Leveraging Distributional Reinforcement Learning for Performance Optimization of Spark Job Scheduling in Cloud Environment

Sumit Kumar[1][a], Vishnu Prasad Verma[2][b] and Santosh Kumar[2][c]
[1]*Deptt. of CSE, UIET, Maharshi Dayanand University, Rohtak, Haryana, 124001, India*
[2]*Deptt. of CSE, IIIT Naya Raipur, Raipur, Chhattisgarh, 493661, India*

Keywords: Apache Spark, Distributed Computing, Job Scheduling, Distributional Deep Reinforcement Learning, Big Data, Performance Optimization, Cloud Computing.

Abstract: Apache Spark is extensively utilized for processing massive data sets in fields like big data analytics and machine learning. However, its performance is closely tied to how jobs are scheduled, and resources are allocated, especially in dynamic cloud settings. The default Spark scheduler can sometimes struggle to efficiently manage resources in diverse clusters, leading to delays, higher costs, and slower job completion. This research introduces a new approach for optimizing Spark job scheduling using Distributional Deep Reinforcement Learning (DDRL). Unlike other methods focusing on average performance, DDRL employs a Rainbow Deep Q-Network to model the entire range of possible outcomes. This allows the system to better understand the risks and uncertainties associated with scheduling decisions. Key features of our approach include multi-step learning for long-term planning, techniques to encourage exploration and exploitation, and strategies for adapting to rapidly changing workloads. Our experiments show that the proposed framework significantly improves Spark's performance. It achieves faster job scheduling, better resource utilization, and lower overall costs than existing methods. These results demonstrate the potential of DDRL as a robust and scalable solution for enhancing Spark scheduling in dynamic cloud environments

1 INTRODUCTION

Big data processing professionals rely heavily on distributed computing frameworks like Apache Spark. Efficient job scheduling is crucial to harnessing Spark's full potential in cloud environments. Although foundational, traditional algorithms like First-In-First-Out (FIFO) and Fair Scheduling struggle with dynamic workloads and heterogeneous clusters, leading to suboptimal resource utilization and longer job completion times (Gandomi et al., 2019).

Key factors influencing Spark job scheduling include data affinity, resource heterogeneity, Service Level Agreement (SLA) objectives, data skew, inter-job dependencies, and workload nature. Data affinity affects execution and transmission times, requiring sophisticated algorithms to minimize completion time (Zhang et al., 2022). Resource heterogeneity, including hardware capabilities and specific task needs, must be considered for optimization, as seen in systems like RUPAM (Islam et al., 2021a). SLA objectives, such as cost minimization and performance improvement, are critical, with RL models helping to meet these in cloud-deployed Spark clusters (Islam et al., 2021b). Data skew and deadline constraints complicate scheduling, necessitating algorithms to manage rental costs and meet deadlines (Gu et al., 2020). If unmanaged, inter-job dependencies can lead to job failures, highlighting tools like the Wing dependency profiler (Cheng et al., 2018). The workload nature, including streaming data, demands adaptive scheduling approaches like A-scheduler (Li et al., 2020) (Xu et al., 2018). Hybrid cloud environments add complexity, requiring algorithms to optimize costs and meet deadlines through various VM Average pricing (Cheng et al., 2017). Efficient scheduling must also minimize communication costs associated with cross-node data transfers, as seen in locality-aware task scheduling algorithms (Fu et al., 2020). These factors underscore the need for advanced and adaptive algorithms to optimize job scheduling in Spark.

[a] https://orcid.org/0009-0000-8676-4017
[b] https://orcid.org/0009-0009-5831-7258
[c] https://orcid.org/0000-0003-2264-9014

Kumar, S., Verma, V. P. and Kumar, S.
Leveraging Distributional Reinforcement Learning for Performance Optimization of Spark Job Scheduling in Cloud Environment.
DOI: 10.5220/0013582600004664
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 3rd International Conference on Futuristic Technology (INCOFT 2025) - Volume 1*, pages 615-623
ISBN: 978-989-758-763-4
Proceedings Copyright © 2025 by SCITEPRESS – Science and Technology Publications, Lda.

However, to resolve the issues mentioned above, the rise of Deep learning-based solutions is proliferating due to emerging applications for futuristic big data processing and leveraging new possibilities for intelligent job scheduling in cloud computing (Cao et al., 2024). Among these, distributional deep reinforcement learning (DDRL) stands out by learning the entire distribution of possible computed rewards, providing a deeper understanding for agents to make correct decisions to optimize job scheduling as compared to traditional Deep Reinforcement Learning (DRL), which emphasizes maximizing predicted maximum return (Li, 2023) for completion of jobs in cloud computing which leverages to solve severe issues influencing Spark job scheduling. DDRL is beneficial in cloud environments where dynamically increasing the size of jobs, execution times, and requirement of distributional resource availability in dynamic changes.

