RESEARCH ARTICLE

Optimistic Static Data Partitioning for Cost-Aware Parallel Sorting in Heterogeneous Distributed Systems

Joeniño Cainday and Junar Landicho

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

ARTICLE HISTORY
Compiled June 5, 2025

ABSTRACT
This research addresses the challenge of efficient static data partitioning for parallel sorting in heterogeneous distributed systems, aiming to minimize both execution time (makespan) and monetary cost. Current methods often struggle with load balancing and resource utilization, particularly in cost-sensitive cloud environments. We introduce Heterogeneous Parallel Sorting by Linear Programming (H-PSLP), an approach that employs a linear programming (LP) model to determine optimal data allocation across nodes with varying performance, memory capacities, and costs. H-PSLP replaces the heuristic, sampling-based partitioning phase of traditional algorithms like Heterogeneous Parallel Sorting by Regular Sampling (H-PSRS), which serves as our primary performance benchmark.

Simulations were conducted using synthetic datasets (uniform and Gaussian distributions) and heterogeneous cluster configurations (4 and 8 nodes). The results demonstrate H-PSLP's superiority over H-PSRS. Across all experiments, H-PSLP achieved an average makespan reduction of 33.3% and an average total cost reduction of 60.7% compared to H-PSRS. Critically, H-PSLP consistently adhered to node memory constraints, preventing the memory overflows that plagued H-PSRS in all tested scenarios. The LP optimization overhead was minimal, and H-PSLP proved robust to different data distributions and scaled effectively. These findings highlight H-PSLP as a more reliable, efficient, and economically viable solution for parallel sorting in heterogeneous environments. Future work will focus on distributed merge strategies and refining the LP model.

KEYWORDS
Data Partitioning, Parallel Sorting, Heterogeneous Distributed Systems, Cost Optimization, Makespan Minimization, Linear Programming, Resource Allocation, Cloud Computing, Load Balancing, Memory Constraints, Static Data Partitioning

1. Introduction

The widespread adoption of cloud computing has heightened the importance of efficient resource allocation in distributed systems. Traditional data partitioning methods—such as uniform or round-robin—often fall short in heterogeneous environments, leading to load imbalance and poor utilization of resources. These challenges are amplified when data is distributed across geographically distant nodes, where differences in hardware capabilities and increased communication overhead introduce latency and reduce overall system throughput [5].

In addition to performance concerns, cost has become a central factor in partitioning decisions with the rise of pay-as-you-go cloud platforms. While premium instances may offer better performance, this is not guaranteed across providers. Consequently, partitioning strategies must account for both execution time and monetary cost. Ideally, partitioning decisions should fall on the Pareto front—where no improvement in one metric, such as speed or cost, is possible without compromising the other [5]. Inefficient strategies can result in inflated expenses or system failures, such as memory overflows. Thus, effective solutions must balance these competing objectives.

This study addresses the problem of static data partitioning for parallel sorting in cost- and

resource-constrained distributed environments. To capture the heterogeneity of computing nodes, we consider variations in usage costs, memory capacities, and processing capabilities. Specifically, we abstract the combined effects of communication latency, bandwidth, and computation speed into a unified processing throughput metric, defined as the relative rate at which a node can complete its assigned tasks. For example, a node with twice the throughput of another can process and communicate data approximately twice as fast. In practice, this throughput (Perf[i]) could be estimated through benchmarking specific instance types or by analyzing historical performance data. To address this multifaceted optimization problem, we propose a linear programming (LP)-based approach that produces cost-aware and load-balanced data partitioning schemes for parallel sorting. The model seeks to minimize execution time (makespan) while also reducing overall resource costs, without incurring significant performance penalties. In contrast to heuristic or sampling-based approaches, the LP formulation guarantees globally optimal solutions under linear constraints. Furthermore, its polynomial-time solvability [3] makes it well-suited for deployment in real-world heterogeneous cloud environments.

2. Related Work

In the domain of parallel sorting, many algorithms operate under the assumption of a homogeneous computing environment. For instance, Parallel Sorting by Regular Sampling (PSRS) selects pivots to divide the dataset into equally sized partitions [4], and commonly used benchmarks are based on synthetically generated data with uniform or similarly idealized distributions. However, such methods do not account for the heterogeneity inherent in real-world distributed systems. Addressing this limitation, Monga and Lodhi [2] introduced a modified version of PSRS designed for heterogeneous environments. Their approach improves load balancing by allocating data in proportion to each node's relative throughput. Nevertheless, their method does not consider other crucial system attributes. Additionally, the reliance on regular sampling introduces computational overhead associated with global sampling and all-to-all redistribution. Their procedure also entails initial local sorting and sampling prior to defining data ranges, which may result in uneven memory usage and load imbalance during the redistribution phase.

