International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Reinforcement Learning Hadoop Map Reduce Parameters Optimization

Nandita Yambem[1], Rashmi S [2], A N Nandakumar [3]

Submitted: 08/01/2025 **Revised:** 22/02/2025 **Accepted:** 01/03/2025

Abstract*:* Among the various techniques for enhancing Hadoop performance—such as intermediate data compression, in-memory management, and parameter tuning—dynamic configuration parameter tuning proves to be the most impactful. However, existing approaches face several challenges: limited adaptability to specific application requirements, isolated parameter tuning without considering interdependencies, and inaccurate linear assumptions in complex environments. To address these issues, this study introduces a reinforcement learning-based optimization framework using Q-Learning. The proposed method dynamically adjusts key Hadoop configuration parameters by continuously learning from job execution metrics such as completion time and wait times in map/reduce phases. It employs a reward-based feedback mechanism to minimize the gap between expected and actual performance, ensuring more accurate, adaptive, and holistic optimization. Additionally, the framework integrates a neural network to predict optimal parameter values, further enhancing decision-making. This approach significantly improves execution efficiency and resource utilization, offering robust adaptability across diverse workloads and operational environments, while aligning closely with service level agreements.

Keywords: *Hadoop, reinforcement learning, q-learning, mapreduce, HDFS*

1. Introduction

Many enterprise organizations are increasingly embracing big data analytics to glean insights from vast reservoirs of digital data housed within their data repositories, employing these insights for diverse objectives such as marketing, advertising, and product design. Among the prominent open-source platforms catering to big data needs, Hadoop stands out. It is engineered to support the execution of large-scale data applications through the map-reduce architecture,

Hadoop serves as a cornerstone for data processing. Analytic applications operate on data stored within the Hadoop Distributed File System (HDFS), facilitating comprehensive data processing. Applications running on Hadoop comprise two crucial components: the Map and Reduce phases.

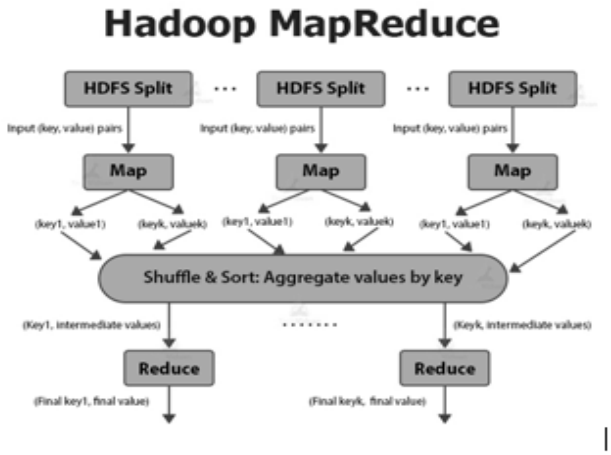

Fig. 1. MapReduce Application Execution

1, 2 Computer Science Engineering(Data Science) Dayananda Sagar College of Engineering,Bangalore, Karnataka 5600111, India, nanditayambem@gmail.com, rashmineha.s@gmail.com
3 AMC Engineering College, Bengaluru, Karnataka 560062, India
inandakumar53@gmail.com

The input data undergoes processing by multiple Map tasks, generating intermediate data in the form of key-value pairs. Subsequently, multiple Reduce tasks process this intermediate data and produce output also in key-value format. Hadoop, functioning as a distributed platform, accommodates the addition of numerous heterogeneous computing nodes. It orchestrates the parallel execution of multiple maps and reduces, ultimately furnishing the final result. The performance of applications within the Hadoop ecosystem hinges on various configuration parameters such as the number of maps, reduces, sort factor, spill percentage, and runtime memory allocation, among others. Among these parameters, setting the optimal values for the number of maps/reduces significantly impacts application performance, considering factors like application nature, available resources within the Hadoop infrastructure, and data volume. While many methods for optimizing the number of maps/reduces rely on a linear correlation between intermediate data volume and the number of reduce tasks, this linear relationship proves inadequate in scenarios involving dynamic resource availability and diverse application characteristics. To address this, our study proposes a machine learning-driven solution utilizing reinforcement learning to optimize Hadoop's configuration parameters. Through a reward/punishment mechanism, the solution continuously learns the optimal number of maps/reduces required to achieve desired service level agreements (SLAs), adapting to changing resource availability and application traits until satisfactory prediction accuracy is attained. The predicted optimal parameter values are then configured within the Hadoop environment for application execution, ensuring dynamic responsiveness to current resource availability and application nuances.