In this work, a novel framework is proposed to utilize the potential of using a DDRL technique to optimize Spark job scheduling efficiently. By integrating the DDRL technique with Spark's scheduling mechanisms, the primary objective of this work is to create a strategy with superior adaptability and robustness, dynamically adjusting to evolving workloads and resource conditions to achieve high performance. Moreover, the primary goal extends to minimizing overall execution time on jobs and maximizing resource utilization to ensure job deadlines are met.

The remainder of the paper is structured as follows: Section 2 provides a related work, Section 3 outlines the proposed methodology, Section 4 offers a detailed analysis of experimental setup, Section 5 analyzes the result and performance comparisons. Finally, Section 6 provides concluding remarks with future scopes.

2 RELATED WORKS

Efficiently scheduling jobs in Apache Spark is crucial for maximizing resource utilization and performance, especially in cloud environments. Researchers have explored various approaches to tackle challenges like ensuring data is processed where it's stored, handling fluctuating workloads, and dealing with diverse computing resources. This section examines Spark's different job scheduling techniques, including traditional and more recent methods. By reviewing these approaches, we can identify areas for improvement and understand the motivations behind the research presented in this work.

The study in (Cheng et al., 2017) integrates a dynamic batching technique with an A-scheduler, which uses an expert fuzzy control mechanism to adjust the length of each batch interval according to the time-varying streaming workload and system processing rate. The research in (Fu et al., 2020) introduces a new task scheduling algorithm for Spark applications that focuses on reducing data transfer between nodes and racks, which can slow down performance and cause network congestion. Using a bipartite graph model, the algorithm aims to find the best way to schedule both map and reduce tasks, ensuring that data locality is optimized for better performance.

The study in (Venkataraman et al., 2016) proposes a performance prediction framework named Ernest that accurately predicts the running time of analytics jobs on specified hardware configurations, focusing on minimizing the training data required for accurate predictions. The paper (Alipourfard et al., 2017) introduces CherryPick, a system that employs Bayesian Optimization to construct performance models for diverse applications. These models are sufficiently precise to identify the best or nearly optimal configuration with minimal test runs. CherryPick characterizes each cloud configuration using parameters such as the number of VMs, the number of CPUs, the speed per CPU core, RAM per core, disk quantity, disk speed, and the VM's network capacity. It utilizes the Gaussian Process as a prior in Bayesian Optimization, recognized for its effectiveness as a surrogate model and computational feasibility for large-scale problems.

The paper (Yadwadkar et al., 2017) presents PARIS, a data-driven system designed to accurately estimate workload performance and associated costs for various user-specified metrics across multiple cloud services operators. PARIS employs a mixed framework that combines offline and online data gathering and processing. It surpasses advanced baselines, such as collaborative filtering and linear interpolation models, by reducing forecast at execution time errors by a factor of 4 for specific workloads on AWS and Azure. This results in a 45% reduction in user costs while maintaining performance. Spark on YARN lacks energy efficiency and deadline control, to overcome this problem the study (Shabestari and Navimipour, 2023) introduces a system designed to reduce energy usage in Spark running on YARN within diverse clusters, while also ensuring deadlines are met. It achieves this with a deadline-aware model, locality-aware executor assignment, and a heuristic task scheduler.

Existing research has made progress in addressing challenges like dynamic resource allocation and data skew in Spark scheduling. However, effectively managing resources in diverse cloud environments remains an ongoing challenge. Distributional Deep Reinforcement Learning (DDRL) offers a potential solution by considering the full range of possible outcomes, not just average rewards. This study extends previous work by developing a framework integrating DDRL with Spark's scheduling mechanisms. The goal is to create a more robust and adaptive system that overcomes limitations identified in prior research.

