A Reinforcement Learning Based Resource Management Approach for Time-critical Workloads in Distributed Computing Environment

Zixia Liu, Hong Zhang, Bingbing Rao, Liqiang Wang
Department of Computer Science, University of Central Florida
Email: zixia, hzhang1982, Robin.Rao@knights.ucf.edu, lwang@cs.ucf.edu

***Abstract*—Many data analyzing applications highly rely on timely response from execution, and are referred as time-critical data analyzing applications. Due to frequent appearing of gigantic amount of data and analytical computations, running them on large scale distributed computing environments is often advantageous. The workload of big data applications is often hybrid, *i.e.*, contains a combination of time-critical and regular non-time-critical applications. Resource management for hybrid workloads in complex distributed computing environment is becoming more critical and needs more studies. However, it is difficult to design rule-based approaches best suited for such complex scenarios because many complicated characteristics need to be taken into account.**

Therefore, we present an innovative reinforcement learning (RL) based resource management approach for hybrid workloads in distributed computing environment. We utilize neural networks to capture desired resource management model, use reinforcement learning with designed value definition to gradually improve the model and use ϵ-greedy methodology to extend exploration along the reinforcement process. The extensive experiments show that our obtained resource management solution through reinforcement learning is able to greatly surpass the baseline rule-based models. Specifically, the model is good at reducing both the missing deadline occurrences for time-critical applications and lowering average job delay for all jobs in the hybrid workloads. Our reinforcement learning based approach has been demonstrated to be able to provide an efficient resource manager for desired scenarios.

***Keywords*-reinforcement learning, time-critical, distributed computing**

I. Introduction

In today's big data era, enormous amount of data is generated continuously and awaiting to be analyzed. Some of the analytical applications, such as accumulative historical data statistics analysis, may not have strict deadline or are more tolerant of response delay. However, more and more data analyzing applications, such as streaming data processing, highly rely on timely response from execution result, and can be referred as time-critical jobs. In this work, we classify time-critical applications into two subcategories, time-critical streaming application with approximately periodical repeating patterns and non-streaming single running time-critical application without repeating patterns.

Due to frequent appearing of gigantic amount of data and deeper analytical workflow in contemporary data analytics, these applications often rely on large scale distributed computing systems which often include multi-cluster/multi-group topological structures due to geographical distribution of resources, internal isolation for resource management purpose, possible hybrid cloud elastic structure of computing capabilities, etc. In real-world environment, it is more likely that the large scale computing resources not only handle time-critical applications but also regular non-time-critical data analytics as well. We call such kind of mixed workloads with time-critical and non-time-critical applications as *hybrid workloads*. Although resource scheduling and management have been well studied in traditional parallel and distributed computing, time-critical big data analytics brings many new challenges such as how to balance multiple performance demands regarding hybrid workloads, how to at best effort fulfill the temporal needs from time-critical applications meanwhile keeping general job delay low.

In this paper, we present a reinforcement learning based resource management approach that takes into consideration many unique features specific to efficient resource utilization on large-scale distributed data analytical systems for hybrid workloads. Our approach coordinates overall cluster-level resource allocation and is compatible with different inner-cluster resource management components. By our approach, time-critical and non-time-critical applications can comprehensively utilize computing resources in an efficient and harmonic way. The experiment results demonstrate that our approach is more effective than rule-based resource management approaches due to better capability of capturing complex characteristics behind scene using reinforcement learning. The major contributions of this paper include:

- Use reinforcement learning based approach with proper neural network to obtain effective resource management solution that outperforms baseline rule-based resource manager.
- Design applicable value function definition in reinforcement learning for evaluating both effects of actions in reducing missing deadline occurrences and reducing average job delay.
- Improve reinforcement learning technique relating to ϵ-greedy strategy, as well as other accommodations to better suit RL approach to underlying practical problem and improve learning effectiveness and efficiency.

978-1-5386-5035-6/18/$31.00 ©2018 IEEE

Authorized licensed use limited to: China University of Petroleum. Downloaded on December 20,2020 at 13:24:02 UTC from IEEE Xplore. Restrictions apply.

- Compare effects of different reinforcement learning models with multiple competitive rule-based resource management schemes in various hybrid workload scenarios.

II. Background

Resource management for time-critical distributed computing faces many challenges, which requires a better understanding of both instantaneous and long-term influence of allocation decision. Traditional rule-based or white-box resource allocation models are inadequate on these goals due to intrinsic system complexity and difficulty in abstracting behavior characteristics. Instead, we tackle this problem by using the cutting-edge reinforcement learning technique, which is good at capturing intricate system features and gradually improving itself along RL process. Reinforcement learning is an important area in machine learning. The goal of which is to gradually learn to perform good actions in response to state representations of different environment status, in earning maximum action value.

Recently, many significant achievements have been accomplished using reinforcement learning technique. For one representative example, Deepmind [1] recently developed a computer Go program called AlphaGo to defeat world champions [2, 3]. In [2], they facilitate Monte-Carlo Tree Search (MCTS) algorithm [4] with their own policy network and value network, which are obtained via techniques including supervised learning and reinforcement learning with deep neural networks. Further in [3], they apply reinforcement learning strategy directly without human knowledge, and are able to achieve model which surpasses previous one [2] in a short interval of time.

In the field of distributed computing, there are also studies focusing on utilizing various approaches in improving system performance and functionality. [5] proposes a search-based automatic parameter tuning method for MapReduce [6] framework. It uses a genetic algorithm to identify near-optimal configuration of several Hadoop platform parameters in minimizing job execution time. [7] proposes an architecture for scientific workflow management systems that supports provenance and atomicity to distributed scientific computations represented as scientific workflows. [8] proposes a framework facilitating execution of a big data computing application with multiple spark clusters. [9] proposes JDS-HNN, a heuristic approach that utilizes Hopfield Neural Network for job scheduling and data replication in a grid, in purpose of minimizing job execution makespan and data file delivery time. [10] and [11] try to deduce machine learning model that could respectively capture the relationship between execution time or percentage performance improvement with parameter configurations. The former is depicted as a regression problem, while the later a binary and multi-category classification problem. [12] proposes a cloud computing configuration optimization method for big data analytical platforms. It focuses on configurations such as selecting appropriate type and numbers of instances, and could distinguish a good configuration efficiently due to Bayesian optimization efficiency.

