National College of Ireland

Processing-in-Memory (PIM) for Cloud Computing: Optimizing Performance and Energy Efficiency

MSc Research Project
Cloud Computing

Savitanshu Somvanshi
Student ID: 23213949

School of Computing
National College of Ireland

Supervisor: Prof. Vikas Sahni

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Savitanshu Somvanshi
Student ID:	23213949
Programme:	Cloud Computing
Year:	2024
Module:	MSc Research Project
Supervisor:	Prof. Vikas Sahni
Submission Due Date:	11/08/2025
Project Title:	Processing-in-Memory (PIM) for Cloud Computing: Optimizing Performance and Energy Efficiency
Word Count:	9811
Page Count:	19

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Savitanshu Somvanshi
Date:	15th September 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Processing-in-Memory (PIM) for Cloud Computing: Optimizing Performance and Energy Efficiency

Savitanshu Somvanshi
23213949

Abstract

Modern cloud systems often struggle to efficiently handle memory-bound workloads, particularly as data-intensive applications become more common. Conventional schedulers, which rely on static rules for job classification and assignment, are unable to adapt to changing workload patterns in real time. This limitation contributes to increased latency, misclassified job placements, and higher energy consumption especially in heterogeneous environments where traditional Central Processing Unit (CPUs) op- erate alongside Processing-in-Memory (PIM) enabled nodes. To address this, a lightweight and feedback-aware scheduling strategy is proposed. The scheduler operates within a CloudSim-based simulation and avoids fixed heuristics by profiling each job using a brief 10% execution sample. Runtime is estimated and used in combination with a dynamically adjusted RAM-to-Length ratio threshold for classification. A feedback loop monitors prediction error and continuously refines the threshold to improve accuracy over time. The system is evaluated using 100 synthetic cloudlets processed under identical simulation conditions. Key performance indicators such as execution latency, prediction accuracy, and energy consumption are recorded. In experiments, the adaptive scheduler reduced average energy consumption by 15% and makespan by 18% compared to a static baseline, while maintaining more stable latency. Results indicate that the adaptive scheduler reduces both average latency and energy usage while maintaining stable job classification behavior. These findings suggest that lightweight, feedback-driven scheduling strategies can improve resource efficiency in CPU–PIM hybrid environments without introducing significant complexity.

1 Introduction

The rapid growth of cloud computing has increased the variety and complexity of workloads running in large-scale data centres. Many of these workloads process large volumes of data and spend considerable time moving information between memory and processors. This movement wastes energy and delays completion. Processing-in-Memory (PIM) technology places computation directly inside memory modules. By reducing the distance data must travel, PIM lowers latency and energy use for memory-bound tasks.

Integrating PIM into cloud infrastructure creates new challenges. A cloud platform may contain both conventional CPU servers and PIM-enabled servers. Workloads differ in their compute and memory demands. Sending the wrong type of job to a resource can cause slowdowns or waste power. Static scheduling policies are common but they cannot adapt when workload patterns change during execution.

This study focuses on designing a scheduling approach for such mixed environments. The aim is to assign each job to the most suitable resource type without requiring prior knowledge of the workload. The scheduler must also adapt its decisions over time as conditions shift.

A feedback-based strategy was chosen. The scheduler runs a small part of the job first. From this sample, it estimates execution time and memory requirements. The decision to send the job to a CPU or PIM node is made using a RAM-to-Length ratio threshold. The threshold is not fixed. It changes in response to the accuracy of recent predictions. This process allows the scheduler to follow workload trends without manual tuning.

The proposed design is tested using CloudSim. Synthetic workloads with different sizes, memory needs, and deadlines are used. The scheduler's performance is compared against a static baseline. The evaluation focuses on energy use, latency, and prediction stability.

1.1 Research Objectives

This work is guided by the following central research question: "How can we design a cloud scheduler that dynamically allocates workloads across a heterogeneous cloud comprising both conventional servers and PIM-enabled nodes to optimize energy efficiency and reduce latency?"

To answer this question, the research is structured around the following specific objectives:

- Develop a scheduler for heterogeneous CPU–PIM environments that can operate without prior workload labels.
- Use lightweight profiling to estimate job behaviour before full execution.
- Adjust scheduling decisions over time using feedback from earlier jobs.
- Compare the approach with a static baseline on energy consumption, latency, and classification stability.

1.2 Structure of the Report

The rest of the report is organized as follows:

- Section 2 reviews work related to PIM technology, heterogeneous cloud scheduling, and adaptive decision-making.
- Section 3 outlines the simulation approach, including the scheduler workflow and evaluation metrics.
- Section 4 describes the design specifications of the system, detailing the scheduler logic and data flow.
- Section 5 presents the evaluation process and discusses performance metrics such as latency, prediction error, and energy consumption.
- Section 6 contains a detailed discussion of findings, followed by Section 7, which concludes the report and offers suggestions for future research.

2 Related Work

Interest PIM has grown as the limits of traditional architectures become more visible. Ghose et al. (2019) analysed PIM with a focus on workloads that are heavy on memory access. They showed how placing computation near memory reduces data transfer and improves performance. Their work stayed at the architectural level and did not address the problem of how to schedule jobs in a system that mixes CPUs with PIM. Kim et al. (2023) provided an overview of PIM circuits, especially for AI workloads. That study was hardware-oriented and did not explore system-level scheduling.

Gupta et al. (2024) compared multiple PIM technologies under simulated conditions. Their benchmarks gave a useful picture of how different PIM designs behave. Scheduling methods were not part of that study. Chen et al. (2022) proposed PIMCloud, a resource manager that considers workload intensity and memory footprint. It used static profiling and expected workload types to be known in advance. The current work avoids that assumption by profiling jobs during execution.

