RESEARCH ARTICLE

Cost-Optimized Data Partitioning for Parallel Sorting on Heterogeneous Distributed Systems using Mixed Integer Programming

Joeniño Cainday and Dr. Junar Landicho

Department of Computer Science, University of Science and Technology of Southern Philippines, Cagayan De Oro City, Philippines

ARTICLE HISTORY
Compiled May 5, 2025

ABSTRACT
This research addresses the optimization of data partitioning for parallel computations in heterogeneous distributed systems, with the dual objective of minimizing financial expenditure and execution time. We develop a Mixed Integer Programming (MIP) model that incorporates critical system constraints including CPU speed, memory capacity, memory latency, network latency, and budget limitations. Our model replaces the heuristic partitioning phase of a parallel quicksort algorithm, which serves as our performance benchmark. Utilizing synthetic datasets representing both workload and node specifications, our simulations focus on pre-execution data partitioning decisions. The goal is to provide a framework for financial and time optimization within cloud computing environments, where efficient resource allocation and budget management are essential. Results demonstrate the effectiveness of our MIP-based approach in achieving improved load distribution and reduced runtime compared to traditional heuristics.

KEYWORDS
Data Partitioning; Mixed Integer Programming; Heterogeneous Distributed Systems; Budget Constraints; Makespan Minimization

1. Introduction

The widespread adoption of cloud computing and serverless architectures has increased the importance of efficient resource allocation in distributed systems. This research addresses the challenge of statically partitioning data across heterogeneous computing environments, where nodes vary in processing speed, memory capacity, network latency, and monetary implications.

We formulate data partitioning as an NP-hard combinatorial optimization problem aimed at minimizing execution time (makespan) while adhering to defined constraints. Our approach replaces traditional heuristic-based partitioning in parallel sorting algorithms with a novel Mixed Integer Programming (MIP) model designed to capture comprehensive system constraints for more balanced, resource-aware data distributions.

1.1. Background and Motivation

Distributed computing enables parallel processing of large-scale datasets across multiple nodes. However, in heterogeneous systems with diverse computational capabilities, simplistic data partitioning strategies often result in load imbalances and inefficient resource utilization [1]. Common approaches like uniform distribution or partitioning based solely on heuristics typically fail to consider system compositions in cloud environments.

Cloud computing introduces new complexities to data partitioning. These environments offer flexibility with pay-as-you-go pricing models where computing resources are available on demand [5]. This necessitates re-evaluating traditional partitioning methodologies to explicitly consider economic factors alongside performance metrics. Effective strategies must balance high performance with budget limitations—minimizing both computational time and overall expenditure for cloud resource utilization.

While Monga and Lodhi [53] demonstrated benefits of heterogeneity-aware allocation, their model primarily considered CPU speed alone. Our work introduces a more sophisticated MIP-based model that integrates multiple system attributes to achieve globally optimized data distributions, providing a more realistic and budget-conscious approach to optimized data processing.

1.2. Cost Considerations in Heterogeneous Distributed Systems

This paper specifically addresses statically partitioning large datasets before executing parallel sorting across simulated heterogeneous environments. Our primary objective is to minimize the overall execution time (makespan) while ensuring we satisfy the defined constraints. As a secondary objective, we aim to identify the most budget-effective solution with an equivalent makespan, using a lexicographic optimization approach.

In this context, cost refers to the pricing of cloud compute nodes. While higher-priced instances often imply better performance, this is not guaranteed, as pricing and performance can vary across vendors. Each simulated node has synthetic attributes including processing speed, memory capacity, network characteristics, and pricing models inspired by real cloud offerings, reflecting typical variations in heterogeneous cloud deployments.

Allocating data across heterogeneous resources inherently involves performance-cost tradeoffs [48]. As highlighted in previous research, balancing processing time with financial implications when assigning datasets to different machines is critical. Effective data partitioning can significantly reduce both runtime and monetary cost, while suboptimal partitioning leads to inflated expenses or resource limitations like memory overflows. Our controlled synthetic environment enables repeatable experiments independent of real-world cloud variability.

