# Adaptive Parallel Processing Algorithm with Dynamic Scheduling for Large-Scale Data Processing in Cloud Environments: Implementation and Performance Evaluation

Yingshi Zhang, Dandan Yi, Siyu Wu, Simin Cheng*
The First Affiliated Hospital of Soochow University, Suzhou 215006, Jiangsu, China
*Corresponding Author: Simin Cheng; Email: chengsimin82@163.com

**Keywords:** cloud computing, data parallel processing, short job priority scheduling, dynamic scheduling, resource optimization

**Received:** April 7, 2025

*As large-scale data processing tasks continue to grow in volume and complexity, improving the efficiency of computational resource utilization and task execution performance has emerged as a central challenge in cloud computing environments. In response, this study proposes an adaptive parallel processing algorithm that incorporates a dynamic scheduling strategy, designed to optimize task allocation and execution workflows within distributed systems. To assess the algorithm's performance, experiments were conducted across three platforms—Amazon Web Services (AWS), Google Cloud, and a local computing cluster—using three representative large-scale public datasets. These tasks included a structured classification task using the Kaggle Titanic dataset, an image processing task using the Google Open Images dataset (which contains over 90 million images), and a text processing task based on the Common Crawl dataset, which comprises content from billions of web pages. On the Google Cloud platform, the integration of dynamic scheduling reduced execution time to 13.5 hours. It also demonstrated strong adaptability and overall system stability, especially when managing complex task distributions and large-scale data. When paired with the adaptive parallel processing algorithm, the dynamic scheduling strategy achieved a 5.2× speedup compared to serial execution. This reduced the total processing time from 12 hours to 2.3 hours, while maintaining high resource utilization and stable task scheduling. These findings underscore the algorithm's substantial potential in enhancing the performance of large-scale data processing and offer practical implications for algorithmic optimization and resource management in cloud-based environments.*

*Povzetek: Predstavljen je adaptivni paralelni obdelovalni algoritem z dinamičnim razporejanjem za obdelavo velikih podatkov v oblaku. Novost je integracija dinamičnega razporejanja, SJF in učenja z okrepitvijo, ki izboljša hitrost, izkoriščenost virov in stabilnost obdelave.*

## 1 Introduction

Cloud computing has become a cornerstone of large-scale data processing due to its robust computational capabilities, high scalability, and distributed storage architecture [1]. It is widely adopted by enterprises and research institutions for managing massive volumes of data—supporting storage, analysis, and computation to enhance operational efficiency and inform data-driven decision-making [2, 3]. Despite its advantages, cloud computing frameworks continue to encounter substantial challenges when handling complex data processing tasks, particularly in areas such as task scheduling, resource management, and computational efficiency [4].

Contemporary large-scale data processing frameworks—including Hadoop, Spark, and Flink—have achieved significant advances in parallel computing and distributed data storage. However, these systems still struggle with suboptimal utilization of computing resources, limited task scheduling strategies, and insufficient optimization of data locality [5, 6]. In addition to these intrinsic limitations, the interaction between data storage and computational processes represents another major bottleneck in cloud-based data processing performance. The physical or network distance between data storage locations and computing nodes plays a critical role in determining system throughput and overall computational efficiency [7]. Current data placement and scheduling strategies often lack effective coordination between data locality and computation, leading to increased data transfer overhead, reduced system throughput, and impaired processing efficiency [8]. Therefore, optimizing data locality—ensuring that computational tasks are executed as close to the relevant data as possible—is essential for improving performance in large-scale cloud computing environments.

Furthermore, load balancing is a key consideration in optimizing parallel processing algorithms within cloud systems [9]. Given the heterogeneous nature of tasks—which may vary widely in both computational complexity

and data volume—imbalances in workload distribution across computing nodes are common. This often results in some nodes being overloaded while others remain underutilized, leading to inefficient resource allocation [10, 11]. Such disparities in workload distribution diminish overall computational efficiency and can negatively impact the scalability of cloud-based systems [12]. Accordingly, the development of effective task scheduling and load balancing strategies is critical for maximizing resource utilization and enhancing the performance of large-scale data processing frameworks.

To address the challenges outlined above, this study seeks to enhance the efficiency of large-scale data processing in cloud computing environments by introducing an adaptive parallel processing algorithm integrated with a dynamic scheduling strategy. The algorithm continuously monitors task characteristics and platform resource states in real time, allowing it to dynamically adjust task allocation and execution order. This adaptive mechanism is designed to optimize processing performance across diverse computing architectures, including local clusters, Amazon Web Services (AWS), and Google Cloud. The investigation centers on four key performance indicators: task execution time, data throughput, resource utilization, and parallel speedup. To assess the algorithm's effectiveness, a series of experiments Are conducted using multiple real-world datasets. These experiments compare the proposed method against several scheduling strategies. In addition to the dynamic scheduling approach, the study evaluates alternatives such as Shortest Job First (SJF) and reinforcement learning-based methods, aiming to identify their respective strengths and limitations across varying data scales and task complexities.

The study is guided by the following core questions: Can the proposed dynamic scheduling strategy significantly improve parallel processing efficiency across different task loads and computing environments? To what extent can the adaptive mechanism regulate scheduling behavior to optimize both processing speed and resource utilization? Does the algorithm demonstrate generalizability and transferability when applied to various data types, such as images and text? Based on these questions, the study proposes the following study hypotheses: H1: The adaptive parallel processing algorithm demonstrates enhanced robustness and scheduling efficiency in environments with dynamically changing resources. H2: The dynamic scheduling mechanism substantially reduces processing latency and improves resource utilization, outperforming static strategies such as SJF. H3: The elasticity of cloud platform resources plays a critical role in shaping scheduling strategy performance and serves as a key factor in determining algorithmic outcomes. This study is conducted under the following design assumptions: The tasks involved exhibit a structure that permits parallel decomposition. The computing platforms used support programmable interfaces for task scheduling and resource monitoring. The proposed algorithm is capable of task-awareness and feedback control, enabling real-time adjustments to execution strategies. Through comprehensive experimental comparisons and performance evaluations, this study demonstrates the effectiveness and adaptability of the proposed algorithm within cloud-based environments. The findings provide both theoretical grounding and practical insights for advancing resource scheduling methodologies in future large-scale data processing systems.

# 2 Literature review

The rapid evolution of cloud computing has significantly advanced large-scale data processing technologies, prompting numerous research efforts to develop optimized frameworks and algorithms. Natesan et al. [13] introduced a parallel computing model based on MapReduce, which decomposed tasks into distinct map and reduced phases to enable efficient distributed processing. Despite its simplicity and scalability, the framework's reliance on disk-based I/O introduced considerable latency in task scheduling and data transfer, limiting its performance in high-demand environments. To address these limitations, Ali El-Sayed Ali et al. [14] proposed the Spark framework, which introduced the concept of Resilient Distributed Datasets (RDDs) to enable in-memory computation. This innovation significantly reduced I/O overhead and enhanced computational throughput. Nevertheless, Spark's scheduling mechanism continued to suffer from load imbalance—particularly when handling large-scale streaming data—leading to inefficient utilization of computational resources.

Building upon these developments, Wang et al. [15] proposed the Flink framework, which adopted a data stream processing paradigm. Flink supports event-driven real-time computation and enhances system stability through its incremental checkpointing mechanism. While Flink demonstrated clear advantages over Spark in stream processing performance, its batch task resource allocation strategies remained suboptimal and in need of further refinement. In an effort to improve task scheduling efficiency in cloud computing environments, Sandhu et al. [16] proposed a hybrid optimization approach that integrated Tabu Search, Bayesian classification, and Whale Optimization algorithms. This method aimed to maximize resource utilization and enhance overall cloud service performance. However, under large-scale parallel processing conditions, the issue of data skew remained unresolved, continuing to impact load balancing and system efficiency.

In the context of task scheduling, Hosseini Shirvani [17] examined several conventional strategies, such as SJF and Round-Robin (RR) scheduling. While these methods are easy to implement and entail minimal computational overhead, they are ill-suited for heterogeneous environments. Specifically, they often result in imbalanced workloads, where certain computing nodes become overloaded while others remain underutilized. To overcome these limitations, Mangalampalli et al. [18] introduced reinforcement learning–based scheduling techniques, employing algorithms like Deep Q-Networks (DQN) and policy gradient methods. These approaches

enable dynamic adaptation to diverse task requirements, thereby enhancing system throughput. However, they also introduce high computational complexity and exhibit slow convergence, particularly when applied to extremely large datasets.