Moreover, the effectiveness of their partitioning strategy may be sensitive to the characteristics of the input dataset. For example, their use of custom dataset construction—potentially employing techniques such as least common multiple (LCM)-based schemes—may not guarantee optimal or balanced partitioning for arbitrary or non-uniform data distributions, potentially leading to suboptimal resource utilization. While a commonly used strategy for approximating ideal continuous allocations in integer-constrained systems is the largest remainder method (also known as Hamilton's method), which is widely applied in apportionment and resource allocation problems, its application in H-PSRS would still be subject to the limitations of sampling-based pivot selection.

In the context of distributed systems, this research distinguishes itself from existing works such as Yoon and Kamal's "Optimal Dataset Allocation in Distributed Heterogeneous Clouds" [5] by adopting a more focused approach for parallel sorting. While Yoon and Kamal broadly tackle optimal dataset allocation across geographically distributed cloud data centers using a multi-objective linear programming (LP) model to find a Pareto front that simultaneously minimizes processing time and monetary cost, this research employs a single, scalarized LP objective for makespan minimization with cost as weighted secondary consideration. This allows for a direct and computationally efficient solution tailored to data partitioning in heterogeneous nodes, simplifying complex inter-node considerations like detailed communication and data transfer speeds into a generalized "throughput" metric, which encapsulates latency, bandwidth, and compute speeds, for practical application in parallel sorting tasks.

In big-data systems like Spark, dynamic partitioning and scheduling algorithms have been proposed. For example,[1] developed a dynamic partitioning strategy for intermediate Spark data to mitigate skew, and a greedy scheduling method that considers node speed. They find that balanced partitioning significantly lowers completion time. Our work differs by focusing on static

initial partitioning with explicit metrics, rather than in-job rebalancing.

3. Methodology

3.1. Baseline Approach: H-PSRS

As established in Section 2, existing heterogeneous sorting approaches, particularly H-PSRS by Monga and Lodhi [2], provide a foundation for load-balanced sorting but have significant limitations in cost optimization and global optimality. We use H-PSRS as our baseline comparison since it represents the current state-of-the-art in heterogeneous parallel sorting.

H-PSRS operates through four sequential phases: local sorting and sampling, pivot selection via coordinator, all-to-all data redistribution, and final merge. While this approach successfully adapts PSRS for heterogeneous environments by proportional workload allocation, our analysis identified key areas for improvement that directly inform our design decisions.

3.1.1. H-PSRS Complexity Analysis

We analyze the computational and space complexity of H-PSRS across its four phases:

Phase I (Local Sorting and Sampling):

- *Time Complexity:* Each node i sorts its proportionally allocated data chunk of size $n_i = n \cdot \frac{\text{Perf}[i]}{\sum_{j=0}^{p-1} \text{Perf}[j]}$, requiring $O(n_i \log n_i)$ time. Sample extraction takes $O((p-1) \cdot \text{Perf}[i])$ time per node. Since operations occur in parallel, the bottleneck is $\max_i(n_i \log n_i + (p-1) \cdot \text{Perf}[i])$.
- *Space Complexity:* Each node requires $O(n_i)$ space for its data chunk and $O(p \cdot \text{Perf}[i])$ space for samples.

Phase II (Pivot Selection):

- *Time Complexity:* The coordinator collects $O(p^2)$ total samples and sorts them in $O(p^2 \log p)$ time, creating a sequential bottleneck that cannot be parallelized.
- *Space Complexity:* The coordinator requires $O(p^2)$ space to store and process all samples from nodes.

Phase III (Data Redistribution):

- *Time Complexity:* Each node partitions its data using $(p-1)$ pivots in $O(n_i \log p)$ time using binary search. All-to-all communication involves $O(n)$ total data movement across the network, with each node potentially sending and receiving $O(n/p)$ data on average.
- *Space Complexity:* Each node requires temporary buffers of size $O(n_i)$ for partitioning and $O(n/p)$ for receiving data from other nodes.

Phase IV (Final Merge and Sort):

- *Time Complexity:* Each node merges p sorted sublists (one from each node) in $O(n_i \log p)$ time, then performs final local sorting in $O(n_i \log n_i)$ time. Coordinator concatenation requires $O(n)$ time.
- *Space Complexity:* Each node uses $O(n_i)$ space for merged data, and the coordinator requires $O(n)$ space for the final result.