3 METHODOLOGY

3.1 Problem Formulation

Let M stand for the overall number of virtual machines (VMs) in the cluster, and L signify the overall number of jobs. The scheduler allocates executors on VMs based on resource capacities and job demands. Each Job's resource demand in CPU and memory is modeled as a multi-dimensional box. The resource constraints are:

$$\sum_{a \in E} (p_{\text{cpu},a} \times y_{aj}) \le r_{\text{cpu},j} \quad \forall j \in S \tag{1}$$

$$\sum_{a \in E} (p_{\text{mem},a} \times y_{aj}) \le r_{\text{mem},j} \quad \forall j \in S \tag{2}$$

An executor is placed for task P in VM j if the binary decision variable y_{aj} is 1; otherwise, it is 0. Executors cannot use combined resources from multiple VMs:

$$\sum_{j \in S} y_{aj} = 1 \quad \forall a \in E \tag{3}$$

The scheduler's total cost is:

$$\text{Cost}_{\text{aggregate}} = \sum_{j \in S} (r_{\text{price},j} \times r_{T,j}) \tag{4}$$

The average job completion time is:

$$\text{AvgT} = \frac{\sum_{b \in J} T_{\text{job},b}}{L} \tag{5}$$

The optimization goal is to minimize:

$$\gamma \times \text{Cost} + (1 - \gamma) \times \text{AvgT} \tag{6}$$

where $\gamma \in [0, 1]$ is a user-defined parameter.

This NP-hard, mixed-integer linear programming (MILP) problem becomes infeasible to solve optimally as the number of jobs, executors, and cluster size increases. While capable of handling large data, heuristic algorithms often lack generalizability and fail to capture workload and cluster architecture nuances.

Our solution involves a cloud-based scheduling system for batch workload balancing. It dynamically assigns incoming batch jobs to clusters with changing resources. Batch jobs $\{job1, job2, \ldots, jobN\}$ arrive sequentially, each consisting of multiple parallel tasks $\{taskj, 1, taskj, 2, \ldots, taskj, O\}$. The Task instances to be executed can be spread across multiple VMs, starting with matching CPU and memory needs.

The scheduling agent plans based on current resource needs and cluster usage. The instance creation process verifies if a VM meets SLA constraints:

$$\text{InsSuccess} = \begin{cases} \text{ins}_{\text{cpu}} \le \text{r}_{\text{cpu}}^{\text{Avl}} \\ \text{ins}_{\text{mem}} \le \text{r}_{\text{mem}}^{\text{Avl}} \end{cases} \tag{7}$$

Table 1: Symbolic Annotations and Their Meaning

Notation	Meaning
M	Total virtual machines in a cluster
L	Total user-requested jobs
O	Overall tasks in ongoing job
P	Quantity of instances required for the task at the moment
α	VM indices, $\alpha = \{1, 2, \ldots, M\}$
β	Job indices, $\beta = \{1, 2, \ldots, L\}$
γ	Task indices in the job, $\gamma = \{1, 2, \ldots, O\}$
δ	The task's instantiation identifiers, $\delta = \{1, 2, \ldots, P\}$
r_{cpu}	Max CPU capacity available in VM r_i, $i \in \alpha$
r_{mem}	Max memory capacity available in VM r_i, $i \in \alpha$
r_{cpuAvl}	Available CPU of VM r_i, $i \in \alpha$
r_{memAvl}	Available memory of VM r_i, $i \in \alpha$
j_{obj}	j-th job within group of jobs, $j \in \beta$
$task_{j,o}$	o-th task in within j-th job, $o \in \gamma$
$insNum_{j,o}$	The quantity of instance in $task_{j,o}$
$insSucc_{j,o}$	Instances created in $task_{j,o}$
$ins_{\text{cpu},p}$	Demand for CPU resources in an instance, $p \in \delta$
$ins_{\text{mem},p}$	Demand for memory resources in an instance, $p \in \delta$

3.2 Optimizing Spark Scheduling with Distributional Reinforcement Learning

Let $\mathcal{S}$ denote the state space of an Apache Spark job in a cloud environment, where each state $s \in \mathcal{S}$ represents a configuration of allocated resources, including CPU cores, memory units, and potentially network bandwidth or disk space. The objective is to