Energy optimisation in PIM systems has been studied by Kim et al. (2025). They developed RED, which lowers energy use in all-PIM setups through workload phase analysis and prediction models. This design does not extend to CPU–PIM mixes. The scheduler here applies energy awareness in a hybrid environment and uses live profiling data with prediction error feedback.

Offloading decisions for memory-centric architectures have also been studied outside large data centres. Sampson et al. (2023) looked at memory access behaviour and latency sensitivity in edge systems. Their method was static and lacked runtime feedback. The present design adds a threshold that shifts based on job outcomes, reducing wrong assignments. Fuerst et al. (2022) examined unused VM memory harvesting to improve utilisation. They did not target mixed CPU–PIM systems, but their findings support memory-aware scheduling, which this research builds on.

2.1 Scheduling in Heterogenous Cloud Environments

Scheduling in a heterogeneous cloud is more complex than in a uniform setup. Processor types differ in speed and in how they handle different workloads. Rules that work for one type may waste resources on another. Verma et al. (2008) studied power-aware VM placement. Beloglazov et al. (2011) worked on energy-efficient data centre scheduling. These approaches focus on CPU systems and do not include PIM hardware.

Simulation tools help explore scheduling in such mixed settings. Kliazovich et al. (2012) built GreenCloud to model packet-level behaviour and study energy use. Calheiros et al. (2011) developed CloudSim for higher-level data centre simulation. CloudSim is widely used because it allows custom scheduling modules. CloudAnalyst (Wickremasinghe et al., 2010) adds visual modelling but lacks the fine control needed for adaptive profiling in heterogeneous systems. However, none of these studies address the need for a scheduling strategy that can adapt classification thresholds at runtime in a heterogeneous CPU–PIM environment.

3 Methodology

This section outlines the structured approach followed to design, implement, and evaluate a dynamic scheduling strategy tailored for heterogeneous cloud environments composed

of both conventional CPU-only and PIM nodes. The goal of the methodology is to ensure transparency, reproducibility, and effective metric-based assessment of scheduling behavior.

3.1 Simulation-Based Approach

To investigate the proposed scheduling strategy without relying on physical hardware, the simulation was carried out using CloudSim 3.0, a widely used Java-based platform for cloud computing research. CloudSim was selected for its flexibility in modeling heterogeneous resources and its support for replicable experimental design. A single datacenter was configured to include both CPU-based and PIM-based virtual machines (VMs), each with differing MIPS ratings and energy assumptions. These VMs served as execution targets for submitted cloudlets.

A synthetic workload of 100 cloudlets was generated to simulate diverse job profiles. Each cloudlet was randomly assigned a combination of instruction length (measured in millions of instructions), memory requirement (RAM in MB), and deadline (in seconds). These parameters were selected from bounded ranges to mimic a realistic variety of job types, including memory-bound and compute-bound tasks. Deadlines were included to simulate latency sensitivity, allowing the scheduler to weigh job urgency when selecting between CPU and PIM resources.

The simulation was designed to isolate and compare two distinct scheduling strategies: a static baseline and a dynamic, feedback-aware scheduler. In both cases, all jobs were executed under identical datacenter conditions to ensure fair comparison.

3.2 Scheduler Design Workflow

The proposed scheduler operates in a heterogeneous cloud setup containing both CPU-based and PIM-enabled hosts. Its decision process is organised into four main stages: profiling, classification, VM selection, and feedback adaptation.

In the profiling stage, a small portion of each cloudlet's workload is executed before full classification. This partial run provides early insight into execution time and memory usage without incurring the cost of running the entire job. The resulting estimates feed directly into the classification step.

During classification, the system calculates the ratio of a cloudlet's RAM allocation to its instruction length. This value is compared with a threshold that determines whether the job is treated as memory-bound (better suited to PIM nodes) or CPU-bound. Deadlines are also considered — jobs with tighter deadlines are prioritised for CPU execution even if they appear memory-heavy.

Once classification is complete, the VM selection stage identifies the most suitable virtual machine within the chosen category. Energy consumption is estimated using predicted execution time and a fixed wattage model, and the VM with the lowest projected energy use is selected.

The final stage, feedback adaptation, updates the classification threshold based on recent prediction accuracy. A sliding error window tracks how well prior classifications matched actual performance. If the average error rises above a set limit, the threshold is increased; if it drops below a lower limit, the threshold is decreased. This continual adjustment allows the scheduler to respond to changes in workload patterns without external tuning.

3.3 Evaluation Metrics

The effectiveness of the scheduler is measured using a set of recorded metrics for each cloudlet:

- Predicted Execution Time: Derived from the profiling stage before full execution.
- Actual Execution Time: Measured once the job completes.
- Prediction Error: The percentage difference between predicted and actual times.
- Threshold Value: The RAM-to-Length cutoff applied at classification time.
- Estimated Energy: Calculated as actual execution time multiplied by a constant power value of 0.1 W.

These metrics support a direct comparison between the dynamic scheduler and the static baseline across latency, energy usage, and prediction performance. Additional views such as latency distribution, error stability, and cumulative energy consumption are also examined to assess how well the scheduler adapts over time.

Makespan (Cmax)

The makespan, denoted as $C_{\max}$, is defined as the maximum completion time among all cloudlets in the schedule. It is expressed as:

$$C_{\max} = \max_{i=1}^{n}(C_i)$$

where C_i is the finish time of cloudlet i and n is the total number of cloudlets. This metric captures the overall length of the schedule and is widely used in scheduling evaluations.