Overall Complexity:

- *Total Time Complexity:* $O(\max_i(n_i \log n_i) + p^2 \log p + n)$
  - The $O(p^2 \log p)$ pivot selection creates a sequential bottleneck that limits scalability
  - Communication overhead $O(n)$ becomes significant in distributed environments
  - Under balanced allocation, $\max_i(n_i \log n_i) \approx O(\frac{n \log n}{p})$
- *Total Space Complexity:* $O(n + p^2)$, where $O(n)$ is distributed across nodes and $O(p^2)$ is concentrated at the coordinator

3.2. Proposed Approach: H-PSLP

Building upon the limitations identified in Section 2, we propose Heterogeneous Parallel Sorting by Linear Programming (H-PSLP). Our approach fundamentally redesigns the resource allocation strategy by replacing heuristic sampling with mathematical optimization, directly addressing the cost-awareness and global optimality gaps in existing methods.

Algorithm 1 Heterogeneous Parallel Sorting by Linear Programming (H-PSLP)

Require: Dataset S of size n, nodes $N_0, \ldots, N_{p-1}$ with performance Perf, memory Mem, cost Cost
Ensure: Sorted dataset S' with optimized makespan and cost
Phase I: Linear Optimization
Solve LP model (Section 3.2.1) to obtain optimal workload allocation $x[i]$

Phase II: Data Partitioning
Apply largest remainder method to convert fractional LP solution to integer allocation

Phase III: Parallel Local Sorting
for each node N_i **in parallel do**
    Sort assigned chunk C_i locally: $O(|C_i| \log |C_i|)$
end for

Phase IV: Multi-way Merge
Coordinator performs p-way merge using min-heap: $O(n \log p)$
return Sorted dataset S'

3.2.1. Linear Programming Formulation

Our LP model optimizes workload allocation by minimizing a scalarized objective that combines makespan and cost. This approach assumes linearity in performance and cost metrics; while this is a common simplification for LP, it may not fully capture complex non-linear behaviors (e.g., network congestion, tiered pricing).

$$\text{minimize} \quad T + \lambda \sum_{i=0}^{p-1} \text{Cost}[i] \cdot \frac{x[i] \cdot \beta}{\text{Perf}[i]} \tag{1}$$

$$\text{subject to} \quad \sum_{i=0}^{p-1} x[i] = n \tag{2}$$

$$x[i] \leq \text{Mem}[i] \quad \forall i \in \{0, \ldots, p-1\} \tag{3}$$

$$T \geq \frac{x[i] \cdot \beta}{\text{Perf}[i]} \quad \forall i \in \{0, \ldots, p-1\} \tag{4}$$

$$x[i] \geq 0 \quad \forall i \in \{0, \ldots, p-1\} \tag{5}$$

where T represents the makespan, $x[i]$ is the workload assigned to node i, β is a computational complexity factor, λ is the cost-performance trade-off parameter, and p is the number of nodes. The factor β normalizes the workload $x[i]$ to a time unit, effectively encapsulating the constant factors associated with the sorting algorithm's per-element operations (e.g., the log factor in $O(N \log N)$ complexity). The parameter λ allows for adjusting the relative importance of execution time versus monetary cost in the optimization; a higher λ places greater emphasis on reducing cost, potentially at the expense of a slightly increased makespan, thus enabling exploration of the Pareto front.

3.2.2. H-PSLP Complexity Analysis

We analyze the computational and space complexity of H-PSLP across all four phases:

Phase I (Linear Optimization):

- *Time Complexity:* The LP formulation contains p variables and $2p + 1$ constraints. Using interior-point methods, the solver requires $O(p^3)$ time in the worst case. Modern specialized LP solvers can achieve better performance, with some reaching $O(p^{2.5})$ for well-conditioned problems.
- *Space Complexity:* The constraint matrix requires $O(p^2)$ space, with additional $O(p)$ space for variables and dual solutions.

Phase II (Data Partitioning):

- *Time Complexity:* Converting fractional LP solutions to integer allocations using the largest remainder method requires $O(p \log p)$ time for sorting fractional parts, plus $O(p)$ for remainder distribution.
- *Space Complexity:* Temporary arrays for fractional parts and sorted indices require $O(p)$ space.