2. Literature Review

Greeshma Lingam et al. [1] introduces the Learning Automata-based MapReduce Scheduling (LA-MRS) algorithm to optimize energy consumption in heterogeneous environments while adhering to user-specified deadlines. The LA-MRS algorithm introduces a reinforcement learning-based approach using Learning Automata to optimize the scheduling of MapReduce jobs by minimizing energy consumption while meeting job deadlines. It dynamically allocates resources based

on feedback from the system, improving efficiency over time. Evaluated using HiBench benchmarks, the algorithm achieves around 25% energy savings compared to traditional methods,

while also ensuring deadline compliance and adaptability in heterogeneous environments. However, it may introduce computational overhead, faces scalability challenges in large-scale systems, and relies heavily on accurate environmental feedback.

Prashanth et al[2] proposes DRLMOTS, a Deep Q-Learning Network-based scheduler that dynamically assigns tasks between cloud and fog nodes to optimize system performance. By analyzing task characteristics and system states, it reduces makespan, lowers energy consumption, and improves fault tolerance significantly compared to traditional models like CNN, LSTM, and GGCN. Experimental results show up to 39.6% energy savings and over 26% makespan reduction. However, the approach may introduce computational overhead, relies heavily on quality training data, and requires further testing for scalability in larger, more dynamic environments

Zaharia et al.[5] achieves improved Hadoop performance through scheduling, by introducing the Longest Approximate Time to End (LATE) algorithm. This algorithm prioritizes jobs that could affect the response time of others, thereby mitigating slowdowns and memory inaccuracies. However, identifying such lagging jobs in a heterogeneous environment proved challenging.
Guo et al. [7] aimed to increase speed-up in the Hadoop platform by exploiting data locality, which reduces cross-switch network traffic by localizing tasks with more intercommunication on the same node. However, prior knowledge of job communication characteristics is required to achieving efficient data locality, which can be difficult to obtain for certain applications.
Chen et al. [] proposed an adaptive intermediate data compression strategy to enhance Hadoop performance. This strategy, driven by a decision algorithm, compresses data based on overhead and application performance, also reducing power consumption by 60%.
Verma et al. [3] devised a strategy to enhance Hadoop performance by employing a data compression approach. By compressing data, they aimed to reduce net bandwidth and the time needed for data transfer between Hadoop nodes. This solution, tested using a word counting task in XSEDE resource, yielded a performance gain of less than 5%.
Ruan et al.[13] increased application speed-up in Hadoop through data compression, achieving a 5% increase for word counting tasks while reducing bandwidth and energy consumption.
Moise et al.[10] improved Hadoop performance by introducing a fast-intermediate layer optimized for concurrency and fault tolerance, reducing overall execution time by enhancing efficiency in read/write processes. The performance gain was improved by 10% due to the fast-intermediate layer.
Liao et al.[19] identified optimal Hadoop configuration parameters using a genetic algorithm, though their approach did not consider the interplay between parameters.
Bhaskar et al.[14] improved Hadoop performance by improvising the memory model, pre-allocating memory based on past execution profiles. While this increased application speed-up, it reduced system throughput.
Chen et al.[12] identified parameters for optimization in Hadoop based on the application's nature, distinguishing between CPU and IO-intensive applications. However, they did not provide an optimization strategy for these parameters.
Malik et al.[17] enhanced Hadoop performance by collocating interactive applications on the same node, reducing communication overhead and energy consumption, resulting in an 8% increase in application speed-up.
Yu et al. [11] focused on increasing application speed-up in Hadoop by developing an efficient data shuffling algorithm to reduce data movement and extra data cycles. Despite achieving a 10% performance gain, the improvement was relatively modest.
Crume et al.[9] addressed Hadoop job speed-up by compressing intermediate data created by map jobs, reducing data shuffling overhead. Their proposed loss compression scheme achieved

significant compression of intermediate data, but decompression overhead offset some of the gains.
Veiga et al.[22]increased application speed-up in Hadoop by optimizing memory allocation and increasing in-memory operations, doubling application speed-up. However, this optimization led to higher resource costs due to in-memory storage usage.
Zhang et al.[7] tackled Hadoop performance enhancement through phase-level scheduling, splitting jobs into multiple phases to increase parallelism and speed-up. While this approach increased performance by 1.3 times in a 10-node Hadoop cluster, splitting applications into phases without congruency posed challenges.

S. Kumar et al.[23] utilized a gradient approach for fine-tuning Hadoop configuration parameters to increase application speed-up, though the approach lacked adaptability to application nature and resource availability.
Nicolae et al.[24] proposed enhancing the efficiency of Hadoop Distributed File System (HDFS) to boost application speed-up within the Hadoop platform. They managed the computational overhead resulting from data access concurrency in HDFS to achieve this aim. Validation of their solution against the Grid 5000 dataset revealed a 5% speed-up improvement over the default HDFS configuration. However, the gain was relatively low compared to the gains obtained through configuration parameter tuning.