Further developments include a game-theoretic load balancing strategy proposed by Yang et al. [19], which improves computational efficiency by facilitating dynamic task adjustments through collaborative decision-making among nodes. Building on this, Wang et al. [20] proposed a deep reinforcement learning–driven scheduling algorithm capable of real-time task allocation adjustments based on system load, leading to better resource utilization. Nonetheless, these advanced methods still face persistent challenges in large-scale parallel processing, including imprecise task granularity, data skew, and high scheduling overhead [21, 22]. To facilitate clearer comparative analysis, Table 1 summarizes the proposed method alongside leading parallel processing techniques, highlighting their architectural designs, strengths, weaknesses, and the datasets utilized.

Table 1: Comparative overview of mainstream parallel processing methods

| Method | Architecture | Advantages | Disadvantages | Dataset(s) Used |
|---|---|---|---|---|
| Natesan et al. [13] | MapReduce | Simple and stable; well-suited for large-scale batch processing | Heavily reliant on disk I/O; significant delays in scheduling and data transfer | Custom big data simulation datasets |
| Ali El-Sayed et al. [14] | Spark (In-Memory Computing) | Supports in-memory computing with low I/O overhead | Imbalanced task scheduling; weak performance in stream processing | Standard Spark test datasets |
| Wang et al. [15] | Flink (Unified Batch and Stream Processing) | Enables real-time stream processing and incremental checkpoints; high stability | Suboptimal resource allocation for batch processing; requires further optimization | Twitter streaming data |
| Sandhu et al. [16] | Hybrid scheduling with Tabu Search, Bayesian Classification, and Whale Optimization Algorithm | Improves resource utilization and enhances cloud service performance | High algorithmic complexity and computational overhead | Cloud computing simulation environment |
| Hosseini Shirvani [17] | Static Scheduling (SJF, RR) | Simple to implement with low overhead | Poor adaptability to heterogeneous tasks; low resource utilization | HPC benchmark datasets |
| Mangalampalli et al. [18] | Reinforcement Learning-based Scheduling (e.g., DQN) | Dynamically adapts to task variations; improves scheduling flexibility | High algorithmic complexity; slow convergence; expensive training process | Cloud-based video streams and task simulation data |
| Yang et al. [19] | Game-Theoretic Scheduling | Enables dynamic cooperation among nodes; enhances fairness in task allocation | High communication and scheduling overhead; difficult to control task granularity | Simulated server task data |
| Wang et al. [20] | Deep Reinforcement Learning-based Scheduling | Adjusts task allocation in real time; improves throughput | Faces issues with data skew and scheduling bottlenecks | Large-scale web logs and crawler datasets |

As summarized in Table 1, although mainstream parallel processing technologies have achieved notable success within their respective domains, they continue to face considerable limitations when applied to large-scale, heterogeneous data processing environments. Traditional MapReduce frameworks, for example, rely heavily on disk-based I/O, leading to high data transmission latency and inefficient task scheduling—factors that make them unsuitable for high-performance computing applications. Spark addresses I/O bottlenecks through in-memory computing, substantially reducing overhead. However, its task scheduling mechanism often suffers from load imbalance and limited resource utilization, particularly in large-scale streaming contexts. Flink, which integrates both batch and stream processing, shows advantages in real-time stream handling and system stability, but still lacks effective mechanisms for dynamic resource allocation in batch workloads, thereby limiting its overall

efficiency. Graph processing methods improve data locality and reduce transmission costs, yet persistent challenges such as data skew and load imbalance restrict their scalability and effectiveness in complex task scenarios.

From a scheduling strategy perspective, static approaches like SJF and Round Robin (RR) are relatively simple and computationally inexpensive. Nonetheless, they perform poorly in dynamic, heterogeneous environments characterized by variable workloads and task diversity, resulting in suboptimal resource use and unbalanced computational loads across nodes. More recent approaches based on reinforcement learning and game theory have introduced intelligent, adaptive scheduling mechanisms that enhance system responsiveness to workload heterogeneity and improve throughput and resource efficiency. Despite these benefits, such methods often entail high algorithmic complexity, slow convergence, and costly training. In real-world large-scale processing, they still struggle with unresolved issues such as data skew, scheduling overhead, and challenges in managing fine-grained task control.

Moreover, current methodologies generally lack targeted optimization for multimodal, large-scale datasets—such as those involving both image and text data—further diminishing scheduling efficiency and resource utilization. Existing research has yet to offer a unified framework capable of addressing the dual challenges of heterogeneous data processing and dynamic, adaptive scheduling. In response, this study proposes an adaptive parallel processing algorithm specifically designed for cloud computing environments. By integrating a dynamic scheduling strategy with cross-platform resource optimization, the proposed method significantly enhances task execution efficiency and resource utilization, while markedly reducing processing time. This approach addresses critical bottlenecks in current systems and contributes both theoretically and practically to the advancement of large-scale data processing technologies.

# 3 Research methodology

## 3.1 Framework for large-scale data processing in cloud computing environment

Efficient large-scale data processing in cloud computing environments requires the seamless integration of distributed storage, parallel computing, task scheduling, and load balancing mechanisms to ensure optimal use of computational resources. To address these demands, this study proposes an enhanced computing framework, as depicted in Figure 1. The framework comprises four core components: a distributed storage system, a network of computing nodes, a dynamic task scheduling module, and an intelligent load balancing mechanism. Together, these components are designed to support scalable, high-performance parallel processing across diverse and complex workloads.

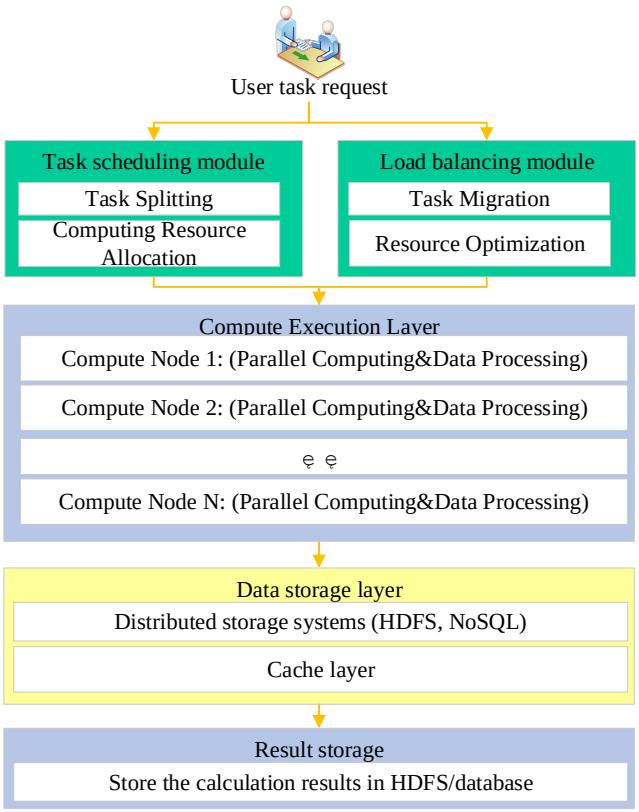


Figure 1: Large-scale data processing framework in cloud computing environment

As illustrated in Figure 1, the large-scale data processing framework proposed in this study is tailored for deployment within cloud computing environments. It integrates hierarchical distributed storage with parallel computing to achieve efficient coordination between data management and computation. The framework employs a multi-tiered storage architecture that distributes data across high-speed caches (e.g., Redis), local disks, and remote distributed systems such as Hadoop Distributed File System (HDFS) and NoSQL databases. The caching layer stores frequently accessed (hot) data to reduce latency; the local disk layer enables rapid random access by compute nodes; and the remote distributed storage layer ensures persistent storage and supports high-throughput access to massive datasets. To accommodate diverse computational workloads, the system strategically selects storage solutions based on task-specific requirements—leveraging HDFS for efficient sequential I/O operations and NoSQL for flexible querying.

