AI-Powered Federated Task Scheduling and Self-Healing Framework in Dynamic Cloud Systems

Umit Demirbaga*, Omer Rana†, Ashiq Anjum‡, Gagangeet Singh Aujla§
*Department of Computer Engineering, Bartin University, Türkiye
†Department of Computer Science and Informatics, Cardiff University, United Kingdom
‡Department of Computing and Mathematical Sciences, University of Leicester, United Kingdom
§Department of Computer Science, Durham University, United Kingdom
Email: udemirbaga@bartin.edu.tr, ranaof@cardiff.ac.uk, aa1180@leicester.ac.uk, gagangeet.s.aujla@durham.ac.uk.

***Abstract*—Federated cloud environments have emerged to integrate multiple cloud providers like AWS, Azure, and Google Cloud seamlessly into cloud computing. Optimising resource utilisation and ensuring high availability in such environments pose significant challenges. This paper comprehensively investigates federated task scheduling algorithms and self-healing mechanisms in autonomous federated cloud setups. The research objectives include the development of an independent task-scheduling algorithm capable of intelligently distributing computing tasks across federated clouds based on workload characteristics, resource availability, and network latency. Furthermore, the study investigates implementing self-healing mechanisms to detect faults and performance degradation, triggering automatic recovery processes for uninterrupted service availability. The proposed approaches are evaluated through real-world experiments, considering diverse cloud workloads and failure scenarios, focusing on resource utilisation efficiency, system performance, and the effectiveness of the self-healing mechanisms in mitigating cloud failures and maintaining seamless operations within the federated environment.**

***Index Terms*—Federated learning, Federated cloud computing, MapReduce, Artificial intelligence, Big data analysis**

I. Introduction

FEDERATED learning is a decentralised machine learning (ML) paradigm that allows several devices or entities to train a single model while storing and protecting their data locally [1]. In federated learning, each device node performs model training using its local data and shares only the model updates with the central server instead of providing raw data. The global model is then updated with the help of these aggregated changes before being transmitted once more to the involved nodes. This privacy-preserving distributed learning paradigm tackles data privacy concerns by decentralising sensitive information, making it especially ideal for situations involving huge and sensitive data, such as mobile devices, edge computing, and Internet of Things (IoT) settings [2].

The concept of federated cloud or cloud federation emerges to overcome the major issues and limitations of single-cloud setups, offering flexibility, scalability, and improved privacy. Creates a unified, connected cloud environment across multiple cloud service providers that aggregates all computing resources and services to accelerate end-user engagements [3]. Given this capability, the federated cloud enables customers to access various custom services such that vendor lock-in risk is mitigated and tailors individualised needs. To handle this, AI systems learn from continuous real-time data collected from cloud resources [4], including task execution times, resource utilisation, network latency, and workload characteristics [5]. By analysing these performance metrics, AI models identify patterns and trends, such as bottlenecks, straggler tasks, or underutilised resources. The learning process involves training machine learning models on this data to detect inefficiencies and predict task delays. These models are continuously updated with new data to adapt to the dynamic conditions of cloud environments.

Some researchers have investigated task rescheduling deployments in cloud computing environments. The authors of [6] propose a novel approach termed MALO, which combines elite-based differential evolution with hybrid antlion optimisation to address the issue of efficient job scheduling in cloud computing. Their approach minimises makespan while maximising resource utilisation, a crucial objective in cloud computing. Extensive experiments were conducted on synthetic and real trace datasets, utilising the CloudSim toolkit [7]. The limitation of this work lies in the need for further exploration and validation across a broader range of complex optimisation problems and real-world cloud computing scenarios. The paper [8] addresses big data task scheduling challenges in cloud computing, focusing on achieving minimal makespan and efficient resource utilisation. The authors present BigTrustScheduling, a three-stage trust-aware scheduling solution that computes VM trust levels, determines task priorities, and facilitates trust-aware scheduling. Nonetheless, a constraint of the proposed solution resides in the need for further exploration and adaptation to other computing environments beyond cloud, IoT, and parallel computing to ensure the generalisability and effectiveness of its trust model.

