Smart Task Scheduling for Cloud-Based Big Data Systems

Nagham Sultan[1*], Wael Hadeed[2], Dhuha Abdullah[3]
[1,2,3] Department of Computer Science, University of Mosul, Mosul, 41002, Iraq
E-mail: naghamajeel@uomosul.edu.iq
*Corresponding author

Keywords: cloud performance, cost optimization, big-data processing, distributed systems, resource management

Received: August 3, 2025

This paper presents a hybrid task scheduling framework for cloud-based big data systems that aims at three main objectives: to improve the system's performance, to decrease the expenses, and to increase the energy efficiency. The conceived system combines a rule-based decision engine with a Long Short-Term Memory (LSTM)-based resource prediction model, enabling real-time job assignment based on task urgency, data locality, and system state. The framework is at the top of Apache YARN; thus, it is compatible with batch jobs (via Hadoop/Spark) as well as streaming tasks (via Kafka/Flink). We reproduced the experiments on a 50-node cluster (n2-standard-16 instances, 16 vCPUs, 64 GB RAM), using real workloads of 100 GB–1 TB batch jobs and 1K–5K event/sec streams. Some of the metrics for evaluating the performance of the experiments are job completion time, throughput, cost per TB processed, and energy consumption (Joules/TB). The results indicate a 32–50% improvement in performance, up to 54% savings in cost when using spot instances, and a 25% reduction in energy consumption compared to baseline schedulers such as YARN, Kubernetes, and Spark.

Povzetek: Predlagana metoda združuje pravila odločanja in LSTM-napovedovanje virov. Dosega občutno hitrejše izvajanje nalog, nižje stroške ter večjo energetsko učinkovitost v primerjavi z obstoječimi rešitvami.

1 Introduction

Big data has exponentially increased to the point that modern computing paradigms are no longer the same, and the IDC expects global data generation to reach 181 zettabytes by 2025 [1]. The data coming from IoT devices, social media, and digital transformation in enterprises creates such a massive flow that the only way to process it effectively is through distributed cloud platforms [2][3]. However, as more organizations migrate their workloads to cloud environments, they encounter weaknesses in existing scheduling architectures, particularly their inability to balance performance requirements with operational cost constraints [4]. The rapid development of cloud computing frameworks such as Hadoop and Spark has enabled virtually limitless scalability [5][6]. Yet, log monitoring in production clusters reveals that 30–45% of resources are wasted because the scheduling mechanisms are inefficient and unchanged [7]. The waste is due to the following three major defects:

- Static Resource Allocation: Systems such as YARN [5] and Mesos rely on fixed resource partitioning and are unable to adjust dynamically to changing workloads. Research [8][9] has previously reported that this results in underutilization of resources during valleys and overutilization during peaks.

- Unidimensional Optimization: The majority of schedulers are tailored to only one objective either to maximize the performance of the system or to minimize the costs but rarely both at the same time because of the difficulty in balancing these opposing goals [10].

- Operational Complexity: Some AI-driven planners signal high-end optimization but are too complicated for a real application; only with a lot of effort and extra infrastructure can they be used [11][12][13]. As an illustration, Fog Bus-like mechanisms offer promising potential, but they are still constrained by high implementation complexity [14]. Furthermore, the money-related problems are very severe: by selecting the wrong VM types or schedules, your costs may increase by 35–60%, and if you are very aggressive in reducing your costs, you may face the occurrence of SLA violations and system performance degradation [4][15].

Task scheduling, the mapping of user tasks to virtual machines and resources, emerges as a central challenge in cloud-based big data systems. Scheduling directly influences execution time, throughput, system load balance, and energy consumption. Despite major advances in virtualization, the scheduling problem is NP-hard, making it computationally complex to solve optimally [16]. Conventional heuristic and mathematical methods do not succeed in scaling when they are confronted with heterogeneous and dynamic big data workloads, whereas existing production schedulers like those in Hadoop and spark are inclined to prioritize fairness over resource efficiency, which leads to the underutilization of the infrastructure [17] [18]. As a result, the decision-makers in the research community have gradually shifted to the use of metaheuristic and intelligent scheduling methods to come up with the closest-to-optimal solutions in a reasonably short time. One of the latest studies illustrates that merging resource allocation with task scheduling can significantly enhance a system's performance, making it an effective strategy [16]. Additionally, Big Data, IoT, and cloud computing convergence introduce even more difficulties. Pal et al. maintain that IoT–cloud ecosystems necessitate schedulers that, in spite of device heterogeneity and bandwidth bottlenecks, result in the least possible end-to-end delay and the highest resource efficiency. Their research on deep learning-based models is the driver of remarkable improvements in QoS as well as adaptive workload distribution [17]. The PAS (Performance-Aware Scheduler) is one of the intelligent frameworks that point out the possibility of achieving a balance of turnaround time with the maintenance of high resource utilization [18].

The main difference between batch and stream workloads is that the latter are continuous systems that are always processing high-velocity data flows; this means that they cannot be delayed for long, and in addition, they should be highly reliable. As an example, pipeline-based linear scheduling can be a winning strategy in real-time big data stream scenarios when memory consumption reduction, congestion avoidance, and throughput improvement are desired [19]. This paper makes contributions as follows:

- Hybrid Scheduling Framework: We unveil a creative hybrid scheduling framework that combines rule-based decision-making with LSTM-based predictive modeling. His framework adjusts in a very interactive way.
- Performance and Cost Efficiency: This is mainly due to the implementation of dynamic task prioritization, resource mapping that is intelligent, and the use of operational pricing data.
- Scalable and Adaptive Design: The system is flexible for both batch (Hadoop/Spark) and streaming (Kafka/Flink) workloads; therefore, it has the capacity for different big data scenarios to scale and adapt.