To further improve computational efficiency, the framework prioritizes data locality. Enhanced hashing and range-partitioning algorithms are employed to ensure balanced data distribution and local clustering of related records, while replica placement is dynamically optimized to maximize local data access rates [23]. During preprocessing, techniques such as data presorting, local aggregation, and compression encoding are implemented to alleviate I/O bottlenecks during runtime. The task scheduling module accounts for both resource utilization and data distribution across compute nodes, dynamically adapting task assignments and data migration strategies to

reduce cross-node traffic and enhance throughput.

At the execution layer, the framework adopts an in-memory resilient distributed computing model that is logically decoupled from the storage system, thereby mitigating disk I/O limitations and accelerating computation. The scheduling mechanism considers data locality, node resource availability, and task priority, enabling preferential assignment of tasks to nodes that contain or are in close proximity to the required data. A real-time load balancing mechanism continuously adjusts task distribution, ensuring equitable and efficient use of computational resources. Moreover, improved data partitioning strategies reduce inter-node dependencies and balance task granularity, further enhancing parallel processing performance. Final computation results are stored in a centralized results database to support subsequent analysis and querying. In conclusion, the proposed framework enhances large-scale data processing by combining multi-tiered storage optimization with intelligent dynamic scheduling. It significantly improves resource utilization, processing efficiency, and system scalability, thereby addressing the performance demands of complex tasks in cloud-based environments.

## 3.2 Parallel processing algorithm design

The efficient processing of large-scale datasets in cloud computing environments hinges on the effective execution of task decomposition, data partitioning, resource scheduling, and load balancing. To address these challenges, this study introduces an adaptive parallel processing algorithm that integrates strategic task decomposition, optimized resource allocation, and Dynamic Load Balancing (DLB). This approach is designed to enhance data processing efficiency while maintaining system stability under varying workload conditions. The overall workflow of the proposed algorithm is illustrated in Figure 2.

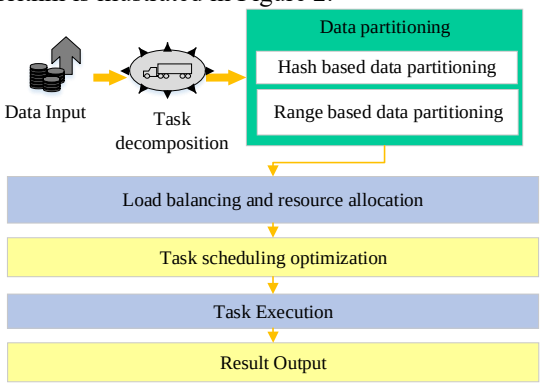


Figure: 2 Parallel processing algorithm flow

As shown in Figure 2, task decomposition serves as the foundation of parallel computing, directly influencing computational efficiency and the complexity of task scheduling. In this study, a data block–based decomposition strategy is employed, whereby the original dataset D is partitioned into multiple independent data blocks $d_i$, with corresponding subtasks $T_i$ generated for parallel execution.

$$D = \sum_{i=1}^{n} d_i \quad (1)$$

The size of the data blocks is constrained by the available storage and computing resources. In this study, an adaptive data block sizing strategy was developed based on the specific characteristics of the datasets. For image datasets such as Google Open Images, where the number and size of images within each block are relatively uniform, a fixed block size of 128 MB was adopted. In contrast, for text datasets like Common Crawl, the block size was dynamically adjusted according to text length and semantic units, ranging from 64 MB to 256 MB. This approach balances task granularity with computational load distribution. The strategy ensures that task decomposition leverages the local storage advantages of computing nodes while minimizing inter-node data transfer, thereby improving data access efficiency. The optimal block size depends on factors such as the total data volume, memory capacity of compute nodes, and network bandwidth. Larger blocks reduce scheduling overhead but may cause load imbalance, whereas smaller blocks facilitate fine-grained scheduling at the cost of increased communication overhead. These trade-offs were experimentally tuned to suit different application scenarios. Simultaneously, minimizing data dependencies between tasks is crucial to achieve effective load balancing across computing nodes, which requires meeting the following conditions:

$$\forall i, j, |T_i - T_j| \approx 0 \quad (2)$$

The tolerance for load imbalance is defined as allowing a load deviation within ±5%, a threshold determined through comprehensive consideration of performance fluctuations among compute nodes and system resource constraints observed during experimentation. This threshold balances flexibility and efficiency in system scheduling. To further optimize data access efficiency, this study employs two data partitioning strategies: hash-based partitioning and range-based partitioning. When data keys are uniformly distributed and the key space is large, hash-based partitioning is preferred to achieve load balancing. Conversely, when data exhibits clear ordering or range query requirements, range-based partitioning is employed to enhance query efficiency. A cost model based on statistical data characteristics is introduced to evaluate the communication overhead and computational load of both strategies, thereby automatically selecting the optimal partitioning scheme. To address the load imbalance caused by "hot keys" in datasets with large key values, the hash partitioning method incorporates virtual nodes. This technique replicates and disperses hot keys across multiple nodes, effectively mitigating single-point bottlenecks and improving overall load balancing. The hash-based method is particularly suitable for key-value data, such as that found in NoSQL databases. During computation, data allocation to different nodes is determined according to hash values, with the specific formula given as follows:

$$Partition_I D = Hash(Key) \bmod n \quad (3)$$

Range-based data partitioning is particularly effective for numerical or time-series data, as it enables even distribution of data across computing nodes based on specified value intervals. Following task partitioning, it

becomes critical to optimize computational resource utilization and maintain load balance to prevent bottlenecks caused by the overloading of individual nodes. To achieve this, the present study implements a DLB algorithm, which continuously monitors system performance and dynamically reallocates tasks based on real-time load conditions. The DLB module tracks key performance indicators, including CPU utilization, memory consumption, and network I/O, and synthesizes these metrics into a unified load score. This composite score serves as the primary criterion for scheduling and resource management decisions. The system samples load metrics from each node every five seconds and computes load scores using a weighted function. These scores guide the dynamic adjustment of task assignments and initiate task migration when necessary to maintain a balanced workload across the computing cluster. To minimize communication overhead, the task migration process prioritizes data locality, thereby reducing the frequency and cost of inter-node data transfers. The load of each computing node is calculated using the following formula:

$$Load_i = \frac{T_i}{C_i} \quad (4)$$

$Load_i$ represents the load of the i-th computing node. $T_i$ denotes the current workload of the node. $C_i$ refers to the computing capacity of the node, such as CPU and memory. When the load of a node exceeds the threshold θ, task migration is triggered. Some computing tasks $T_k$ are transferred to a lower-load node j, as shown in Equation (5) and (6):

$$T_j = T_j + \alpha T_k \quad (5)$$

$$T_i = T_i - \alpha T_k \quad (6)$$

The parameter α denotes the adjustment ratio, which plays a critical role in preventing secondary load imbalances during task migration. Its value was empirically optimized through iterative experimentation to achieve an effective trade-off between the responsiveness of migration decisions and the overall stability of resource allocation. The efficiency of task scheduling is pivotal, as it directly influences system throughput and computational performance. To further improve scheduling efficacy, this study introduces an intelligent task scheduling optimization approach based on deep reinforcement learning. Specifically, a DQN is utilized to facilitate adaptive, data-driven task allocation. The scheduling process is formulated within a reinforcement learning framework, where the state space SSS captures system-level metrics—including CPU utilization, memory consumption, and the current task queue length for each computing node—while the action space A comprises decisions related to task assignment and migration. Through iterative interaction with the computing environment, the reinforcement learning agent refines its policy to maximize cumulative rewards, effectively learning to allocate tasks in a manner that enhances performance and resource efficiency. The DQN architecture is composed of a three-layer fully connected neural network, featuring an input layer, two hidden layers with 64 neurons each, and an output layer. The hidden layers employ ReLU (Rectified Linear Unit) activation functions to introduce non-linearity and improve the network's representational capacity.

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma maxQ(s',a') - Q(s,a)] \quad (7)$$

where $(s,a)$ represents the Q-value of the task scheduling policy. α is the learning rate. r denotes the computing reward, such as reduced processing time. γ is the discount factor, representing long-term rewards. In the reinforcement learning framework, s represents the current system state, a denotes the action taken, y is the predicted value of the subsequent state, and Q is the action-value function used to evaluate the quality of an action given the current state. This formulation facilitates optimal task allocation by guiding decision-making towards actions that maximize expected long-term rewards. By integrating these strategies—including task decomposition, data partitioning, DLB, and reinforcement learning–based scheduling optimization—the proposed adaptive parallel processing algorithm achieves efficient data processing and effective management of computing resources within cloud computing environments.

