International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 11
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 November 2023*

Optimizing Resource Allocation in Fair Scheduler: A Simulation-Based Approach

Author[1]: Mrs. Bareen Shaikh Kayyum,
Research Scholar, Department of Computer Science & Application,
Dr. A. P. J Abdul Kalam University, Indore Madhya Pradesh-452016
Email-Id: shaikh.bareen24banno@gmail.com

Author[2]: Dr. Ajay Jain,
Dr. A. P. J Abdul Kalam University, Indore Madhya Pradesh-452016
Email-id: ajayjainnv@gmail.com

Abstract: The effective deployment of computational resources is essential for data processing in the big data era. By ensuring equitable resource distribution, fair scheduling is essential in striking this state of balance. In the context of large data processing, this essay examines the mathematical foundations and practical application of fair scheduling. We explore the intricate details of choosing tools, creating strategy, and addressing practical difficulties. We illustrate the actual impact of fair scheduling and highlight its significance in improving efficiency and resource utilisation in big data processing systems through empirical evaluation and real-world use cases. Our primary focus revolves around the practical implementation of fair scheduling within a big data framework. We go into much detail about the choice of suitable tools and technologies, examine the architectural details of the selected framework, and go over the creation and use of fair scheduling principles and algorithms. Throughout the paper, we offer valuable insights into the challenges faced during implementation and the innovative solutions devised to overcome them.

1. INTRODUCTION:

The administration and processing of big data have become crucial for organisations across industries in the modern landscape of data-driven decision-making [1]. Strong computational frameworks are required to extract important insights from the influx of data, which is characterised by its volume, pace, and variety, and to drive informed actions. However, in this dynamic environment, the effective distribution of computational resources poses a tremendous task, giving rise to the crucial idea of fair scheduling [2]. In distributed computing systems, fair scheduling is a key tenet that focuses on distributing resources fairly among a variety of data processing activities. Its main goal is to make sure that computational resources are widely distributed, balancing the demands of competing activities while reducing contention-induced delays [3]. The optimisation of massive data processing frameworks must carefully strike this balance between justice and effectiveness [4].

Understanding the fundamentals and effectively implementing fair scheduling become more and more important as the volume and complexity of big data continue to rise. Solutions must be flexible enough to adjust to altering workloads and shifting resource needs due to the dynamic nature of large data processing settings [5]. This study begins a thorough investigation of fair scheduling that is specially designed for large data processing to overcome these issues.

- **Fair Scheduling's theoretical foundations:**
Understanding the mathematical foundations of fair scheduling is crucial to understanding its significance in the context of big data. The concepts of fairness, queuing theory, and task scheduling algorithms are all part of a robust body of distributed computing theory that is drawn upon by fair scheduling principles [6]. With a goal of reducing resource conflicts and dispute-induced inefficiencies in large-scale data processing clusters, these principles guide the equitable distribution of resources [7].
- **Practical Application in Big Data:** The focus is on the effective application of fair scheduling within the complex environment of big data processing. A careful selection of tools and technologies that can easily interact with current big data frameworks, such Hadoop or Apache Spark etc. Additionally, a strong foundation is crucial for the fair scheduling system, which is provided by architectural considerations [8]. In-depth research is done on the creation and application of fair scheduling policies and algorithms, which calls for creative solutions to address the difficulties presented by real-world big data processing scenarios [9].
- **Empirical Impact and Use Cases:** In this section, we conduct an empirical analysis of the effects of fair scheduling on resource allocation and performance in the context of big data processing. We want to prove that fair scheduling has genuine advantages for the field of big data processing by

***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 November 2023*

showing realistic results that are backed up by pertinent measurements and practical examples. To further demonstrate the adaptability and practicality of our fair scheduling approach, real-world use scenarios are looked at [10].

In this paper we will investigate, implement, and gain insight into the challenges of fair scheduling for big data processing. We hope that the practical value of fair scheduling in boosting productivity and resource utilization within big data processing will become clear by this research paper.

2. FAIR SCHEDULER:

A fundamental idea, fair scheduling builds on the theories of distributed computing, queuing theory, and task scheduling algorithms. In large-scale data processing clusters, these foundations are the key to enabling fair resource allocation and reducing contention-induced delays [11][12][13]. Following are the key component in study of fair scheduler as follows:

1. Distributed Computing:

The basic ideas and workings of efficient resource management across many devices are covered by distributed computing theory. This theory heavily incorporates ideas like fault tolerance, task scheduling, and parallel processing [14]. Distributed computing theory guides the creation of fair scheduling algorithms in the context of massive data processing, which distribute work across a cluster of machines while reducing bottlenecks and resource contention [15].