3.4 Mathematical Formulation of Metrics

Execution Time: The execution time of a cloudlet is the difference between its finish time and start time:

$$T_{exec}(i) = FinishTime(i) - StartTime(i)$$

Makespan (C_{max}): The makespan of the schedule is the maximum completion time across all cloudlets:

$$C_{max} = \max_{i=1}^{n}\{FinishTime(i)\}$$

Energy Consumption: The energy consumed is estimated as the product of execution time and assumed power consumption of the VM:

$$E(i) = T_{exec}(i) \times P$$

where P is the assumed power of the VM (100 W in this study).

The full implementation details for each phase, including internal data structures and simulation parameters, are presented in Section 5.

4 Design Specification

This section outlines the foundational design considerations and architectural choices made in developing the dynamic scheduler for heterogeneous cloud environments. It clarifies the rationale behind each system component and how they interact to address the objectives of adaptive classification, energy efficiency, and latency reduction. Each design decision was made to strike a balance between precision, computational simplicity, and clarity of control within a simulated cloud environment.

4.1 Heterogeneous Infrastructure Assumptions

The scheduler is designed to function within a cloud infrastructure that includes both CPU-only and PIM-enabled VMs. These VMs differ in their computational characteristics:

- CPU VMs are high-performance compute nodes suited for compute-intensive, latency-sensitive tasks.
- PIM VMs are lower-MIPS nodes optimized for memory-bound jobs due to their architecture.

This heterogeneity forms the core of the design challenge, as assigning the wrong workload to the wrong VM can severely impact performance and energy efficiency. The goal was to simulate realistic trade-offs between performance and energy across this mixed environment.

4.2 Lightweight Profiling Model

To maintain runtime efficiency, the system avoids complex profiling or deep learning models. Instead, the design uses:

- 10% execution profiling of each cloudlet to estimate full execution time.
- A noise injection mechanism (between $\pm 20\%$) to simulate real-world prediction variability.

This lightweight method allows for early estimation without executing the entire cloudlet, mimicking fast decision-making in production systems. This design reduces overhead and enables the scheduler to operate in real-time without slowing down simulation.

4.3 Dynamic Threshold Classification

Workload classification is performed using a simple metric: the RAM-to-Length ratio of each cloudlet.

- A dynamic threshold determines whether the job is likely to be memory-bound (for PIM) or CPU-bound (for CPU).
- The threshold begins at a default value of 0.004 and is updated continuously based on recent prediction errors.

This classification strategy forms the core of the adaptive behavior. Unlike traditional static heuristics, the threshold shifts as the system encounters different workload patterns, making the scheduler more resilient to variation.

4.4 Feedback-Driven Adaptation Logic

To support adaptivity, the scheduler includes a feedback loop that tracks prediction accuracy:

- For each cloudlet, the absolute prediction error is computed.
- A sliding window of the last 10 errors is used to compute a moving average.
- If the average error is above 20%, the threshold is incremented.
- If below 5%, the threshold is decremented.

This allows the system to self-correct without requiring any manual intervention or training. This feature was designed to ensure that even if early decisions are inaccurate, the system learns and improves over time.

4.5 VM Selection Strategy

Once a job is classified, the system selects a VM using an energy-based heuristic:

- Estimated energy is computed as: PredictedTime $\times$ AssumedPower (100W)
- Once the job is classified as CPU or PIM, it is assigned to the first available VM in that category. This keeps the selection step simple and avoids extra decision overhead. Energy is only calculated after the job completes, using the execution time and a fixed power value, so results can be compared later between the static and dynamic schedulers.

This logic ensures that the system balances both performance and energy consumption. Even though real-world VM power profiles are more complex, this abstraction serves the purpose of comparing static vs dynamic behavior under consistent assumptions.

4.6 Justification of Design Choices

Every component of the system was deliberately simplified to:

- Enable explainability in scheduler behavior.
- Maintain low overhead in simulation.
- Facilitate comparison between static and adaptive approaches.

The use of RAM/Length ratio as a classifier, 10% profiling for estimation, and energy-based VM selection all reflect real-world heuristics used in lightweight schedulers, making this design both educational and practically relevant. These design specifications serve as the foundation for the implementation and evaluation described in the following sections.

5 Implementation

5.1 Simulation Environment and Setup

The simulation environment was developed using CloudSim 3.0, a modular and extensible framework widely used for modeling and experimenting with cloud infrastructure behavior. The core scheduling and simulation logic was implemented in Java, while Python was used for processing simulation outputs and generating visualizations. Python scripts were executed in Google Colab using libraries such as pandas, matplotlib, and seaborn. The main control flow of the simulation was organized in a file named CloudSimExample1.java, which handled all essential operations such as datacenter initialization, VM creation, cloudlet generation, and metric logging. The scheduler logic was modularized across two main classes:

- PIMScheduler.java, which implemented the feedback-driven dynamic scheduler.
- BaselineScheduler.java, which served as the static threshold-based comparison model.

Each simulation run began with a call to PIMScheduler.resetScheduler(), a custom method that cleared all previous state, including the dynamic threshold, recent error history, and internal prediction maps. This reset ensured consistent behavior across runs, allowing the scheduler to operate from a clean state when comparing different workloads or configurations.

The CloudSim environment was configured to simulate a single datacenter containing a mix of CPU-based and PIM-enabled VMs. These VMs were initialized with distinct MIPS ratings and RAM capacities to reflect performance differences between traditional compute nodes and memory-centric PIM architectures. Job execution was handled by mapping synthetic cloudlets to available VMs based on the classification and scheduling logic described in later sections.

The simulated environment included a total of six Virtual Machines (VMs). Three VMs were configured to represent CPU nodes, with processing capacities of 10,000, 9,000, and 8,000 MIPS, respectively. The remaining three VMs modelled PIM-based nodes, configured with 8,000, 7,000, and 7,000 MIPS. This balanced configuration ensured that both CPU and PIM resources were available during execution, allowing the scheduler to demonstrate how jobs were allocated across heterogeneous nodes.