Approaches including reinforcement learning have been investigated to be applied to computing resource management. [13] introduces a parallel temporal-difference reinforcement learning algorithm for achieving optimal cloud resource scaling decision. [14] employs deep learning strategy in reinforcement learning to accomplish resource management in a cluster from gradually accumulated experience and the result called DeepRM is comparable to state-of-the-art heuristics. [15] presents a novel multi-agent reinforcement learning method for load balancing problems of grid computing resources composed of multiple clusters with large-scale computing jobs. [16] proposes to enable model-free control in distributed stream data processing systems using deep reinforcement learning, which aims at minimizing tuple processing time in average. However, none of the aforementioned researches handles hybrid time-critical workload in distributed computing environment.

III. Problem Description

The overall computing resource is defined as multiple computer clusters with specific capability. To define computing capability, we use the concept of executor in Yarn [17], which is a stationary basic allocation unit of a combination of virtual CPUs (*a.k.a.,* vCPUs) and memory. This concept is general and is widely adopted in popular distributed computing systems such as Apache Hadoop and Spark. Clusters can be heterogeneous in the sense that they provide overall different numbers of executor units. In this paper, we assume executors in a cluster are identical to different applications so that they provide the same computing performance.

The workload deployed to the overall computing resource is hybrid. More specifically, it can be a combination of three categorical applications: (1) streaming applications that repeat executions of themselves with different batches of arriving streaming data, which intrinsically contain temporal desire thus are time-critical; (2) non-streaming time-critical applications without repeating patterns; and (3) regular non-time-critical applications. Each application may have different duration, execution time on single executor and different computing capacity needs in unit of executors. Streaming applications can also possibly experience streaming input data fluctuation. In such a case, the streaming application will reflect this as a different desire for numbers of executors in order to maintain equivalent workload on single executor to better achieve temporal requirement in data fluctuation. In addition, we assume that computing resources are released between intervals of batch executions of streaming applications for less idling resource occupation.

At each moment, multiple applications may possibly arrive at the scheduling pool, which mimics the practical

Authorized licensed use limited to: China University of Petroleum. Downloaded on December 20,2020 at 13:24:02 UTC from IEEE Xplore. Restrictions apply.

scenario where multiple users submit jobs simultaneously. At each moment, the resource management module picks one of the awaiting applications to deploy onto one of the computing clusters for actual execution. Selecting which application and which cluster is crucial in determining the overall performance of all applications.

Internal scheduling within a cluster is accomplished by local scheduler. The global resource management module does not interact or intervene with the operation of the local internal schedulers. As a matter of fact, this design brings up one advantage of the global resource management module, that is, our proposed reinforcement learning is capable of coordinating with different local internal scheduling schemes, which greatly enhances the generality and applicability of the proposed approach to different practical scenarios.

For time-critical applications, if the actual execution time exceeds its temporal deadline requirement, this triggers the "missing deadline" (MDL) event. It can be caused by prolonged resource allocation waiting time and/or insufficient executor allocation due to resource competition among concurrent applications. For both time-critical and non-time-critical applications, we also generally care about the average job execution delay ratio, where a lower average of job execution delay ratio represents more efficient overall application running.

A critical problem that the global resource manager needs to solve is how to schedule a hybrid workload for minimal total missing deadline events and average job delay ratio. Hence it may need to abstract cluster running statistics, application characteristics, workload pattern, and internal local scheduler behavior. The achieved resource management module should be capable of achieving favorable balance in goals and provide effective resource management scheme.

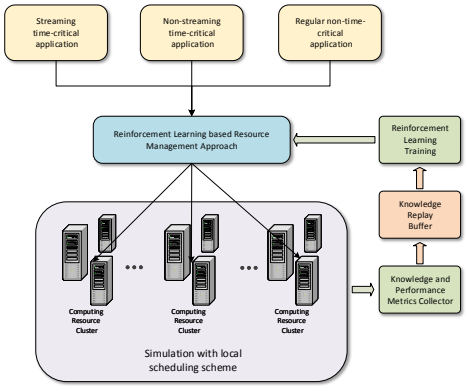


Figure 1. An illustration of the architecture of our approach.

IV. The Reinforcement Learning Based Approach

As depicted in Figure 1, we propose a resource management approach based on reinforcement learning for hybrid workloads in distributed computing environment. Recall the main purpose is to select an application from possible multiple candidates and identify a suitable cluster for it among all available ones for application deployment.

A. Concept Definition and Value Function Design

The *environment* of our reinforcement learning can be described as follows: the overall distributed computing resource is composed of multiple computing clusters. These clusters may provide different numbers of executors, with each executor a fixed basic combinational allocation unit of vCPU and memory resources. The applications awaiting to be allocated can be categorized as three classes: streaming time-critical, non-streaming time-critical, and regular non-time-critical.

The *state* of the environment is defined as a vector of several components, with details shown in Table I according to experiment setup. (1) For each cluster, a component is used for describing its computing capability in terms of executors, discrete resource utilization statistics in the past 100 time slices if available, and total occurrences of missing deadlines for time-critical applications in current episode. (2) Another component is for job profile. Without loss of generality, we use streaming time-critical application as an example for description. The profile includes: (a) The category of job: streaming time-critical application, non-streaming time-critical application, or non-time-critical application; (b) Discrete probability distribution of resource utilization due to streaming data fluctuation; (c) Single executor occupation time, execution deadline for batch running, and overall duration of the streaming application. Applications in the other two categories can nonetheless use the same state format with intuitive modifications. For all state representations, we uniformly expand or shrink individual value range accordingly to maintain similar data range for different dimensions in state vector.