Phase III (Parallel Local Sorting):

- *Time Complexity:* Each node i sorts its assigned chunk C_i of size n_i in $O(n_i \log n_i)$ time. Since sorting occurs in parallel, the bottleneck is $\max_i(n_i \log n_i)$. Under optimal LP allocation, this approaches $O(\frac{n \log n}{p})$ per node.
- *Space Complexity:* Each node requires $O(n_i)$ space for its data chunk. Total distributed space is $O(n)$.

Phase IV (Multi-way Merge):

- *Time Complexity:* The p-way merge using a min-heap processes n elements with $O(\log p)$ operations per element, resulting in $O(n \log p)$ total time. This phase is centralized at the coordinator, which could become a bottleneck for extremely large datasets if the coordinator node's resources are insufficient.
- *Space Complexity:* The coordinator requires $O(n)$ space for the final sorted array and $O(p)$ space for the min-heap structure.

Overall Complexity:

- *Total Time Complexity:* $O(p^3 + p \log p + \frac{n \log n}{p} + n \log p)$
  - For large datasets where $n \gg p^3$, this simplifies to $O(\frac{n \log n}{p} + n \log p)$
  - The LP overhead $O(p^3)$ becomes negligible as n grows, making the algorithm scale well with data size
- *Total Space Complexity:* $O(n + p^2)$, where $O(n)$ is for data storage and $O(p^2)$ is for LP formulation

Comparison with H-PSRS:

- *Time Improvement:* H-PSLP eliminates the $O(p^2 \log p)$ sampling bottleneck and $O(n)$ all-to-all communication overhead present in H-PSRS
- *Space Improvement:* H-PSLP requires $O(p^2)$ space for LP formulation vs. H-PSRS's $O(p^2)$ for sample storage, but eliminates temporary redistribution buffers
- *Scalability:* H-PSLP scales better with increasing node count p due to the elimination of quadratic sampling overhead

3.3. Key Advantages of H-PSLP

Compared to H-PSRS, our approach offers several advantages:

(1) **Global Optimality:** LP formulation guarantees globally optimal solutions under linear

constraints, unlike heuristic sampling approaches.
(2) **Cost Awareness:** Explicit incorporation of monetary costs in resource allocation decisions, crucial for cloud environments.
(3) **Memory Constraint Handling:** Direct enforcement of memory limits prevents system failures due to resource exhaustion.
(4) **Reduced Communication Overhead:** Elimination of all-to-all redistribution phase significantly reduces network traffic.
(5) **Robustness to Input Data Characteristics:** The workload allocation strategy is determined by node capabilities and costs, making the algorithm's performance less sensitive to the specific statistical distribution of the input data, providing consistent optimization across diverse datasets.

4. Experimental Setup

This section details the simulation framework, cluster configurations, dataset characteristics, performance metrics, and experimental design used to evaluate the proposed H-PSLP algorithm against the H-PSRS baseline and a sequential sorting approach.

4.1. Simulation Environment and Tools

The experiments were conducted using a custom-built Python simulation framework. Key libraries included Python's `multiprocessing` for modeling parallel execution, `heapq` for H-PSLP's merge phase, and Google's `OR-Tools` (with the CBC solver) for the Linear Programming (LP) optimization in H-PSLP. Data generation and plotting utilized `numpy` and `matplotlib`.

4.2. Cluster Configuration Generation

Heterogeneous cluster configurations were generated for each experimental run, varying in node count and specific capabilities. Key parameters for cluster generation included:

- **Node Count (p):** Experiments were run with $p = 4$ and $p = 8$ nodes.
- **Node Throughput (Perf$[i]$):** Relative processing throughput for each node was randomly sampled from an integer range of [1, 10].
- **Node Memory (Mem$[i]$):** Total cluster memory was set to 1.5 times the dataset size (memory_overprovision_factor = 1.5). This total memory was then distributed among the nodes using random fractions that sum to unity.
- **Node Billing Cost (Cost$[i]$):** Billing rates for each node were sampled uniformly from a range of [0.1, 1.0] arbitrary cost units per unit of simulated time.
- **Random Seeds:** Seeds 4 and 5 were used to generate different cluster configurations and initial data orderings.

4.3. Heterogeneous Performance Simulation

The simulation of processing times on heterogeneous nodes was achieved as follows:

(1) **Base Sequential Throughput:** For each dataset, a `base_throughput` (in µs/element) was established by performing a sequential sort on the entire dataset. The formula used was: base_throughput = (time_end_seq − time_start_seq) × 10^6/data_size.
(2) **Simulating Task Execution Time on Nodes:** For any task on a simulated node i, the actual execution time of the Python operation was measured. This `actual_time_sec` was scaled using its assigned `relative_throughput` (Perf$[i]$): simulated_delay_µs = (actual_time_sec × 10^6)/Perf$[i]$.