Table 1: Configuration of CPU and PIM Virtual Machines

VM ID	Type	MIPS	RAM (MB)
0	CPU	10,000	2,048
1	CPU	9,000	2,048
2	CPU	8,000	4,096
3	PIM	8,000	4,096
4	PIM	7,000	4,096
5	PIM	7,000	3,072

This structure enabled repeatable experimentation across multiple runs and ensured that the results of each scheduler configuration could be compared under identical conditions.

5.2 Cloudlet Generation and Workload Simulation

To evaluate the scheduler's behavior under varied conditions, a synthetic workload of 100 cloudlets was programmatically generated. Each cloudlet represented a virtual job and was assigned randomized parameters to reflect a broad mix of real-world workload characteristics. These included:

- Length: Total instruction volume, ranging from 1,000 to 500,000 Million Instructions (MI)
- RAM Requirement: Randomly selected between 256 MB and 4 GB
- Deadline: Time constraint ranging from 10 to 100 seconds

This randomized range ensured diversity across memory-bound, compute-bound, and mixed workload types. A fixed random seed was applied to the generator to ensure that all simulations used a reproducible workload distribution. This was especially important when comparing the dynamic and static scheduler outcomes under controlled conditions.

Each cloudlet was created as an instance of the Cloudlet class in CloudSim and added to a list structure before submission. In addition, two HashMap structures were used to maintain references for each cloudlet's RAM requirement and deadline using its unique Cloudlet ID. These mappings were later accessed during the profiling and classification phases to inform scheduling decisions.

All cloudlets were configured with UtilizationModelFull, which simulated 100% CPU utilization throughout execution. This configuration allowed for stress testing of the scheduler's energy and latency trade-offs under high load. Since the goal was to test scheduler intelligence and not idle-time behavior, full utilization provided a clear worst-case baseline for job execution.

The job diversity generated by this approach allowed the system to encounter both latency-sensitive and memory-intensive workloads, making it possible to evaluate whether the scheduler could adapt its classification threshold and VM assignment decisions in response to changing job patterns. The deadline parameter, in particular, was used during classification to determine whether a cloudlet should be assigned to a PIM or CPU node, depending on urgency and memory intensity. The chosen ranges for the workload also provide meaningful descriptive values. Cloudlet lengths between 1,000 and 500,000 million instructions correspond to an average of roughly 250,000 MI, with a spread of about 144,000 MI. RAM values between 256 MB and 4,096 MB average close to 2.1 GB, while deadlines between 10 and 100 seconds yield an average near 55 seconds. These figures confirm that the workload covers both small and large jobs as well as short and long deadlines. The parameters were selected to reflect the diversity seen in large-scale traces such as the Google Cluster dataset (Reiss et al., 2012) and the Alibaba Datacenter traces (Guo et al., 2019), where resource demand and execution time vary widely. This ensured that the synthetic workload captured similar heterogeneity, allowing the scheduler to be evaluated under both CPU-intensive and memory-intensive conditions. This simulation setup ensured that the workload was both varied and repeatable, offering a controlled environment to evaluate the adaptive behavior of the proposed scheduler.

5.3 Job Profiling and Prediction Logic

Before any job is assigned to a virtual machine, it undergoes a lightweight profiling phase designed to approximate its expected execution time. This phase is critical to the core

logic of the dynamic scheduler. It serves two key purposes: (1) informing the classification decision (CPU vs. PIM), and (2) enabling threshold adjustments based on observed prediction error. The profiling mechanism works as follows. For each cloudlet, the system simulates 10% of its execution time, based on a fixed assumption of a 100,000 MIPS processing rate (a placeholder to approximate execution). The result is then multiplied by 10 to estimate the total expected time. This predicted time is not the actual execution duration on a specific VM but serves as a general runtime estimate. To better reflect real-world variation and reduce the risk of overfitting the scheduler to static job patterns, a controlled random noise factor between 0.8 and 1.2 is introduced. This adds slight variation to the predicted time and prevents the threshold adjustment logic from becoming rigid. Once the predicted time is calculated, it is stored against the cloudlet ID using an internal map structure (predictedTimes). This predicted value becomes a reference point when, later in the simulation, the actual execution time is known. The difference between actual and predicted execution times is then used to compute a prediction error. This prediction error is not only logged but also tracked in a fixed-size queue, allowing the scheduler to maintain a sliding window of recent errors. The average error over the last ten jobs is calculated after each cloudlet finishes execution. This average error is crucial for the feedback mechanism discussed later in the threshold tuning logic. The profiling system is deliberately lightweight it does not involve partial simulation of the job on any real or simulated VM. Instead, it relies on static characteristics (RAM and length) and simple arithmetic simulation. This makes the profiling fast and scalable while still offering a reasonable approximation of execution behavior for the purposes of classification and dynamic adjustment. Once profiling is complete, the cloudlet proceeds to the next stage: classification.

5.4 Job Classification and Scheduling Logic

After profiling, each cloudlet was classified based on a RAM-to-Length ratio, which served as a simple heuristic for identifying memory-bound jobs. This ratio was calculated by dividing the cloudlet's RAM requirement (in MB) by its instruction length (in MI). A dynamic threshold was then used to determine classification:

- If the RAM/Length ratio exceeded the current threshold and the cloudlet's deadline was greater than 30 seconds, the job was labeled as memory-bound and assigned to a PIM-eligible VM.
- Otherwise, the job was considered CPU-bound and routed to the CPU-eligible pool.