In order to direct this model, our primary goal is to reduce the time spent on completing work, yet we have to ensure that energy and monetary costs are kept to the minimum in a realistic, large-scale cloud environment. We are able to do this by implementing LSTM-based resource estimation, spot-aware node selection, and DVFS-based energy control.

Section 2 gives a thorough literature survey, which includes detailed coverage of the existing research works regarding cloud and big data task scheduling. Section 3 begins with the explanation of the proposed system architecture that is a three-layer design (application, control, and data plane). Section 4 describes the methodology in detail, including the hybrid scheduling algorithm and program optimization strategies. Section 5 reports on and discusses the results, comparing the system performance with that of baseline schedulers. Section 6 ends the paper, recapping the main themes and indicating possible directions for further research.

2 Related work

This section reviews prior research chronologically, highlighting key contributions and gaps that motivate our work:

The suggestions made by Buyya et al. in 2015 [4] comprised the infrastructure and architectural modifications to the cloud analytics systems.

Their proposal was the first to adopt the concept wherein the system can accommodate spikes and troughs to maximize profits. However, economic factors ignored some options for using resources efficiently and cost-effectively. Ilager et al. in ref. [11] were among the first to exploit the power of AI in the orchestration and management of resources in cloud environments in 2020. The challenge of using such an approach is figuring out where to get the training samples online. Their deep learning algorithm reduces the number of low-level tasks performed by the hypervisor. The method outperforms traditional VM management in terms of response and reduces productive task latency by 60 percent, adding to the time saved by the network. Chen et al. [16], in 2020, introduced a study aiming to utilize the Whale Optimization Algorithm (WOA) for cloud task scheduling; consequently, they came up with a new (IWC) version for improving the solution search range. The result of the simulation demonstrates that IWC outperforms other metaheuristics both in the sense of the convergence speed and the accuracy; further, it achieves better system resource utilization in both small- and large-scale experiments.

Tuli in Ref [14] came up with an inventive idea, Fog Bus, which uses blockchain and edge technologies to create a real-time and secure computing platform in 2020. Nevertheless, the level of consensus delay in the system had gone too far, reaching 300 ms, which already rendered it a non-real-time scheduler. Tantalaki et al. [19] proposed in 2020 a linear, pipeline-based scheduling model for distributed stream processing systems; as an example, the authors mentioned Apache Storm. The main focus of the solution is to manage communication efficiently between different tasks, which will improve throughput by 25-45% compared to other schedulers, and reduce memory usage by up to 45%. The method attributes the increase in performance in big data stream processing to load balancing and buffer diminishing.

In Ref. [2], Berisha et al. proposed performing a comparative study of the analytics techniques deeply in use in cloud-based big data. In one scenario (e.g., with 10 TB of data spread across eight nodes), the authors observed that their main concept significantly reduced the average query time of HDFS from 17.9 to 10.8 ms. However, the drawback was that the nodes were configured according to the initial setup, mainly due to the simple resource allocation model used in this study from 2022. Bal and team in [20] came up with a hybrid machine learning approach (RATS-HM), which not only deals with security but also resource allocation and task scheduling in cloud computing through intelligent co-operation. Along with a remarkable resource utilization, energy-saving, and response time performance over the conventional schedulers, the method significantly shows the advantages of AI and metaheuristics combined for multi-objective optimization. Li et al. 2022 introduced their work PAS, a scheduler for Hadoop YARN and Spark, which is able to adjust policies dynamically based on predicted job completion times. By comparing the system with the latest schedulers, the authors show that their method can shorten the average turnaround time by 25% and the time to complete the whole set of jobs by 15%. This directly leads to performance and resource utilization improvement in big data processing environments [18].

In 2023, Zhu proposed the concept of optimizing network scheduling for large volumes of data workloads through cross-zone traffic pricing, which has been shown to be more affordable [8]. Pal et al. 2023 highlight the difficulty in scheduling tasks for the IoT-cloud environment and come up with the Deep Learning Algorithm for Big Data Task Scheduling System (DLA-BDTSS) as a possible solution. The paper illustrates the usage of a multi-objective strategy to reduce make span and maximize resource consumption, thus outperforming the other algorithms by 8.43%. The proposed method utilizes deep learning to escape from local optima and accelerate convergence speed; hence, it is suitable for scheduling tasks in big data environments [17]. As shown in Table 1, most prior works focus on a single dimension either performance, cost, or energy without addressing them together. None combine dynamic pricing, workload prediction, and power control in a real hybrid workload setting. This work does: it aligns cost-aware placement, LSTM-based estimation, and DVFS-based energy saving in one unified scheduler.