(3) **Cost Calculation:** The monetary cost for a task on node i was: task_cost = simulated_delay_µs × Cost[i].
(4) **LP Model Time Estimation:** The `base_throughput` was used by H-PSLP's LP model to estimate processing time for `data_assigned[i]` items on node i: estimated_time_µs_for_LP = (data_assigned[i] × base_throughput)/Perf[i].

4.4. Dataset Generation and Characteristics

Datasets were synthetically generated:

- **Dataset Size (N):** Determined using the LCM scheme: $N = k \times \sum_{i=0}^{p-1} \text{Perf}[i] \times \text{LCM}(\text{Perf}[0], \ldots, \text{Perf}[p-1])$, where $k(\text{data\_scale}) = 300$.
- **Data Distributions:**
  - **Uniform Distribution:** Integers randomly sampled from $[0, N]$.
  - **Gaussian Distribution:** Integers sampled from a normal distribution with a mean of 252,000 and a standard deviation of 100,000, clipped to $[0, N]$.
- **Data Element Type:** Integers.
- **Visual Representation of Datasets:** Plots illustrating the distribution of each dataset are presented alongside their respective experimental run results in Section 5.2.

4.5. Performance Metrics

Key performance indicators included:

- **Makespan:** Total execution time (µs).
- **Total Cost:** Cumulative monetary cost.
- **Preprocessing Overhead (µs):** Time for H-PSRS initial data partitioning (coordinator) or H-PSLP solver execution.
- **Memory Utilization:** Percentage of node memory used.
- **LP Model Estimates (for H-PSLP):**
  - `m̂akespan`: Predicted makespan for the parallel local sorting phase.
  - `ĉost`: Predicted total cost for the parallel local sorting phase.
- **Speedup:** Sequential Baseline Makespan / Parallel Algorithm Makespan.

4.6. Experimental Design and Procedure

Experiments covered:

- Node Counts: 4, 8.
- Data Distributions: Uniform, Gaussian.
- Random Seeds: 4, 5.

A total of 8 distinct simulation runs were analyzed. Each run involved executing the Sequential Baseline, H-PSRS, and H-PSLP, and recording metrics.

5. Results

This section presents a comprehensive performance evaluation of the sequential sorting baseline, the Regular Sampling approach (simulating H-PSRS), and the Static Optimized approach (simulating H-PSLP).

5.1. Performance Metrics Calculation

For clarity, the primary metrics are calculated as follows:

H-PSRS (Regular Sampling Approach):

- **Makespan:** Sum of [Coordinator Initial Partition Time + Max(Node Local Sort & Sample Time) + Coordinator Pivot Selection Time + Max(Node Local Data Partition & Transfer Time) + Max(Node Final Sort Time) + Coordinator Final Merge Time].
- **Total Cost:** Sum of costs from all individual steps across all nodes.

H-PSLP (Static Optimized Approach):

- **Makespan:** Solver Time + Max(Node Local Sort Time) + Min-Heap Merge Time.
- **Total Cost:** Solver Cost + Sum(All Node Local Sort Costs) + Min-Heap Merge Cost.

5.2. Detailed Results per Experimental Run

We detail the results for each of the 8 provided runs with performance metrics. Node configurations varied per run. Full details for each run's configuration can be found in the Appendix.

5.2.1. Run 1: Uniform Distribution, 4 Nodes, Seed 4

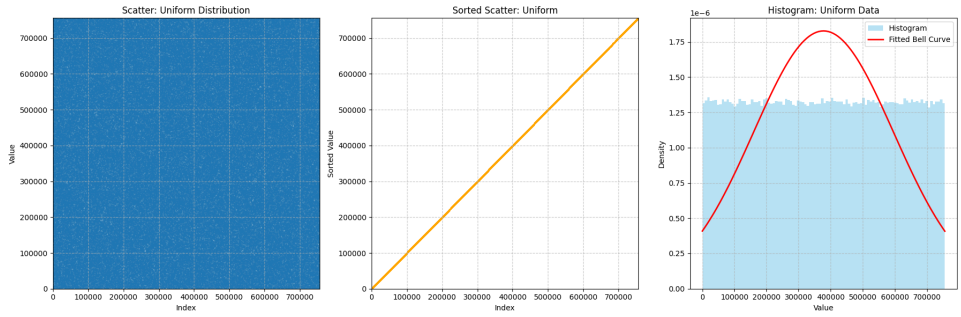


Figure 1.: Dataset distribution for Run 1 - Uniform, 4 Nodes, Seed 4

Cluster & Data: 4 Nodes, Data Size = 756,000 elements.
Sequential Baseline: Time = 180,655.9 µs, Throughput = 0.238963 µs/element.