Developing an autonomous task scheduling algorithm that intelligently allocates computing tasks across diverse cloud resources while accommodating workload diversity and resource heterogeneity presents a formidable challenge. The dynamic nature of federated clouds demands a real-time scheduling mechanism capable of adapting to varying conditions, optimising resource utilisation, and ensuring efficient task distribution. Additionally, implementing fault detection mechanisms suitable for identifying issues within federated cloud components

and initiating automatic recovery processes is essential to ensure high availability. To this end, we investigate the following research questions (RQs) in this paper:

- **(RQ1)** How can real-time monitoring techniques effectively detect straggler tasks in dynamic cloud systems?
- **(RQ2)** What are the most effective strategies for dynamically reallocating straggler tasks across federated clouds to minimise delays and optimise resource utilisation?
- **(RQ3)** How does managing stragglers impact the overall system performance, resource efficiency, and service availability in federated cloud environments?

A. Contributions

A thorough assessment of the suggested AI-driven federated task scheduling and self-healing approaches is sought through extensive real-world tests, taking into account a variety of cloud workloads and failure scenarios to assess their efficacy, scalability, and applicability within the federated cloud ecosystem. By focusing on these goals, this research hopes to support the creation of a reliable and effective distributed computing architecture while optimising and improving resource utilisation, fault resilience, and overall performance in federated cloud settings. To this end, we propose an AI-powered self-healing framework for federated cloud systems. It focuses on detecting straggler tasks and autonomously reallocating them to ensure high availability and optimal resource utilisation. Here are the key contributions of the paper:

- Developing a real-time straggler detection mechanism which identifies delayed tasks that hinder performance in dynamic cloud environments.
- Proposing an AI-driven task reallocation framework, which autonomously relocates straggler tasks to alternative cloud clusters, thus reducing delays and improving resource utilisation.
- Conducting comprehensive performance evaluations on federated clouds, demonstrating the impact of the self-healing framework on improving resource efficiency, and system reliability.

II. Proposed System: FedCloudX

In this section, we introduce FedCloudX, a Python-based system that serves as a comprehensive task-scheduling and resource management system within the federated cloud environment, empowered with full control over both cloud systems. By gathering real-time monitoring data from a time-series database, FedCloudX gains insights into the performance-related metrics of the Hadoop clusters, allowing it to efficiently detect and address slow tasks, commonly referred to as stragglers. Leveraging this monitoring capability, FedCloudX continuously evaluates the system's health status, identifying instances of stragglers or tasks awaiting execution in the queue, indicative of resource insufficiency within the main cloud provider. In such scenarios, FedCloudX dynamically initiates the scheduling process, intelligently allocating the stragglers or waiting tasks to the Hadoop cluster in the secondary cloud provider. Through this proactive task allocation mechanism, FedCloudX optimises resource utilisation, ensures timely task execution, and maintains seamless operations across the federated cloud. The advanced capabilities of FedCloudX contribute to enhanced performance, scalability, and self-healing, facilitating efficient big data processing and empowering the federated cloud ecosystem with heightened responsiveness and adaptability to fluctuating workloads.