Table 1: Summary of related work and identified research gaps

Study (Author, Year)	Method Utilized	Dataset / Environment	Metrics Evaluated	Main Results	Identified Gaps
Buyya et al., 2015 [4]	Rule-based cloud elasticity	Simulated Cloud Platform	SLA, elasticity, utilization	Scalable architecture for peak/off-peak loads	No cost optimization or energy-awareness
Ilager et al., 2020 [11]	Deep learning for VM orchestration	AI-based VM control, simulation	Latency, response time	60% latency reduction via dynamic control	No integration of cost or energy models
Chen et al., 2020 [16]	Improved Whale Optimization (IWC)	Sim Cloud-based experiments	Convergence speed, resource use	Better accuracy and utilization in large/small task scenarios	Lacks real-time constraints and energy metrics
Tuli, 2020 [14]	Fog Bus with blockchain & edge	Fog + blockchain testbed	Security, latency	Real-time secure framework; but 300ms delay	Not scalable; no cost model; unsuitable for real-time
Tantalaki et al., 2020 [19]	Linear scheduling in stream systems	Apache Storm	Throughput, memory usage	25–45% throughput boost; 45% memory reduction	Focus only on streams, not hybrid workloads
Berisha et al., 2022 [2]	Analytics method comparison (HDFS)	8-node Hadoop Cluster (10 TB)	Query time	Reduced HDFS query time from 17.9 to 10.8 ms	No dynamic resource allocation; static setup
Bal et al., 2022 [20]	Hybrid ML (RATS-HM) for scheduling	ML-enhanced cloud simulation	Resource use, energy, latency	Energy and response time improvements over traditional schedulers	Limited to simulated workloads
Li et al., 2022 [18]	PAS scheduler (job prediction)	Hadoop YARN, Spark	Turnaround time, make span	25% less turnaround; 15% make span reduction	No cost or energy-aware placement
Zhu, 2023 [8]	Network-aware cross-zone scheduling	Multi-zone cloud topology	Pricing efficiency	Cost-effective network-aware task scheduling	Lacks ML or performance adaptation
Pal et al., 2023 [17]	DL-based DLA-BDTSS scheduler	IoT-cloud simulation	Make span, resource consumption	8.43% make span improvement; better convergence	No spot pricing; limited to DL; lacks cost-energy balance
Tantalaki et al., 2020 [19]	Linear scheduling in stream systems	Apache Storm	Throughput, memory usage	25–45% throughput boost; 45% memory reduction	Focus only on streams, not hybrid workloads
Berisha et al., 2022 [2]	Analytics method comparison (HDFS)	8-node Hadoop Cluster (10 TB)	Query time	Reduced HDFS query time from 17.9 to 10.8 ms	No dynamic resource allocation; static setup

Bal et al., 2022 [20]	Hybrid ML (RATS-HM) for scheduling	ML-enhanced cloud simulation	Resource use, energy, latency	Energy and response time improvements over traditional schedulers	Limited to simulated workloads
Li et al., 2022 [18]	PAS scheduler (job prediction)	Hadoop YARN, Spark	Turnaround time, make span	25% less turnaround; 15% make span reduction	No cost or energy-aware placement
Zhu, 2023 [8]	Network-aware cross-zone scheduling	Multi-zone cloud topology	Pricing efficiency	Cost-effective network-aware task scheduling	Lacks ML or performance adaptation
Pal et al., 2023 [17]	DL-based DLA-BDTSS scheduler	IoT-cloud simulation	Make span, resource consumption	8.43% make span improvement; better convergence	No spot pricing; limited to DL; lacks cost-energy balance

3 System overview

We propose a hybrid task scheduling framework for cloud-based big data systems that jointly optimizes performance, cost, and energy. The prototype runs on Apache YARN, supports batch (Hadoop/Spark) and streaming (Kafka/Flink) jobs, and uses three pillars: (1) dynamic priority scoring (urgency + data locality), (2) cost-aware placement using real-time spot prices, and (3) energy-aware execution via ECO/DVFS without violating SLAs. A lightweight LSTM resource estimator guides safe consolidation and node selection.

3.1 System architecture

The smart scheduling system that was conceived is basically a very efficient and effective way to command and manage the execution of tasks in a large-scale cloud-based big data environment. This system comprises three distinct layers: the application layer, the control plane, and the data plane, with each layer serving a specific function aimed at improving performance, reducing costs, and enhancing energy efficiency. The architectural concept is implemented using Apache YARN as a base, is compatible with batch (Hadoop/Spark) and stream (Kafka/Flink) processing frameworks, and makes use of machine learning modules for the anticipatory scheduling decision. The overall system architecture is shown by Figure 1. The architecture design consists of three cooperating layers:

3.1.1 Application layer

This layer is the initial point through which jobs come in, and it manages the main data processing and classification.

a) Job ingress and classification: Via the API, the work is segregated into either latency-sensitive or cost-sensitive categories based on the nature of the job and its SLA requirements.

b) Pre-processing unit: Just the minimal filtering and normalization are applied to the job to be scheduled. For example, time critical jobs are adjusted to hot-tier data paths.

c) LSTM Resource Estimator: A simple, pre-trained LSTM model predicts resource envelopes. (CPU, memory), and expected execution time; thus, placement of the task is done more accurately and resource consolidation becomes efficient.

3.1.2 Control layer

This layer governs decision-making, scheduling policies, and job assignment to optimal compute nodes. It includes the following main components:

a) Hybrid scheduler module: One of the core components of the system is the Hybrid Scheduler, which carries out task scoring, selects the proper nodes for computation, and manages energy-aware execution. Upon the arrival of a new task, the scheduler initiates the creation of a weighted function to determine the task's priority score. This function takes into account and balances the urgency of the task's deadline and its data locality. After the priority is set, the module goes on to decide the most fitting scheduling path according to the kind of task. If the task is found to be cost-sensitive, it is rerouted to the cheapest node available; this process is done by using up-to-date spot pricing data extracted from AWS. On the other hand, the latency-sensitive tasks are taken to the node with the shortest data path where they can receive the fastest execution, with further preference to nodes in the SSD-based hot storage zone for the sake of low-latency execution. Moreover, in a situation where the system is running in ECO mode, the scheduler, by initiating Dynamic Voltage and Frequency Scaling (DVFS), will allow for the selected node to have reduced power consumption.

b) Cost optimizer: This unit is responsible for monitoring real-time spot pricing data and taking adaptive bidding decisions that would lead to the minimum execution cost without compromising the performance or the stability of the system. The spot instance prices were fetched through the AWS EC2 Spot Price History API (which is updated every 5 minutes). To solve the problem of API latency and rate limits, a cache with a 10-minute TTL was used as a fallback that could be relied upon when new data was not available.

c) Broker & admission manager: One of the essential functions this system performs is verifying job requests that come in an authentic way, as well as checking system load (conditions) and dispatching the tasks that have been approved for execution on the data plane in a safe manner. By maintaining secure communications between the control and data planes, it ensures that the scheduling flow continues to be of high quality and reliable.