Table 1.: Performance metrics for Run 1

Metric	H-PSRS	H-PSLP
Makespan (µs)	38,686.0	**29,793.6**
Total Cost ($)	56,206.33	**28,788.23**
Preprocessing Overhead (µs)	392.990	375.450
H-PSRS Memory Overflow	N3 (425.4%), N1 (233.9%)	N/A
H-PSLP Max Mem. Util.	N/A	N1 (100.0%)
LP Model Predicted Makespan (sort) (µs)	N/A	**14,044.3**
Actual Max Local Sort Time (µs)	N/A	16,422.7
LP Model Predicted Cost (sort) ($)	N/A	**17,655.42**
Actual Sum of Local Sort Costs ($)	N/A	21,189.53

5.2.2. Run 2: Uniform Distribution, 4 Nodes, Seed 5

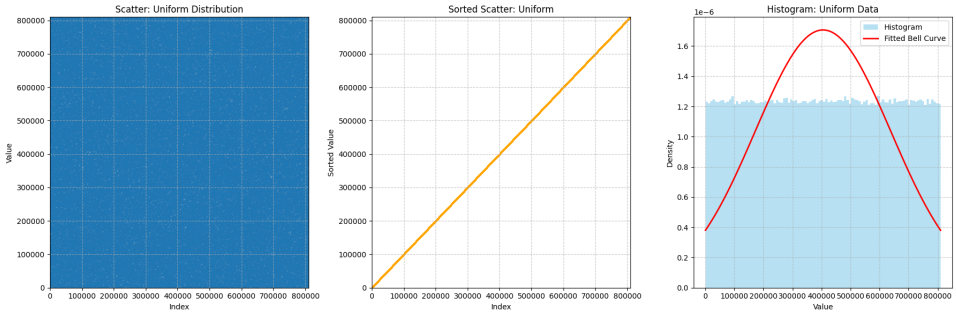


Figure 2.: Dataset distribution for Run 2 - Uniform, 4 Nodes, Seed 5

Cluster & Data: 4 Nodes, Data Size = 810,000 elements.
Sequential Baseline: Time = 186,810.6 µs, Throughput = 0.230630 µs/element.

Table 2.: Performance metrics for Run 2

Metric	H-PSRS	H-PSLP
Makespan (µs)	34,005.6	**20,633.4**
Total Cost ($)	30,512.35	**13,557.69**
Preprocessing Overhead (µs)	258.199	288.044
H-PSRS Memory Overflow	N0 (1599.0%)	N/A
H-PSLP Max Mem. Util.	N/A	N1 (99.8%)
LP Model Predicted Makespan (sort) (µs)	N/A	**9,145.8**
Actual Max Local Sort Time (µs)	N/A	9,790.4
LP Model Predicted Cost (sort) ($)	N/A	**9,921.22**
Actual Sum of Local Sort Costs ($)	N/A	10,049.08

5.2.3. Run 3: Uniform Distribution, 8 Nodes, Seed 4

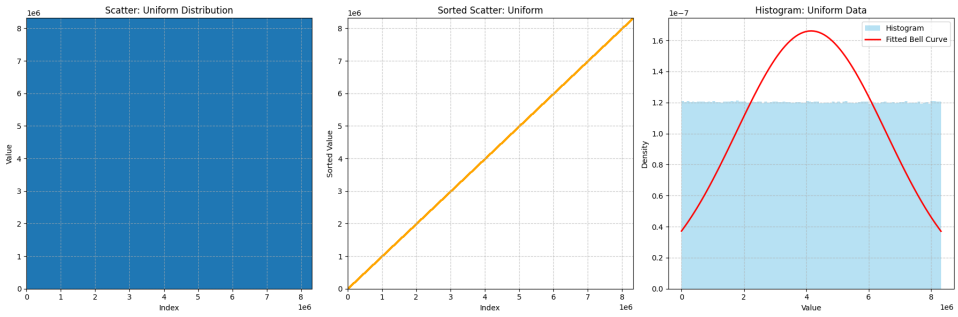


Figure 3.: Dataset distribution for Run 3 - Uniform, 8 Nodes, Seed 4

Cluster & Data: 8 Nodes, Data Size = 8,316,000 elements.
Sequential Baseline: Time = 2,726,167.9 µs, Throughput = 0.327822 µs/element.