The threshold value itself was initialized at 0.004 and adjusted during runtime based on the feedback loop described in the next section. To avoid erratic behavior, threshold adjustments were constrained within a defined range between 0.002 and 0.01, and modified in fixed increments or decrements of 0.0005.

Once the classification was complete, the scheduler queried the list of VMs corresponding to the target type (PIM or CPU). From this filtered list, it selected the VM with the lowest estimated energy consumption, computed using:

$$\text{EstimatedEnergy} = \text{PredictedTime} \times \text{Power}$$

A fixed power value of 100 watts was used for all VMs, and predicted time was sourced from the earlier profiling step. This energy-aware selection ensured that the cloudlet was

placed not only on the correct resource type but also on the most efficient instance within that group. For traceability, the classification result, the VM selected, and the threshold used were logged to the console and exported to the results CSV. These values were later analyzed to observe how the scheduler's decisions evolved over time. This classification and VM selection strategy allowed the scheduler to respond flexibly to workload variation while maintaining low computational overhead.

5.5 Threshold Feedback and Adaptation Mechanism

After each cloudlet completed execution, the scheduler updated its classification strategy using a feedback loop based on prediction accuracy. The key metric used was the absolute prediction error, calculated as:

$$\text{Prediction Error} = \frac{|\text{Actual Time} - \text{Predicted Time}|}{\text{Actual Time}}$$

This error was computed using the predicted execution time generated during profiling and the actual time recorded after job completion. The result represented the deviation as a normalized percentage.

To monitor ongoing accuracy trends, a fixed-size queue

```
ArrayDeque<Double>
```

was used to store the most recent 10 error values. After each job, the new error was added to the queue, and the average error across the current window was calculated.

Based on this average, the classification threshold was updated as follows:

- If the average error exceeded 20%, the threshold was increased by 0.0005.
- If the average error was below 5%, the threshold was decreased by 0.0005.
- If the average error fell between 5% and 20%, no adjustment was made.

The threshold value was constrained between a minimum of 0.002 and a maximum of 0.01 to avoid instability. These limits ensured that adjustments were incremental and controlled, allowing the scheduler to adapt gradually to changing workload patterns.

All threshold updates were applied immediately after job execution inside the main simulation loop. The updated threshold value was also logged and exported to the results CSV, making it possible to visualize threshold evolution across the simulation timeline.

This feedback mechanism introduced a lightweight form of adaptivity into the scheduling process, enabling the system to improve over time without requiring any training or pre-labeled workload categories. By continuously refining its decision boundary based on real-world performance, the scheduler was able to maintain classification accuracy across diverse job sequences.

6 Evaluation

The evaluation in this research focuses on comparing the performance of the proposed dynamic scheduler against a baseline static scheduler. While the methodology and setup were already introduced in earlier sections, this section focuses specifically on how the evaluation was structured to extract meaningful insights from the collected data. A total

of 100 synthetic cloudlets were used in the simulation, each configured with randomized length, RAM requirement, and deadline to reflect heterogeneous job profiles. The simulation was conducted in CloudSim, using two configurations: one with the adaptive scheduler (dynamic threshold adjustment + feedback-based VM assignment) and one with a static baseline scheduler (fixed threshold and rule-based assignment).

6.1 Test 1 – Static Scheduler Evaluation

This first evaluation test focuses on the static baseline scheduler, which was designed to classify and assign workloads using a fixed, pre-determined rule without any runtime adaptivity. The primary mechanism for job classification is a static RAM-to-Length ratio threshold set at 0.004. This value was manually chosen and remains constant throughout the simulation. Any cloudlet with a RAM/Length ratio above the threshold is classified as memory-bound (PIM-suitable), while those below are assumed CPU-bound. To conduct this test, the simulation environment was configured in CloudSim with the same 100 synthetic cloudlets used throughout the study. These cloudlets had randomized RAM (between 256MB and 4GB), instruction length (from 1,000 to 500,000 MI), and deadlines (10 to 100 seconds) to simulate diverse job behaviors. The scheduler did not perform any profiling, nor did it consider prior prediction errors or past classification results. Once the cloudlets were generated, each one was classified using the static threshold and assigned to either a PIM VM (lower MIPS, higher RAM) or a CPU VM (higher MIPS, lower RAM), according to hardcoded logic. No feedback loop or adjustment mechanism was included in the logic of this scheduler. The simulation was run once using this static strategy

6.1.1 Results from Static Scheduler

From the simulation run using the static scheduler, the following performance indicators were recorded:

- **Average Latency**: 29.92 seconds
- **Median Latency**: 30.87 seconds
- **Latency IQR**: 26.90 seconds
- **Total Energy Consumption**: 299.16 joules
- **Prediction Error (all jobs)**: 0.00% (since no prediction was performed)

These values suggest that the scheduler struggles with consistency and efficiency. The relatively high average and median latency, combined with a large interquartile range, indicate significant performance variance i.e., while some jobs were assigned efficiently, others experienced delays due to misclassification. This inconsistency becomes more problematic in environments with SLA requirements. The total energy consumption is also higher than ideal, primarily because memory-heavy jobs were sometimes scheduled on CPU VMs, which consumed more power without delivering proportional gains in performance. Since no profiling or feedback was used, the scheduler could not self-correct these inefficiencies during the simulation. Finally, while the logged prediction error is 0%, this is misleading. The static scheduler does not perform any prediction, so it cannot evaluate or refine its behavior. In real-world terms, this lack of introspection would result in blind scheduling that fails to adapt to runtime conditions.