3.1.3 Data layer

This layer represents the physical and virtual compute resources where jobs are executed.

a) Compute Cluster: The cluster is made up of 50 n2-standard-16 different nodes (16 vCPU, 64 GB RAM).

b) Storage Tiering: Storage in the data plane is managed through a combination of performance tiering and format optimization. Tasks are allocated based on latency requirements to either SSD-backed hot zones for low-latency tasks or HDD-based cold zones for large-scale batch jobs with less sensitivity to latency, while data is stored using a hybrid of HDFS and columnar formats to maximize scan efficiency and throughput during execution.

c) Telemetry Agents: Monitor CPU, memory, I/O, and queuing metrics and send data to the control plane for real-time adaptation.

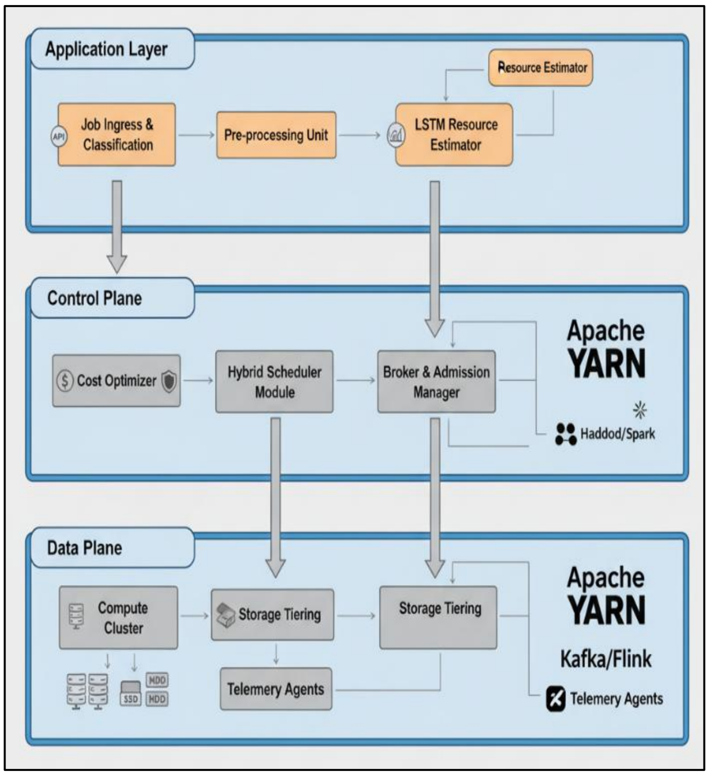

Figure 1: System architecture

3.2. Hardware Components

a) Cluster: 50 nodes of the type n2-standard-16, 16 vCPU, 64 GB RAM.

b) Storage tiers: SSD (hot zone) for latency-critical jobs; HDD (cold zone) for throughput/batch.

c) Coordinator/Broker: The master node that takes care of the submissions, admission, and arbitration.

d) Energy measurement: Intel RAPL was the tool used to measure energy per TB for ECO mode tests.

3.3. Software components

a) Scheduling substrate: Apache YARN.

b) Batch/Stream stack: Hadoop/Spark (batch), Kafka/Flink (stream).

c) Hybrid Scheduler: Priority scorer, node selector (cost/latency paths), ECO/DVFS controller.

d) Cost Optimizer: Real-time AWS/GCP spot price integration with adaptive bidding.

e) Resource Estimator: LSTM model trained on historical executions to predict envelopes/runtime.

f) Monitoring & security: Lightweight agents deployed on each node collect runtime telemetry such as CPU usage, memory consumption, and I/O throughput and report them back to the control plane for adaptive decision-making.

3.4 Implementation and reproducibility

The experiments were all run on a 50-node cluster of n2-standard-16 Google Cloud instances (16 vCPU, 64 GB RAM), operating Ubuntu 20.04. Batch and streaming jobs used Hadoop 3.3.4, Spark 3.4.1, Kafka 3.3.2, and Flink 1.16.2, respectively. Their execution was managed by Apache YARN. The hybrid scheduler, written in Python 3.10, was developed based on YARN Application Master extensions. The LSTM-based resource predictor was first implemented in TensorFlow 2.13, trained offline on 50,000 synthetic task traces emulating Google Cluster workloads, and was then continuously updated via lightweight online feedback loops.

4 Methodology

To address the constantly changing and resource-sensitive nature of scheduling in cloud-based big data environments, we propose a hybrid scheduling methodology that combines rule-based decision-making with learning-driven optimization. The primary goal of this model is to balance the three critical concerns of modern cloud platforms: latency requirements, cost efficiency, and energy awareness.

In order to simulate real-world scenarios, we evaluated the proposed framework on a cluster of 50 nodes running Apache YARN, using a balanced workload of batch processing jobs (100 GB, 1 TB datasets on Hadoop/Spark) and streaming jobs (1K–5K events/sec via Kafka/Flink). A dataset of 50,000 task executions was used to train an LSTM-based resource estimator, which provides predictive guidance for resource envelopes and expected runtimes. The dataset used for training consisted of

50,000 task execution records. Each record included the following features: task type (batch or stream), input data size, SLA deadline slack, cluster node load (CPU and memory), data locality score, and historical runtime. The data was synthetically generated to simulate realistic cloud workloads based on patterns observed in the Google Cluster Trace.