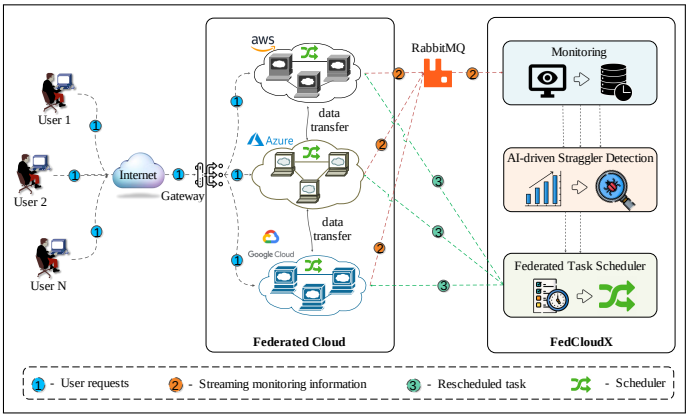


Fig. 1. FedCloudX system architecture in cloud federation

FedCloudX comprises three pivotal components, depicted in Fig. 1. The first component, *Real-time Monitoring and Performance Insights*, is central to continuously collecting and analysing performance-related metrics from the Hadoop cluster. Through this component, FedCloudX gains valuable insights into the health status of the federated cloud infrastructure, which promptly stragglers and identifies nodes where resource availability becomes insufficient. The *AI-Driven Straggler Detection* model constitutes the second essential component, which employs statistical methods and AI algorithms to analyse metrics obtained by SmartMonit to detect stragglers. Following identifying these stragglers, the *Federated Task Scheduling* component schedules them concurrently across multiple cloud providers using the streaming data from the straggler detection algorithm to address the challenges, including network latency, and resource availability.

A. Real-time Monitoring and Performance Insights

FedCloudX employs SmartMonit [9], a real-time big data monitoring system, to collect log data from cloud-based big data systems, including job execution time, task progress, data block locations, node resource utilisation (CPU/memory), network consumption, and disk usage. Through such real-time data analysis, FedCloudX can gain valuable insight into the overall operation of the federated cloud system.

B. AI-Driven Straggler Detection for Task Scheduling

The performance of MapReduce jobs is severely hampered by stragglers, increasing the chance of data loss and task execution delays. We employ artificial intelligence algorithms to examine performance indicators gathered and stored by SmartMonit to tackle this difficulty.

Combining the statistical technique proposed by [10] with the Isolation Forest-based approach, we develop Algorithm 1 to detect stragglers in a streaming data environment. The algorithm takes two data streams as input, namely the execution time (τ) and progress (ρ), representing the execution time and progress metrics of mapper tasks, respectively. The algorithm then proceeds to normalise the τ and ρ data (lines 2 and 3), followed by calculating the performance metric σ for each mapper (lines 4 to 6). Then we use the Isolation Forest model to find σ number of stragglers based on P from line 8. The results (σ) are then added to the list of stragglers (Line 11). Line 14 adds a small waiting period before the algorithm retrieves the next streaming data set. It runs in the loop (line 2 to line 3) and can make real-time discovery with quick reactions in streaming big data systems.

Algorithm 1: AI-Driven real-time straggler detection

```
   Input  : τ, Streaming execution time of each task
   Input  : ρ, Streaming progress of each task
   Output: List of stragglers σ
1  while Streaming data is available do
2      τ̂ ← normalizeData(τ) // Normalize τ̂
3      ρ̂ ← normalizeData(ρ) // Normalize ρ̂
4      P ← [] // Initialize P list
5      for i ← 1 to length(τ) do
6          P [i] ← ρ̂[i] / τ̂[i] // Calculate performance (P)
7      end
8      σ ← detectStragglers(P) // Detect σ using Isolation
         Forest
9      for i ← 1 to length(σ) do
10         if σ [i] = −1 then
11             List of σ ← σ;
12         end
13     end
14     Wait () // Wait before fetching the next set of data
15 end
```

We first introduce a model for the streaming execution time and progress as stochastic processes to explore some mathematical complexity of the algorithm. Consider a normalised execution time process $\hat{\tau}(t)$ and the corresponding normalised progress process $\hat{\rho}(t)$ as functions of time t. Driven by these stochastic processes, we can characterize what is referred to as a joint Probability Density Function (PDF) $f_{\hat{\tau},\hat{\rho}}(\hat{\tau},\hat{\rho},t)$ which models the probability that a specific execution time and progress level are observed at time t.

$$f_{\hat{\tau},\hat{\rho}}(\hat{\tau},\hat{\rho},t) = \lim_{\Delta\hat{\tau},\Delta\hat{\rho}\to 0} \frac{\Pr[\hat{\tau} \le \hat{\tau}+\Delta\hat{\tau}, \hat{\rho} \le \hat{\rho}+\Delta\hat{\rho}, t \le T]}{\Delta\hat{\tau}\Delta\hat{\rho}} \tag{1}$$