MIP provides a powerful mathematical framework for complex optimization problems involving both discrete and continuous decision variables subject to linear constraints [8]. This capability makes MIP well-suited for modeling cost-aware data partitioning in heterogeneous systems:

- Discrete variables represent partitioning decisions (assigning specific data blocks to particular nodes)
- Continuous variables model resource utilization levels and associated costs
- The objective function represents total financial expenditure
- Constraints reflect resource limitations and system characteristics

Through this approach, MIP effectively finds optimal or near-optimal partitioning

strategies that minimize expenditure while satisfying performance requirements.

1.3. Research Contributions

This paper contributes the following:

(1) A MIP formulation for initial data partitioning in parallel quicksort on heterogeneous systems;
(2) An integration of the MIP model into an existing heterogeneous sorting framework;
(3) A performance evaluation using synthetic datasets with extended metrics;
(4) A reproducible experimental setup and discussion of directions for future work.

2. Related Work

Heterogeneous scheduling and data allocation have been studied in various contexts. [48] considered dataset allocation across geo-distributed clouds with two objectives (processing time and cost) [48]. They formulated a linear program to place data blocks on VMs to minimize weighted sum of time and cost, demonstrating Pareto trade-offs. Similarly, [49] used an integer linear program for data assignment in a hybrid heterogeneous processing environment. These works focus on resource assignment rather than within-job data partitioning, but they highlight the utility of mathematical programming for cloud costs.

Classic scheduling theory demonstrates that even simple load-balancing on unrelated machines is NP-hard [50]. For instance, scheduling jobs to minimize makespan in admits a 2-approximation algorithm but cannot be solved optimally in polynomial time beyond trivial cases [50]. This complexity motivates the treatment of cost-aware data partitioning as a combinatorial optimization problem, which is closely related to but with additional cost considerations. This can be extended to multi-objective formulations (e.g., cost and makespan), though even the single-objective version is NP-hard, reducible to scheduling on unrelated parallel machines [50].

In the domain of parallel sorting, many algorithms assume a homogeneous machine model. For instance, Parallel Sorting by Regular Sampling (PSRS) chooses pivots to create equally-sized partitions [51], and typical benchmarks use randomly-generated data with uniform or other distributions [51]. These benchmarks (e.g. uniform 32-bit integer inputs) guide our synthetic data choices. However, PSRS and related methods do not account for cost or heterogeneous speeds.

In big-data systems like Spark, dynamic partitioning and scheduling algorithms have been proposed. [52], for example, developed a dynamic partitioning strategy for intermediate Spark data to mitigate skew, and a greedy scheduling method that considers node speed [52]. They find that balanced partitioning significantly lowers completion time [52]. Our work differs by focusing on static initial partitioning with explicit cost metrics, rather than in-job rebalancing. To the best of our knowledge, prior work has not explicitly applied MIP to cost-aware static data partitioning in heterogeneous cloud sort workloads.

We build upon the model proposed by Monga and Lodhi [**?**], which partitions data across heterogeneous nodes based on CPU performance to balance load during parallel sorting. However, their approach does not consider critical real-world factors such as network latency or resource cost—key limitations in cloud and serverless settings.

Moreover, their method performs initial local sorting and sampling before defining data ranges, which can lead to uneven memory usage and load imbalance during the redistribution phase.

3. Problem Formulation

We now formalize the partitioning problem based on the system characteristics introduced in Section 1.3. To ensure full control over system parameters and facilitate reproducibility, we operate within a synthetic environment. Each node is defined by parameters such as processing speed, memory capacity, network latency, bandwidth, and cost, inspired by real-world cloud configurations (e.g., AWS, GCP). This abstraction enables rigorous evaluation of partitioning strategies without relying on live infrastructure.

3.1. System Model

We partition the dataset into contiguous blocks of size d, assigning each block to a unique node. Consider a heterogeneous cluster with N nodes, where each node i is characterized by the following parameters:

- d_i: Assigned data volume (in units of d)
- r_i: Processing rate (in d/t — data units per time unit)
- m_i: Maximum data volume the node can handle at once (in units of d)
- ℓ_i: Network latency (in time units t)
- b_i: Network bandwidth (in d/t — data units per time unit)
- u_i: Usage billing rate (in c/t — cost units per time unit)

The units above are illustrative and can be adapted depending on context. For instance, processing speed r may be measured in records/ms, memory m in megabytes (MB), and cost u in USD/hour. In this work, we use normalized or synthetic units to model relative performance and cost differences between nodes, without binding the system to a specific infrastructure or currency.