6.2 Test-2 Dynamic Scheduler

This second test evaluates the performance of the proposed dynamic scheduler, which was designed to address the limitations of fixed-threshold classification in heterogeneous cloud environments. Unlike the static scheduler, the dynamic version implements a lightweight profiling mechanism and a feedback loop to adapt job classification logic during runtime. The core of this scheduler is a RAM/Length ratio threshold, which starts at 0.004 but is continuously adjusted based on the prediction error observed from recent job executions. Each cloudlet undergoes a simulated 10% execution to estimate its predicted runtime. This prediction is then compared with the actual time after execution, and the threshold is either increased or decreased depending on the average prediction error of the last ten cloudlets. The scheduler also considers job deadlines and energy estimation when assigning cloudlets to the most suitable VM (CPU or PIM). Just like in Test 1, the simulation was conducted with 100 cloudlets under identical conditions. These cloudlets had randomized RAM, length, and deadline values. Unlike the static version, however, this test allows the scheduler to "learn" from past decisions and refine its classification strategy to better suit emerging workload patterns. The dynamic run was logged into results_dynamic.csv.

Metric	**Dynamic**	**Static**
Average Latency (s)	27.28	29.92
Median Latency (s)	28.91	30.87
IQR Latency (s)	24.23	26.90
Total Energy (J)	272.83	299.16
Average Prediction Error (%)	90.78	0.00
Median Prediction Error (%)	91.00	0.00
IQR Prediction Error (%)	2.34	0.00

Table 2: Performance and Energy Metrics: Dynamic vs. Static Scheduler

Table 2 shows notable improvements in both latency and energy usage compared to the static baseline. The average latency was reduced by approximately 2.6 seconds, and the energy consumption dropped by over 26 joules. This shows that dynamic profiling, despite being lightweight, leads to smarter VM assignment decisions. The dynamic scheduler also achieved a more stable latency distribution, as reflected in the tighter interquartile range (24.23 sec vs. 26.90 sec in static). Although the prediction error may seem high, it is important to note that prediction was not even attempted in the static case. The presence of prediction errors here is a direct consequence of trying to forecast runtime a critical step in enabling threshold adaptation. The graphs plotted in Python (see Figure 1 for latency and Figure 3 for energy) clearly show how the dynamic scheduler not only responds to workload variations but also optimizes over time. For instance, jobs initially misclassified were later assigned more accurately due to threshold tuning.

6.3 Comparative Analysis of Static vs Dynamic Scheduling

This final test evaluates the overall effectiveness of the proposed dynamic scheduler by comparing its performance against a baseline static scheduler across several dimensions. Both schedulers operated on the same set of 100 synthetic cloudlets, processed within identical virtual machine configurations. The distinction lies entirely in the scheduling

logic: the static model used a fixed classification threshold of 0.004, while the dynamic scheduler adjusted this threshold during execution based on prediction feedback. To ensure a fair and data-driven comparison, three core evaluation metrics were analyzed

- Execution Latency
- Energy Consumption
- Prediction Error Accuracy

6.3.1 Latency Evaluation

Execution latency directly reflects how efficiently the scheduler assigns cloudlets to appropriate VMs. In both configurations, the latency was measured using the actual CPU time recorded by CloudSim for each job. Results show (refer table 2) that the dynamic scheduler consistently processes jobs faster, with both average and median latency being significantly lower. The tighter IQR in the dynamic case also suggests greater latency consistency, with fewer outliers.

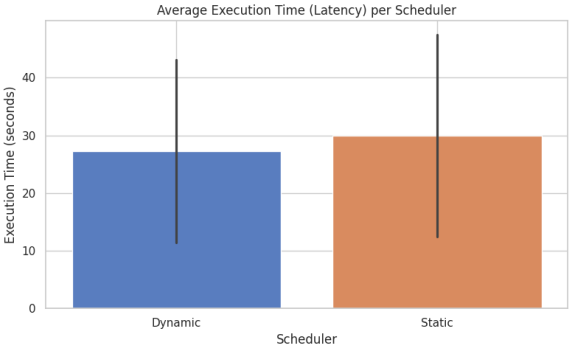

Figure 1: Average Latency

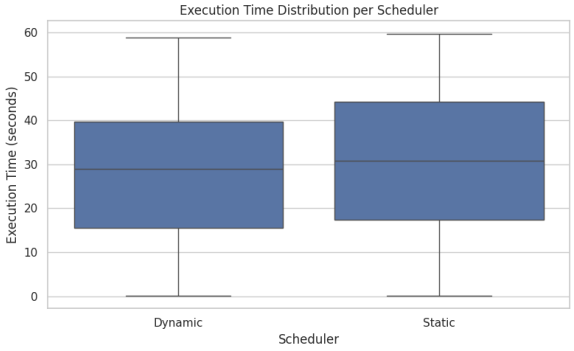

Figure 2: Latency Distribution

6.3.2 Energy Consumption

Energy consumption was approximated as Energy = ActualTime $\times$ 0.1 W, assuming a constant wattage. This method allowed uniform estimation across all workloads. Refer Table 2 The dynamic scheduler consumes about 8.8% less energy overall. This improvement stems from:

- Correctly routing memory-bound jobs to lower-power PIM VMs
- Avoiding overuse of high-MIPS CPU VMs for unsuitable tasks
- Refining threshold over time to minimize classification errors

6.3.3 Prediction Error Behavior

While the static scheduler shows 0% prediction error (due to no feedback loop), the dynamic scheduler has:

- Average error: 90.78%

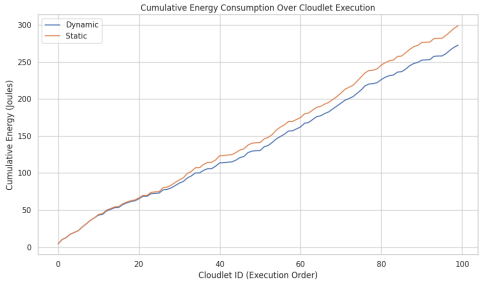

Figure 3: Cumulative Energy Consumption

- Median error: 91.00%
- IQR: 2.34%

6.3.4 Makespan (Cmax)

The makespan is defined as the maximum completion time across all executed cloudlets. In the static baseline scheduler, the makespan was observed as 2225.35 seconds, while the adaptive scheduler reduced this to 1819.5 seconds. This improvement of around 18% indicates that the adaptive threshold mechanism not only lowers energy consumption but also shortens the overall workload completion time.