, where T denotes the total streaming duration. Integrating this joint PDF over the entire time interval yields the marginal PDFs $f_{\hat{\tau}}(\hat{\tau})$ and $f_{\hat{\rho}}(\hat{\rho})$, which characterises the distributions of normalised execution time and progress, respectively:

$$f_{\hat{\tau}}(\hat{\tau}) = \int_{-\infty}^{+\infty} f_{\hat{\tau},\hat{\rho}}(\hat{\tau},\hat{\rho},t)d\hat{\rho} \tag{2}$$

$$f_{\hat{\rho}}(\hat{\rho}) = \int_{-\infty}^{+\infty} f_{\hat{\tau},\hat{\rho}}(\hat{\tau},\hat{\rho},t)d\hat{\tau} \tag{3}$$

The normalisation here refers to scaling the execution time values and progress to a range between 0 and 1, where 0 represents the start, and 1 represents the completion of the execution or task. For instance, if a task is halfway through execution, $\hat{\rho}(t)$ would be approximately 0.5 at that time.

C. Federated Task Scheduling for Self Healing and Resilience

When faced with stragglers or there is a task in the queue waiting to be processed in a situation where the resources of the current cloud provider's cluster become insufficient, FedCloudX dynamically initiates the scheduling process. In response to these scenarios, the system intelligently allocates the stragglers and waiting tasks to the Hadoop cluster in another cloud provider [11]. By leveraging real-time monitoring data, FedCloudX continuously evaluates the performance of the clusters, detecting instances of stragglers and assessing the adequacy of resources in the main cloud provider's cluster. Finally, the system promptly and adaptively allocates tasks to an alternate cluster, optimising resource utilisation and ensuring timely task execution.

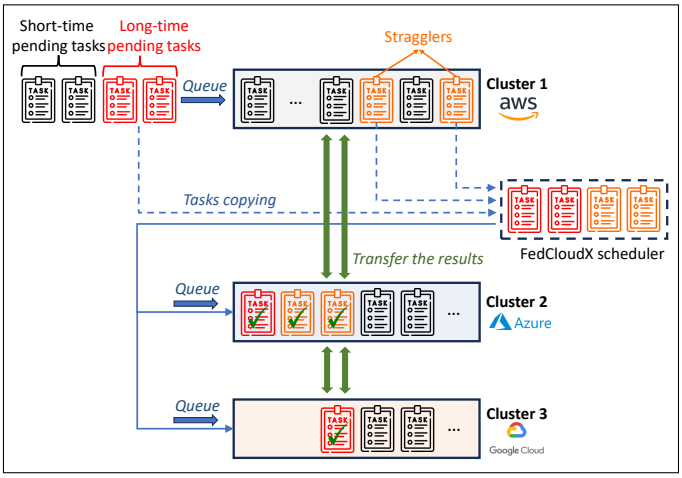


Fig. 2. An example of task scheduling in FedCloudX

Fig. 2 illustrates the task rescheduling mechanism of FedCloudX, showcasing the dynamic reallocation of straggling and long-time pending tasks among distinct cloud clusters to optimise resource utilisation and enhance overall system performance. By dynamically detecting and rescheduling them to different clusters within other cloud systems, the solution overcomes the problem of straggling work within a cluster. Using statistical and ML techniques, notable delays are detected in real-time by tracking their progress and execution times and comparing them. Concurrently, FedCloudX identifies long-time pending tasks in the queue, awaiting processing for extended periods. FedCloudX enhances overall system performance by reallocating these stragglers and long-time pending tasks to underutilised clusters in separate cloud systems.

We develop Algorithm 2 to improve task scheduling performance and reduce job completion time in a MapReduce framework, focusing on two major issues: `"STRAGGLER"` that take longer to complete due to various reasons like data