Table 3.: Performance metrics for Run 3

Metric	H-PSRS	H-PSLP
Makespan (µs)	445,299.3	**311,808.0**
Total Cost ($)	816,014.97	**284,321.96**
Preprocessing Overhead (µs)	4635.303	8935.205
H-PSRS Memory Overflow	N3 (1454.8%), N0 (979.9%)	N/A
H-PSLP Max Mem. Util.	N/A	N2 (99.7%)
LP Model Predicted Makespan (sort) (µs)	N/A	**125,524.1**
Actual Max Local Sort Time (µs)	N/A	122,323.7
LP Model Predicted Cost (sort) ($)	N/A	**303,521.59**
Actual Sum of Local Sort Costs ($)	N/A	274,580.60

5.2.4. Run 4: Uniform Distribution, 8 Nodes, Seed 5

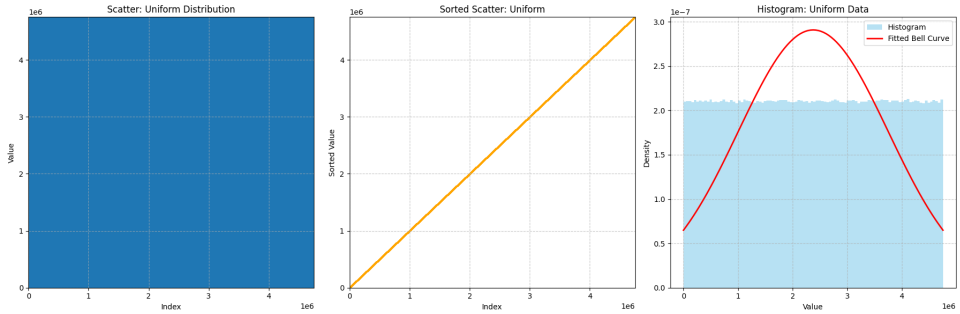


Figure 4.: Dataset distribution for Run 4 - Uniform, 8 Nodes, Seed 5

Cluster & Data: 8 Nodes, Data Size = 4,752,000 elements.
Sequential Baseline: Time = 1,512,629.2 µs, Throughput = 0.318314 µs/element.

Table 4.: Performance metrics for Run 4

Metric	H-PSRS	H-PSLP
Makespan (µs)	272,898.9	**113,946.7**
Total Cost ($)	306,606.23	**110,491.35**
Preprocessing Overhead (µs)	1610.111	361.807
H-PSRS Memory Overflow	N4 (205.9%), N5 (1422.2%), N1 (170.2%)	N/A
H-PSLP Max Mem. Util.	N/A	N5 (100.1%)
LP Model Predicted Makespan (sort) (µs)	N/A	**47,603.4**
Actual Max Local Sort Time (µs)	N/A	43,836.4
LP Model Predicted Cost (sort) ($)	N/A	**136,047.90**
Actual Sum of Local Sort Costs ($)	N/A	87,424.32

5.2.5. Run 5: Gaussian Distribution, 4 Nodes, Seed 4

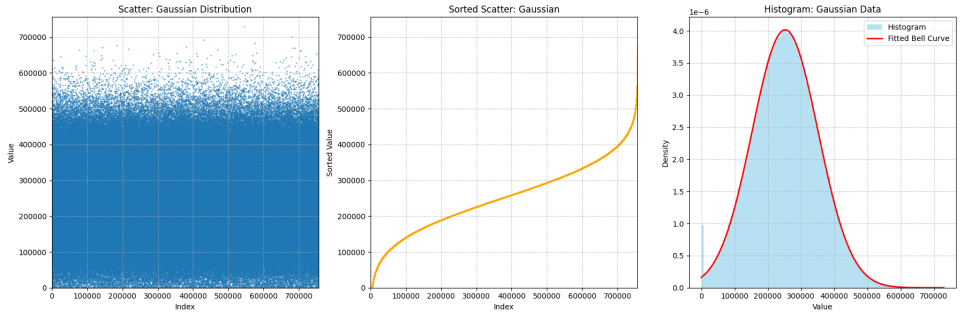


Figure 5.: Dataset distribution for Run 5 - Gaussian (mean 252k, std 100k), 4 Nodes, Seed 4

Cluster & Data: 4 Nodes, Data Size = 756,000 elements.
Sequential Baseline: Time = 178,218.0 µs, Throughput = 0.235738 µs/element.