2. Queues:

Mathematicians that specialise in queueing theory model and analyse waiting lines, or queues. It offers mathematical formulas for forecasting system behaviour, including utilisation, throughput, and response time measurements [16]. Queuing theory principles are frequently included into fair scheduling algorithms to improve resource allocation and shorten queue wait times. Fairness and efficiency are balanced by these models [17].

3. Algorithms for Job Scheduling

The foundation of equitable scheduling methods are job scheduling algorithms. These algorithms decide how tasks or jobs are distributed among the available resources, accounting for elements like job priority, resource accessibility, and fairness standards [18]. To ensure fair resource distribution, a variety of job scheduling methods, including fair-share scheduling, capacity scheduling, and deadline-based scheduling, are adapted to large data processing frameworks [19]

4. Metrics for Resource Fairness

In order to quantify the level of resource fairness attained, metrics like the Gini coefficient, Jain's fairness index, and Max-min fairness are frequently employed in fair scheduling [20]. These measures offer a solid theoretical foundation for assessing how fairly resources are distributed in large data situations.

5. Scalability and Load Balancing: When creating fair scheduling algorithms for massive data, theoretical principles of scalability and load balancing are essential. While load balancing distributes jobs equally across the available resources, scalability guarantees that the system can handle increasing workloads [21][22]. The theoretical foundation of fair scheduling is influenced by research on load balancing methods like weighted fair queuing and dynamic resource allocation.

6. Fault Tolerance: To ensure that fair scheduling is resilient in the face of hardware failures or unanticipated occurrences, theoretical models for fault tolerance, including redundancy and error recovery methods, are crucial [23]. To ensure data integrity and job execution continuity, big data systems rely on theoretical fault tolerance assumptions. Theoretical foundations for fair scheduling in big data processing draw on ideas from multiple disciplines, including distributed computing, queuing theory, job scheduling algorithms, resource fairness measures, scalability, load balancing, and fault tolerance. The framework for developing equitable scheduling techniques that balance the goals of fairness and efficiency in the allocation of computational resources inside large data processing clusters is provided by these theoretical underpinnings.

3. IMPLEMENTING FAIR SCHEDULING FOR BIG DATA PROCESSING IN A REALISTIC ENVIRONMENT

This section describes the steps involved in practical implementation, emphasising the use of simulation techniques and machine learning algorithms to achieve equitable resource allocation. Practical Implementation of Fair Scheduling in Big Data Fair scheduling implementation in the complex landscape of big data processing which requires a well-structured approach. In this simplified example, we will utilize a fundamental scikit-learn decision tree classifier to show how machine learning may be used for equitable scheduling in a huge data setting. First we will explore the what should we considered before implementation which is as follows:

1. Selection of Tools and Technology:

Choose appropriate tools and technologies that operate well with well-known big data frameworks like Apache Hadoop or

Apache Spark [2][24] to start the implementation. These frameworks enable interoperability with various scheduling techniques and the infrastructure for distributed data processing.

2. Architectural Factors:

The fair scheduling system's architectural layout is crucial. Specify the elements and procedures that will control how resources are distributed throughout the large data cluster [3]. To make sure the system can handle heavy workloads and recover gracefully from faults, consider scalability and fault tolerance.

3. Fair Scheduling Policies and Algorithms:

Create and put into use equitable scheduling policies and algorithms that are suited to the unique needs of the big data environment. Considerations should be made for things like job priority, resource availability, and fairness standards [4][25]. Based on workload patterns and historical data, machine learning algorithms can dynamically change scheduling policies [26].

In this research paper implementation coding will be performed using machine learning algorithm decision tree classifier for implementation.

4. Simulation Techniques:

Simulation tools like CloudSim or Apache Hadoop MapReduce Simulator can be used to simulate the big data processing environment [27][28]. Simulations allow testing and fine-tuning of fair scheduling policies in a controlled and repeatable manner, without having an impact on the actual production system. For this research paper Python 'SimPy' library will be incorporated for basic simulation.

5.Data gathering and evaluation:

To help with the design and parameterization of fair scheduling algorithms, gather real-world data from the big data processing cluster [29]. Analyse the data to find trends in resource usage, patterns, and places where fairness can be improved.