Table I
State Representation in Reinforcement Learning Model

Vector Component	**Dimensions**
Cluster ($i = 1 \ldots 5$)	
Sequence number	1
Capability	1
Occupation statistics (latest 100 steps)	100
Current total missing deadline	1
	515 ($\mathbf{103 \times 5}$)
Application	
Category	1
Discrete resource utilization distribution	10
Single executor occupation time	1
Execution deadline	1
Duration	1
	14
Overall state vector	**529** ($\mathbf{515 + 14}$)

The *action* space of our reinforcement learning model is composed of a number of actions equaling the number of candidate clusters from 1 to C. In our experiment, $C = 5$.

Authorized licensed use limited to: China University of Petroleum. Downloaded on December 20,2020 at 13:24:02 UTC from IEEE Xplore. Restrictions apply.

An *episode* is defined as the successful finishing of a fixed number of applications, where each application finishes its execution (for non-streaming applications) or a number of repeating executions (for streaming applications) during its lifespan.

The *value* of an action with respect to assigning a time-critical application i to a cluster can be defined as follows:

$$value_i = -\lambda \cdot MDL_i - \gamma \cdot MDL_{t>t_i} - \eta \cdot DelayResult_i \quad (1)$$

Similarly, the *value* of an action with respect to assigning a regular non-time-critical application i to a cluster can be defined as follows:

$$value_i = -\gamma \cdot MDL_{t>t_i} - \eta \cdot DelayResult_i \quad (2)$$

where MDL_i is the accumulative occurrence of missing temporal deadline events of application *i* during its lifespan. $MDL_{t>t_i}$ is total number of missing deadlines for all applications in entire resource that happen after t_i when application *i* is deployed and before eventual termination of *i*. The delay result $DelayResult_i$ is the average value of all running time ratio of application *i* in possible multiple executions during the episode. A running time ratio of application *i* is the ratio of its actual running time over its optimal expected running time. λ, γ and η are hyper-parameters. In our experiment $\lambda = 1$, $\gamma = 0.02$ and $\eta = 0.1$.

The *value* of an action here is equivalent as accumulated rewards an action received in an episode. Definition of the *value* function can be interpreted as follows. Our purpose for resource management for hybrid workloads is to increase the overall resource utilization of all clusters by assigning newly coming applications, meanwhile achieving multiple performance considerations. For each application, we would like the average job delay result remains low. However, for any time-critical application, we also want to greatly restrict any temporal deadline missing. Consequently, these should be comprehensively taken into consideration by desired resource manager. Specifically, our value function not only depicts the influence of selected action to the occurrence of application's own missing deadline events, but also takes into account the influence of it to all missing deadline events happened after the action, regardless of whether the event is solely related to application i or not. This ensures the global vision of action value evaluation, and enables consideration of correlated influence of multiple actions during action selection performed by the approach.

B. Neural Network and Reinforcement Learning Method Design

To capture important characteristics of efficient resource management decision, we utilize a neural network of three layers, including two fully connected hidden layers of 2000 and 500 neurons respectively and one output layer with neurons equal to the number of available actions.

We use reinforcement learning framework to improve the neural network model served as a value function estimator. Given feature vector input regarding to a single application, the network outputs value estimation for each possible actions. Action is selected based on ϵ-greedy strategy.

To accomplish the training process, in each episode, we gather resource management actions made by the neural network model and attach them to the knowledge replay buffer, which consists of at most 50000 latest resource management knowledge. At the end of an episode, each action taken is evaluated and their values are supplemented to corresponding items in the knowledge replay buffer. Therefore, our approach can be categorized as a Monte-Carlo method [18] instead of Temporal Difference method [19]. After finishing of each episode, if enough knowledge is accumulated in the knowledge replay buffer, neural network training process will be carried out with 1000 randomly selected samples from knowledge replay buffer.

C. Strategies in Accommodating and Improving RL based Approach

We briefly describe the special strategies and considerations we have taken in the process of better accommodating the reinforcement learning based model in the desired resource management problems as follows.

1) Enabling resource management target selection among multiple applications: Note that for the designed neural network, it accepts feature vector of a single application awaiting resource allocation, and the output is the value estimation of all actions corresponding to allocating the aforementioned application to according cluster. However, it is possible that there are multiple applications awaiting scheduling by our resource manager at certain moments. We would like our reinforcement learning based approach to be able to deliberately choose the most suitable application among all awaiting ones at each time slice.

A fundamental requirement for neural network is the stationary length of input vector, which is obviously achievable when the input is with respect to one application, but not likely when input vector is used for describing multiple jobs, where the number of jobs remains variable and unbounded. Considering features of adopted reinforcement learning process, where an action is chosen based on the largest values given by all available actions, we decide to modify the outer-layer of the action selection process, instead of the neural network input vector, to accommodate the need in choosing among multiple awaiting applications.

In our modification, the neural network remains unchanged, that is, its input is still with respect to one single application. However, by utilizing the uniform meaning of values in the output, in each time slice, if there are multiple jobs awaiting, we would feed each of them to the neural network individually, then append all values together as one vector. The vector is then passed through the ϵ-greedy

Authorized licensed use limited to: China University of Petroleum. Downloaded on December 20,2020 at 13:24:02 UTC from IEEE Xplore. Restrictions apply.

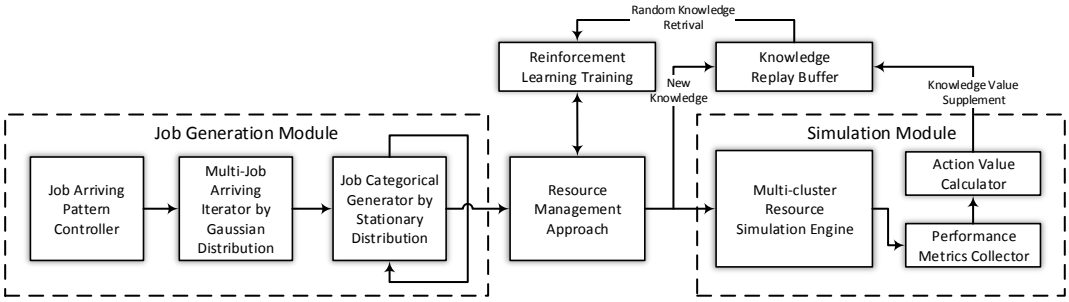


Figure 2. Reinforcement learning procedure in single episode.

process as described below. For any position selection in vector, the position is then translated back to corresponding application and its action. This enables the choice among multiple awaiting jobs without modifying neural network input.