3.2. Derived Parameters

We derive additional parameters from the node characteristics to facilitate the MIP formulation:

3.2.1. Processing Time

$$x_i = \frac{d_i}{r_i} \tag{1}$$

where x_i is the time required by node i to process its assigned data volume d_i.

3.2.2. Transfer Time

$$y_i = \ell_i + \frac{d_i}{b_i} \tag{2}$$

where y_i is the communication overhead for node i, computed from latency ℓ_i and bandwidth b_i.

3.2.3. Total Cost

$$w = \sum_{i=1}^{N} u_i \cdot x_i \tag{3}$$

where w denotes the cumulative cost of utilizing all selected nodes.

3.3. Optimization Objectives

The primary objective of our MIP model is to minimize the total execution time (makespan) of the parallel sorting process. This ensures that the overall completion time is minimized, taking into account both computation and communication overheads. Formally, the primary objective is defined as:

$$\min z = \max_{i=1}^{N}(x_i + y_i) \tag{4}$$

where z represents the maximum time taken by any node i to complete its assigned data processing and communication tasks. This formulation captures the makespan of the entire parallel sorting operation, ensuring that the slowest node determines the overall completion time. In addition to minimizing makespan, we aim to choose the most cost-effective solution among equivalent combinations. To achieve this, we introduce a secondary objective that minimizes the total financial expenditure. This is incorporated into the model using a lexicographic optimization approach:

$$\min \ z + \varepsilon \cdot w \tag{5}$$

In this formulation, w represents the cumulative cost of utilizing the selected nodes, and ε is set to a negligible value (e.g., 10^{-6}) to prioritize makespan minimization while breaking ties in favor of cost efficiency. This approach ensures that among solutions with equivalent makespan, the one with the lowest cost is selected.

3.4. Constraint Definitions

The proposed MIP model is subject to several constraints that reflect system limitations and ensure a feasible allocation of data across nodes. These constraints incorporate resource boundaries (e.g., memory and budget), performance considerations (e.g., makespan), and completeness of the partitioning scheme.

3.4.1. Makespan Constraint

$$z \geq \frac{d_i}{r_i} + \ell_i + \frac{d_i}{b_i} \quad \forall i \in \{1, \ldots, N\} \tag{6}$$

This constraint ensures that the total execution time, represented by the variable z, is at least as large as the time taken by any individual node i to both process its assigned data and transfer it to the next stage. It combines computation time $\frac{d_i}{r_i}$ with communication latency ℓ_i and data transfer time $\frac{d_i}{b_i}$.

3.4.2. Memory Constraint

$$d_i \leq m_i \quad \forall i \in \{1, \ldots, N\} \tag{7}$$

Each node has a limited memory capacity m_i, and this constraint ensures that the volume of data d_i assigned to node i does not exceed its available memory.

3.4.3. Coverage Constraint

$$\sum_{i=1}^{N} d_i = D \tag{8}$$

This constraint enforces full data allocation: the entire dataset of size D must be partitioned and distributed among the N available nodes without omission or duplication.

3.4.4. Non-negativity Constraint

$$d_i \geq 0 \quad \forall i \in \{1, \ldots, N\} \tag{9}$$

This standard constraint ensures that each node receives a non-negative volume of data, reflecting the physical impossibility of assigning negative quantities.

3.4.5. Budget Constraint (Optional)

$$0 \leq \sum_{i=1}^{N} u_i \cdot d_i \leq B \tag{10}$$

In addition to node-specific parameters, we optionally define a user-defined input B, representing the maximum allowable expenditure. This parameter must be a non-negative number, with a default value of 0. Including this constraint models scenarios where financial limitations must be respected.

4. Heterogeneous PSRS Baseline Algorithm

This is the model of Monga and Lodhi [53] which we will use as a baseline. The algorithm is designed to sort a list of items using multiple workers, each with different speeds. The main steps of the algorithm are as follows:

Algorithm 1: Heterogeneous PSRS Baseline Algorithm

Inputs:

- *data*: Input data to be sorted
- n: Total number of data items
- p: Number of processors
- $perf[0 \dots p-1]$: Relative performance of each processor

Output:

