogeneous Computing Systems,” in Proceedings of the Eighth Heterogeneous Computing Workshop, IEEE Computer Society, 1999, p. 30.

[13] W. Cirne, D. Paranhos, F. Brasileiro, L. Fabrício, and W. Góes, “On the Efficacy, Efficiency and Emergent Behavior of Task Replication,” Large Distrib. Syst. Parallel Comput., vol. 33, no. 3, pp. 213–234, 2007.

[14] J. O. Gutierrez-Garcia and K. M. Sim, “A family of heuristics for agent-based elastic Cloud bag-of-tasks concurrent scheduling,” Futur. Gener. Comput. Syst., vol. 29, no. 7, pp. 1682–1699, 2013.

[15] Z. Zheng, R. Wang, H. Zhong, and X. Zhang, “An approach for cloud resource scheduling based on parallel genetic algorithm,” ICCRD2011 - 2011 3rd Int. Conf. Comput. Res. Dev., vol. 2, pp. 444–447, 2011.

[16] H. Zhong, K. Tao, and X. Zhang, “An approach to optimized resource scheduling algorithm for open-source cloud systems,” Proc. - 5th Annu. ChinaGrid Conf. ChinaGrid 2010, pp. 124–129, 2010.

[17] H. C. Lau, W. C. Wan, S. Halim, and K. Toh, “A software framework for fast prototyping of meta-heuristics hybridization,” Int. Trans. Oper. Res., vol. 14, no. 2, pp. 123–141, 2007.

[18] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009.

[19] R. S. Sutton and A. G. Barto, Introduction to Reinforcement Learning, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[20] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, “CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms,” Softw. Pr. Exper., vol. 41, no. 1, pp. 23–50, Jan. 2011.