2) Improved ϵ-greedy strategy for more effective and efficient RL process: In reinforcement learning, there is a strategy with the name ϵ-greedy strategy. It can expand the exploration by randomly selecting actions with ϵ probability, so that the vision of the reinforcement learning is not restricted by current capability of the obtained model. It also provides opportunities to escape out of local optimum, and enhances chances to reach global optimum.

However, one disadvantage of the ϵ-greedy strategy is that it solely expands exploration by random actions, therefore no other possible valuable prior knowledge is utilized. This may hinder efficiency in achieving better evolved resource manager via reinforcement learning model. In light of this, we innovatively apply an improved ϵ-greedy strategy by default for more effective and efficient RL process. Specifically, the improved ϵ-greedy strategy uses both the random action and action from our baseline resource manager to increase exploration. The details of it are stated as below.

$$action = \begin{cases} \text{random action} & r_1 < \epsilon_1 \ \ \& \ \ r_2 < \epsilon_2 \\ \text{baseline guided action} & r_1 < \epsilon_1 \ \ \& \ \ r_2 \geq \epsilon_2 \\ \underset{j \in J, a \in A}{\arg\max} \ \ \mathrm{Q}_a(s_j) & r_1 \geq \epsilon_1 \end{cases} \tag{3}$$

When deciding an action, a randomized value from 0 to 1 is compared to current ϵ_1 value. If less than, an action is selected with ϵ_2 possibility being random and $1 - \epsilon_2$ probability being the action selected by our baseline resource manager; otherwise, action with largest value given by network outputs is selected. Thus, knowledge from the baseline approach could be utilized to enhance reinforcement learning effectiveness and efficiency meanwhile the random exploration is carried out. This improved strategy is shown in Eqn. 3, where J is the set of scheduling awaiting jobs, A is the action set and $\mathrm{Q}_a(s_j)$ gives value estimation of action a for application j, r_1, r_2 are randoms in $[0, 1]$. In experiments, we set $\epsilon_2 = 0.5$, and ϵ_1 linearly decrements from 0.8 to 0.00001 in Episode 1 to Episode 1900 (total episode is 2000).

3) Training with randomized workloads: In specifying single application characteristics in a workload, there are multiple configuration parameters that can be varied, such as discrete resource utilization distribution for streaming application, single executor occupation time, execution deadline, and duration for application. These parameters define the execution features of an application. Whenever an application is generated, especially in verifying the performance effect of resource management approaches, we implement proper randomness to these parameters to expand generality of the workload.

The question remains whether in the reinforcement learning training process, application characteristic randomness, which enhances workload generality, will bring difficulty or even hinder the training process. We consequently verify the answer to this question in several circumstances and have made the following observation:

For different value function designs, it is possible that the broad generality of workload by application randomness may bring great difficulty in training process and harm the effectiveness of reinforcement learning model. Using one sampling of randomized workload as a fixed workload for the entire training process could mostly alleviate the problem, and surprisingly, the resulting model ends up reasonably well for using in randomized workload environment. However, after refining the value function design to be what we present in this work, we discover that the effectiveness of the training process remains well even for randomized workload and the performance of the resulting model is further greatly improved comparing to the model obtained by training with a fixed sampling of randomized workload. The underlying reason behind this is apparent since the trained model with randomized workload has better knowledge and vision from the better workload generality. We thus apply training with randomized workloads in experiments.

D. RL Training Algorithm for Resource Management

The main algorithm for RL training is presented in Algorithm 1. A detailed presentation of the procedure for

Authorized licensed use limited to: China University of Petroleum. Downloaded on December 20,2020 at 13:24:02 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1 Resource Management and Training Algorithm

$\boldsymbol{i}$: Current episode; $\boldsymbol{t}$: Time,
$\boldsymbol{N}$: Total number of episodes,
$\boldsymbol{NoW}$: Predefined number of applications in workload,
$\boldsymbol{fNoW}$: Finished number of applications in workload,
$\boldsymbol{RGA}$: Number of randomly Generated Applications,
$\boldsymbol{vecV}$: Value vector for jobs in job pool,
$\boldsymbol{nn}$: Neural network model,
$\boldsymbol{kb}$: Knowledge replay buffer

kb=[]; initialize ϵ_1
for i in range(N) **do**
  t=0; RGA=0; $fNoW = 0$
  while $fNoW < NoW$ **do**
    if $RGA < NoW$ **then**
      RGA += $generateJobs(jobPool)$
    end if
    vecV=[]
    for all job j in $jobPool$ **do**
      generate feature vector s for j
      vecV.$append$(nn.$eval(s)$)
    end for
    if $random(0,1) < \epsilon_1$ **then**
      if $random(0,1) < \epsilon_2$ **then**
        action=randint(0,len(vecV)-1)
      else
        action=$action_{baseline}$
      end if
    else
      action=$argmax$(vecV)
    end if
    $takeAction$(action)
    knowl=$generateKnowledge$(action)
    kb.$push$(knowl)
    $fNoW+ = removeFinishedJobs()$
    t++
  end while
  update $value$ for new knowledges in episode i
  if $len(kb) > batchsize$ **then**
    minibatch=$random.sample$(kb,batchsize)
    Train neural network by stochastic gradient descent
  end if
  ϵ_1 decrements
end for

one single episode is shown in Figure 2. The entire training process is composed by N training episodes. In our experiments, $N = 2000$. Obtained knowledge in each episode is pushed into the knowledge replay buffer. At the end of each episode, the value of each new knowledge is calculated and supplemented to the knowledge buffer. If sufficient knowledge exists, the neural network training is launched where the newly obtained model is used in the next episode. The weight of the neural network is updated via Stochastic Gradient Descent (SGD) with mean square error.