- Sorted data

Phase 1: Local Sorting and Sampling
for *each processor* i **do**
  $size_i = \frac{n \cdot perf[i]}{\sum perf}$
  Assign $size_i$ items to processor i
  Processor i performs sequential quicksort on its local data
  Select $L = (p-1) \cdot perf[i]$ samples from local sorted data
  Send samples to the designated coordinator
Phase 2: Pivot Selection
Designated coordinator gathers and sorts all samples
Selects $p-1$ global pivots and broadcasts them to all processors
Phase 3: Data Redistribution
for *each processor* **do**
  Partition local data using global pivots
  Send partitioned data to corresponding processors
  Retain the portion corresponding to its final range
Phase 4: Final Sorting and Merge
Each processor sorts its final local data
Coordinator concatenates all sorted segments for final output

4.1. Limitations of the Baseline Algorithm

While the Heterogeneous PSRS algorithm by Monga and Lodhi [**?**] offers an effective strategy for distributing sorting tasks across processors of varying speeds, it introduces several inefficiencies. The algorithm performs local sorting before establishing the global data partitions, which can result in skewed data distributions during the redistribution phase. This may lead to certain processors receiving significantly larger portions of data, causing load imbalance and increased memory consumption. Furthermore, the algorithm does not account for communication cost or network latency, which are vital considerations in distributed and cloud-based environments. These shortcomings motivate the need for a more adaptive approach that considers both performance and cost constraints when assigning data ranges.

5. MIP-Guided Sorting Algorithm for Heterogeneous Distributed Systems

We propose a modified algorithm that replaces the sampling and redistribution stages with a Mixed Integer Programming (MIP) formulation that statically partitions the global key range across nodes.

5.1. Bucket Assignment

Given the total dataset size D and a heterogeneous set of N nodes with parameters $(r_i, m_i, \ell_i, b_i, u_i)$, we solve the MIP formulation (Section **??**) to compute optimal partition sizes d_i such that:

- Processing time is balanced according to each node's capability
- Communication and processing costs are minimized
- Constraints on memory, load, and budget are respected

Once d_i values are determined, each node is assigned a fixed key range. For example (illustrative only):

Worker	**Percentage of Total Data**	**Assigned Range**
A (4x speed)	50%	[0, 49]
B (3x speed)	30%	[50, 79]
C (2x speed)	20%	[80, 100]

5.2. Proposed Algorithm

Algorithm 2: Hybrid Bucket-Quicksort

Inputs:

- D: Total dataset
- N: Number of nodes
- r_i: Processing rate of node i
- m_i: Memory capacity of node i
- b_i: Network bandwidth of node i
- ℓ_i: Network latency of node i
- u_i: Usage cost rate of node i

Output:

- Sorted data

Phase 1: Static Partitioning via MIP
Solve MIP to determine partition sizes d_i for each node
Assign key ranges to nodes based on d_i and global key distribution
Phase 2: Data Distribution
for *each item* x *in* D **do**
  Route x to processor i such that x lies in i's assigned key range
Phase 3: Local Sorting
for *each processor* i **do**
  Sort received data using sequential quicksort
Phase 4: Final Merge (Optional)
Concatenate sorted outputs from all nodes

5.3. Distribution Strategies

5.3.1. Option A: Centralized Bucketing (Low Latency, High Memory)

- A single coordinator receives all input data and routes items to target nodes
- Requires 2× dataset size in memory
- Low latency and simple coordination, but limited scalability

5.3.2. Option B: Asynchronous Routing (Low Memory, High Latency)

- Each item is routed asynchronously to its responsible node
- Lower memory footprint, suited for streaming or distributed sources
- May incur higher latency due to decentralized routing
- Increases billing costs due to more frequent node usage

6. Theoretical Complexity Analysis

A rigorous complexity analysis highlights the theoretical advantages of our MIP-guided approach over the baseline PSRS algorithm, particularly in heterogeneous environments.