3.Research Gap

Among the various methods employed to improve performance in Hadoop, such as intermediate data compression, in-memory management, and configuration parameter tuning, optimizing the configuration parameters stands out as particularly effective in yielding higher performance gains. However, despite its effectiveness, several challenges hinder the identification of the optimal values for these configuration parameters in existing solutions.
Firstly, there is often poor adaptivity to the characteristics of specific applications. For instance, consider a scenario where two different applications are running on the same Hadoop cluster: one application may require more memory allocation while the other may prioritize processing speed. Existing solutions may struggle to adapt to these diverse application needs.
Secondly, many existing solutions optimize each individual parameter in isolation, without considering its influence on other parameters. This lack of holistic optimization can jeopardize the reliability of execution. For example, optimizing the number of mapper tasks without considering the impact on reducer tasks may lead to inefficient resource allocation and suboptimal performance.
Thirdly, the assumption of a linear relationship between configuration parameters and intermediate data volume often falls short in heterogeneous environments and for applications with different characteristics. For instance, the relationship between the number of reducers and intermediate data volume may vary significantly depending on factors such as data skew and processing logic.
Some techniques emphasized the resource overhead introduced by in-memory optimization, where increased speed-up came at the cost of higher memory usage, making it unsuitable for resource-constrained systems. This highlights a gap in balancing performance with efficient resource utilization.
Solutions were proposed gradient-based tuning mechanism, but it lacked adaptability to changing workloads or application behavior, reducing its effectiveness in dynamic Hadoop

environments. The inability to respond to runtime variability remains a significant limitation.
Furthermore, their solution focused solely on file system enhancements and did not address broader MapReduce-level optimizations or integrate with parameter tuning strategies.
To address these challenges, this work proposes a solution that takes into account the specific characteristics and requirements of individual applications running on Hadoop clusters. By considering the interplay between different parameters and adapting dynamically to changes in workload and environment, this solution aims to achieve more robust and efficient optimization of configuration parameters.

4.Proposed Solution

This work proposes the utilization of Q-Learning reinforcement learning for tuning Hadoop parameters in order to tackle the mentioned challenges.

1.1Test run

To initiate a trial run of the application, various configuration parameters are adjusted, and job statistics are gathered. The parameters under consideration in this study are:

- io. sort.mb (C1)
- io.file.buffer.size (C2)
- io.sort.spill.percent (C3)
- io.sort.factor (C4)

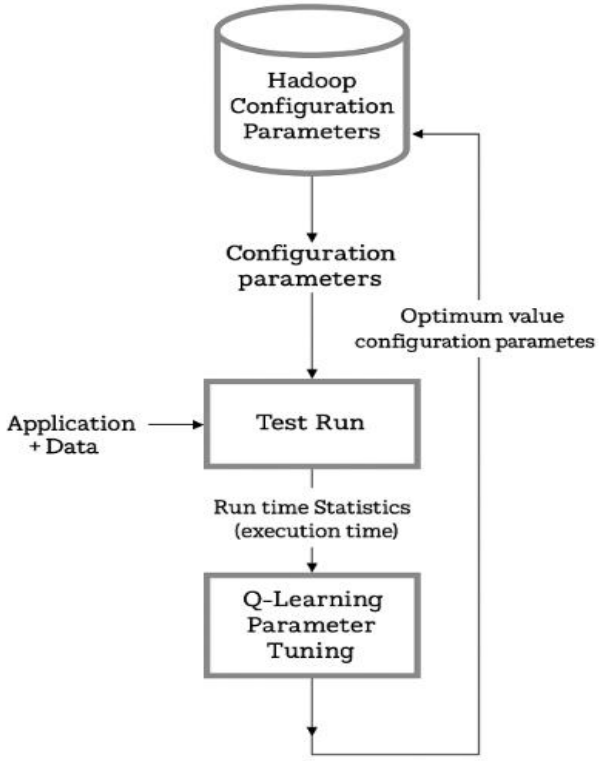

Fig. 2. Proposed Architecture

These parameters are then fine-tuned based on the collected job statistics, which encompass:

- Job completion time (P1)
- Wait time in the map phase (P2)
- Wait time in the reduce phase (P3)

The architectural layout of the solution is illustrated in Figure 3.