V. Experiment Results

In this section, we present experiment design and results. The experiment is launched in our designed simulator, in a computing resource of 5 clusters with different number of executors defined by $[500, 800, 1200, 1300, 1900]$. For inner-cluster local scheduler, a First-in-First-out (FIFO) scheduler is adopted, which is popularly provided as a default scheduler in various big data platforms. First, we introduce design of job arriving patterns and the baseline rule-based resource managers. Then we present experimental results regarding the performance comparison of RL based resource management approach to baseline models.

A. Job Arriving Patterns

To thoroughly testify the effectiveness of designed reinforcement learning based approach, we adopt three statistical patterns as job arriving patterns in resource management experiments.

1) By Bernoulli process: In the first job arriving pattern, we assume at each simulation time step, a job arriving event happens with a stationary probability $\rho = 0.08$, where each experiment is fully independent. This obeys the definition of "*Bernoulli process*". When such an event happens, we let the number of arriving jobs be decided following a rectified discrete Gaussian distribution: $Num_{job} = \max(\text{round}(\text{N}(\mu, \sigma^2)), 1)$. For Bernoulli pattern, $\mu = 1.5,\ \sigma = 1$, for other two patterns, $\mu = 3,\ \sigma = 1$.

The category of arriving jobs also follows a stationary distribution, with β_1, β_2, and β_3 being probabilities for regular non-time-critical, non-streaming time-critical and streaming time-critical, respectively. Here $\beta_1 + \beta_2 + \beta_3 = 1$, and remains the same for other job arriving patterns as well. For all three patterns, $\beta_1 = 0.5,\ \beta_2 = 0.25$, and $\beta_3 = 0.25$. From the property of Bernoulli process, the probability of having a new job arriving event with interval i is equivalent as the probability of having a first success in i consecutive yet independent Bernoulli experiments, of which the expression can be written as: $P(i) = (1-\rho)^{i-1} \cdot \rho \cdot I_{i>0}(i)$. Here $I_{set}(x)$ is the indicator function, it is 1 when $x \in set$, otherwise 0.

2) By Uniform distribution: In the second job arriving pattern, we assume the occurring interval of job arriving event follows a discrete uniform distribution in $[a, b]$, here $a = 1,\ b = 39$. That is, in our selected range, the occurring probability for job arriving event with each interval time i is the same. Thus, the probability function can be presented as:

$$P(i) = \frac{1}{|b - a + 1|} I_{i \in [a,b]}(i) \tag{4}$$

Authorized licensed use limited to: China University of Petroleum. Downloaded on December 20,2020 at 13:24:02 UTC from IEEE Xplore. Restrictions apply.

3) By Beta distribution: In the third job arriving pattern, we assume the occurring interval of job arriving event follows a modified discrete version of Beta distribution. α, β and M are pattern parameters, where in our experiment $\alpha = 4,\ \beta = 2,\ M = 30$. The probability of job arriving event with interval i therefore can be written as:

$$P(i) = \frac{I_{i>0}(i) \cdot \Gamma(\alpha+\beta) \cdot [B\left(\frac{i}{M};\alpha,\beta\right) - (B\left(\frac{i-1}{M};\alpha,\beta\right)]}{\Gamma(\alpha)\Gamma(\beta)} \tag{5}$$

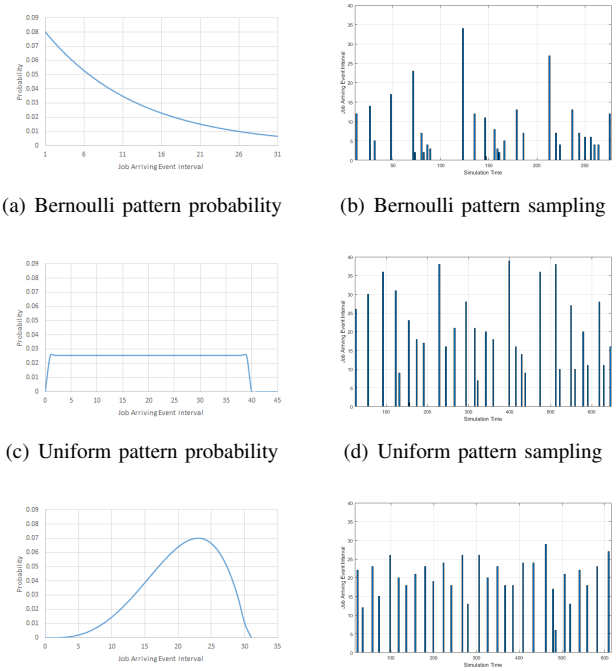
where,

$$B(x;\alpha,\beta) = \int_0^x t^{\alpha-1}(1-t)^{\beta-1}dt \tag{6}$$

and $\Gamma(x)$ is defined as the Gamma function:

$$\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt \tag{7}$$

The probability of job arriving event with interval time i for all three job arriving patterns, and corresponding sampling of first 30 job arriving events in an episode, could be seen in Figure 3.



(a) Bernoulli pattern probability (b) Bernoulli pattern sampling

(c) Uniform pattern probability (d) Uniform pattern sampling

(e) Beta pattern probability (f) Beta pattern sampling

Figure 3. Job arriving pattern probability and pattern sampling

B. Rule-based Baseline Resource Managers

To compare with the effectiveness and justify the necessity of the proposed reinforcement learning based approaches, we put forward rule-based resource management approaches as the baseline models. These different rule-based resource management methods are designed in purpose of expanding baseline solution generality. During designing, we at best effort design competitive ruled-based solutions that utilize available information reasonably and at its utmost; meanwhile keeping variance and generality of solutions in mind.

The rule-based approaches are shown as follow:

- *Random action (Random)*: Both the target job when multiple jobs are awaiting scheduling and the scheduling action for this job are selected randomly.
- *Smallest first-P (SF-P)*: When multiple jobs are awaiting scheduling, the job with smallest computing capacity requirement will be selected. This scheduler will examine cluster utilization percentage in the nearest past 100 time slices, and select the one with the lowest average percentage as the destination for the target job.
- *Largest first-P (LF-P)*: Same as previous, except that the job with largest requirement in computing capacity is selected among multiple awaiting jobs.
- *Smallest first-E (SF-E)*: $SF\text{-}E$ is the same with $SF\text{-}P$ in selecting target job among multiple awaiting ones. But it will examine cluster resource utilization in the nearest past 100 time slices, and select the one with the largest average number of available executors as the destination for the target job.
- *Largest first-E (LF-E)*: Same as previous, except that the job with largest requirement in computing capacity is selected among multiple awaiting jobs.