6.1. Time Complexity

6.1.1. PSRS (Baseline)

The original Heterogeneous PSRS algorithm involves four distinct phases, each contributing to the overall time complexity:

- **Phase 1: Local Sorting and Sampling:** Each of the p processors sorts its local data of size $O(n/p)$ on average, taking $O\left(\frac{n}{p}\log\frac{n}{p}\right)$ time using quicksort. Sampling takes $O(p)$ time per processor. The communication of samples to the coordinator takes $O(p^2)$ in the worst case if all processors send their $O(p)$ samples sequentially. Thus, this phase has an average time complexity of $O\left(\frac{n}{p}\log\frac{n}{p}+p^2\right)$.
- **Phase 2: Pivot Selection:** The coordinator sorts $O(p^2)$ samples, which takes $O(p^2\log p^2)=O(p^2\log p)$ time. Broadcasting the $p-1$ pivots to all p processors takes $O(p)$ time in parallel or $O(p^2)$ sequentially.
- **Phase 3: Data Redistribution:** Each processor partitions its $O(n/p)$ local data using $p-1$ pivots, which takes $O(n/p\cdot p)=O(n)$ in the worst case if the pivots are poorly chosen. The redistribution of data among processors can take up to $O(n)$ in the worst-case scenario where one processor receives almost all the data.
- **Phase 4: Final Sorting and Merge:** Each processor sorts its received data, which can be up to $O(n)$ in the worst case, leading to $O(n\log n)$ for a single processor if load balancing is poor. The coordinator then merges the p sorted segments, which takes $O(n)$ time.

Combining these phases, the average-case time complexity of the original PSRS algorithm is dominated by the local sorting and pivot selection, yielding $O\left(\frac{n}{p}\log\frac{n}{p}+p^2\log p+n\right)$. Under ideal load balancing, this simplifies to $O\left(\frac{n\log n}{p}+p^2\log p\right)$. The worst-case time complexity, particularly due to potential imbalances in data redistribution, can reach $O(n\log n)$ if one processor ends up with most

of the data, or even $O(n^2/p)$ if local sorting on an imbalanced partition degrades to quadratic time.

6.1.2. MIP-Guided Algorithm

Our proposed MIP-guided algorithm significantly streamlines the process:

- **Phase 1: Static Partitioning via MIP:** The MIP formulation is solved once as a precomputation step. The complexity of solving a general MIP is NP-hard. However, for a fixed number of nodes N, the complexity can be bounded by a polynomial function of the input size, which in our case relates to the number of nodes and the complexity of the constraints. Assuming an efficient solver, and considering the number of variables (proportional to N) and constraints (also proportional to N), a loose upper bound can be considered around $O(N^k)$ for some constant k, potentially up to $O(N^3)$ for simpler formulations or higher for more complex ones. This is a one-time overhead.
- **Phase 2: Data Distribution:**
  - **Option A (Centralized Bucketing):** The coordinator processes each of the n items once to route it, taking $O(n)$ time.
  - **Option B (Asynchronous Routing):** Each of the n items is routed directly to a processor. Assuming the routing decision is constant time, this phase takes $O(n)$ time in total, potentially distributed across the network.
- **Phase 3: Local Sorting:** Each of the N processors sorts its assigned data of size d_i. Since the MIP aims to balance the load based on processing rates, we expect $d_i \approx O(n/N)$. Thus, the local sorting on each processor takes approximately $O\left(\frac{n}{N} \log \frac{n}{N}\right)$. Executed in parallel, the time for this phase is $O\left(\frac{n}{N} \log \frac{n}{N}\right)$.
- **Phase 4: Final Merge (Optional):** If a final merged output is required, the coordinator concatenates the N sorted segments, taking $O(n)$ time. This step can be avoided if the output is acceptable as a set of distributed sorted partitions.

Therefore, the overall time complexity of the MIP-guided algorithm, after the initial MIP solving, is dominated by the data distribution and local sorting phases, resulting in an average and worst-case time complexity of $O(N^k + n + \frac{n}{N} \log \frac{n}{N})$. If N is significantly smaller than n, this simplifies to $O(n + \frac{n}{N} \log \frac{n}{N})$. The one-time MIP solving overhead $O(N^k)$ becomes negligible for large datasets processed multiple times.

6.2. Space Complexity

6.2.1. PSRS (Baseline)

- **Local Storage:** Each processor needs to store its assigned data, which is $O(n/p)$ on average.
- **Sampling:** Each processor generates $O(p)$ samples, and the coordinator stores $O(p^2)$ samples.
- **Redistribution Buffers:** During data redistribution, processors might need temporary buffers to hold data being sent or received, potentially up to $O(n/p)$ in size per processor.
- **Final Sorted Segments:** Each processor holds a sorted segment of approximately $O(n/p)$.

The total space complexity of the PSRS algorithm is $O(n)$ as the total data is parti-

tioned and stored across the processors. The coordinator requires additional space for the samples, $O(p^2)$.