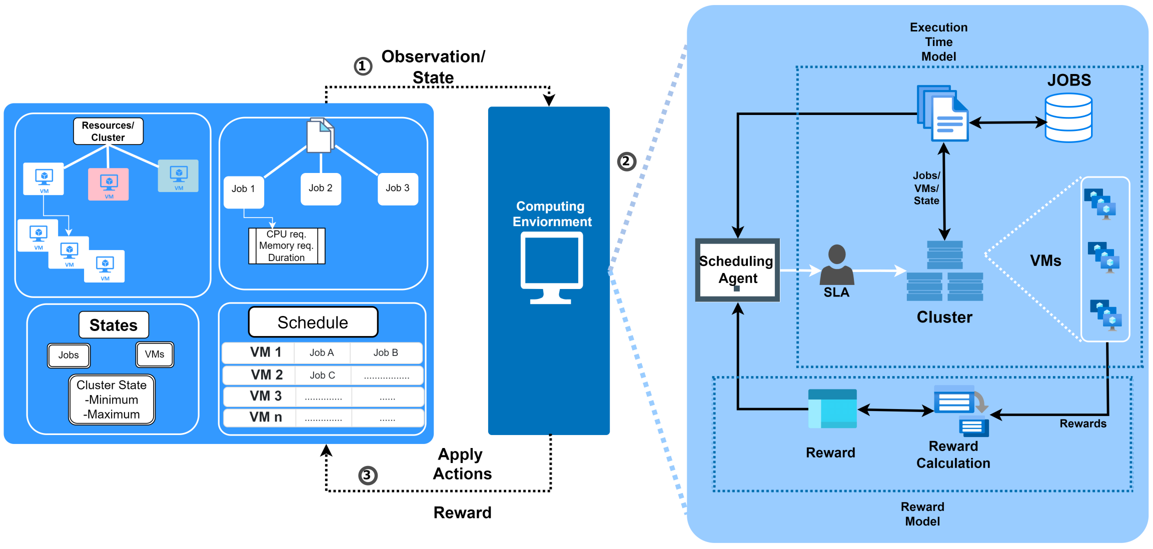

Figure 1: Proposed DRL Framework

determine an optimal policy, π^*, that specifies the resource allocation strategy to minimize the expected cost while accounting for the inherent uncertainty in job execution times. These times are influenced by data size and distribution, resource availability, cluster heterogeneity, and shuffle operations. Traditional scheduling algorithms often rely on average resource utilization metrics, which may not effectively capture this variability.

3.2.1 Policy Optimization

The objective to find the policy π^* that minimizes the expected cost is determined by:

$$\pi^* = \arg\min_{\pi} \mathbb{E}_{s\sim S}\left[\mathbb{E}_{a\sim\pi(s)}\left[\mathbb{E}_{T\sim P(T|s,a)}\left[C(s,a,T)\right]\right]\right], \tag{8}$$

The policy π^* is optimized using DDRL. Instead of predicting average rewards, DDRL models the entire reward distribution, accounting for variability and risks in resource allocation decisions. DDRL optimizes CPU, memory, and network bandwidth allocation in Spark, considering average execution time and outliers. This results in robust resource management that adapts to workload fluctuations and cloud environment uncertainties.

3.2.2 Model Formulation

The success of DDRL for multi-objective optimization depends on the state space, action space, and reward function. These components for batch task load balancing scheduling are shown in Fig: 1 and are specified as follows:

Temporal State: The current scheduling environment is a two-part, one-dimensional vector. The first component of this vector represents cluster resource load:

$$s1 = [l_{Cpu1}, l_{Mem1}, \ldots, l_{CpuN}, l_{MemN}, l_{Cpu\_avg}, l_{Mem\_avg}] \tag{9}$$

Where $l_{Cpu1}, \ldots, l_{CpuN}$ are the CPU loads and $l_{Mem1}, \ldots, l_{MemN}$ are the memory loads of all VMs Cloud infrastructure. l_{Cpu_avg} and l_{Mem_avg} the mean workloads. The machine load is normalized between 0 and 1 for better precision and efficacy.

The second component of the vector represents a task instance's resource needs:

$$s_2 = [taskType, insCpu, insMem, insNum] \tag{10}$$

where $insCpu$ and $insMem$ are the CPU and memory needs, $taskType$ is the task index, and $insNum$ is the overall number of instances. The overall state of the scheduling environment is:

$$s = s_1 + s_2 \tag{11}$$

Action Space: A virtual machine is chosen by the scheduling agent for instance creation. Each of the M distinct actions corresponds to one of the M machines in the cluster:

$$A = \{a \mid a \in \{1, 2, \ldots, M\}\} \tag{12}$$

Reinforcement Signal: The RL model's training relies on the Reinforcement Signal, also known as reward function r, with signals indicating the quality of the agent's actions. Positive signals/rewards signify beneficial actions, while negative signals/rewards denote detrimental ones. SLA limitations are analyzed