4.1 Workflow

The end-to-end workflow of the proposed system is as follows:

1. Submit: A broker node authenticates the incoming job and extracts metadata (size, SLA constraints, and job type).

2. Tag & Estimate: The Task Tagger classifies the job as either latency-sensitive or cost-sensitive. The LSTM resource estimator predicts its CPU, memory, and runtime requirements.

3. Score: One after the other, tasks are scored according to a value of priority determined on the basis of deadline urgency and data locality with a weighted function, as shown in Eq. 1.

$$priority(T) = wu\,.U(T) + w\ell.L(T) \quad \dots \quad \text{Eq. 1}$$

4. Select Node:
   - Cost-sensitive jobs: The cost optimizer gets the AWS spot pricing API as input and selects the cheapest viable node based on predicted capacity with sufficient resources.
   - Latency-sensitive jobs: The Latency Optimizer examines the cluster topology and data zones to identify the path with the lowest latency and selects the SSD hot zone.

5. Energy Policy: When the cluster works in ECO mode, the node selected goes through the process of Dynamic Voltage and Frequency Scaling (DVFS) to lower power consumption.

6. Dispatch & Monitor: The process is dispatched via YARN. The runtime telemetry (CPU, memory, latency, and throughput) is continuously monitored.

7. Adaptation: Execution traces are returned to the estimator to provide better prediction and to change the scheduling thresholds. The system supports real-time adaptation by employing the lightweight online learning method whereby it updates the LSTM model's scheduling thresholds based on the telemetry it receives. While the main LSTM weights are kept unchanged for the sake of stability, the threshold parameters (e.g., urgency cutoff, predicted runtime deviation bounds) are changed after every 50 job completions using exponential smoothing of recent prediction errors. In this way, the scheduler can adapt to the changes in cluster dynamics without retraining the full model during runtime.

4.2 Hybrid scheduling algorithm

The algorithm in detail explains how the process of optimal node N for task T is done, based on the task type and the current condition of the cluster. Algorithm 1, highlights the decision-making path: first assigning priority, then selecting between cost-aware and latency aware strategies, and finally applying an energy-saving policy if ECO mode is active.

```
Input: Task T, Cluster State C
Output: Optimal Node N
1: T. priority ← calculate_priority(T)
2: if T. type == "cost-sensitive":
3: N←CostOptimizer.select_node (T, C.spot_prices)
4: else:
 5: N←LatencyOptimizer.select_node (T, C.topology)
6: if C.energy_mode == "ECO":
7:     N ← apply_energy_policy(N)
8: return N
```

Algorithm 1 shows the hybrid scheduling with a note on the input: Task T, cluster state C, output: optimal node N. Then the analyzed time complexity is O(n), where n is the number of candidate nodes. The hybrid scheduling algorithm is an orderly means of decision-making used to distribute tasks to cloud resources in the most profitable way without limiting task priority, cost, or energy constraints. The complete process is below:

Step 1: Priority calculation

Every task T is evaluated on the basis of its need and data locality. The weighted priority function is defined as:

$$priority(T) = wu\,.U(T) + w\ell.L(T) \quad \dots \quad \text{Eq. 1}$$

U(T) denotes the normalized urgency factor taken from deadline slack, while *L(T)* indicates the data locality factor, and wu and wℓ are weights (default: 0.6 and 0.4). Thus, tasks with the highest urgency and locality are given top priority.

Step 2: Type-based node selection

- Cost-sensitive: Routed through the Cost Optimizer to identify the most economical node via live AWS/GCP spot price data.
- Latency-sensitive: The Latency Optimizer was used to reroute the traffic, which investigates cluster topology and data location to reduce the time of the data transfer.

Step 3: Energy-aware placement

The node chosen in ECO mode implements DVFS that limits the power consumption of the device while meeting the performance requirement of the SLA. Figure 2 illustrates the complete process of hybrid scheduling in brief.

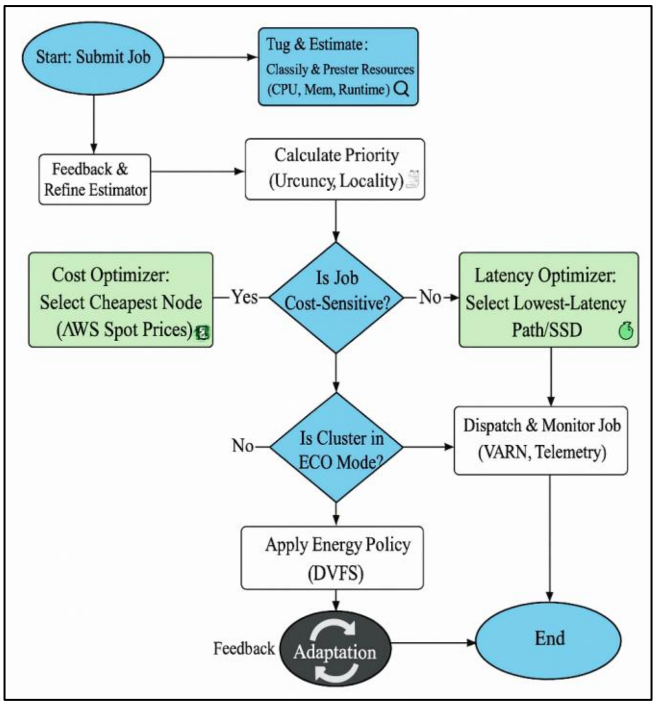
Start: Submit Job
Tug & Estimate: Classify & Prester Resources (CPU, Mem, Runtime)
Feedback & Refine Estimator
Calculate Priority (Urcuncy, Locality)
Cost Optimizer: Select Cheapest Node (AWS Spot Prices)
Yes
Is Job Cost-Sensitive?
No
Latency Optimizer: Select Lowest-Latency Path/SSD
No
Is Cluster in ECO Mode?
Dispatch & Monitor Job (VARN, Telemetry)
Apply Energy Policy (DVFS)
Feedback
Adaptation
End