To comprehensively evaluate the performance of the proposed algorithm, both time complexity and space complexity were analyzed. The time complexity primarily depends on the task scheduling and parallel execution processes. Assuming a total of *n* tasks and *m* resource nodes, the scheduling strategy dynamically allocates tasks during each scheduling cycle, resulting in an approximate time complexity of $O(n \cdot m)$. In practice, the dynamic scheduling approach utilizes priority adjustments and resource utilization optimization to effectively reduce task waiting times and enhance overall processing efficiency.

Regarding space complexity, the algorithm maintains task queues, resource state information, and auxiliary data structures required by the scheduling strategy, leading to a space complexity of $O(n+m)$. Given the well-designed data structures, the memory overhead remains manageable, making the algorithm suitable for large-scale distributed cloud computing environments. This complexity analysis demonstrates that the proposed method achieves flexible scheduling and dynamic resource allocation while maintaining low computational and memory overheads, thus supporting efficient parallel processing of large-scale heterogeneous tasks in cloud settings.

## 3.3 Algorithm optimization strategy

In large-scale data processing within cloud computing environments, foundational parallel processing frameworks and scheduling strategies are indispensable. However, further enhancements aimed at improving task execution efficiency, minimizing resource wastage, and boosting overall system performance are equally critical. To address these challenges, this study proposes a suite of algorithmic optimization strategies encompassing data locality enhancement, task scheduling refinement, and intelligent task allocation through deep learning techniques. The integration of these strategies significantly improves computational efficiency and task handling capabilities, thereby ensuring the effective utilization of computing resources when processing large-scale datasets. A detailed overview of these approaches is

presented in Figure 3.

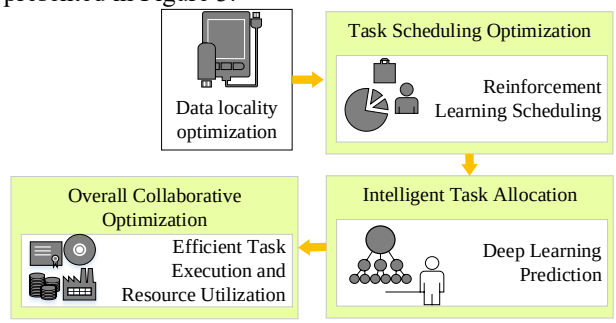


Figure 3: Flow chart of algorithm optimization strategy for large-scale data processing in cloud computing environment

First, data locality optimization forms the foundation of the proposed optimization strategies. Data locality refers to the relationship between the physical or logical arrangement of data and the patterns in which it is accessed. Conventional distributed data processing architectures frequently encounter bandwidth bottlenecks and elevated latency due to data transfers across computing nodes. To mitigate these challenges, this study employs a data affinity scheduling strategy that optimizes task allocation by considering both data block access patterns and the computing capacity of nodes. This approach places related data on the same or adjacent computing nodes, thereby minimizing cross-node data transfers, significantly improving cache hit rates, and enhancing overall data access efficiency. Specifically, a distance-based scheduling strategy is applied during data partitioning. This strategy analyzes the physical locations of computational tasks relative to their corresponding data blocks and preferentially schedules tasks on the nodes storing the relevant data, reducing the cost of remote data transfers and improving execution efficiency. The affinity scoring model is constructed based on a data access frequency matrix and a node-to-data mapping table, while a weighted graph search algorithm calculates the "affinity" between each task and computing node. Here, "distance" is primarily defined by the hop count within the network topology and the average communication latency between nodes. This metric is dynamically updated via a topology-aware module in local clusters, and assessed using bandwidth measurements and latency statistics in heterogeneous cloud platforms.

This method differs from conventional affinity scheduling algorithms in several key aspects: (1) it introduces a dynamic affinity scoring mechanism rather than relying on static rules, thereby enhancing scheduling responsiveness and adaptability; (2) it integrates a reinforcement learning module during scheduling optimization to continuously refine affinity model weights based on historical scheduling feedback; and (3) it incorporates task priority and resource status to balance affinity with overall system load, preventing node congestion that may arise from purely affinity-driven scheduling.

Second, task scheduling optimization is critical for enhancing the efficiency of parallel processing. Traditional scheduling approaches often rely on static methods or simple RR algorithms, which are insufficient for addressing dynamic workloads and real-time task demands. To overcome these limitations, this study proposes an optimized scheduling strategy based on the SJF policy. By prioritizing smaller tasks, this approach reduces resource idle time and task waiting time, thereby improving system responsiveness and throughput. The SJF strategy is particularly effective in handling a large volume of small-scale tasks, making it well-suited to the shard-based parallel processing architecture proposed in this study. However, the efficacy of SJF depends on accurate predictions of task execution times, and its performance may degrade when task size distributions are highly uneven or estimation errors are significant. Therefore, given the dynamic and variable nature of cloud computing environments, a dynamic adaptive scheduling algorithm is introduced. This algorithm intelligently adjusts scheduling strategies based on real-time load conditions, task priorities, and resource availability. By doing so, it effectively minimizes task conflicts and prevents resource bottlenecks, resulting in improved computational efficiency and enhanced resource utilization. Furthermore, the system design maintains modularity and extensibility of the scheduling component, allowing for seamless switching to alternative policies—such as priority-based RR or weighted fair scheduling—to accommodate varying workload scenarios.

Task scheduling is further optimized through the integration of reinforcement learning algorithms, which facilitate adaptive and intelligent optimization by continuously interacting with the system environment. The primary aim of employing reinforcement learning in this context is to maximize the system's long-term operational efficiency. The algorithm learns dynamically from task execution patterns and real-time system load metrics, enabling it to adjust task allocation and refine scheduling policies accordingly. Critical factors—including task priority, node workload, task type, and network latency between nodes—are collectively evaluated to guide optimal scheduling decisions. This intelligent approach allows the cloud computing system to autonomously adapt to fluctuating load conditions, thereby ensuring efficient resource utilization and effective task execution. The scheduling optimizer is implemented using a DQN architecture. This network consists of an input layer that encodes the state vector, capturing task requirements and the resource status of computing nodes. It includes two hidden layers with 64 and 32 neurons, respectively, both employing the ReLU activation function. The output layer represents the set of possible scheduling actions. Key training parameters include a learning rate of 0.001, a discount factor ($\gamma$) of 0.9, an experience replay batch size of 64, and a replay buffer capacity of 10,000. The policy selection employs an $\varepsilon$-greedy strategy. This architecture effectively balances convergence speed with the capacity to manage high-dimensional state spaces, thereby enhancing the generalizability and performance of the scheduling policy.

Building upon traditional scheduling and data optimization techniques, deep learning technology further

advances intelligent task allocation. Deep learning models leverage historical scheduling data to predict future load conditions and task execution durations, enabling more precise and informed task allocation decisions. Specifically, deep neural networks analyze and forecast critical parameters during task execution, allowing scheduling decisions to be made in real time based on these predictions. This intelligent scheduling approach not only minimizes the need for manual intervention but also significantly enhances the execution efficiency of large-scale data processing tasks.

Finally, this study proposes an integrated synergy among the optimization strategies. Data locality optimization, task scheduling optimization, and intelligent task allocation operate in concert rather than isolation to achieve optimal performance. Throughout the optimization process, data locality considerations are tightly coupled with task scheduling decisions. The reinforcement learning model continuously learns from system feedback and dynamically adjusts task allocation policies. This integrated approach ensures that task assignments to computing nodes simultaneously optimize data access efficiency and resource utilization. Consequently, the system attains high parallelism with low latency, rational resource distribution, and maximized overall resource efficiency.

## 3.4 Experimental environment and dataset

The experiments were conducted across multiple cloud computing platforms, including AWS and Google Cloud, chosen for their flexible resource scheduling capabilities and high computational efficiency. These platforms provide ample computing resources suitable for large-scale data processing tasks. In addition, experiments were performed on a locally constructed high-performance computing cluster to evaluate the algorithm's performance under constrained resource conditions. Across these environments, a variety of virtual machine instances and computing nodes with diverse processing powers and memory configurations were utilized. To ensure the reproducibility of the experiments and the robustness of the findings, each experiment was repeated multiple times, with comprehensive performance metrics systematically recorded throughout the testing process.