Algorithm 2: Federated task scheduling algorithm

```
   Input : List of σ // predictions from Algo. 1
1          localClusterQueue ← []
2          remoteClusterQueue ← []
3          schedulingDelay ← 5 // Set to 5 seconds
4          maxWaitingTime ← 30 // Set to 30 seconds
5  Function submitTask(taskId, priority, startTime, executionTime):
6      newTask ← Task(taskId, priority, startTime, executionTime,
         'WAITING');
7      localClusterQueue ← task;
8  Function scheduleTasks():
9      while localClusterQueue is not empty do
10         task ← localClusterQueue;
11         if task.status = 'WAITING' and currentTime -
             task.startTime > maxWaitingTime then
12             Set task.status = 'STRAGGLER';
13             remoteClusterQueue ← task;
14         end
15         else
16             Wait (schedulingDelay sec.);
17             if task.status = 'WAITING' then
18                 remoteClusterQueue ← task;
19             end
20         end
21     end
22     while remoteClusterQueue is not empty do
23         task ← remoteClusterQueue;
24         if task.status = 'STRAGGLER' then
25             Set task.status = 'WAITING';
26             localClusterQueue ← task;
27         end
28     end
29 submitTask(1, 2, currentTime, 10) // Example task submission
     with priority 2 and execution time 10 seconds
30 scheduleTasks();
```

skew or resource contention, and "WAITING" jobs that are waiting to be executed in the queue. This algorithm monitors two task queues: the local and remote cluster queues. First, tasks are added to the local cluster queue upon submission. Then, the algorithm selects "STRAGGLER" jobs depending on the predefined maximum waiting time "WAITING" tasks in the local queue are also transferred to the remote cluster queue after a brief wait. They are returned to the local cluster queue after they have completed their execution on the remote cluster. The algorithm's lines 1 through 30 carry out the following steps: To store tasks sent to the local cluster, line 1 initialises the queue as an empty list. The remote cluster queue is then initialised as an empty list by Line 2 to hold tasks detected as stragglers and moved to the distant cluster. Subsequently, the scheduling delay is set to 5 seconds by Line 3, indicating the time the algorithm waits before searching the local cluster queue for stragglers. Line 4 sets the maximum waiting time to 30 seconds as a threshold to determine if a task becomes a straggler based on its waiting time in the local queue. Furthermore, the `submitTask` function (Line 6) creates a new task object with the given parameters (taskId, priority, startTime, executionTime), and its status is set to "WAITING". The task is then promptly added to the local cluster queue in Line 7. The `scheduleTasks` function, outlined in Line 10, starts its operation by retrieving the next task from the front of the local cluster queue for processing. As the algorithm progresses, Line 11 checks if the current task is in "WAITING" status and whether its waiting time in the local queue exceeds the maximum waiting time (30 seconds). If this condition is met, the task is identified as a straggler, and its status is set to "STRAGGLER" in Line 12. Consequently, the task is moved from the local cluster queue to the remote cluster queue for execution on a different cluster in Line 13. On the other hand, if the task's status is still "WAITING" after a short scheduling delay (5 seconds) in Line 16, it is determined that the task is better suited for execution on the remote cluster. Subsequently, it is added to the remote cluster queue in Line 18. The algorithm continues its task management by processing tasks in the remote cluster queue in Lines 23 and 25. If a task is identified as a straggler (status is "STRAGGLER"), it is moved back to the local cluster queue in Line 26 after it finishes execution on the remote cluster. Throughout the algorithm's execution, Line 29 exemplifies the submission of tasks to the local cluster queue, showcasing different priorities and execution times. Ultimately, Line 30 culminates the process by calling the `scheduleTasks` function, thus initiating the scheduling algorithm and ensuring smooth task distribution across clusters, mitigating straggler problems.

$$S_i = \begin{cases} W \times e^{-\lambda \int_{\mathrm{ST}_i}^{t} d\tau}, & \text{if } S_i = S \\ S, & \text{if } S_i = W \text{ and } t - \mathrm{ST}_i > MWT \\ S_i, & \text{otherwise} \end{cases} \tag{4}$$

$$\text{Where } \lambda: \text{ Rate of decay, } \lambda = -\frac{1}{MWT} \tag{5}$$

The presented formula (Eq. 4) encapsulates the mathematical expression characterising the dynamic adjustment of a computational task's status (S_i) within the algorithm. In this context, the symbol W signifies the waiting tasks, and S represents the stragglers. The equation employs an exponential decay function, $e^{-\lambda \int_{\mathrm{ST}_i}^{t} d\tau}$, where λ denotes the rate of decay. If S_i is identified as a S, its status transforms to W governed by the decay function. The parameter MWT represents the threshold for the maximum waiting time, and the rate of decay (λ) is calculated as $-\frac{1}{MWT}$ (Eq. 5).