A test run is conducted using the application and data, and learning occurs based on the results to fine-tune the Hadoop configuration parameters.
Here is the significance of each configuration parameter:

- io.sort.mb: This parameter governs the size of the in-memory buffer utilized for storing map results. A lower configuration value may trigger spills, leading to performance degradation. By default, this parameter is set to 100 MB.
- io.file.buffer. size: Control over the size of buffers allocated
- for I/O operations is determined by this parameter. Opting for a smaller value may result in application slowdowns, particularly for applications with a high volume of I/O operations.
- io.sort.spill. percent: Once the data in the in-memory buffer surpasses the value specified by this parameter, it is spilled to the hard disk. Excessive spilling can negatively impact application performance.
- io.sort.factor: This parameter dictates the number of files (streams) to be merged during the sorting process of map tasks. While the default value is 10, increasing it enhances the utilization of physical memory and reduces overhead in I/O operations.

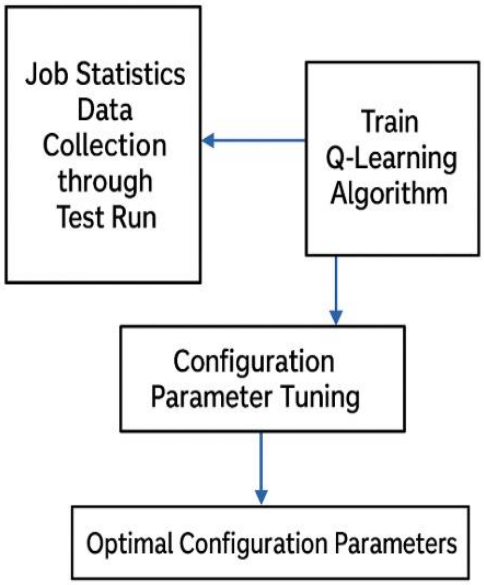

Fig. 3. Proposed implementation with Reinforcement Learning

1.2Reinforcement Learning

The process of reinforcement learning (RL) is outlined below:

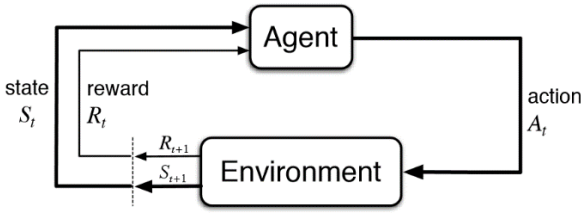

Fig. 4. Process of RL

Fig 4. illustrates the RL process, which revolves around four fundamental concepts. The agent controls the three key concepts of state, action, and reward. An objective function is formulated based on the minimization of deviation between expected and actual execution times.

Continuously, the agent makes decisions during each profile run and evaluates the result of the objective function. Based on these outcomes, it refines its decision-making process for subsequent iterations. The objective function for this study is computed as:

$$F = \frac{1}{E_T - O_T} \quad (1)$$

Where O_T is the actual execution time and E_T denotes the expected completion time of the application. When the disparity between the expected and actual execution times is zero, the objective function reaches its maximum value. Thus, the agent continuously strives to maximize the objective function.
States represent the decision-making factors that influence the execution time of the application, while actions denote the decisions made by the agent, such as increasing or decreasing the number of maps/reduces. These actions can yield positive or negative rewards. Positive rewards are assigned when an action maximizes the objective function, whereas negative rewards are given when an action leads to a reduction in the objective function. Over time, the agent learns to prioritize actions that consistently yield positive rewards.

In this study, four Q-Learning models are employed, with each model corresponding to one of the configuration parameters considered. The number of states utilized for the Q-Learning model is four: start, stop, favor, and de-favor. Within each state, there are four actions available:

- Increase the value of the configuration parameter.
- Decrease the value of the configuration parameter.

The degree of increment or decrement for each parameter is determined based on the observed values of profile parameters (P1, P2, P3). Predictions for the values of configuration parameters (C1, C2, C3, C4) are generated using a neural network with the following configuration:
Inputs: P1, P2, P3

Outputs: C1 | C2 | C3 | C4

Number of layers: 3

Number of neurons in layer 1: 3

Number of neurons in layer 2: 9

Number of neurons in layer 3: 1

Activation function: Relu

The neural network is trained using past profile history. At each stage, the agent selects one action and observes the resulting response, which is the value of the objective function. Positive rewards are assigned if the objective function is maximized, while negative rewards are given otherwise. The Boltzmann distribution function is employed to select one action among the
four available actions in a given state. This function calculates the probability of selecting each action based on the quality of choosing that action at the current state.

$$p(s_k, a_k = i) = \frac{e^{Q(s_k, a_k = i)/t_n}}{\sum_{j=1}^{N_a} e^{Q(k, a_j)/t_n}} \quad i = 1, \dots N_a \quad (2)$$

Where $Q(s_k, a_k = i)/t_n$is the state-action value function that evaluates the quality of choosing action $a_k = i$ at state s_k.
N_a is the number of actions.
t_n is the time varying parameter controlling the degree of exploration versus exploitation.
Reinforcement learning is executed for all job profile data, and eventually stabilizes in the state that maximizes the reward. The number of maps and reduces at this state represents the optimal configuration setting for achieving the desired completion time of the application.