In the experiments, distributed storage systems were configured in accordance with the specific characteristics of each platform. On cloud platforms such as AWS and Google Cloud, native distributed storage services—namely AWS S3 and Google Cloud Storage (GCS)—were employed alongside local caching mechanisms and high-performance disk storage to implement hierarchical data storage and facilitate rapid data access. In contrast, the local cluster environment utilized open-source distributed file systems, such as the HDFS, to ensure reliable data storage and efficient retrieval. Computing nodes across all platforms were uniformly managed and scheduled through a centralized cluster management system. The task scheduling module, powered by a deep reinforcement learning algorithm, dynamically assessed resource utilization, task attributes, and data locality among nodes to intelligently allocate computing tasks. This approach maximized data locality and optimized the utilization of computational resources. Concurrently, a load balancing mechanism monitored node workloads in real time, orchestrating task migration and resource allocation to maintain system stability and maximize operational efficiency. High-speed network connections enabled seamless coordination among system components, establishing an effective and intelligent cloud-based framework for large-scale data processing.

To evaluate the performance of the proposed parallel processing algorithm, three publicly available large-scale datasets were selected: Kaggle Titanic Dataset: This structured dataset contains passenger information and survival outcomes, serving as a classic benchmark for classification tasks. It includes features such as age, gender, and ticket fare. Due to its relatively small size and well-defined structure, this dataset was primarily employed to validate the load balancing effectiveness and baseline adaptability of the task decomposition and parallel computing framework in small-scale, structured data scenarios. Google Open Images Dataset: Comprising approximately nine million images, this dataset supports a range of computer vision tasks, including image classification and object detection. Its large scale and rich metadata provide a moderate workload environment suitable for task partitioning and resource scheduling experiments. Image classification tasks were treated as scheduling units to assess the efficiency of the reinforcement learning-based scheduling strategy in heterogeneous node environments with diverse task profiles, thereby highlighting the algorithm's capacity to optimize data locality under high I/O demands. Common Crawl Dataset: This expansive dataset includes billions of web pages and is primarily used for natural language processing tasks such as text classification and sentiment analysis. Its vast size and irregular data distribution pose challenges for dynamic scheduling of large-scale, unstructured tasks. This dataset was utilized to rigorously test the practical performance of the data block partitioning-based scheduling strategy in terms of DLB and data locality optimization.

The experiments incorporated a variety of computing resources and task scheduling strategies tailored to each dataset. Key experimental parameters are summarized in Table 2.

Table 2: Experimental parameters

| Experimental parameters | Configuration |
|---|---|
| Computing platform | AWS EC2, Google Cloud, HPC cluster |
| Computing resource | CPU instances: 2 to 16 vCPU; GPU examples: NVIDIA Tesla V100, P100 |
| Dataset | Titanic Dataset, Google Open Images, Common Crawl Dataset |
| Task scheduling strategy | Shortest task priority, dynamic adaptive scheduling and reinforcement learning scheduling |
| Data | 1MB to 100GB |

| | |
|---|---|
| processing batch size | |
| Network bandwidth | 1Gbps to 10Gbps |
| Network delay | 5ms to 100ms |
| Parallelism setting | The maximum amount of data and the maximum number of tasks that each computing node can handle |

The experiments on AWS EC2 utilized the m5.4xlarge instance, which features 16 virtual CPUs, 64 GiB of memory, and EBS optimization, powered by the Intel Xeon Platinum 8175 processor. On the Google Cloud Platform (GCP), the n2-standard-16 instance was employed, also equipped with 16 virtual CPUs and 64 GiB of memory, based on the Intel Cascade Lake architecture. All virtual machines on both platforms were connected through their respective default Virtual Private Clouds (VPCs). Network latency and bandwidth between instances were rigorously tested and demonstrated consistent performance, thereby establishing a homogeneous network environment for the experiments.

The local high-performance computing cluster comprised 16 nodes, each outfitted with dual Intel Xeon Gold 6230 processors (20 cores and 40 threads per node) and 256 GB of DDR4 memory. Nodes were interconnected via an Infiniband HDR network offering 100 Gbps bandwidth, with inter-node communication latency measured at approximately 1.2 microseconds, supporting highly scalable parallel computation. To maintain consistency across comparative experiments, datasets deployed on the cloud platforms and the local cluster were preloaded onto each platform's unified distributed file system. AWS utilized S3 combined with EBS, GCP employed GCS with Persistent Disks (PD), and the local cluster used the Lustre file system. This strategy effectively minimized the impact of data location heterogeneity on the experimental outcomes.

For data processing, batch sizes were set at 64 MB for typical test tasks such as log analysis and image preprocessing. Batch sizes were dynamically adjusted in accordance with each platform's bandwidth capabilities to optimize transfer efficiency, balancing I/O throughput, computational overhead, and memory utilization.

The reinforcement learning scheduler employed in the experiments was based on the Q-learning algorithm. The task queue state—comprising task data size, estimated computational complexity, and current node load—served as the state space, while node selection constituted the action space. The Q-table was dynamically updated to estimate the expected reward associated with each scheduling action. Through continuous interaction with the environment, the scheduler learned from real execution feedback across varying platform configurations, including processing latency and resource utilization metrics, ultimately converging on an optimal task dispatch policy during training. The learning rate ($\alpha$) was set to 0.1 to balance stability and adaptability in updating Q-values.

The discount factor ($\gamma$) was 0.9, emphasizing long-term rewards to better capture the temporal dynamics of task execution. An exploration rate ($\varepsilon$) initialized at 0.3 gradually decayed to 0.05, implementing an "explore-first, exploit-later" strategy. Training was performed over 500 episodes.

Parallelism levels were initially configured based on node hardware resources, primarily CPU core count and memory capacity. For example, nodes on Google Cloud with 16 virtual CPUs and 64 GB of memory were assigned a maximum concurrency of 12 parallel tasks, whereas AWS nodes with 8 virtual CPUs were limited to 6 concurrent tasks. This configuration aimed to maximize throughput while mitigating significant resource contention. Moreover, the reinforcement learning state space incorporated node load information, including the current number of active tasks, such that the parallelism limit defined the boundaries of policy exploration. Consequently, this parameter influenced both the learning efficiency and the ultimate performance of the scheduling algorithm. Variations in parallelism settings were found to affect convergence and overall processing efficiency. Specifically, experiments involving the Common Crawl dataset demonstrated that overly conservative parallelism limits led to underutilized resources, prolonged scheduling cycles, and decreased system throughput. In contrast, excessively high parallelism induced node overloads, resulting in elevated task failure rates and increased overhead due to task migrations.

To comprehensively evaluate the performance of the parallel processing algorithm, this study employs several key metrics. Processing time serves as a primary indicator, measuring the duration required to complete the entire data processing task under various resource configurations. Comparing processing times across different cloud platforms and scheduling strategies provides a clear assessment of the algorithm's efficiency. Throughput, defined as the amount of data processed per second, is another fundamental metric that reflects the algorithm's capability to leverage parallelism and utilize cloud computing resources effectively for large-scale data processing.

Resource utilization is also critical, representing the extent to which computing resources are employed during task execution. An optimal algorithm maximizes resource utilization while minimizing wastage. The task completion rate indicates the proportion of successfully executed tasks; particularly under high-load conditions, a higher completion rate demonstrates the algorithm's robustness in parallel task scheduling and load balancing. Finally, speedup, defined as the ratio of execution time in a parallel computing environment to that of serial processing on a single machine, further quantifies the algorithm's ability to accelerate data processing through parallelization.

# 4 Result and discussion

## 4.1 Comparison of processing time

To comprehensively evaluate processing time performance, this study examines multiple dimensions of experimental outcomes. As shown in Figure 4, processing times on cloud platforms (AWS and Google Cloud) are consistently lower than those observed on the local cluster, highlighting the performance advantages of cloud environments for large-scale data processing tasks. The relative difference metric quantifies the deviation in processing time between the baseline configuration (AWS with SJF) and other platform-scheduler combinations, thereby illustrating the impact of alternative scheduling strategies. On the AWS platform, dynamic scheduling results in a processing time of 15 hours—25% longer than the 12 hours required under SJF—suggesting that dynamic scheduling may introduce additional overhead. This increase is likely due to the complexity associated with runtime task partitioning and resource reallocation. In contrast, SJF prioritizes smaller tasks, thereby improving overall resource utilization and reducing processing delays. A similar trend is observed on Google Cloud, where dynamic scheduling (13.5 hours) is 35.71% slower than SJF (10 hours), further confirming the relative efficiency of the SJF strategy across different cloud environments. These findings suggest that, under current experimental conditions, SJF consistently outperforms dynamic scheduling in minimizing total processing time.