C. Evaluation Metrics

Whenever evaluating approaches, all participating approaches will be tested for 50 independent testing episodes. Each testing episode is independent in the sense that its workload consisting of 500 jobs is entirely randomly generated following designated job arriving pattern. However, in each testing episode, this same workload of 500 jobs are submitted to all approaches. To enable performance comparison, we present multiple evaluation metrics as follow.

Firstly, as previously stated, there are two major performance factors we take into consideration as shown below.

- *TMDL*: Total occurrence of missing temporal deadline events in all clusters of the overall computing resource during one episode, with respect to the resource management approach.
- *AJDR*: Average job delay ratio is defined as the average job running overhead percentage for all 500 jobs in one episode with respect to certain resource management approach. Specifically, it is defined as:

$$AJDR = \sum_{i=1}^{J}\left(\frac{100 \cdot \sum_{j=1}^{N_i}\left(\frac{AR_{ij}}{OR_{ij}} - 1\right)}{N_i}\right)/J \tag{8}$$

where $J = 500$ is the total number of jobs in one episode. N_i is the total repeating runtime of job i, it is 1 for non-streaming application and larger than 1 for streaming applications. AR_{ij} and OR_{ij} are the actual

Authorized licensed use limited to: China University of Petroleum. Downloaded on December 20,2020 at 13:24:02 UTC from IEEE Xplore. Restrictions apply.

running time and optimal expected running time of job i in its j-th running, respectively.

Then, besides these two direct metrics, to evaluate system performance in a more integrated and well-rounded way with considering both performance metrics as mentioned before, we include one more quantitative measurement and four more comparative methods.

1) Quantitative Measurement: The quantitative measurement is presented in Eqn. 9. It is designed in purpose of concisely evaluating both major performance metrics TMDL and AJDR in combination. With the reciprocal operation, the eventual $Eval_{app}$ can be interpreted as an evaluation score where higher score implies better performance, with 0 being the lower bound. The weights in linear combination are chosen considering data scales in separate metrics.

$$Eval_{app} = (0.02 \cdot \text{TMDL} + \text{AJDR})^{-1} \tag{9}$$

2) Comparative Methods: Comparative methods will be used in different configurations for comparing proposed RL approaches with baseline approaches (the best of candidates is chosen) in 50 testing episodes of workloads. The methods are constructed based on deciding *win*(1), *lose*(0), and optionally, *even*(0.5) for proposed RL approach in each testing episode. The sum of which is then multiplied by 2 to convert into a 100 basis. At all times, $Score_{RL} + Score_{Base} = 100$, with a winning-for-all-rounds RL approach during testing scored at 100. Difference in score definition decides its strictness.

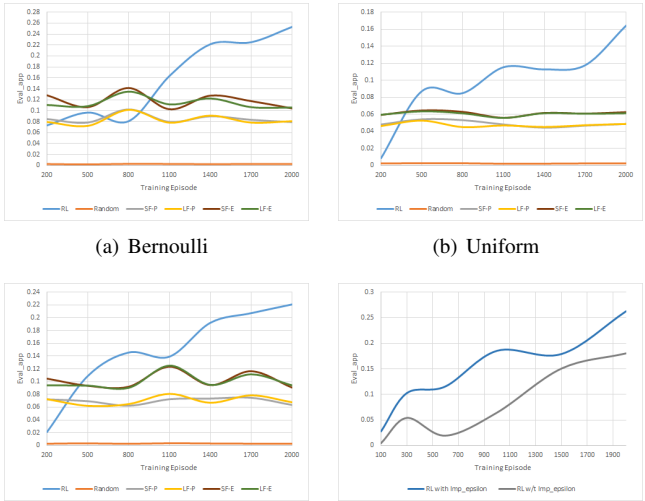
- Score-A: If RL approach outperforms the baseline in **both** metrics in a testing episode, it scores 1, otherwise 0. It is a very strict evaluation standard for RL approach, in purpose of showing absolute dominant percentage of RL approach over baseline.
- Score-B: Same as Score-A, but additionally adds the 'even' case, that is, the RL approach receives 0.5 if it outperforms in only one of either metrics.
- Score-C: RL receives 1 if its evaluation $Eval_{app}$ is higher than that of the baseline solution, calculated by Eqn. 9. Otherwise, it receives 0.
- Score-D: The percentage changes respectively in TMDL and AJDR from baseline to RL approach are computed and added together. If the overall percentage change is negative (thus implies improved performance for RL approach), RL gets 1, otherwise 0.

D. Performance Comparison

In this section, we present performance comparison of RL approach with multiple baselines for all job arriving patterns. The RL approach is constructed by description in Section IV, with employing all modification strategies mentioned in Section IV-C (SF-E as baseline in improved ϵ-greedy method). Each RL approach is trained for 2000 episodes.

For each job arriving pattern, we present performance comparison of RL with other baseline approaches along different training episodes. Following by quantitative metric and score comparison of RL at final training episode with the best baseline approach.

Furthermore, we present the comparison of RL approaches with and without utilization of our improved ϵ-greedy strategy in reinforcement learning process. We also present the result of applying the obtained RL approach to workloads with intentionally varied computing capability requirement statistics, which demonstrates the generality of our obtained RL resource management approach.



(a) Bernoulli (b) Uniform

(c) Beta (d) RL comparison

Figure 4. (a-c) Performance comparison of RL approach and different baselines for Bernoulli, Uniform and Beta job arriving pattern respectively in different training episodes. (d) Performance comparison of RL approach with and without our improved ϵ-greedy method in different training episodes.