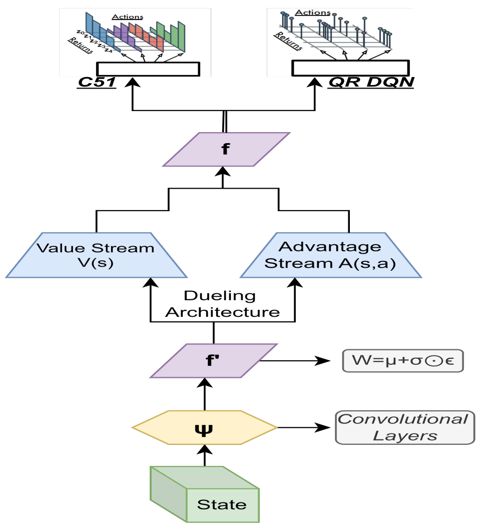

Figure 2: Network architectures for Rainbow-DQN distributional RL algorithms

first; if criteria are breached, the VM fails to meet instance creation requirements, resulting in $r = -1$. The reward is then adjusted based on the cluster's load balancing variances, calculated to distinguish between average cluster load and individual cluster load VM scheduling for the current instance. Load differences in CPU and memory are represented by $dvCpu$ and $dvMem$.

$$dvCpu = lCpu_{avg} - lCpu_i \tag{13}$$

$$dvMem = lMem_{avg} - lMem_i \tag{14}$$

Load balancing aims to keep all VM load as close to the cluster's average load as possible. A positive divergence value indicates that the chosen virtual machine's load surpasses the cluster's average load, hence such a machine should not be picked. When SLA restrictions are satisfied, the reward mechanism is defined as follows:

$$r = \begin{cases} 1, & \text{if } dvCpu \geq 0 \text{ and } dvMem \geq 0 \\ \frac{dvCpu+dvMem}{2}, & \text{otherwise} \end{cases} \tag{15}$$

3.3 Rainbow DQN: An Integrated Approach to Deep Reinforcement Learning

In Spark job scheduling, the RL environment simulates real workloads and includes cluster resource constraints in the state space. A DRL agent's actions, such as placing an executor, yield immediate rewards and update the state based on VM and task changes, as illustrated in figure 1. Agents aim to complete all executors and collect episodic rewards by managing resource availability and demand constraints. Rainbow DQN enhances deep reinforcement learning by combining Double Q-learning, Prioritized Experience Replay, Dueling Network Architectures, Multi-step Learning, DRL, and Noisy Nets into one framework, addressing DQN limitations and improving performance and stability, as shown in figure 2.

Double Q-learning

Double Q-learning reduces overestimation bias by separating action selection and assessment. Rainbow DQN maintains two networks: the online network (θ_1) for action selection and the target network (θ_2) for assessment. The update rule is:

$$Z(y,a;\theta_1) \leftarrow Z(y,a;\theta_1) + \alpha \Big[r + \gamma Z(y', \arg\max_{a'} Z(y',a';\theta_1);\theta_2) - Z(y,a;\theta_1) \Big] \tag{16}$$

Prioritized Experience Replay

Prioritized Experience Replay (PER) assigns a priority to each experience based on the temporal-difference (TD) error. Experiences with larger TD errors are more likely to be sampled. The sampling probability for an experience j is:

$$P(j) = \frac{p_j^{\beta}}{\sum_k p_k^{\beta}} \tag{17}$$

where p_j is the priority of experience j and β controls the prioritization level. Priority p_j is adjusted by the absolute TD error and a constant ε:

$$p_j = |\delta_j| + \varepsilon \tag{18}$$

Dueling Network Architectures

The Dueling Network Architecture separates the representation of state values from action advantages. The Q-value is divided into two parts: the state value function $U(x)$ and the advantage function $A(x,b)$. The Q-values are computed as:

$$Q(x,b;\theta_1,\beta,G) = U(x;\theta_1,G) + \Big(A(x,b;\theta_1,\beta) - \frac{1}{|B|}\sum_{b'} A(x,b';\theta_1,\beta)\Big) \tag{19}$$

Multi-step Learning

Multi-step learning improves the learning process by considering cumulative rewards over multiple steps. The k-step return $R_t^{(k)}$ is:

$$R_t^{(k)} = \sum_{m=0}^{k-1} G^m r_{t+m} + G^k U(x_{t+k}) \tag{20}$$

The Q-value update rule is:

$$Z(y_t, b_t) \leftarrow Z(y_t, b_t) + \alpha \left[R_t^{(k)} - Z(y_t, b_t) \right] \tag{21}$$

Distributional Reinforcement Learning

DRL models the distribution of returns. Rainbow DQN uses Categorical DQN (C51) to approximate the return distribution:

$$Z(x, b) = \sum_{i=1}^{N} p_i \delta_{z_i} \tag{22}$$

where z_i are support points and p_i are probabilities.

Noisy Nets

Noisy Nets introduce noise into network parameters to facilitate exploration and are defined as:

$$W = \mu + \sigma \odot \varepsilon \tag{23}$$

Where W are noisy weights, μ and σ are learnable parameters, and ε is a noise vector sampled from a standard Gaussian distribution.