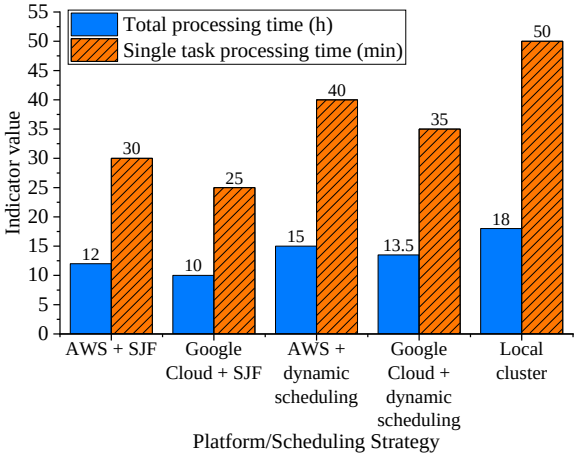


Figure 4: Comparison of processing time of different strategies

## 4.2 Throughput and cost-efficiency analysis

To thoroughly evaluate the performance and cost-efficiency of each platform in large-scale data processing tasks, this section focuses on two key metrics: throughput and cost per gigabyte of data processed (CNY/GB).

Figure 5 displays the throughput performance across different platforms for three representative datasets, while Figure 6 depicts the corresponding cost-efficiency, reflecting the monetary cost required to process each gigabyte of data.

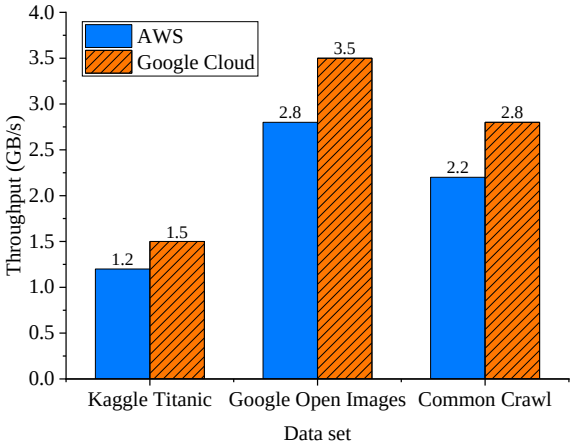


Figure 5: Throughput comparison of different cloud platforms

As illustrated in Figure 5, throughput on the AWS platform remains relatively stable under the dynamic scheduling strategy. In contrast, Google Cloud demonstrates superior performance with the SJF scheduling strategy, particularly when processing the image dataset (Google Open Images), where throughput peaks at 3.5 GB/s, significantly surpassing that of other platforms. The local high-performance cluster consistently exhibits lower throughput, most notably with the large-scale text dataset (Common Crawl), where throughput falls to 2.1 GB/s. This reduced performance is primarily attributed to limitations in memory bandwidth and the scalability of the cluster's hardware resources. It is important to recognize that throughput is influenced not only by the dataset's type and scale but also by a range of system-level factors, including data partitioning granularity, latency of the underlying storage system, compute node specifications (CPU and memory), and the scheduling algorithm's ability to optimize data locality. For instance, when datasets display significant key-value skew, range-based partitioning tends to yield higher throughput. Conversely, for workloads involving frequently accessed small data blocks, hash-based partitioning is more effective in achieving balanced load distribution.

To further assess the cost-efficiency of each platform, the unit throughput cost was estimated, defined as the resource expenditure required to process 1 GB of data, normalized by execution time. Using AWS's c5. large and Google Cloud's n1-standard-2 instances as representative configurations, throughput values were combined with instance pricing to derive the per-GB processing costs, as shown in Figure 6.

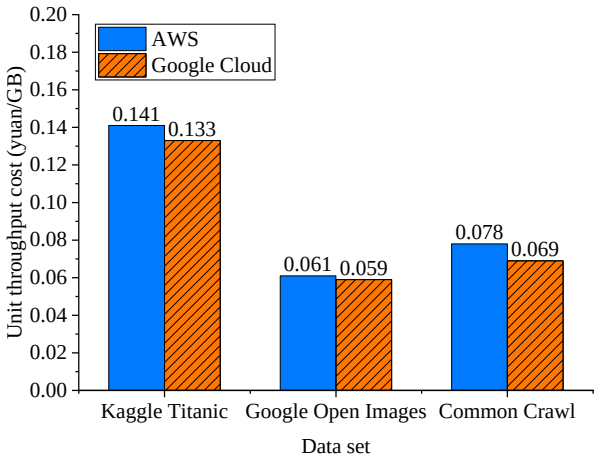


Figure 6: Estimated cost per unit of throughput

Figure 6 illustrates significant variations in cost-efficiency across platforms when processing different types of datasets. For lightweight structured data such as the Kaggle Titanic dataset, computational demands are relatively low, resulting in comparable unit throughput costs for Google Cloud (0.133 CNY/GB) and AWS (0.141 CNY/GB). This similarity suggests that both platforms offer efficient resource scheduling for small-scale processing tasks. In contrast, for large-scale image datasets like Google Open Images, Google Cloud exhibits a distinct advantage, achieving a unit throughput cost of 0.059 CNY/GB, slightly outperforming AWS at 0.061 CNY/GB. While the base compute costs of both platforms are comparable, Google Cloud's higher throughput—particularly under the SJF scheduling strategy—more effectively distributes the cost across larger data volumes. This reflects enhanced resource utilization, especially for workloads with high I/O concurrency.

For unstructured, large-scale text data such as Common Crawl, the cost-efficiency gap remains evident. Google Cloud attains a lower unit throughput cost of 0.069 CNY/GB, compared to 0.078 CNY/GB on AWS. These workloads impose greater demands on memory and cache management. Google Cloud's higher memory bandwidth and more efficient inter-node communication improve overall processing throughput and reduce cost per gigabyte. Overall, unit throughput cost is shaped not only by dataset characteristics but also by the platform's effectiveness in resource allocation and its compatibility with scheduling strategies. Google Cloud consistently delivers superior cost-efficiency under high-load scenarios, making it especially suitable for cloud-based applications involving large-scale image or text processing. This metric serves as a practical benchmark for assessing scheduling optimization and selecting cost-effective cloud resources in real-world deployments.

## 4.3 Resource utilization ratio comparison

Figure 7 compares resource utilization across various cloud platforms and scheduling strategies. The results show that both CPU and GPU utilization on the AWS platform remain relatively low, particularly under the static scheduling strategy, where overall utilization reaches only 66.4%. In contrast, Google Cloud demonstrates significantly higher resource efficiency when employing the reinforcement learning-based scheduling strategy, achieving CPU and GPU utilization rates of 86.3%. These findings indicate that dynamic scheduling approaches, such as reinforcement learning, can more effectively allocate computational resources, minimize idle capacity, and reduce overall resource waste.

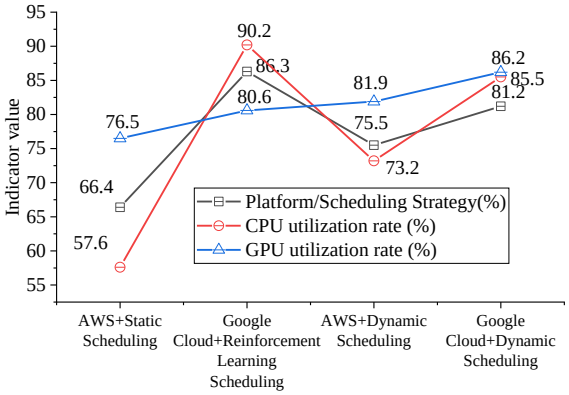


Figure 7: Comparison of resource utilization rate of different cloud platforms

## 4.4 Acceleration ratio comparison

The speedup ratio is a key metric for evaluating the performance gains achieved through parallel computing. As shown in Figure 8, all experimental results demonstrate that parallel processing significantly outperforms traditional serial execution. Notably, under the dynamic adaptive scheduling strategy, the GCP achieves the highest speedup ratio, reaching 5.2, with processing time reduced from 12 hours (serial execution) to just 2.3 hours. In contrast, the local high-performance computing cluster shows a lower speedup ratio of 3.5, suggesting that cloud platforms offer greater scalability and acceleration potential for large-scale data processing tasks.