III. Experimental Setup and Evaluation

To establish a federated cloud environment, we meticulously configured Hadoop clusters within distinct cloud service providers and ensured seamless communication between these clusters. This federated cloud architecture allows us to conduct successful experiments by leveraging the combined capabilities of both cloud providers.

A. Experimental setup

For conducting the experiments, we utilised two prominent cloud service providers, AWS and Azure, to establish separate four-node Hadoop clusters, each node has 4 CPUs and 16 GB of memory in a federated cloud environment.

1) Setting clusters communication: It takes a long setup process with many components and parameters to connect the two Hadoop clusters in AWS and Azure. To ensure compatibility, virtual networks (VNets) in Azure and virtual private clouds (VPCs) in AWS must be set up. Network Security Groups, or NSGs, are configured in Azure to control traffic entering and exiting clusters while maintaining strict authorisation. Firewall rules are adjusted to permit communication on specific ports needed by Hadoop components. Cluster configuration parameters are changed to enable cross-cluster communication by specifying essential network information, such as IP addresses or domain names.

2) Data replication and synchronisation: To ensure real-time data consistency between clusters, we employ Event-Driven Synchronisation, which entails installing systems that initiate data synchronisation operations automatically when particular events or changes occur in one cluster. So, we used Azure Functions and AWS Lambda to execute the code responding to cluster data change events. These serverless functions, which preserve real-time data synchronisation between the clusters, are triggered by events generated by AWS and Azure services like Amazon S3 and Azure Blob Storage. In addition, we employ AWS DMS (Database Migration Service) and Azure Data Factory for routine data replication tasks that provide continuous data synchronisation and consistency across the clusters. This approach focuses on managing storage across multiple NameNodes. Our system, by contrast, ensures real-time, automated synchronisation across cloud platforms, providing greater flexibility in managing data consistency and task execution across federated environments.

B. Data Transmission Time Calculation in Federated Cloud

Moving these responsibilities to a new Hadoop cluster hosted in a different cloud architecture is one noteworthy strategy used in this work. A necessary precondition for successfully rescheduling is the availability of the required data blocks in the desired cloud cluster. This study also aims to assess the effectiveness of the suggested method using a customised algorithm designed for this particular situation, which will further enhance self-healing and resource allocation strategies in federated cloud systems. We use Eq. 6 to do this.

$$\mathcal{N} = \frac{\mathcal{T} \times 8}{\mathcal{C} \times \mathcal{U} \times 3600 \times \mathcal{H}} \tag{6}$$

, where $\mathcal{N}$ denotes the Number of Days needed for data transmission in an AI-driven fault-tolerant system and federated job scheduling inside independent federated cloud environments. The data volume in Terabytes to be transported is indicated by $\mathcal{T}$. The available network bandwidth, denoted by $\mathcal{C}$, is measured in CIRCUIT gigabits per second (Gbps). The proportion of network utilisation is shown by $\mathcal{U}$. Last, $\mathcal{H}$ denotes the total number of hours available daily, which is essential for figuring out how long data transmission will take. A key tool for assessing the effectiveness of data migration in cloud systems is the formula[1].

C. Performance and Efficiency Analysis

1) Straggler Detection Evaluation: The integrated logic used for straggler identification is shown in Fig. 3, which combines statistical methods with the ML algorithm Isolation Forest to improve the detection process' efficacy and accuracy.