6.2.2. MIP-Guided Algorithm

- **Partition Storage:** Each of the N nodes stores its assigned partition of size d_i, with $\sum d_i = n$. Thus, the total storage for the data is $O(n)$.
- **Data Distribution:**
  - **Option A (Centralized Bucketing):** The coordinator temporarily holds the entire dataset for routing, requiring $O(n)$ additional space, leading to a total of $O(2n)$ peak memory usage at the coordinator.
  - **Option B (Asynchronous Routing):** Each item is routed individually, minimizing the need for a central buffer. The memory footprint remains $O(n)$ distributed across the nodes.
- **Local Sorting:** Each processor sorts its local partition d_i in-place or requires $O(d_i)$ auxiliary space, which is within the $O(n)$ total.

The space complexity of the MIP-guided algorithm is $O(n)$ in the distributed asynchronous routing scenario (Option B). The centralized bucketing strategy (Option A) incurs a temporary $O(2n)$ space requirement at the coordinator. The space required for storing the MIP model and its solution is dependent on the solver and the complexity of the formulation but is typically much smaller than the dataset size n, especially for a fixed number of nodes N.

6.3. Complexity Comparison

In this section, we formally analyze and compare the time complexity of the baseline PSRS algorithm and the proposed MIP-guided algorithm. The analysis emphasizes both average-case behavior and worst-case scenarios under heterogeneous processing environments.

6.3.1. Baseline PSRS Algorithm

The PSRS algorithm executes in four distinct phases: local sorting and sampling, pivot selection, data redistribution, and final sorting or merging. The overall time complexity is characterized as follows:

$$\mathcal{O}\left(\frac{n}{p}\log\frac{n}{p} + p^2\log p + n\right) \tag{11}$$

Under ideal load balancing and efficient communication, this simplifies to:

$$\mathcal{O}\left(\frac{n\log n}{p} + p^2\log p\right) \tag{12}$$

However, due to dynamic pivot selection and possible load imbalance, the worst-case time complexity may degrade to:

$$\mathcal{O}(n \log n) \tag{13}$$

This degradation arises when a significant portion of data is allocated to a single processor due to poor pivot selection.

6.3.2. Proposed MIP-Guided Algorithm

The MIP-guided algorithm separates the partitioning step as a one-time optimization via a Mixed Integer Programming formulation. After solving the MIP, the data is distributed based on a precomputed partitioning strategy that accounts for processor heterogeneity. The total time complexity is expressed as:

$$\mathcal{O}\left(N^k + n + \frac{n}{N} \log \frac{n}{N}\right) \tag{14}$$

Here, N is the number of processors and k is a constant that depends on the complexity of the MIP formulation. This overhead is incurred once and amortized over multiple executions. The dominant runtime costs after MIP solving are data distribution and local sorting.

6.3.3. Summary

Table 1. Asymptotic Time Complexity Comparison

Metric	**PSRS (Baseline)**	**MIP-Guided Algorithm**
Average Time	$\mathcal{O}\left(\frac{n \log n}{p} + p^2 \log p\right)$	$\mathcal{O}\left(N^k + n + \frac{n}{N} \log \frac{n}{N}\right)$
Worst-Case Time	$\mathcal{O}(n \log n)$	$\mathcal{O}\left(N^k + n + \frac{n}{N} \log \frac{n}{N}\right)$
Preprocessing Overhead	None	$\mathcal{O}(N^k)$ (amortized)
Load Balancing	Data-dependent (dynamic)	Optimized (static)

The MIP-guided algorithm offers theoretical advantages in both average and worst-case runtime. By removing the need for sampling and pivot-based partitioning at runtime, it eliminates a primary source of imbalance and communication overhead. In contrast, the PSRS algorithm is more susceptible to skewed load distributions and requires additional synchronization steps, which may limit scalability on heterogeneous hardware. Although solving the MIP model incurs an initial computational cost, this cost becomes negligible in scenarios involving repeated sorting tasks or large datasets. As such, the MIP-guided approach is especially advantageous in environments where preprocessing is feasible and performance consistency is critical. The MIP-guided algorithm demonstrates superior asymptotic performance under realistic workload and platform assumptions, particularly when amortizing the MIP overhead across repeated executions or in systems with significant processor heterogeneity.