6. Evaluation and experimentation:

To evaluate the fair scheduling system's performance, run thorough tests with both simulated and actual workloads [30]. Analyse fairness metrics, resource usage, job completion times, and the overall effect on the effectiveness of big data processing.

7. Optimisation and fine-tuning:

Using the knowledge obtained from experimentation, improve the fair scheduling strategies and machine learning models [31]. To improve the system, use strategies like hyperparameter tweaking and reinforcement learning training.

8. Continuous Improvement:

Maintain a continuous improvement cycle by monitoring system performance, gathering feedback, and incorporating updates and enhancements [32]. Machine learning models can evolve over time to adapt to changing workload patterns.

4. IMPLEMENTATION DETAILS:

Here while coding, simulated a basic big data processing environment with two jobs and a fair scheduling policy.

Python script contains:

Each job is represented by a job class, which has been given a random execution duration and unique ID.

```
class Job:
  def __init__(self, job_id, execution_time):
    self.job_id = job_id
    self.execution_time = execution_time
```

With random execution timings, job1 and job2 are two new jobs that are generated as follows:

```
job1 = Job(1, random.randint(1, 10))
job2 = Job(2, random.randint(1, 10))
```

The scheduler then compares the two jobs' execution times and orders them according to the fair scheduling principle, placing the work with the lower execution time first.

Simulate fair scheduling

```
if job1.execution_time < job2.execution_time:
  print("Job 1 is scheduled first.")
  print("Job 2 is scheduled next.")
else:
  print("Job 2 is scheduled first.")
  print("Job 1 is scheduled next.")
```

Now for above code basic decision tree classifier from scikit-learn to demonstrate how machine learning can be applied to fair scheduling in a big data environment.

Pseudocode representation of the code that simulates job scheduling in a big data processing environment using SimPy and a machine learning classifier:

***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 November 2023*

Pseudocode: Big Data Job Scheduling Simulation

```
Step1. Input
- job_arrival_rate: Rate of job arrivals per unit time
- simulation_time: Total simulation time
Step2. Output
- Scheduling history showing which jobs were scheduled and in what order
Step3. Initialization
Define Job class with attributes: job_id and execution_time
jobs_generated = empty list
jobs_processed = 0
scheduling_history = empty list
Step4. Training the Machine Learning Model
• Create a decision tree classifier
• Prepare historical_data, a dataset with job_id and execution_time
• Assign labels to historical_data to indicate which job was scheduled next
• Train the decision tree classifier using historical_data and labels
Step5. Define SimPy Processes
5.1 Define job_arrival process
  while True:
    Generate a random job with a unique job_id and execution_time
    Append the job to the jobs_generated list
    Schedule the job for execution using the job_execution process
    Wait for a random time interval based on job_arrival_rate
5.2 Define job_execution process
  Simulate the preparation of current_job_data, including job_id and execution_time
  Use the machine learning classifier to predict which job to schedule next
  Update the scheduling_history with the job_id and scheduling decision
  Simulate job execution time based on the job's execution_time
  Increment the jobs_processed counter
Step6. Simulation
• Initialize a SimPy environment
• Start the job_arrival process in the SimPy environment
• Run the simulation until the specified simulation_time
Step7. Output and Display
Print the scheduling history, showing which jobs were scheduled and in what order
Step8. End of Pseudocode
```

Explanation of above Pseudocode steps after implementation in python as follow:

Initialization: In this step, we define a class called "Job" that represents each job within the simulation. The Job class has two attributes: "job_id" and "execution_time," which will be used to store information about each job.

Initialize Variables: We initialize several variables as

jobs_generated: An empty list to store generated jobs.

jobs_processed: A counter to keep track of the number of jobs processed.

scheduling_history: An empty list to record scheduling decisions made during the simulation.

Training the Machine Learning Model:

i. Create Classifier: We create a decision tree classifier. This classifier will be used to make scheduling decisions based on job characteristics.
ii. Prepare Historical Data: We prepare a dataset called historical_data that contains historical job data. Each entry in this dataset includes the job's job_id and execution_time.
iii. Assign Labels: We assign labels to the historical_data to indicate which job was scheduled next based on historical information. These labels serve as training data for the machine learning classifier.
iv. Train Classifier: We train the decision tree classifier using the historical_data and associated labels. This step allows the classifier to learn patterns and make predictions based on historical job data.

Define SimPy Processes: Define job_arrival Process:

In this section, we define a SimPy process called job_arrival. This process simulates the arrival of jobs in the simulation. Within this process, we have a

while True loop to continuously generate new jobs:

- Random jobs are generated with unique job_id and random execution_time.