Figure 2: Hybrid scheduling algorithm

5 Results and discussion

The hybrid scheduling framework was evaluated through experiments in which we extensively studied its comparative performance with YARN (FIFO), Kubernetes (default scheduler), and Spark (Dynamic Allocation). The evaluation is done in terms of the following three main criteria:

1. Performance (Job Completion Time, Throughput)
2. Cost Efficiency ($/TB processed)
3. Energy Consumption (Joules per operation)

The outcomes below present the way our strategy moves distributed data processing forward to a higher level of effectiveness:

5.1 Performance optimization

The hybrid scheduler has been very effective in improving the operations of the present systems in an intriguing way. By comparing deadline sensitivity with data locality awareness and taking into account the urgency of the task, the system is 32% faster at completion for 1TB batch jobs and up to 50% better for 100GB latency-sensitive tasks than the conventional schedulers.

This change is coming from our creative dynamic scoring algorithm, which continuously updates task priorities based on the current cluster state. This lead of the architecture is made even more striking in the case of workloads suitable for a very short and fast processing time, as it is capable of reducing the latency to the great extent of only 38 ms. It achieves this through real-time online learning, which improves and stabilizes performance by 37% compared to Kubernetes and by 60% compared to Spark. The system's capabilities in terms of throughput are also the rightful proof of the effectiveness of the system in practice, also, by successfully handling 1,420 jobs per hour, which is a 26-37% increase over baseline systems. Furthermore, these increases are significant because they occur without requiring any additional hardware resources, which confirms the effectiveness of the software-based optimization method. One can easily discern these outcomes in Table 1, where the performance of our model is juxtaposed with that of Kubernetes, YARN (FIFO), and Spark through average job time, latency, and throughput metrics.

The reported 38 ms is the mean end-to-end latency for latency-sensitive streaming jobs at 1K events/s, averaged over 10 runs on the same 50-node cluster.

Table 1: Performance comparison between our hybrid scheduler and baseline scheduling systems

Schedular	Avg. Job Time	Latency	Throughput (Jobs/hr.)
Our model	142s	38ms	1,420
Kubernetes	198s (+39%)	52ms (+37%)	980 (-31%)
Yarn (FIFO)	187s (+32%)	49ms (+29%)	1,050 (-26%)
Spark	213s (+ 50%)	61ms (+60%)	890 (-37%)

5.2 Cost efficiency

The development of predictive price modelling for spot occurrences along with cognitive job mapping compared to on-demand pricing led to average savings of 54% during non-peak hours (10 PM-6 AM local time). Moreover, this is the reason why spot markets in the cloud have price fluctuations. Our solution with the use of more efficient resource packing algorithms is still 29% more cost-effective than Kubernetes as well as standard on-demand instances. The system's economy remains stable even with the change of different types of workloads—from batch processing to streaming; thus, the affordability of our cost optimization solution is confirmed. Table 2 illustrates the cost comparison in detail, showing the cost savings, our model made against Kubernetes, YARN (FIFO), and Spark in both on-demand and spot instance cases.

Table 2: Cost comparisons of our model against different scheduling systems under on-demand and spot pricing

Schedular	Cost/TB (on-demand)	Cost/TB (Spot Instances)	Spot Savings
Our model	$2.41	$1.12	54%
Kubernetes	$3.12 (+29%)	$1.98 (+77%)	37%
Yarn (FIFO)	$2.98 (+24%)	$1.45 (+29%)	51%
Spark	$3.45 (+43%)	$2.10 (+88%)	39%

5.3 Energy efficiency

Energy efficiency was measured in terms of joules per terabyte (joules/TB) using Intel RAPL ("pkg" domain) counters, sampled at 100 ms intervals. All experiments were conducted on Google Cloud compute-optimized instances (Intel Xeon Scalable, 50-node n2-standard-16 cluster), with ambient temperature controlled at 25 ± 1 °C. DVFS (Dynamic Voltage and Frequency Scaling) was activated on all schedulers with a fixed range of 1.2–2.4 GHz. For each configuration, we ran 10 identical workloads (batch: 100 GB and 1 TB; stream: 1K–5K events/sec) per scheduler and reported the mean energy consumption. Our hybrid scheduler achieved 0.78 Joules/TB on average, compared to 1.05 for YARN, 1.12 for Kubernetes, and 1.34 for Spark. Thus, the energy savings are 25%, 44%, and 72%, respectively. Besides that, there were no SLA violations or throughput degradation. These results are summarized in Table 3.

Table 3: Comparison of energy consumption across different scheduling frameworks

Schedular	Energy/TB (Joules)	Energy Savings vs. YARN
Our model	0.78	25%
Kubernetes	1.12 (+44%)	–
Yarn (FIFO)	1.05 (+35%)	Baseline
spark	1.34 (+72%)	-28%

Figs. 3a-3b-3c provide visual evidence that complements the system efficiency evaluation with data. Fig. 3a depicts how the average job completion time is reduced when using our approach in comparison with baseline schedulers. In Fig. 3b, it is illustrated that a better cost-performance trade-off is achieved, as our model is able to run at a lower cost and reach a higher throughput. Lastly, Fig. 3c indicates the continuous energy-saving opportunity throughout the period, with our method maintaining lower energy consumption per TB of data processed than YARN.