5.Novelty in the Proposed Solution

The key contributions of this work are

- The parameter tuning approach in this study considers multiple attributes of job completion time and wait times for map/reduce tasks. Unlike previous methods that solely rely on intermediate data volume, this approach offers a more comprehensive perspective on performance optimization. By incorporating various metrics related to job execution, such as completion time and wait times, the tuning process becomes more refined and aligned with the specific requirements of the application. This approach enhances the effectiveness of parameter optimization, leading to improved overall performance in Hadoop environments.
- One significant aspect of this solution is its adaptability to both resource availability and application characteristics. Traditional approaches often lack flexibility and struggle to accommodate dynamic changes in resource availability or variations in application requirements. However, in this work, the parameter tuning methodology is designed to dynamically adjust to fluctuations in resource availability and adapt to the unique characteristics of different applications. This adaptability ensures that the tuning process remains responsive and relevant in diverse operational scenarios, thereby enhancing its practical utility and effectiveness.
- Another noteworthy contribution of this work is the adoption of reinforcement learning (RL) techniques for modelling the relationship between Hadoop configuration parameters and job execution times. Unlike conventional methods that rely on simplistic linear models, RL offers a more sophisticated and dynamic approach to parameter optimization. By leveraging RL algorithms, the model can capture complex interactions and dependencies between configuration parameters and job performance. This enables more accurate and nuanced adjustments to be made, leading to optimized configurations that better align with the specific requirements and constraints of the Hadoop environment. As a result, the overall efficiency and effectiveness of the parameter tuning process are significantly enhanced, contributing to improved performance outcomes in Hadoop-based systems.

6.Results

The efficacy of the proposed parameter tuning approach based on reinforcement learning is evaluated using the following configuration.

Table 1. Testing Parameters

Dataset	**PUMA (Wikipedia and Movies-database) [21]**
Applications tested	Word count, K-means
Solutions compared	Default Hadoop, Selective parameter tuning proposed by Chen et al[12]
Performance parameters	Execution time

The performance of the proposed solution is compared against parameter optimization method introduced by Chen et al [12]. Execution times are measured across various dataset volumes, focusing on the word count and K-means application.

The result for work count application is given below.

Table 2. Execution Time for Different Volume for Word Count

Data volume (MB)	Default Hadoop	[12]	Reinforcement Learning
128	100	60	46
256	170	100	77.3
512	240	150	103.04
1024	430	270	186
2048	840	510	432

The proposed reinforcement learning framework has achieved a 29% decrease in execution time compared to the solution given in [12] and an impressive 110% decrease compared to default Hadoop. This improvement is attributed to reinforcement learning's ability to optimize the number of maps and reduce execution time. The experiment, conducted with a constant data volume of 128MB, measured execution time for varying numbers of iterations, yielding the results presented below for the word count application.

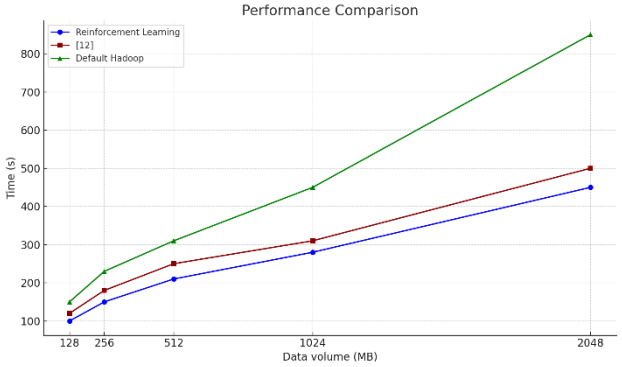

Fig. 5. Execution time for word count

Table 3. Execution Time For Different Iteration For Word Count

Iterations	Default Hadoop	[12]	Reinforcement Learning
1	100	60	46
2	107	62	43
3	105	60	40
4	104	61	39
5	100	62	35