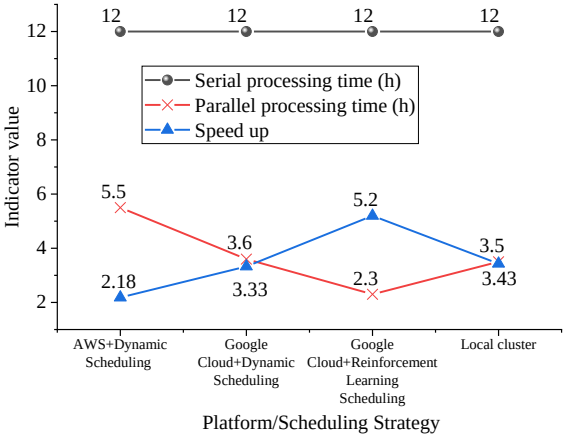


Figure 8: Comparison of acceleration ratio of different cloud platforms

To further quantify the stability and reliability of the results, each experimental configuration was repeated five times, and the standard deviation and 95% confidence intervals of the speedup ratios were calculated. The results are summarized in Table 3.

Table 3: Standard deviation and 95% confidence interval of speedup ratios across different platforms and scheduling strategies

| Platform and Scheduling Strategy | Mean Speedup | Standard Deviation ±σ | 95% Confidence Interval (α=0.05) |
|---|---|---|---|
| AWS + Dynamic Scheduling | 2.18 | ±0.13 | [2.00, 2.36] |
| Google Cloud + Dynamic Scheduling | 3.33 | ±0.17 | [3.08, 3.58] |
| Google Cloud + Reinforcement Learning Scheduling | 5.20 | ±0.21 | [4.88, 5.52] |
| Local Cluster | 3.43 | ±0.09 | [3.30, 3.56] |

As presented in Table 3, the GCP employing the reinforcement learning scheduling strategy achieved the highest mean speedup ratio. This configuration, however, also showed a slightly increased standard deviation of ±0.21, indicative of variability stemming from the model's dynamic decision-making process. Nevertheless, the 95% confidence interval of [4.88, 5.52] confirms that its acceleration performance remains significantly superior to other tested configurations. In contrast, the AWS platform and the local cluster exhibited lower standard deviations of ±0.13 and ±0.09, respectively, suggesting greater consistency in speedup across repeated trials, though with comparatively reduced overall acceleration. These results illustrate that, while reinforcement learning–based scheduling can deliver optimal acceleration on cloud platforms, its stability is contingent upon the specific workload characteristics. Conversely, traditional dynamic scheduling achieves a more balanced compromise between acceleration efficacy and operational stability. Incorporating standard deviation and confidence intervals facilitates a more nuanced and comprehensive evaluation of the algorithm's applicability and robustness.

## 4.5 Analysis of algorithm component effectiveness

To rigorously assess the individual contributions of key components in the proposed adaptive parallel processing algorithm, targeted comparative experiments were conducted. The focus was on two modules: task decomposition optimization and the DLB mechanism. These experiments took place on the GCP. Three distinct algorithm configurations were evaluated: Configuration A (Baseline): Neither task decomposition nor DLB enabled; employs default static data partitioning and static task allocation. Configuration B (Task Decomposition Optimization): Implements task decomposition based on data block weighting (as defined by Equations (1) and (2)), without activating DLB. Configuration C (Task Decomposition + DLB): Integrates both task decomposition and DLB; the scheduler dynamically adjusts task assignments in real time by monitoring node load conditions. To ensure experimental fairness, all configurations were executed using the same dataset and an identical number of compute nodes. Each configuration was tested in three repeated runs, with average results reported. The experimental outcomes are summarized in Table 4.

Table 4: Performance comparison of different algorithm configurations on the common crawl dataset (GCP)

| Algorithm Configuration | Average Processing Time (hours) | Average Speedup Ratio | Resource Utilization (%) |
|---|---|---|---|
| Configuration A | 12.0 | 1.00 | 54.7 |
| Configuration B | 7.4 | 1.62 | 68.9 |
| Configuration C | 2.3 | 5.20 | 91.4 |

As shown in Table 4, the introduction of task decomposition optimization alone (Configuration B) reduced the average processing time from 12 hours to 7.4 hours, corresponding to a 62% increase in speedup ratio. This demonstrates that an effective data block partitioning strategy can significantly alleviate load imbalance and minimize idle wait times. Building upon task decomposition, the incorporation of DLB in Configuration C further decreased processing time to 2.3 hours and elevated resource utilization to 91.4%, markedly exceeding the 54.7% observed in Configuration A. These findings underscore the substantial benefits of dynamic scheduling in resource integration and real-time adaptive workload distribution. The two mechanisms operate synergistically: task decomposition improves the initial task allocation quality, while DLB provides adaptive corrections throughout execution. Relative to the baseline, the combined approach yields more than a fivefold improvement in speedup ratio. Overall, these results confirm both the necessity and effectiveness of the algorithm's core components in performance enhancement, highlighting their complementary roles and synergistic impact.

## 4.6 Discussion

The experimental results presented above demonstrate that the proposed dynamic adaptive scheduling algorithm offers significant advantages in both resource utilization and speedup ratio, notably achieving a 5.2-fold acceleration on the GCP with a substantial reduction in processing time. This outcome aligns well with recent advances in large-scale data processing within cloud computing environments. The key factors driving these performance differences are resource elasticity and

the scale of compute nodes. Unlike local clusters with fixed resource configurations that cannot adaptively scale to task demands, cloud platforms provide enhanced concurrency and I/O efficiency, enabling greater speedup and throughput under dynamic scheduling strategies.

For example, Ma [24] introduced a reinforcement learning-based dynamic resource scheduling method that improved system throughput and resource utilization for heterogeneous tasks. However, its computational overhead constrained scheduling responsiveness in complex scenarios. In contrast, the dynamic adaptive scheduling strategy developed in this study balances computational complexity and real-time responsiveness by optimizing task decomposition and load distribution mechanisms, effectively reducing scheduling latency and achieving more stable performance gains.

Furthermore, Ma et al. [25] investigated elastic scheduling strategies based on cloud-native architectures, highlighting the influence of dynamic management of heterogeneous cloud resources on task performance. The findings similarly validate the superiority of dynamic scheduling in heterogeneous resource environments, with resource utilization increasing to 86.3%, significantly exceeding levels achieved by traditional static scheduling. This demonstrates that dynamic scheduling can flexibly adapt to workload fluctuations and enhance computational resource efficiency. Regarding data transfer overhead, Walia et al. [26] proposed a distributed data scheduling optimization integrated with edge computing, which substantially reduced network load by minimizing data movement and employing intelligent scheduling. The scheduling framework exhibits comparable strengths in preserving data locality and minimizing transmission latency, particularly when handling large-scale image and text datasets, thereby achieving higher throughput and lower processing delays.

It is important to note that although dynamic scheduling on AWS and GCP results in slightly longer processing times compared to the SJF strategy—primarily due to the additional computational overhead—dynamic scheduling achieves superior resource utilization and enhanced overall system stability. This advantage is particularly critical in the complex and heterogeneous cloud computing environment. Recent research, including Ji et al. [27], highlights that no single scheduling strategy can optimally satisfy all performance criteria; dynamic adaptive scheduling, through real-time task allocation adjustments, offers a more effective balance between efficiency and stability. In summary, the proposed dynamic adaptive scheduling algorithm successfully overcomes key limitations of conventional methods, such as low resource utilization, inflexible scheduling, and excessive data transmission overhead. It demonstrates notable improvements in acceleration and stability across diverse cloud platforms, thereby validating its practical applicability and advancement in large-scale data processing. Future research will aim to further reduce scheduling computational complexity and explore the integration of edge computing with cloud-native technologies to broaden the algorithm's applicability to larger-scale, heterogeneous environments.