3.4 Designing the Agent by Integrating the Enhancements

Rainbow DQN combines six enhancements for superior performance: Double Q-learning reduces overestimation bias by using two network weights (current and target). Prioritized Experience Replay accelerates learning by sampling critical transitions more frequently. The Dueling Network Architecture splits the output into value and advantage streams for more accurate value estimates. Multi-step Learning incorporates multi-step returns, capturing longer-term dependencies by considering cumulative rewards over multiple steps. Distributional Q-Learning predicts a distribution over returns rather than a single Q-value, providing a richer value function representation. Noisy Networks replace standard layers with noisy layers, enhancing exploration through stochasticity in network parameters.

These enhancements produce a robust, efficient, consistent learning process, improving performance in various RL tasks. Rainbow DQN also includes functions ψ and f. The feature extractor ψ, typically convolutional layers, processes the input state into a lower-dimensional feature representation. The Q-value function f, the final fully connected layers, takes $\psi(x)$ and produces Q-values for each action, incorporating the enhancements of Rainbow DQN.

Table 2: Hyperparameters for Agent and Environment

Specifications	Value
Rfixed	10000
Batch Size	128
Eval. Episodes	10
Policy Opt. Priority (γ)	[0.0, 0.25, 0.50, 0.75, 1.0]
Q-Network Layers	200
Policy Eval. Interval	1000
Epsilon	0.001 or 0.1-1
Iterations	10000
Learning Rate	0.001/0.0009
Optimizer	Adam
Discount Factor	0.9
Job Dur. Inc. for Bad Placement	30%
Collect Steps/Iter. (DQN)	10
Collect Episodes/Iter. (RE)	10
Replay Buffer Size	10000
AvgTmin, AvgTmax	Profiled from Actual Runs of Task
Valid Action Reward	-1 or +1
AvgT, Costmax	Dyn. Calculated
Invalid Action Reward	-200
Target Update Period	100
N-Step Update	3
Alpha, Beta (Prioritized Buf.)	0.6, 0.4

4 EXPERIMENTAL SETUP

DRL was integrated using TensorFlow, employing a deep neural network policy architecture with multiple layers and feature inputs. The training procedure involved iterative episodes with an exploration-exploitation strategy to minimize job execution times and optimize resource usage. Various VM instance types and price models were chosen to train and assess an agent for cost minimization, analogous to an actual cloud setting. The study employed three distinct VM types, differentiated by CPU specifications and memory capacities. The DRL agents were required to have CPU cores and 128 GB of RAM. TensorFlow 2.11.0, TF-Agent 0.16.0, and Python 3.7.8 were installed on the machines. We developed a TF-agent compatible environment to train multiple DRL agents with various objectives; the hyperparameter for agent and environment is shown in table 2. Reward signals were designed to maximize cost efficiency while reducing average job length. This environment, named RB DDRL uses TensorFlow agents and can be expanded for new rewards, continuous states, and additional DRL agents. For the workload, we utilized the BigDataBench (Wang et al., 2014) benchmark suite, selecting three distinct applications as jobs for the cluster: WordCount (CPU-

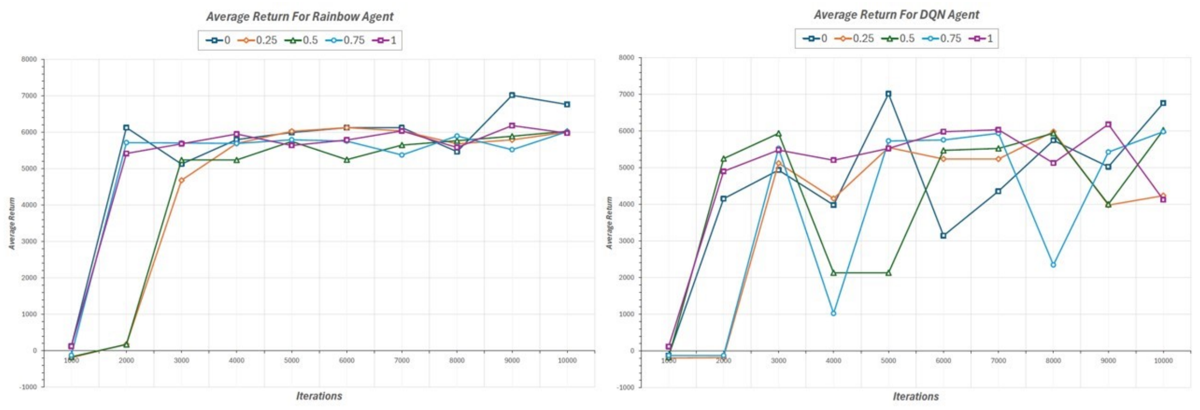

Figure 3: Average return for different γ values of Rainbow DQN and DQN agent

intensive), PageRank (network or I/O-intensive), and Sort (memory-intensive). Job requirements were generated using a uniform distribution, with CPU cores ranging from 1 to 6, memory ranging from 1 to 10 GB, and the total number of executors ranging from 1 to 8.