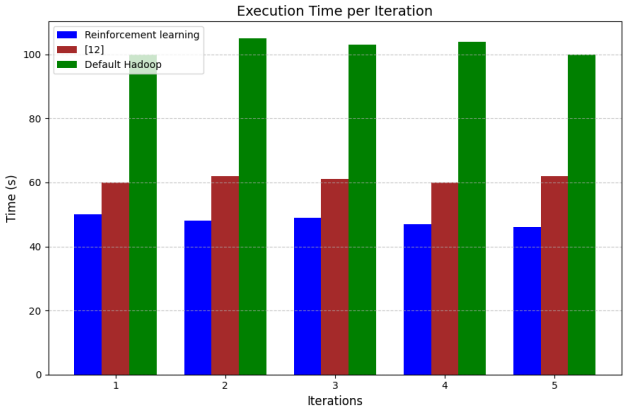

Fig. 6. Execution time over iterations for word count

The proposed reinforcement learning algorithm demonstrates superior capability in learning optimal performance parameters compared to the solution in [12]. Over time, the execution time exhibits a standard deviation of 3.84 minutes, whereas solution in [12] shows a standard deviation of 2.34 minutes. Notably, the proposed solution adapts dynamically in learning execution times, as opposed to solution [12]. Across five iterations, the ensemble-based solution in the proposed method exhibits a standard deviation of 6.05 minutes, in contrast to the 2.34 minutes seen in solution [12]. This indicates a 40% increase in adaptability in the proposed reinforcement solution compared to the solution in [12]. Below are the results for the K-means clustering application.

Table 4. Execution Time for Different Volume for K-Means

Data volume (MB	Default Hadoop	[12]	Reinforcement Learning
128	73.6	96	160
256	123.68	160	272
512	164.86	240	384
1024	297.6	432	688
2048	691	816	144

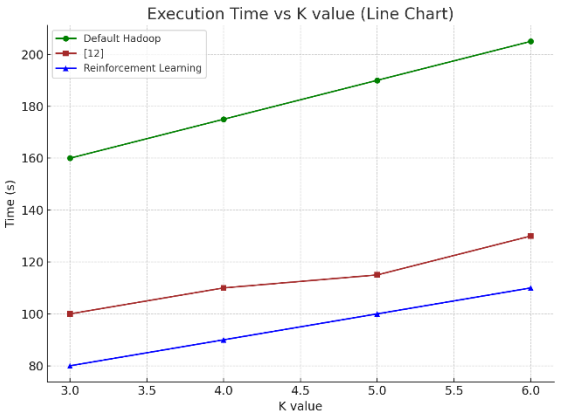

Fig. 7. Execution time for K means clustering

The proposed reinforcement learning framework has achieved a significant improvement, with a 30% decrease in execution time compared to solution [12], and an even more impressive 105% decrease compared to the default Hadoop configuration. This notable enhancement can be attributed to reinforcement learning's capacity to determine the optimal number of maps, thus effectively reducing execution time in the proposed solution.

Furthermore, to gauge the effectiveness of the proposed approach, execution time was measured across various iterations for K-means clustering, with a fixed value of k set to 3. The results of these measurements are detailed below.

Table 5. Execution Time for Different Iterations for K-Means

Iterations	Default Hadoop	[12]	Reinforcement Learning
1	160	96	73.6
2	161	96.2	70.3
3	162	97	67.2
4	162.3	96	68.6
5	167.4	96.2	64.7

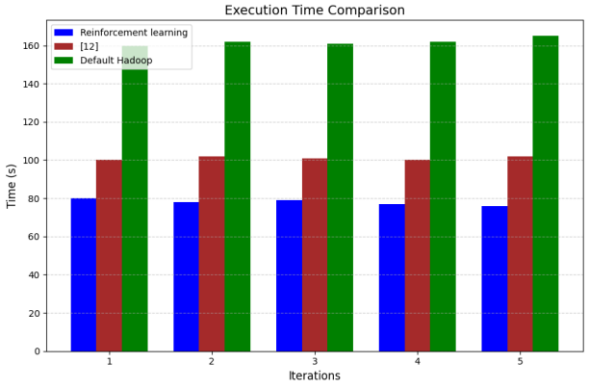

Fig. 8. Execution time over iteration for K-mean clustering

The proposed reinforcement learning algorithm demonstrates superior capability in learning optimal performance parameters compared to solution [12]. As the algorithm iterates, it refines its understanding of the system, resulting in a gradual decrease in execution time by over 12% over the course of the iterations.

Moreover, to comprehensively assess the algorithm's effectiveness, execution time was measured for various values of K, yielding the results provided below.

The relationship between execution time and the value of K exhibits a linear trend, wherein an increase in K leads to a proportional increase in execution time.