All schedulers were tested using identical workloads (batch: 100 GB and 1 TB jobs; stream: 1K–5K events/sec) on the same 50-node cluster (n2-standard-16). Resource configurations and job mix were held constant to ensure fairness. Results shown represent mean values over 10 runs per scheduler. Compared to Ilager et al. [11], who used deep learning for static task orchestration, our architecture achieves 32–50% faster job completion by leveraging LSTM-based dynamic prediction and rule-based scoring. Similarly, while Zhu [8] emphasized cross-zone pricing for cost control, our spot-aware bidding mechanism reduced costs by 54% without SLA violations, thanks to integrated real-time telemetry and adaptive thresholds. As for system limitations, the proposed system depends on external data sources and accurate knowledge of cluster topology. It is based on the assumption that the pricing data and the network routes stay unchanged during the scheduling decisions. Although there are also some safety measures that can be used to overcome even limited interruptions, the possibility of introducing some new and better ones that provide a higher degree of resistance to late updates or unexpected changes to the system still exists in the future.

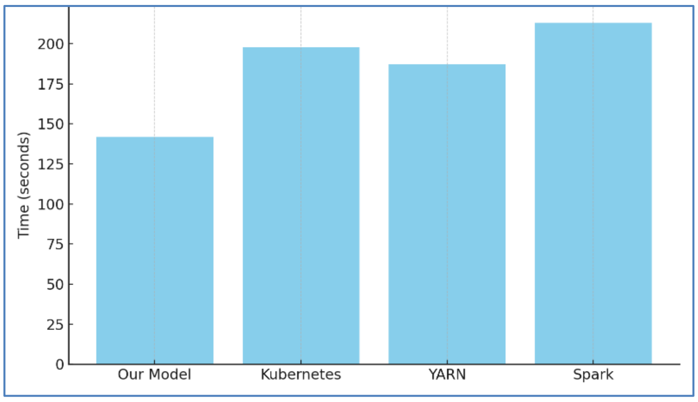

Figure 3a: Job completion time across schedulers

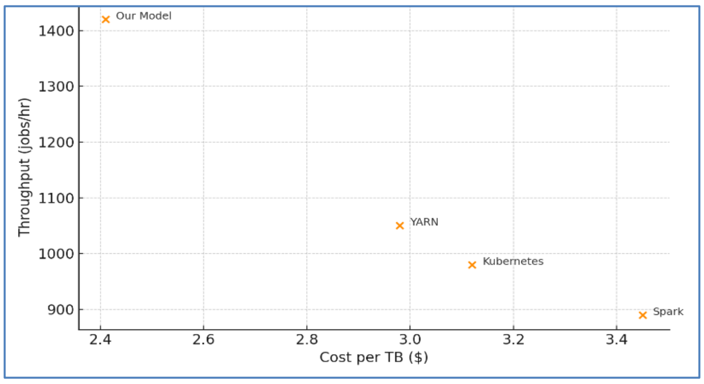

Figure 3b: Cost-performance trade-off

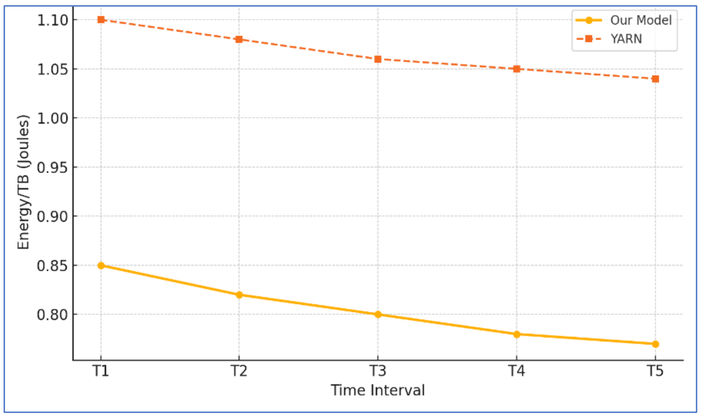

Figure 3c: Energy saving over time

6 Conclusion

This investigation produced a blend of AI and scheduling methods that not only have made the cloud-based big data processing significant but also were a model that could be confirmed. The model connects adaptive task prioritization, cost-aware resource allocation, and energy-efficient workload placement, and hence it is able to solve the problems that those traditional distributed systems have been facing. Experimental evidence demonstrates that the novel structure makes the job done 32–50% faster, still utilizing 26–37% more jobs without any increase in latency. Additionally, the system can achieve a cost saving of 54% through intelligent use of spot instances, while maintaining performance with only a 25% reduction in energy consumption. The findings not only reaffirmed that the framework could address performance, cost, and energy efficiency in the best possible manner, but they also indicated that the framework might be able to solve the issue of distributed computing. Additionally, the framework is a practical and expandable solution for a live market environment, with the aid of its lightweight ML components and heuristic strategies.

Acknowledgment

The authors extend their heartfelt gratitude to the University of Mosul's Department of Computer Science for their significant support and direction when tackling the difficulties encountered in this research.