For three job arriving patterns, the performance comparisons of RL approach with different baselines, with respect to training episodes are shown in Figure 4(a), 4(b) and 4(c). When computing $Eval_{app}$ in Figure 4, TMDL and AJDR are averaged respectively over 50 testing episodes. For all job arriving patterns, it is observable that among five baselines, the Random baseline performs the worst; SF-P and LF-P although perform slightly differently, are on average at a similar level; as well, SF-E and LF-E perform at a very similar level with slight differences occasionally.

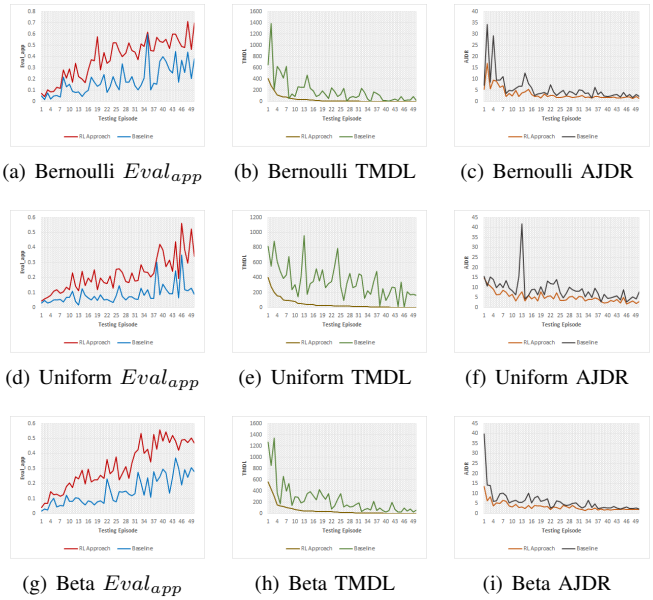
From Figure 4(a), 4(b) and 4(c) we can see that our proposed RL approach for all job arriving patterns gradually improves itself, surpassing all opponents in early training episodes, and eventually achieves big performance advantage over all baseline approaches. This fulfills our desire in achieving good resource management approach.

When further looking into the final model, which is the RL approach at final training episode, we first select its best opponent. By examining the overall performance of all baseline approaches, in all three job arriving patterns, SF-E

Authorized licensed use limited to: China University of Petroleum. Downloaded on December 20,2020 at 13:24:02 UTC from IEEE Xplore. Restrictions apply.

remains performing as the best baseline, it is thus selected for pairwise comparison with final RL approach for all three job arriving patterns.

The comparison of corresponding RL approach (at final training episode) and best baseline SF-E in 50 testing episodes for three job arriving patterns can be found in Figures 5. Specifically, for Bernoulli pattern: 5(a), 5(b), 5(c); for Uniform pattern: 5(d), 5(e), 5(f); and for Beta pattern: 5(g), 5(h), 5(i), which are all related to $Eval_{app}$, TMDL and AJDR metrics, respectively. It is worth mentioning that for $Eval_{app}$ metric, the **higher** value means better performance. Whereas for the later two metrics TMDL and AJDR, the **lower** means the better performance. It is apparent that our final RL approach consistently performs very well in all three job arriving patterns, and outperforms SF-E in all three metrics with significant difference.



(a) Bernoulli $Eval_{app}$ (b) Bernoulli TMDL (c) Bernoulli AJDR

(d) Uniform $Eval_{app}$ (e) Uniform TMDL (f) Uniform AJDR

(g) Beta $Eval_{app}$ (h) Beta TMDL (i) Beta AJDR

Figure 5. Comparison of RL (at final training episode) with the best baseline (SF-E) for $Eval_{app}$, TMDL and AJDR metrics in different job arriving patterns. Graphs are showing for 50 testing episodes used for comparison, sorted by RL TMDL in convenience of viewing. Three rows correspond to *Bernoulli*, *Uniform* and *Beta*, respectively. Three columns correspond to $Eval_{app}$, TMDL and AJDR, respectively. For $Eval_{app}$, the **higher** the better. For TMDL and AJDR, the **lower** the better.

The average statistics of aforementioned 50 testing episodes with respect to all three job arriving patterns can be found in Table II:

Bernoulli pattern: For TMDL, the average belongs to RL and Baseline are 35.04 and 189.34. RL approach achieves to reduce TMDL by a significant ratio of 5.40. It also reduces AJDR by ratio 1.78. For four scoring metrics, it suffices to say that even for Score-A, the most strict standard for RL, it achieves 98 out of 100. That is, according to testing, it can dominant baseline in both TMDL and AJDR simultaneously with approximately 98 percentages probability. Other scores are even higher.

Uniform pattern: For TMDL, RL approach achieves to reduce TMDL by a significant ratio of 7.55, it also receives reduction in AJDR by 2.08. For four scoring metrics, the scores keep showing RL approach as dominant solution, with Score-A, the most strict one, being 96 out of 100.

Beta pattern: Once again, RL approach achieves to reduce TMDL by a good ratio of 4.40, and receives reduction in AJDR by 1.79. For four scoring metrics relating to 50 testing episodes, all scores are identically 100.

By examining performance comparison, we are therefore confident to consider achieved RL approach as a good resource management approach for designated scenario and it becomes a much better resource management solution than aforementioned rule-based baselines.

After showing major performance comparison, we would like to supplement additional experiments. Firstly, we want to verify influence of the improved ϵ-greedy strategy described in Section IV-C2. We use Bernoulli process pattern and compare the performance of obtained RL approach with and without the improved ϵ-greedy method as shown in Figure 4(d). The improved ϵ-greedy method (with SF-E as baseline) indeed improves both the effectiveness and efficiency of RL approach. The RL approach with improved ϵ-greedy strategy gains much better final performance. And even at middle stage of RL process, it already achieves comparable performance to the one at final episode without the improved ϵ-greedy method. This justifies the necessity in utilizing proposed improved ϵ-greedy strategy.