Table 6. Execution Time Varying K Value In K Means

K value	Default Hadoop	[12]	Reinforcement Learning
3	160	96	73.6
4	170	104.2	78.3
5	183	108.5	84.2
6	201	121	88.7

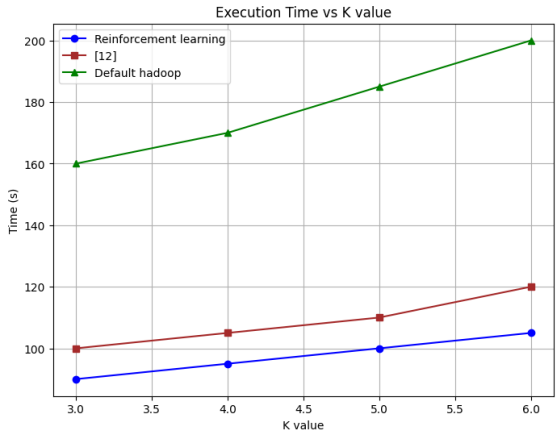

Fig. 9. Execution time with varying k value

However, it's noteworthy that the rise in execution time for K-means clustering from a K value of 3 to 6 is merely 17%, a notably lower increase compared to the 20% observed in solution [12], and even lower than the 20.39% observed in default Hadoop configurations.

Furthermore, to provide a comprehensive understanding of system performance, the percentage of system utilization was measured across various applications, including word count and K-means clustering with different values of K. The detailed results are presented below.

Table 7. System Utilization (%)

Applications	Default Hadoop	[12]	Reinforcement Learning
Word count	74	78	81
K means (K=3)	76	85	87
K means (K=4)	77	86	89
K means (K=5)	78	87	91
K means (K=6)	78	88	92

The average system utilization in the proposed setup consistently surpasses that of solution [12] by a minimum margin of 3.2%. In comparison to the default Hadoop configuration, this improvement is even more significant, with the proposed system achieving an average utilization rate that is 11.4% higher. This indicates a marked enhancement in resource utilization efficiency in the proposed framework when compared to both solution [12] and the default Hadoop setup

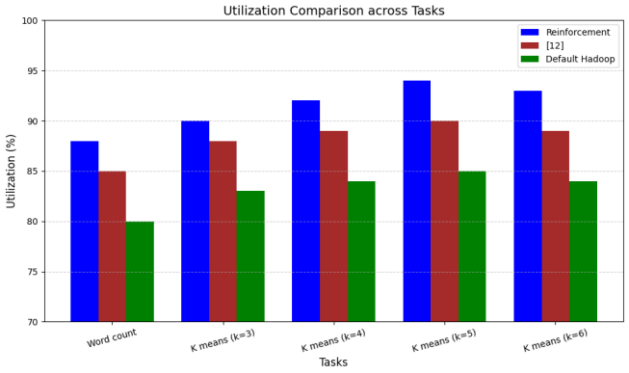

Fig. 10. Utilization for different applications

7.Conclusion

This work introduces a novel approach to Hadoop configuration parameter tuning, leveraging reinforcement learning techniques. Diverging from conventional methods, our approach involves learning the optimal parameter values based on multiple attributes. By considering these diverse attributes, our proposed solution demonstrates a remarkable ability to accurately predict the optimal configuration parameters for Hadoop. As a result, our solution achieves a notable 110% increase in speed compared to the default Hadoop configuration. This advancement underscores the efficacy of our approach in significantly enhancing the performance of Hadoop systems.

References

[1] Greeshma Lingam , “Reinforcement learning based energy efficient resource allocation strategy of MapReduce jobs with deadline constraint”, Cluster Computing ,2023, 26:2719–2735, Springer

[2] Prashant Choppara and Sudheer Mangalampalli, “ An efficient deep reinforcement learning based task scheduler in cloud-fog environment”, Cluster Computing ,2025,28:67, Springer.

[3] A. Verma, L. Cherkasova, and R. Campbell. Resource Provisioning Framework for MapReduce Jobs with Performance Goals. ACM/IFIP/USENIX Middleware, pages 165–186, 2011.