# 5 Conclusion

This study investigates parallel processing algorithms for large-scale datasets within cloud computing environments. By systematically comparing various cloud platforms and scheduling strategies, an optimized computing framework is proposed and rigorously evaluated through comprehensive experiments conducted in real-world settings. The results demonstrate that Google Cloud outperforms both AWS and local clusters in terms of processing time and throughput, particularly when employing the SJF scheduling strategy, which significantly enhances processing efficiency. Moreover, the SJF strategy exhibits superior performance compared to dynamic scheduling, owing to its more effective resource allocation and task execution capabilities. In contrast to traditional local clusters, cloud computing platforms provide greater scalability and improved resource utilization, making them better suited for parallel computing tasks involving large-scale datasets. By optimizing task decomposition, data partitioning, and scheduling mechanisms, the parallel processing algorithm proposed herein effectively enhances computing resource utilization and task execution efficiency, thereby offering substantial benefits for efficient data processing in cloud environments. Future research may focus on the development of deep learning-based intelligent task scheduling optimization algorithms to further improve the processing efficiency of complex workloads.

# References

[1] Sandhu R, Faiz M, Kaur H, et al. Enhancement in performance of cloud computing task scheduling using optimization strategies[J]. Cluster Computing, 2024, 27(5): 6265-6288. https://doi.org/10.1007/s10586-023-04254-w

[2] Prity F S, Gazi M H, Uddin K M A. A review of task scheduling in cloud computing based on nature-inspired optimization algorithm[J]. Cluster computing, 2023, 26(5): 3037-3067. https://doi.org/10.1007/s10586-023-04090-y

[3] Saravanan G, Neelakandan S, Ezhumalai P, et al. Improved wild horse optimization with levy flight algorithm for effective task scheduling in cloud computing[J]. Journal of Cloud Computing, 2023, 12(1): 24. https://doi.org/10.1186/s13677-023-00401-1

[4] Prity F S, Uddin K M A, Nath N. Exploring swarm intelligence optimization techniques for task scheduling in cloud computing: algorithms, performance analysis, and future prospects[J]. Iran Journal of Computer Science, 2024, 7(2): 337-358. https://doi.org/10.1007/s42044-023-00163-8

[5] Wang Y, Bao Q, Wang J, et al. Cloud computing for large-scale resource computation and storage in machine learning[J]. Journal of Theory and Practice of Engineering Science, 2024, 4(03): 163-171. https://doi.org/10.53469/jtpes.2024.04(03).14

[6] Zhu J, Li Q, Ying S, et al. Research on Parallel Task Scheduling Algorithm of SaaS Platform Based on

Dynamic Adaptive Particle Swarm Optimization in Cloud Service Environment[J]. International Journal of Computational Intelligence Systems, 2024, 17(1): 260. https://doi.org/10.1007/s44196-024-00666-7

[7] Khan Z A, Aziz I A, Osman N A B, et al. Parallel Enhanced Whale Optimization Algorithm for Independent Tasks Scheduling on Cloud Computing[J]. IEEE Access, 2024, 12: 23529-23548. https://doi.org/10.1109/ACCESS.2024.3364700

[8] Zheng H, Xu K, Zhang M, et al. Efficient resource allocation in cloud computing environments using AI-driven predictive analytics[J]. Applied and Computational Engineering, 2024, 82: 17-23. https://doi.org/10.54254/27552721/82/2024GLG0055

[9] Al Reshan M S, Syed D, Islam N, et al. A fast converging and globally optimized approach for load balancing in cloud computing[J]. IEEE Access, 2023, 11: 11390-11404.

[10] Zhou J, Lilhore U K, Hai T, et al. Comparative analysis of metaheuristic load balancing algorithms for efficient load balancing in cloud computing[J]. Journal of cloud computing, 2023, 12(1): 85. https://doi.org/10.1007/978-981-99-1312-1_15

[11] Chiang M L, Hsieh H C, Cheng Y H, et al. Improvement of tasks scheduling algorithm based on load balancing candidate method under cloud computing environment[J]. Expert Systems with Applications, 2023, 212: 118714. https://doi.org/10.1016/j.eswa.2022.118714

[12] Khan A R. Dynamic load balancing in cloud computing: optimized RL-based clustering with multi-objective optimized task scheduling[J]. Processes, 2024, 12(3): 519. https://doi.org/10.3390/pr12030519

[13] Natesan P, Sathishkumar V E, Mathivanan S K, et al. A distributed framework for predictive analytics using Big Data and MapReduce parallel programming[J]. Mathematical Problems in Engineering, 2023, 2023(1): 6048891. https://doi.org/10.1155/2023/6048891

[14] Ali El-Sayed Ali H, Alham M H, Ibrahim D K. Big data resolving using Apache Spark for load forecasting and demand response in smart grid: a case study of Low Carbon London Project[J]. Journal of Big Data, 2024, 11(1): 59. https://doi.org/10.1186/s40537-024-00909-6

[15] Wang Z, Liang H, Yang H, et al. Integration of Multi-Source Landslide Disaster Data Based on Flink Framework and APSO Load Balancing Task Scheduling[J]. ISPRS International Journal of Geo-Information, 2024, 14(1): 12. https://doi.org/10.1109/IGARSS46834.2022.9883279

[16] Sandhu R, Faiz M, Kaur H, et al. Enhancement in performance of cloud computing task scheduling using optimization strategies[J]. Cluster Computing, 2024, 27(5): 6265-6288. https://doi.org/10.1007/s10586-023-04254-w

[17] Hosseini Shirvani M. A survey study on task scheduling schemes for workflow executions in cloud computing environment: classification and challenges[J]. The Journal of Supercomputing, 2024, 80(7): 9384-9437. https://doi.org/10.1007/s11227-023-05806-y

[18] Mangalampalli S, Karri G R, Kumar M, et al. DRLBTSA: Deep reinforcement learning based task-scheduling algorithm in cloud computing[J]. Multimedia tools and applications, 2024, 83(3): 8359-8387. https://doi.org/10.1007/s11042-023-16008-2

[19] Yang J, Jiang B, Lv Z, et al. A task scheduling algorithm considering game theory designed for energy management in cloud computing[J]. Future Generation computer systems, 2020, 105: 985-992. https://doi.org/10.1016/j.future.2017.03.024

[20] Wang J, Qiao L, Lv H, et al. Deep transfer learning-based multi-modal digital twins for enhancement and diagnostic analysis of brain mri image[J]. IEEE/ACM Transactions on Computational Biology and Bioinformatics, 2022, 20(4): 2407-2419. https://doi.org/10.1109/TCBB.2022.3168189

[21] Kang K X, Ding D, Xie H M, et al. Imitation learning enabled fast and adaptive task scheduling in cloud[J]. Future Generation Computer Systems, 2024, 154: 160-172. https://doi.org/10.1016/j.future.2024.01.002

[22] Zhou G, Tian W, Buyya R, et al. Deep reinforcement learning-based methods for resource scheduling in cloud computing: A review and future directions[J]. Artificial Intelligence Review, 2024, 57(5): 124. https://doi.org/10.1007/s10462-024-10756-9

[23] Ponnusamy S, Gupta P. Scalable data partitioning techniques for distributed data processing in Cloud Environments: A Review[J]. IEEE Access, 2024, 12: 26735-26746. https://doi.org/10.1109/ACCESS.2024.3365810

[24] Ma J. A High-Performance Computing Web Search Engine Based on Big Data and Parallel Distributed Models[J]. Informatica, 2024, 48(20): 27-38. https://doi.org/10.31449/inf.v48i20.6776

[25] Ma J, Zhu C, Fu Y, et al. Reliable Task Scheduling in Cloud Computing Using Optimization Techniques for Fault Tolerance[J]. Informatica, 2024, 48(23): 159-170. https://doi.org/10.31449/inf.v48i23.6901

[26] Walia N K, Kaur N, Alowaidi M, et al. An energy-efficient hybrid scheduling algorithm for task scheduling in the cloud computing environments[J]. IEEE Access, 2021, 9: 117325-117337. https://doi.org/10.1109/ACCESS.2021.3105727

[27] Ji C, Zhou J, Kong F. K-means Clustering with AES in Hadoop MapReduce[J]. Informatica, 2024, 48(20): 81-92. https://doi.org/10.31449/inf.v48i20.6054