Secondly, although our RL resource management approach demonstrates good generality by being suitable to randomly generated hybrid workloads in multiple job arriving patterns, we intend to further apply stresses to the obtained RL approach by the following experiments:

For an already obtained RL model, we vary several statistical characteristics during workload generation in testing. The newly randomly generated workloads, although following the same job arriving pattern as in training, show significantly different computing capability desires than ones during training. This consequently brings challenges to RL approach generality. Using uniform distribution job arriving pattern, we generate two new testing workload patterns, "eased" uniform and "stressed" uniform, where randomly generated jobs have statistically **less** (**more**) computing capability requirement than original ones respectively. We test obtained final RL approach by original uniform pattern and SF-E against new workload patterns. The result is shown in Table III.

As expected, comparing to original version, it can be seen that the performance of RL and SF-E in the sense of TMDL and AJDR both simultaneously improve (deteriorate) in the "eased" ("stressed") version, respectively, due to changes in computing capability requirement statistics. However, regardless of the pattern change, the originally obtained RL

Authorized licensed use limited to: China University of Petroleum. Downloaded on December 20,2020 at 13:24:02 UTC from IEEE Xplore. Restrictions apply.

Table II
PERFORMANCE COMPARISON OF RL APPROACH AND SF-E FOR THREE DIFFERENT ARRIVING PATTERNS

Metric	Bernoulli arriving pattern			Uniform arriving pattern			Beta arriving pattern		
	RL Approach	Baseline	Ratio	RL Approach	Baseline	Ratio	RL Approach	Baseline	Ratio
TMDL	35.04	189.34	5.40	46.34	349.9	7.55	56.52	248.86	4.40
AJDR	3.24	5.78	1.78	5.18	10.80	2.08	3.40	6.07	1.79
Score-A	98	2	49	96	4	24	100	0	∞
Score-B	99	1	99	98	2	49	100	0	∞
Score-C	100	0	∞	100	0	∞	100	0	∞
Score-D	100	0	∞	100	0	∞	100	0	∞

approach remains performing very well in either cases. It means that the generality of our RL approach is further fortified. And this concludes our experiment section.

Table III
PERFORMANCE COMPARISON OF RL AND SF-E FOR UNIFORM ARRIVING PATTERN WITH EASED AND STRESSED WORKLOADS

Metric	Eased workloads			Stressed workloads		
	RL	Baseline	Ratio	RL	Baseline	Ratio
TMDL	16.36	140.52	8.59	350.44	1027.86	2.93
AJDR	3.12	4.87	1.56	13.76	19.96	1.45
Score-A	96	4	24	92	8	11.5
Score-B	98	2	49	95	5	19
Score-C	100	0	∞	94	6	15.67
Score-D	100	0	∞	96	4	24

VI. CONCLUSION

In this paper, we analyze reinforcement learning based approach for resource management of hybrid workloads in large-scale distributed big data computing environment. By comparing its performance with multiple baseline solutions in various job arriving patterns, as well as comparing to other RL approach version, we are able to demonstrate the effectiveness and generality of obtained RL approach. It is observed during testing that the TMDL metric is improved by up to 7.55 times, while the AJDR metric is improved by up to 2.08 times. In conclusion, we successfully obtain better RL based resource management approaches for hybrid workloads in distributed big data computing environment.

Acknowledgements: This work was supported in part by NSF-1836881.

REFERENCES

[1] Deepmind, "Deepmind," https://deepmind.com/.
[2] D. Silver et al., "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
[3] D. Silver et al, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.
[4] G. M. J.-B. C. Chaslot, "Monte-carlo tree search," Ph.D. dissertation, Maastricht University, 2010.
[5] G. Liao, K. Datta, and T. L. Willke, "Gunther: Search-based auto-tuning of mapreduce," in *European Conference on Parallel Processing*. Springer, 2013, pp. 406–419.
[6] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.
[7] L. Wang, S. Lu, X. Fei, A. Chebotko, H. V. Bryant, and J. L. Ram, "Atomicity and provenance support for pipelined scientific workflows," *Future Generation Computer Systems*, vol. 25, no. 5, pp. 568–576, 2009.
[8] Z. Liu, H. Zhang, and L. Wang, "Hierarchical spark: A multi-cluster big data computing framework," in *Cloud Computing (CLOUD), 2017 IEEE 10th International Conference on*. IEEE, 2017, pp. 90–97.
[9] J. Taheri, A. Y. Zomaya, P. Bouvry, and S. U. Khan, "Hopfield neural network for simultaneous job scheduling and data replication in grids," *Future Generation Computer Systems*, vol. 29, no. 8, pp. 1885–1900, 2013.
[10] N. Yigitbasi, T. L. Willke, G. Liao, and D. Epema, "Towards machine learning-based auto-tuning of mapreduce," in *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE, 2013, pp. 11–20.
[11] G. Wang, J. Xu, and B. He, "A novel method for tuning configuration parameters of spark based on machine learning," in *High Performance Computing and Communications*. IEEE, 2016, pp. 586–593.
[12] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang, "Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics." in *NSDI*, vol. 2, 2017, pp. 4–2.
[13] E. Barrett, E. Howley, and J. Duggan, "Applying reinforcement learning towards automating resource allocation and application scalability in the cloud," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 12, pp. 1656–1674, 2013.
[14] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. ACM, 2016, pp. 50–56.
[15] J. Wu, X. Xu, P. Zhang, and C. Liu, "A novel multi-agent reinforcement learning approach for job scheduling in grid computing," *Future Generation Computer Systems*, vol. 27, no. 5, pp. 430–439, 2011.
[16] T. Li, Z. Xu, J. Tang, and Y. Wang, "Model-free control for distributed stream data processing using deep reinforcement learning," *Proceedings of the VLDB Endowment*, vol. 11, no. 6, pp. 705–718, 2018.
[17] V. K. Vavilapalli et al., "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 5.
[18] N. Metropolis, "Monte carlo method," *From Cardinals to Chaos: Reflection on the Life and Legacy of Stanislaw Ulam*, p. 125, 1989.
[19] A. G. Barto, "Temporal difference learning," *Scholarpedia*, vol. 2, no. 11, p. 1604, 2007.

Authorized licensed use limited to: China University of Petroleum. Downloaded on December 20,2020 at 13:24:02 UTC from IEEE Xplore. Restrictions apply.