[4] Y. Chen, A. Ganapathi, and R. H. Katz, “To compress or not to compress-compute vs. io tradeoffs for mapreduce energy efficiency,” in Proceedings of the first ACM

SIGCOMM workshop on Green networking. ACM, 2010, pp. 23–28.
[5] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica. Improving mapreduce performance in heterogeneous environments. In USENIX Symposium on Operating Systems Design and Implementation (OSDI), volume 8, page 7, 2008
[6] Y Chen, S Alspaugh, R Katz, "Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads",2012, arXiv preprint arXiv:1208.4174
[7] Qi Zhang, "PRISM: Fine-Grained Resource-Aware Scheduling for MapReduce", 2015 IEEE
[8] Zhenhua Guo , Geoffrey Fox , Mo Zhou, Investigation of Data Locality in MapReduce, Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012), p.419-426, May 13-16, 2012 [doi>10.1109/CCGrid.2012.42]
[9] Adam Crume , Joe Buck , Carlos Maltzahn , Scott Brandt, Compressing Intermediate Keys between Mappers and Reducers in SciHadoop, Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, p.7-12, November 10-16, 2012 [doi>10.1109/SC.Companion.2012.12]
[10] Nandita Yambem, AN Nandakumar, "AMPO: Algorithm for MapReduce Performance Optimization for enhancing big data analytics", IEEE,2017
[11] W. Yu, Y. Wang, X. Que, and C. Xu, "Virtual shuffling for efficient data movement in mapreduce," IEEE Transactions on Computers, vol. 64, no. 2, pp. 556–568, 2015
[10] D. Moise, T.-T.-L. Trieu, L. Boug´e, and G. Antoniu, "Optimizing intermediate data management in mapreduce computations," in Proceedings of the first international workshop on cloud computing platforms. ACM, 2011, pp. 1–7 .
[11] B. Nicolae, D. Moise, G. Antoniu, and al. BlobSeer: Bringing high throughput under heavy concurrency to Hadoop Map/Reduce applications. In Procs of the 24th IPDPS 2010, 2010. In press
[12] Chen, Xiang & Liang, Yi & Li, Guang-Rui & Chen, Cheng & Liu,Si-Yu. (2017). Optimizing Performance of Hadoop with Parameter Tuning. ITM Web of Conferences. 12. 03040. 10.1051/itmconf/20171203040.
[13] G. Ruan, H. Zhang, and B. Plale, "Exploiting mapreduce and data compression for data-intensive applications," in Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery. ACM, 2013, pp. 1–8
[14] Bhaskar, Archana & Ranjan, Rajeev. (2019). Optimized memory model for hadoop map reduce framework. International Journal of Electrical and Computer Engineering (IJECE). 9. 4396. 10.11591/ijece.v9i5.pp4396-4407.
[15] Nandita Yambem A. N. Nandakumar, "Enhanced Performance of Hadoop Parameters Using Hybrid Meta Heuristics Optimization Techniques", International Journal of Intelligent Systems and Applications in Engineering,2024, Volume 12,Issue No.3 , 1508-1513
[16] Veiga, Jorge & Expósito, Roberto & Taboada, Guillermo & Touriño, Juan. (2018). Enhancing in-memory efficiency for MapReduce-based data processing. Journal of Parallel and Distributed Computing. 120. 10.1016/j.jpdc.2018.04.001.
[17] Maria Malik, Hassan Ghasemzadeh, Tinoosh Mohsenin, Rosario Cammarota, Liang Zhao, Avesta Sasan, Houman Homayoun, and Setareh Rafatirad. 2019. ECoST: Energy-Efficient Co-Locating and Self-Tuning MapReduce Applications. In Proceedings of the 48th International Conference on Parallel Processing (ICPP 2019).
[18] C, K. and X, A. (2020), Task failure resilience technique for improving the performance of MapReduce in Hadoop. ETRI Journal, 42: 748-760. https://doi.org/10.4218/etrij.2018-0265
[19] Liao G., Datta K., Willke T.L. (2013) Gunther: Search-Based Auto-Tuning of MapReduce. In: Wolf F., Mohr B., an Mey D. (eds) Euro-Par 2013 Parallel Processing. Euro-Par 2013. Lecture Notes in Computer Science, vol 8097. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-40047-6_42
[20] S. Kumar, S. Padakandla, L. Chandrashekar, P. Parihar, K. Gopinath and S. Bhatnagar, "Scalable Performance Tuning of Hadoop MapReduce: A Noisy Gradient Approach," *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, Honolulu, CA, 2017, pp. 375-382, doi: 10.1109/CLOUD.2017.55
[21] https://engineering.purdue.edu/~puma/datasets.htm
[22] J Veiga, RR Expósito, GL Taboada, J Touriño, "Enhancing in-memory efficiency for MapReduce-based data processing", Journal of Parallel and Distributed Computing , April 2018, 323-338
[23] S. Kumar, S Padakandla, L Chandrashekar, P Parihar, K Gopinath , "Scalable performance tuning of hadoop mapreduce: a noisy gradient approach",IEEE,2017
[24] B Nicolae, D Moise, G Antoniu, L Bougé, M Dorier, "BlobSeer: Bringing high throughput under heavy concurrency to Hadoop Map-Reduce applications",IEEE,2010