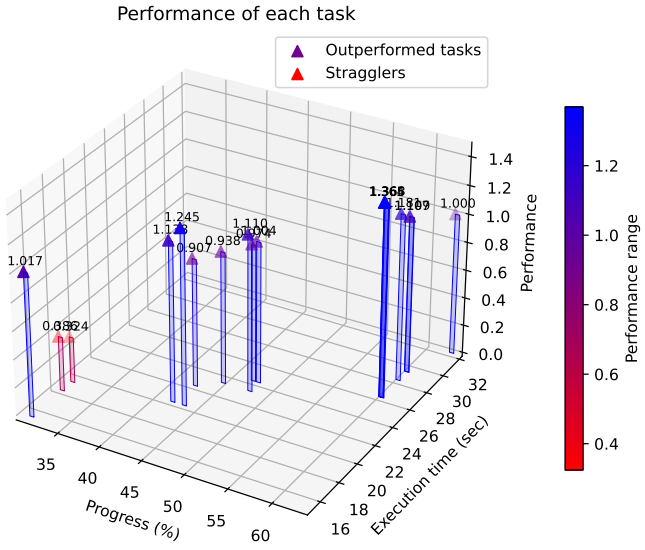
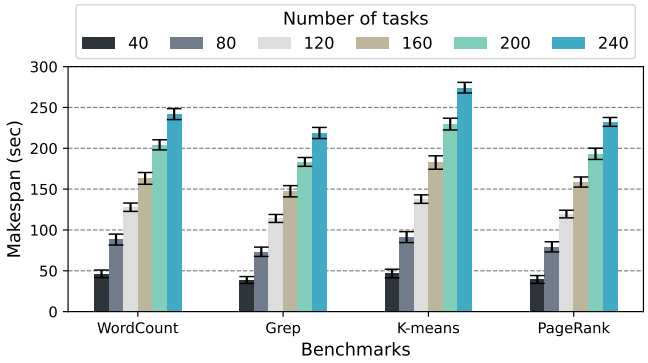


Fig. 3. Straggler detection in FedCloudX

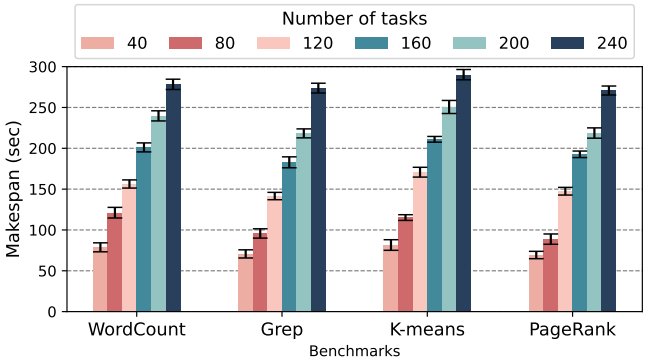
2) Makespan Development Evaluation: To evaluate the system's performance, we carried out in-depth tests, each performed five times for improved accuracy over a wide range of task quantities, from 40 to 240. Fig. 4(a) showcases the remarkable mitigation of straggler-related issues when FedCloudX is employed, while Fig. 4(b) highlights the system's proficiency in handling long-time pending jobs. These results, supported by error rate data, underscore the efficacy of FedCloudX in significantly improving the overall performance and efficiency of cloud-based computing environments, offering a promising solution for enhancing the reliability and productivity of cloud systems. The results in Fig. 5 demonstrate that applying the FedCloudX leads to a noteworthy reduction in makespan. FedCloudX not only improves the makespan by around 15% for straggler cases and around 22% for long-time pending jobs.

3) CPU and Memory Consumption of FedCloudX: Table I provides a detailed breakdown of the memory and CPU utilisation associated with two distinct components within the FedCloudX system, where *straggler detection* consumes 3.78% of system memory and 5.23% of CPU resources, while *task rescheduling* imposes slightly higher demands, utilising 4.08% of memory and 4.61% of CPU resources.

[1]https://docs.aws.amazon.com/whitepapers/latest/overview-aws-cloud-data-migration-services/time-and-performance.html



(a) Makespan evaluation of stragglers rescheduling



(b) Makespan evaluation of long-time pending jobs rescheduling

Fig. 4. Makespan evaluation of different sized jobs for stragglers and long-time pending jobs across different benchmarks with FedCloudX