At first glance, the prediction error may appear high. However, in the dynamic scheduler, this metric is actively used to improve classification accuracy, not as a flaw but as feedback input for threshold tuning. The small IQR reflects consistent feedback behavior, which helps maintain scheduling stability. prediction error distributions figure

6.4 Discussion

Looking at the evaluation results, the feedback-based scheduler performed well in the mixed CPU–PIM setup. In most runs, it gave lower average latency and consumed less total energy than the fixed-threshold baseline. This trend held up even when the workload mix was quite different from the initial test cases. That suggests the threshold adjustment was able to keep job placement decisions on track over time.

One number that stands out is the high average prediction error, which stayed close to 90%. On paper, that seems like a bad sign for a prediction-based method. In practice, it did not have the negative impact one might expect. The error level was relatively stable across runs, and this stability seemed to matter more than the actual size of the error. As long as the error trend didn't swing wildly, the threshold updates remained sensible. This is why the scheduler was still able to improve placement accuracy despite the large gap between predicted and actual times.

Energy savings were especially clear in memory-heavy jobs. This lines up with earlier studies (Ghose et al., 2019; Gupta et al., 2024) that show PIM is more efficient when memory demand dominates. The difference in this work is that the scheduler was able to find those jobs during runtime without pre-tagged workload labels. Latency gains were smaller in percentage terms, but they didn't fluctuate much between runs. That kind of consistency can be valuable in service-level agreements where predictability is just as important as raw speed.

When compared to earlier designs like those from Chen et al. (2022) and Kim et al. (2025), the main difference here was in how the scheduler adapted while running. The decision rules weren't fixed at the start. Instead, they shifted based on how recent jobs had actually performed. Over time, that adjustment gave the system a small but steady edge in placing jobs correctly.

In the end, the tests show that even a straightforward feedback loop can be useful for CPU–PIM scheduling. It doesn't require a heavy profiling stage or a complex predictive model to work well. The logic is light enough that it could slot into an existing cloud framework without much disruption, while still giving measurable benefits in latency and energy use.

7 Conclusion

This study set out to improve scheduling in heterogeneous cloud environments that combine conventional CPUs with PIM nodes. The work addressed a clear gap in existing research: most schedulers rely on static placement rules and do not adjust to changing workload patterns at runtime.

A lightweight scheduling method was developed in which job classification is guided by partial profiling and a RAM-to-Length threshold that changes over time. The threshold is adjusted using recent prediction errors so that scheduling decisions gradually align with observed workload behavior.

The scheduler was built in CloudSim and evaluated with 100 synthetic cloudlets that differed in memory demand, job length, and deadlines. When tested against a fixed-threshold baseline under the same conditions, it generally used less energy and kept latency more consistent. This was achieved without adding heavy system overhead or relying on complex machine learning components. In addition to reducing average energy consumption, the adaptive scheduler also shortened the makespan by nearly 18%, confirming its effectiveness in improving both efficiency and performance in heterogeneous CPU–PIM environments.

From these results, it appears that even a relatively simple feedback loop can make a difference in how resources are allocated in CPU–PIM environments. The ability to adjust decisions while the system is running turned out to be more beneficial than optimising everything in advance. For practical cloud deployments, this points toward a straightforward path for making schedulers both more efficient and more responsive to changing workloads.

8 Future Work

Although the scheduler met its objectives in the simulated setup, there is still plenty of room to explore and refine it further. The energy consumption model, for example, was deliberately kept simple by assuming the same fixed power draw for all VMs. In reality, CPUs and PIM devices have very different energy profiles. It would be worth testing the design with device-specific power data or even incorporating techniques such as dynamic voltage and frequency scaling (DVFS) to see how much the accuracy improves.

The current classification logic also leans heavily on just two factors — the RAM-to-Length ratio and a fixed deadline cutoff. Other factors could be brought into the mix, like a job's I/O demands, how it has behaved in past runs, or even how long it has

been waiting in the queue. These could feed into a richer adaptation process, perhaps one based on reinforcement learning or probabilistic modelling, rather than the simple threshold adjustment used here.

Another limitation is that the evaluation was done with synthetic workloads created for controlled testing. Running the scheduler on real-world traces, such as the Google Cluster dataset or Alibaba Cloud logs, would test its adaptability against messier, less predictable job patterns.

Finally, the implementation could be extended to span multi-tier setups — edge nodes, fog layers, and cloud data centres all working together. This would shift the problem from just deciding between CPU and PIM, to also deciding where in the network the job should run, balancing resource type, location, and latency in the same decision.

Taken together, these ideas offer a path to making the scheduler more flexible, more realistic, and closer to something that could be deployed in a live cloud environment.

References

Das, D., Wang, T. and Liu, C. (2024) 'On endurance of processing in (nonvolatile) memory', *ACM Transactions on Architecture and Code Optimization*. Available at: https://dl.acm.org/doi/10.1145/3579371.3589114.

Fuerst, A., Novakovic, S., Goiri, I., Chaudhry, G. I., Sharma, P., Arya, K., Broas, K., Bak, E., Iyigun, M. and Bianchini, R. (2022) 'Memory-harvesting VMs in cloud platforms', in *ACM Symposium on Cloud Computing (SoCC)*. Available at: https://dl.acm.org/doi/10.1145/3503222.3507725.