5 RESULT ANALYSIS

5.1 Convergence and Effectiveness of DRL

Figure 3 shows Rainbow DQN's convergence under normal task arrival patterns in comparison to DQN agent's, which depicts the efficiency of the agent as it very quickly adapted and balanced between exploration and exploitation and achieved convergence, whereas DQN could not converge as efficiently as Rainbow and showed numerous and many deviations and fluctuations. We trained agents with different gamma values over 10,000 iterations to study their impact on Average return and average rewards. Episodic rewards vary based on cluster conditions and task criteria. Figure 5 shows every agent action's step-by-step reward gained for it. The reward pattern demonstrates Rainbow DQN's effectiveness in optimizing immediate and long-term rewards despite occasional drops due to the epsilon-greedy policy, which helped find the most optimal one. Rainbow DQN balances resource allocation and task execution times in Spark scheduling. The policy π^* outperformed traditional heuristics, exceeding theoretical expectations in real-world efficiency. The ease and quickness of convergence and the multiple layers ensure more optimal usage of resources and execution of jobs. The quicker the convergence is acquired, the more efficiently the jobs will be executed, increasing deadline adherence and promoting resource utilization.

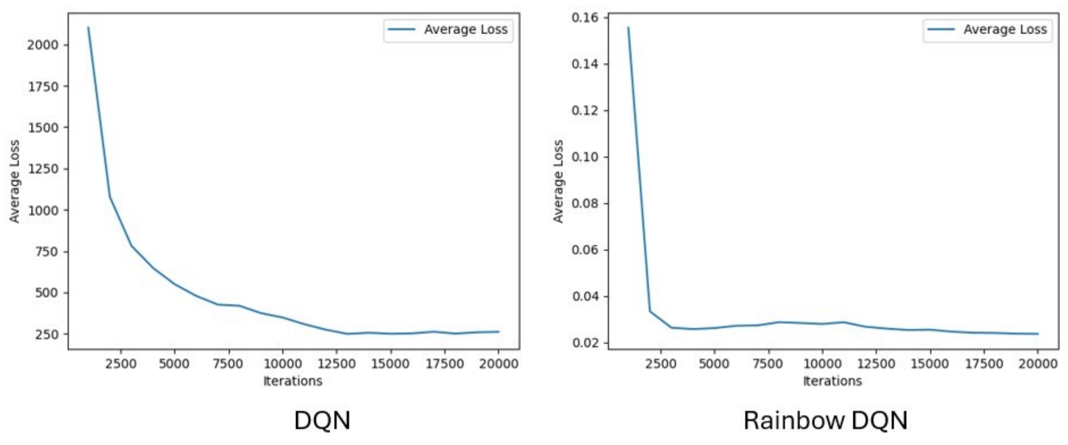

Figure 4: Comparison of Loss in DQN vs Rainbow DQN

5.2 Comparison of Loss in DQN and Rainbow DQN

The optimized policy π^* using Rainbow DQN significantly outperforms traditional scheduling methods DQN, reducing loss as shown in figure 4. This enhances efficiency in execution time, CPU/memory utilization, and job completion times. Rainbow DQN effectively manages execution time uncertainties ($P(T \mid s,a)$) and adapts dynamically to real-world variability, surpassing deterministic approaches. It also shows promising reward results, indicating optimal performance and effective learning from initial negative rewards (Fig: 5).