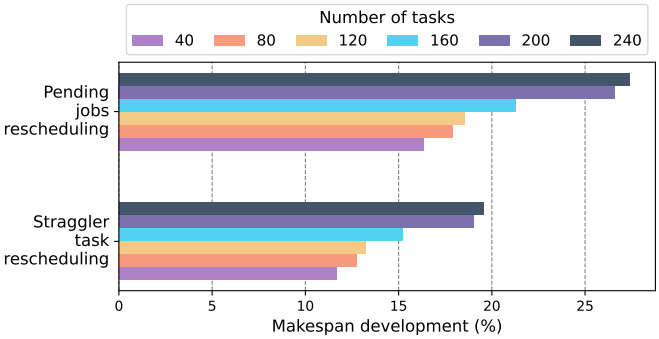


Fig. 5. Percentage-based makespan improvement using FedCloudX

TABLE I
RESOURCE OVERHEAD CAUSED BY FEDCLOUDX COMPONENTS

Components	Mem (%)	CPU (%)
FedCloudX straggler detection	3.78	5.23
FedCloudX task rescheduling	4.08	4.61

IV. CONCLUSION

This study aims to provide an AI-powered solution for identifying and mitigating straggler jobs in federated cloud environments. A key focus has been developing a real-time monitoring system that identifies stragglers and an autonomous task scheduling algorithm that dynamically reallocates these delayed tasks across federated cloud resources. The proposed system effectively enables the accomplishment of tasks very fast and optimal resource utilisation using AI and real-time data, as well as removing stragglers from existence. Furthermore, by introducing self-healing mechanisms, we can quickly overcome performance degradations to increase system resilience and availability of the services. Their approaches have been tested extensively in the real world, and they have achieved significant improvements in efficiency and scalability, as well as handling various workloads and failure scenarios.

ACKNOWLEDGMENT

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) for project CHEDDAR: Communications Hub For Empowering Distributed ClouD Computing Applications And Research [Grant number EP/X040518/1].

REFERENCES

[1] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.

[2] Z. Zhang, L. Wu, D. He, Q. Wang, D. Wu, X. Shi, and C. Ma, "G-vcfl: grouped verifiable chained privacy-preserving federated learning," *IEEE Transactions on Network and Service Management*, vol. 19, no. 4, pp. 4219–4231, 2022.

[3] U. Demirbaga and G. S. Aujla, "Mapchain: A blockchain-based verifiable healthcare service management in iot-based big data ecosystem," *IEEE Transactions on Network and Service Management*, vol. 19, no. 4, pp. 3896–3907, 2022.

[4] Y. Wu, "Cloud-edge orchestration for the internet of things: Architecture and ai-powered data processing," *IEEE Internet of Things Journal*, vol. 8, no. 16, pp. 12 792–12 805, 2020.

[5] Ü. Demirbaga, G. S. Aujla, A. Jindal, and O. Kalyon, "Big data monitoring," in *Big Data Analytics: Theory, Techniques, Platforms, and Applications*. Springer, 2024, pp. 155–170.

[6] L. Abualigah and A. Diabat, "A novel hybrid antlion optimization algorithm for multi-objective task scheduling problems in cloud computing environments," *Cluster Computing*, vol. 24, pp. 205–223, 2021.

[7] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.

[8] G. Rjoub, J. Bentahar, and O. A. Wahab, "Bigtrustscheduling: Trust-aware big data task scheduling approach in cloud computing environments," *Future Generation Computer Systems*, vol. 110, pp. 1079–1097, 2020.

[9] U. Demirbaga, A. Noor, Z. Wen, P. James, K. Mitra, and R. Ranjan, "Smartmonit: Real-time big data monitoring system," in *2019 38th symposium on reliable distributed systems (SRDS)*. IEEE, 2019, pp. 357–3572.

[10] Ü. Demirbaga, G. S. Aujla, A. Jindal, and O. Kalyon, "Debugging big data systems for big data analytics," in *Big Data Analytics: Theory, Techniques, Platforms, and Applications*. Springer, 2024, pp. 171–192.

[11] Y. F. Ugurluoglu, D. Williams, and M. Xia, "Enhancing genetic algorithm-based process parameter optimisation through grid search-optimised artificial neural networks," in *2023 28th International Conference on Automation and Computing (ICAC)*. IEEE, 2023, pp. 1–6.