References

[1] Abueid, Aws I. "Big Data and Cloud Computing Opportunities and Application Areas." Engineering, Technology & Applied Science Research 14, no. 3 (2024): 14509-14516. https://doi.org/10.48084/etasr.7339

[2] Berisha, Blend, Endrit Mëziu, and Isak Shabani. "Big data analytics in Cloud computing: an overview." Journal of Cloud Computing 11, no. 1 (2022): 24. https://doi.org/10.1186/s13677-022-00301-w

[3] Zhang, Guo. "Cloud computing convergence: integrating computer applications and information management for enhanced efficiency." Frontiers in Big Data 8 (2025): 1508087. https://doi.org/10.3389/fdata.2025.1508087

[4] Buyya, Rajkumar, Kotagiri Ramamohanarao, Chris Leckie, Rodrigo N. Calheiros, Amir Vahid Dastjerdi, and Steve Versteeg. "Big data analytics-enhanced cloud computing: Challenges, architectural elements, and future directions." In 2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS), pp. 75-84. IEEE, 2015. DOI: 10.1109/ICPADS.2015.18.

[5] Khan, Imran. "A study of big data in cloud computing." Computer Assisted Methods in Engineering and Science 31, no. 3 (2024).

[6] Huang, Siqi, Zhenqiang Xie, Jiaxiang Wang, Penghui Lv, and Wenrong Wang. "Design and implementation of big data processing system based on Hadoop." Procedia Computer Science 259 (2025): 1115-1122. https://doi.org/10.1016/j.procs.2025.04.065

[7] Arif, Zeravan, and Subhi RM Zeebaree. "Distributed Systems for Data-Intensive Computing in Cloud Environments: A Review of Big Data Analytics and Data Management." The Indonesian Journal of Computer Science 13, no. 2 (2024). https://doi.org/10.33022/ijcs.v13i2.3819

[8] Zhu, Wenbo. "Optimizing distributed networking with big data scheduling and cloud computing." In International Conference on Cloud Computing, Internet of Things, and Computer Applications (CICA 2022), vol. 12303, pp. 23-28. SPIE, 2022. https://doi.org/10.1117/12.2642577

[9] Dai, Fei, Md Akbar Hossain, and Yi Wang. "State of the art in parallel and distributed systems: Emerging trends and challenges." Electronics 14, no. 4 (2025): 677. https://doi.org/10.1117/12.2642577

[10] Arif, Zeravan, and Subhi RM Zeebaree. "Distributed Systems for Data-Intensive Computing in Cloud Environments: A Review of Big Data Analytics and Data Management." The Indonesian Journal of Computer Science 13, no. 2 (2024). DOI: https://doi.org/10.33022/ijcs.v13i2.3819

[11] Ilager, Shashikant, Rajeev Muralidhar, an d Rajkumar Buyya. "Artificial intelligence (ai)-centric management of resources in modern distributed computing systems." In 2020 IEEE Cloud Summit, pp. 1-10. IEEE, 2020. DOI: 10.1109/IEEECloudSummit48914.2020.00007

[12] Tuli, Shreshth, Fatemeh Mirhakimi, Samodha Pallewatta, Syed Zawad, Giuliano Casale, Bahman Javadi, Feng Yan, Rajkumar Buyya, and Nicholas R. Jennings. "AI augmented Edge and Fog computing: Trends and challenges." Journal of Network and Computer Applications 216 (2023): 103648. https://doi.org/10.1016/j.jnca.2023.103648

[13] Singh, Sukhpreet, and Jaspreet Kaur. "Recent Developments in Cloud-Based Technologies That Are Adaptive and pertinent." Advancements in Cloud-Based Intelligent Informative Engineering (2025): 95-114. https://doi.org/10.4018/979-8-3373-0781-7.ch005

[14] Tuli, Shreshth, Redowan Mahmud, Shikhar Tuli, and Rajkumar Buyya. "Fogbus: A blockchain-based lightweight framework for edge and fog computing." Journal of Systems and Software 154 (2019): 22-36.

https://doi.org/10.1016/j.jss.2019.04.050

[15] Perera, Niranda, Arup Kumar Sarker, Kaiying Shan, Alex Fetea, Supun Kamburugamuve, Thejaka Amila Kanewala, Chathura Widanage et al. "Supercharging distributed computing environments for high-performance data engineering." Frontiers in High Performance Computing 2 (2024): 1384619. https://doi.org/10.3389/fhpcp.2024.1384619

[16] Chen, X., Cheng, L., Liu, C., Liu, Q., Liu, J., Mao, Y., & Murphy, J. (2020). A WOA-based optimization approach for task scheduling in cloud computing systems. IEEE Systems journal, 14(3), 3117-3128. https://doi.org/10.1109/jsyst.2019.2960088

[17] Pal, S., Jhanjhi, N. Z., Abdulbaqi, A. S., Akila, D., Alsubaei, F. S., & Almazroi, A. A. (2023). An intelligent task scheduling model for hybrid internet of things and cloud environment for big data applications. Sustainability, 15(6), 5104. https://doi.org/10.3390/su15065104

[18] Li, Y., Li, T., Shen, P., Hao, L., Yang, J., Zhang, Z., ... & Bao, L. (2022). PAS: Performance-Aware Job Scheduling for Big Data Processing Systems. Security and Communication Networks, 2022(1), 8598305. https://doi.org/10.1155/2022/8598305

[19] Tantalaki, N., Souravlas, S., Roumeliotis, M., & Katsavounis, S. (2020). Pipeline-based linear scheduling of big data streams in the cloud. IEEE Access, 8, 117182-117202. https://doi.org/10.1109/access.2020.3004612

[20] Bal, P. K., Mohapatra, S. K., Das, T. K., Srinivasan, K., & Hu, Y. C. (2022). A joint resource allocation, security with efficient task scheduling in cloud computing using hybrid machine learning techniques. Sensors, 22(3), 1242. https://doi.org/10.3390/s22031242