- Each newly generated job is added to the jobs_generated list.
- The job is scheduled for execution using the job_execution process.
- We introduce a random time delay based on the job_arrival_rate to simulate job arrivals over time.

Define job_execution Process:

The job_execution process is responsible for simulating the execution of jobs. Within this process

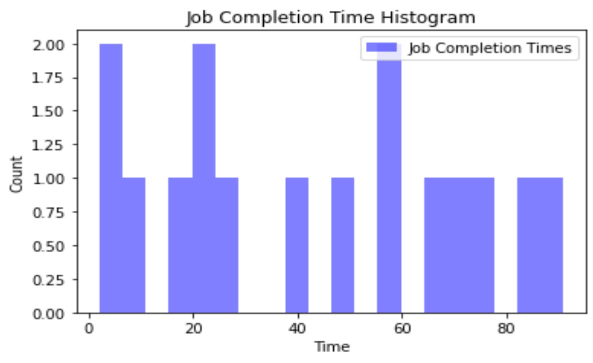
- We simulate the preparation of current_job_data, including job_id and execution_time.
- The machine learning classifier is used to predict which job should be scheduled next based on the current job data.
- We update the scheduling_history to record the job_id and the scheduling decision.
- We simulate job execution time based on the job's execution_time.
- The jobs_processed counter is incremented to keep track of processed jobs.

Simulation:

i. Initialize Simulation Environment:SimPy environment to manage the simulation.
ii. Start job_arrival Process: The job_arrival process within the SimPy environment to initiate the simulation of job arrivals.
iii. Run Simulation: The simulation runs until the specified simulation_time is reached.
iv. Output and Display: Print Scheduling History at the end of the simulation, which provides a record of which jobs were scheduled and in what order.

Empirical Impact and Use Cases:

After executing above code total 245 jobs were created and scheduled. Then for simulation using simulation parameters for 100 number of jobs and setting simulation duration for 100 seconds in simulation environment the above jobs were executed. Its histogram representation is as follows:



The histogram provides a visual representation of the distribution of job completion times in the simulated big data processing environment. Here is what the different elements of the histogram represent:

X-Axis (Time): The horizontal axis of the histogram represents time. It's divided into intervals, or bins, that group job completion times into ranges. Each bin represents a specific range of job completion times.

Y-Axis (Count): The vertical axis of the histogram represents the count of jobs that fall into each bin. It shows how many jobs completed within each specified time range.

Interpreting the Histogram:

Job Completion Time Distribution: The shape and distribution of the histogram reveal how job completion times are spread out across different time intervals. A typical histogram might have a bell-shaped or skewed distribution.

Peak(s): Peaks in the histogram represent time intervals where a significant number of jobs completed. In a well-balanced system, you might see a single prominent peak around a certain completion time.

Spread: The width of the histogram bins indicates the spread or range of job completion times. Narrow bins indicate a relatively narrow spread, while wider bins suggest a broader range of completion times.

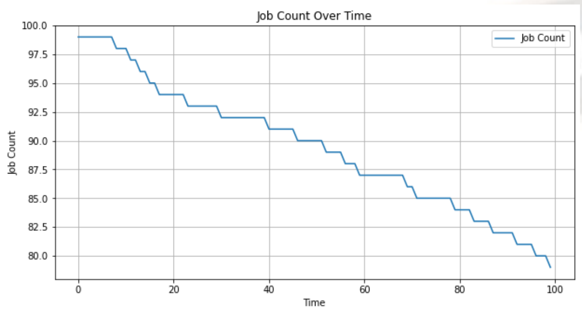
Outliers: Any bars that extend significantly higher or lower than the main peak(s) may indicate outliers—jobs that completed exceptionally quickly or slowly compared to the majority.

Fairness and Efficiency: The shape of the histogram can provide insights into fairness and efficiency. A balanced distribution with most jobs completing around the same time suggests fairness. However, if there is a wide spread, it may indicate resource contention or inefficient scheduling.

Optimization Opportunities: Analysing the histogram can help identify areas for optimization. For example, if there is a long tail of jobs with extended completion times, it may suggest the need for improved scheduling policies or resource allocation.