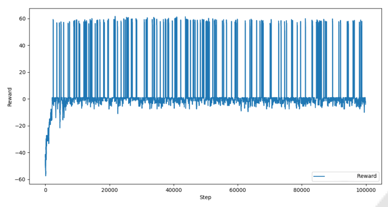

Figure 5: Reward for each step for γ= 1.0

6 CONCLUSION AND FUTURE DIRECTION

This study demonstrates the effectiveness of DRL in enhancing Apache Spark scheduling within cloud environments. By modeling uncertainties in job execution times and dynamically adapting resource allocations, DDRL-based policies significantly improved cost efficiency and performance metrics. The reward pattern highlights Rainbow DQN's ability to optimize both immediate and long-term rewards despite occasional drops due to the epsilon-greedy policy. Experiments showed that DDRL outperformed DQN in minimizing loss and improving resource utilization. The practical implications include cost savings through efficient resource allocation, operational efficiencies via automated management, and scalability to meet evolving computational demands. Leveraging DRL for Spark scheduling offers a promising path toward advancing distributed computing efficiency in cloud environments. Future work will improve algorithm convergence, refine reward functions, handle dynamic workloads, and integrate factors such as network bandwidth and disk I/O. Automated hyperparameter optimization will reduce complexity and enhance DRL performance. Additionally, further exploration of DRL algorithms, deeper integration with cloud-native features, and broader applicability across diverse Spark workloads and infrastructures will be prioritized.

REFERENCES

Alipourfard, O., Liu, H. H., Chen, J., Venkataraman, S., Yu, M., and Zhang, M. (2017). Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 469–482.

Cao, Z., Deng, X., Yue, S., Jiang, P., Ren, J., and Gui, J. (2024). Dependent task offloading in edge computing using gnn and deep reinforcement learning. *IEEE Internet of Things Journal*.

Cheng, D., Chen, Y., Zhou, X., Gmach, D., and Milojicic, D. (2017). Adaptive scheduling of parallel jobs in spark streaming. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pages 1–9. IEEE.

Cheng, D., Zhou, X., Wang, Y., and Jiang, C. (2018). Adaptive scheduling parallel jobs with dynamic batching in spark streaming. *IEEE Transactions on Parallel and Distributed Systems*, 29(12):2672–2685.

Fu, Z., Tang, Z., Yang, L., and Liu, C. (2020). An optimal locality-aware task scheduling algorithm based on bipartite graph modelling for spark applications. *IEEE Transactions on Parallel and Distributed Systems*, 31(10):2406–2420.

Gandomi, A., Reshadi, M., Movaghar, A., and Khademzadeh, A. (2019). Hybsmrp: a hybrid scheduling algorithm in hadoop mapreduce framework. *Journal of Big Data*, 6(1):1–16.

Gu, H., Li, X., and Lu, Z. (2020). Scheduling spark tasks with data skew and deadline constraints. *IEEE Access*, 9:2793–2804.

Islam, M. T., Karunasekera, S., and Buyya, R. (2021a). Performance and cost-efficient spark job scheduling based on deep reinforcement learning in cloud computing environments. *IEEE Transactions on Parallel and Distributed Systems*, 33(7):1695–1710.

Islam, M. T., Wu, H., Karunasekera, S., and Buyya, R. (2021b). Sla-based scheduling of spark jobs in hybrid cloud computing environments. *IEEE Transactions on Computers*, 71(5):1117–1132.

Li, D., Hu, Z., Lai, Z., Zhang, Y., and Lu, K. (2020). Coordinative scheduling of computation and communication in data-parallel systems. *IEEE Transactions on Computers*, 70(12):2182–2197.

Li, S. E. (2023). Deep reinforcement learning. In *Reinforcement learning for sequential decision and optimal control*, pages 365–402. Springer.

Shabestari, F. and Navimipour, N. J. (2023). An energy-aware resource management strategy based on spark and yarn in heterogeneous environments. *IEEE Transactions on Green Communications and Networking*.

Venkataraman, S., Yang, Z., Franklin, M., Recht, B., and Stoica, I. (2016). Ernest: Efficient performance prediction for {Large-Scale} advanced analytics. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 363–378.

Wang, L., Zhan, J., Luo, C., Zhu, Y., He, Y., Gao, W., Jia, Z., Shi, Y., Zhang, S., Zheng, C., Lu, G., Zhan, K., Li, X., and Qiu, B. (2014). Bigdatabench: a big data benchmark suite from internet services. *Proceedings - International Symposium on High-Performance Computer Architecture*.

Xu, L., Butt, A. R., Lim, S.-H., and Kannan, R. (2018). A heterogeneity-aware task scheduler for spark. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 245–256. IEEE.

Yadwadkar, N. J., Hariharan, B., Gonzalez, J. E., Smith, B., and Katz, R. H. (2017). Selecting the best vm across multiple public clouds: A data-driven performance modeling approach. In *Proceedings of the 2017 symposium on cloud computing*, pages 452–465.

Zhang, X., Li, X., Du, H., and Ruiz, R. (2022). Task scheduling for spark applications with data affinity on heterogeneous clusters. *IEEE Internet of Things Journal*, 9(21):21792–21801.