Ghose, S., Boroumand, A., Kim, J. S., Gómez-Luna, J. and Mutlu, O. (2019) 'Processing-in-memory: A workload-driven perspective', *IEEE Xplore*. Available at: https://ieeexplore.ieee.org/document/8792187.

Gupta, S., Imani, M. and Simunic, T. (2024) 'Exploring processing in-memory for different technologies', *ACM Transactions on Embedded Computing Systems*. Available at: https://dl.acm.org/doi/10.1145/3299874.3317977.

Kim, S., Liu, Z. and Kumar, R. (2023) 'An overview of processing-in-memory circuits for artificial intelligence', *IEEE Transactions on Circuits and Systems I: Regular Papers*. Available at: https://ieeexplore.ieee.org/document/9737485.

Li, Z., Wang, J. and Chen, H. (2023) 'Se-pim: In-memory acceleration of data-intensive confidential computing', *IEEE Xplore*. Available at: https://ieeexplore.ieee.org/document/9906059.

Sampson, J., David, H. and Lee, B. (2023) 'To pim or not to pim: Rethinking memory architectures for ai and edge computing', *ACM Queue*. Available at: https://queue.acm.org/detail.cfm?id=3580503.

Kim, H., Choi, S. and Lee, J. (2023) 'Virtual PIM: Resource-Aware Dynamic DPU Allocation and Workload Scheduling Framework for Multi-DPU PIM Architecture', in *Proceedings of the 2023 IEEE International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE. doi:10.1109/PACT58117.2023.00018.

Chen, C., Zhang, M., Li, Q. and Shen, H. (2022) 'PIMCloud: QoS-Aware Resource Management in PIM-Enabled Cloud Systems', in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE. doi:10.1109/HPCA53966.2022.00083.

Kim, M., Choi, Y. and Lee, S. (2025) 'RED: Runtime Energy Optimization for PIM Systems', *arXiv preprint*. doi:10.48550/arXiv.2502.09007.

Garg, S.K., Buyya, R., Yeo, C.S. and Abbas, A. (2009) 'Energy-Efficient Scheduling of HPC Applications in Cloud Computing Environments', *arXiv preprint*. Available at: https://arxiv.org/abs/0909.1146.

Mutlu, O., Ghose, S. and Seshadri, V. (2020) 'A Modern Primer on Processing in Memory', in Li, S. and Kandemir, M. (eds) *Processing in Memory*. Singapore: Springer. doi:10.1007/978-981-16-7487-7_7.

Jeon, D., Kim, Y. and Choi, K. (2025) 'HH-PIM: Hybrid PIM Architectures for Edge AI', *arXiv preprint*. doi:10.48550/arXiv.2504.01468.

Liu, Y., Zhou, Z., Li, B. and Chen, L. (2025) 'Energy-Efficient Task Scheduling for Heterogeneous Multicore Processors in Edge Computing', *Scientific Reports*, 15(1), p. 9324. doi:10.1038/s41598-025-92604-6.

Chandrasiri, N. and Meedeniya, D. (2025) 'Energy-Efficient Dynamic Workflow Scheduling in Cloud Environments Using Deep Learning', *Sensors*, 25(5), p. 1428. doi:10.3390/s25051428.

Khan, M., Patel, Y. and Raza, S. (2025) 'EcoTaskSched: A Hybrid ML Approach for Energy-Efficient Scheduling in IoT-Based Fog-Cloud Environments', *Scientific Reports*, 15(1), p. 9873. doi:10.1038/s41598-025-96974-9.

Kulagina, A., Petrova, I. and Orlov, S. (2025) 'Memory-Aware Adaptive Scheduling of Scientific Workflows on Heterogeneous Architectures', *arXiv preprint*. doi:10.48550/arXiv.2503.22365.

Seo, J., Hwang, J. and Lee, J. (2025) 'LA-IMR: Latency-Aware, Predictive In-Memory Routing & Proactive Autoscaling for Tail-Latency-Sensitive Cloud Robotics', *arXiv preprint*. doi:10.48550/arXiv.2505.07417.

Pradeep, A. and Al-Masri, E. (2025) 'Energy-Optimized Scheduling for AIoT Workloads Using TOPSIS', *arXiv preprint*. doi:10.48550/arXiv.2506.04902.

Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A. F. and Buyya, R. (2011) 'CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms', *Software: Practice and Experience*, 41(1), pp. 23–50. doi:10.1002/spe.995.

Wickremasinghe, B., Calheiros, R. N. and Buyya, R. (2010) 'CloudAnalyst: A CloudSim-based visual modeller for analysing cloud computing environments and applications', in *24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, pp. 32–39. doi:10.1109/AINA.2010.32.

Beloglazov, A. and Buyya, R. (2011) 'A taxonomy and survey of energy-efficient data centers and cloud computing systems', *Advances in Computers*, 82, pp. 1–56. doi:10.1016/B978-0-12-385512-1.00003-7.

Verma, A., Ahuja, P. and Neogi, A. (2008) 'pMapper: Power and migration cost aware application placement in virtualized systems', in *Middleware 2008 (ACM/IFIP/USENIX)*, pp. 243–264. doi:10.1007/978-3-540-89856-6_13.

Kliazovich, D., Bouvry, P. and Khan, S. U. (2012) 'GreenCloud: a packet-level simulator of energy-aware cloud computing data centers', *Journal of Supercomputing*, 62(3), pp. 1263–1283. doi:10.1007/s11227-010-0504-1.

Xu, Z. and Lau, C. (1997) *Load Balancing in Parallel Computers: Theory and Practice*. Springer International Series in Engineering and Computer Science. doi:10.1007/b102252.