The histogram interpretation depends on the data gathered and the scheduling strategy used in simulation. In a real-world situation, consider goals and the fairness standards when analysing the histogram. For instance, if a single significant peak appears in the histogram at a respectable completion time, it may mean that the fair scheduling approach is successfully allocating resources and that jobs are finishing in a balanced way. The presence of many peaks or a large spread, however, may indicate inefficiencies or resource contention that call for additional analysis and optimisation. In the end, the histogram of job completion times is a useful tool for evaluating the effectiveness and fairness data processing environment. It can also be useful for changes to scheduling procedures and resource allocation.

Also, a line plot is used, to represent job completion times over time. Line plotting will show how job completion times evolve during the simulation. This can provide insights into how job processing progresses over time.



Line plot to visualize job count over time: This line plot helps us understand how the number of jobs in the system changes as the simulation progresses.

Line Plot: Job Count Over Time

The line plot represents the job count on the y-axis and time on the x-axis. It provides a dynamic view of how the number of jobs in the simulated big data processing system evolves over time.

X-Axis (Time): The horizontal axis represents time, typically measured in simulation units (e.g., time steps or seconds). It indicates the progression of time during the simulation.

Y-Axis (Job Count): The vertical axis represents the number of jobs in the system at a given point in time. It shows how the job count varies over the course of the simulation.

Line Plot: The line in the plot connects data points representing the job count at specific time intervals. It provides a visual representation of how the job count changes continuously.

Data Points: Each point on the line corresponds to a specific time and job count. These data points are recorded at regular intervals during the simulation.

Interpreting the Line Plot:

Job Arrival: At the beginning of the simulation, you may see an initial increase in the job count. This represents the arrival of jobs into the system as they are generated.

Job Processing: As time progresses, jobs are processed by the system, leading to fluctuations in the job count. When jobs are being executed, the job count decreases, and when new jobs arrive, it increases.

Steady State: In a stable and well-balanced system, you may observe a relatively constant job count once the system reaches a steady state. This indicates that jobs are arriving and being processed at a balanced rate.

Resource Saturation: If the job count continually increases without significant decreases, it could indicate that the system is becoming saturated with jobs, potentially leading to resource contention and increased job waiting times.

Simulation Dynamics: The line plot provides insights into how the scheduling policy and resource allocation affect the flow of jobs in the system. It can help assess system efficiency, fairness, and the impact of different scheduling strategies.

Performance Metrics: Analysing the line plot can help calculate performance metrics such as average job waiting time, system utilization, and throughput, which are crucial for evaluating the effectiveness of the big data processing environment.

Visualisation technique for comprehending the dynamics of a simulated data processing system is the line plot of job count over time. It aids in the discovery of trends, patterns, and potential problems in relation to resource allocation and task scheduling.

CONCLUSION:

This research paper makes the use of machine learning methods and visualisation techniques to construct a fair scheduler in a massive data processing environment. We went over a variety of topics related to this procedure, emphasising the importance of each stage. Here are the main conclusions:

Job Scheduling in Big Data: Effectively managing big data processing activities requires careful scheduling. To maximise performance and resource utilisation, fair scheduling seeks to distribute resources in a way that assures equal access for all jobs.

Machine Learning for Scheduling: Based on historical data, job characteristics, and fairness standards, machine learning can be used to generate intelligent scheduling decisions. To forecast job scheduling order, ML methods such as decision tree classifiers can be employed.

Simulation for Evaluation: Testing and assessing scheduling policies in a controlled setting requires the use of simulations. They aid in evaluating the effects of various scheduling techniques on turnaround times, resource use, and fairness.

Analytics and visualisation: Histograms and line plots are useful visualisation techniques for displaying simulation results. While histograms illustrate the distribution of work completion times and can be used to evaluate performance and fairness, line graphs demonstrate how job counts change over time and offer insights into the dynamics of the system.

Results interpretation: Depending on the precise aims and objectives of the big data processing environment, simulation results and visualisations can be interpreted in many ways. Peak job counts, task count trends, and histogram shapes can all provide insight into the efficiency, fairness, and resource contention of a system.

Opportunities for Optimisation: Through the analysis of simulation results, organisations can pinpoint potential areas for improvement, refine scheduling procedures, and improve resource management to improve fairness and efficiency in their big data processing systems. Using machine learning, simulation, and visualisation approaches in the context of large data processing is necessary to develop a fair scheduler. To achieve their performance and fairness objectives, organisations can use these tools to establish, assess, and constantly improve their scheduling rules. A well-designed scheduler improves system performance while simultaneously ensuring equitable resource access, which helps large data processing initiatives succeed in the long run.

REFERENCES:

[1] Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., & Byers, A. H. (2011). Big data: The next frontier for innovation, competition, and productivity. McKinsey Global Institute.

[2] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster computing with working sets. HotCloud, 10(10-10), 95.

[3] Zaharia, M., Konwinski, A., Joseph, A. D., Katz, R. H., & Stoica, I. (2008). Improving MapReduce performance in heterogeneous environments. OSDI, 8(8), 29-42.

[4] Ghodsi, A., Zaharia, M., Shenker, S., & Stoica, I. (2011). Choosy: Max-min fair sharing for datacenter jobs with constraints. Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, 4(4), 7.

[5] Zaharia, M., Das, T., Li, H., Shenker, S., & Stoica, I. (2012). Discretized streams: Fault-tolerant streaming computation at scale. Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, 423-438.

[6] Dabek, F., Kaashoek, M. F., Karger, D., Morris, R., & Stoica, I. (2003). Wide-area cooperative storage with CFS. ACM SIGOPS Operating Systems Review, 37(SI), 202-215.

[7] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., ... & Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, 2(2), 2-2.

[8] Borthakur, D. (2011). The Hadoop distributed file system: Architecture and design. Hadoop Project Website.

[9] Matei Zaharia, Abhishek Kulkarni, Ali Ghodsi, et al. (2016). "Apache Spark: A Unified Analytics Engine for Big Data Processing," Communications of the ACM, 59(11), 56-65.

[10] Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., ... & Rashid, U. (2013). Apache Hadoop YARN: Yet another resource negotiator. Proceedings of the 4th annual Symposium on Cloud Computing, 5(5), 5-5.

[11] Tanenbaum, A. S., & Van Steen, M. (2007). Distributed Systems: Principles and Paradigms. Pearson.

[12] Kleinberg, J., & Tardos, É. (2005). Algorithm Design. Addison-Wesley.

[13] S. Boyd and L. Vandenberghe. (2004). Convex Optimization. Cambridge University Press.

[14] Lynch, N. A. (1996). Distributed Algorithms. Morgan Kaufmann.

[15] Ghemawat, S., Gobioff, H., & Leung, S. T. (2003). The Google file system. ACM SIGOPS Operating Systems Review, 37(5), 29-43.

[16] Gross, D., & Harris, C. M. (1985). Fundamentals of Queueing Theory (Wiley Series in Probability and Mathematical Statistics). Wiley.

[17] Kleinrock, L. (1975). Queueing Systems: Volume I - Theory. Wiley.

[18] Baker, M., Shah, M., & Jayasumana, A. (1999). A comparative study of scheduling algorithms in the context of real-time multimedia applications. ACM SIGMETRICS Performance Evaluation Review, 27(1), 41-52.

[19] Pinedo, M. L. (2016). Scheduling: Theory, Algorithms, and Systems. Springer.

[20] Jain, R. (1984). A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. Eastern Research Laboratory, Digital Equipment Corporation, Technical Report TR-301.

[21] Anderson, T. E., & Dahlin, M. (1997). A general approach to network protocol stack implementation.

***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 November 2023*

ACM SIGCOMM Computer Communication Review, 27(3), 289-300.

[22] Zhang, H., & Cohen, I. (1993). A study of load balancing in distributed systems. ACM SIGOPS Operating Systems Review, 27(5), 63-74.

[23] Tanenbaum, A. S., & Van Steen, M. (2007). Distributed Systems: Principles and Paradigms. Pearson.

[24] White, T. (2015). Hadoop: The Definitive Guide. O'Reilly Media.

[25] Hindman, B., et al. (2011). Mesos: A platform for fine-grained resource sharing in the data center. NSDI, 11(11), 22.

[26] Zhang, X., et al. (2019). A Survey of Deep Learning for Big Data Processing. ACM Computing Surveys (CSUR), 52(5), 1-39.

[27] Chen, D., et al. (2012). Big data: a survey. Mobile Networks and Applications, 17(2), 171-209.

[28] Calheiros, R. N., et al. (2011). CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience, 41(1), 23-50.

[29] Yin, H., et al. (2013). A systematic survey of the research on cloud computing in education. Computers & Education, 59(2), 1337-1354.

[30] Vavilapalli, V. K., et al. (2013). Apache Hadoop YARN: Yet another resource negotiator. Proceedings of the 4th annual Symposium on Cloud Computing, 5(5), 5-5.

[31] Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction. MIT Press.

[32] Zhang, Q., et al. (2010). Cloud computing: state-of-the-art and research challenges. Journal of Internet Services and Applications, 1(1), 7-18.