Table 5.: Performance metrics for Run 5

Metric	H-PSRS	H-PSLP
Makespan (µs)	40,108.8	**30,114.0**
Total Cost ($)	54,000.01	**26,420.90**
Preprocessing Overhead (µs)	334.683	477.432
H-PSRS Memory Overflow	N3 (425.4%), N1 (233.9%)	N/A
H-PSLP Max Mem. Util.	N/A	N1 (100.0%)
LP Model Predicted Makespan (sort) (µs)	N/A	**13,854.7**
Actual Max Local Sort Time (µs)	N/A	18,009.5
LP Model Predicted Cost (sort) ($)	N/A	**17,417.16**
Actual Sum of Local Sort Costs ($)	N/A	21,842.52

5.2.6. Run 6: Gaussian Distribution, 4 Nodes, Seed 5

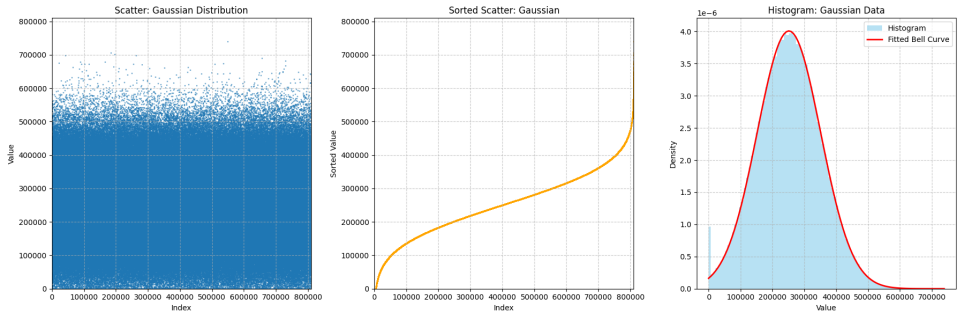


Figure 6.: Dataset distribution for Run 6 - Gaussian (mean 252k, std 100k), 4 Nodes, Seed 5

Cluster & Data: 4 Nodes, Data Size = 810,000 elements.
Sequential Baseline: Time = 195,203.4 µs, Throughput = 0.240992 µs/element.

Table 6.: Performance metrics for Run 6

Metric	H-PSRS	H-PSLP
Makespan (µs)	36,290.1	**22,019.5**
Total Cost ($)	33,712.17	**12,447.08**
Preprocessing Overhead (µs)	323.418	313.903
H-PSRS Memory Overflow	N0 (1599.0%)	N/A
H-PSLP Max Mem. Util.	N/A	N1 (99.8%)
LP Model Predicted Makespan (sort) (µs)	N/A	**9,556.7**
Actual Max Local Sort Time (µs)	N/A	11,038.9
LP Model Predicted Cost (sort) ($)	N/A	**10,366.95**
Actual Sum of Local Sort Costs ($)	N/A	12,135.29

5.2.7. Run 7: Gaussian Distribution, 8 Nodes, Seed 4

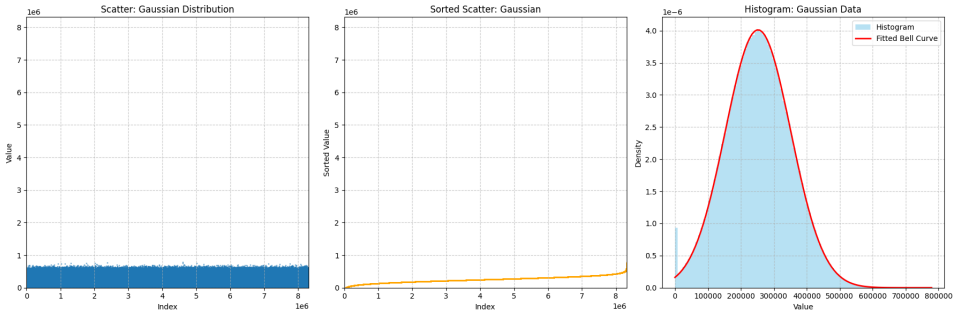


Figure 7.: Dataset distribution for Run 7 - Gaussian (mean 252k, std 100k), 8 Nodes, Seed 4

Cluster & Data: 8 Nodes, Data Size = 8,316,000 elements.
Sequential Baseline: Time = 2,850,812.1 µs, Throughput = 0.342810 µs/element.

Table 7.: Performance metrics for Run 7

Metric	H-PSRS	H-PSLP
Makespan (µs)	446,139.2	**319,017.1**
Total Cost ($)	839,765.87	**292,759.19**
Preprocessing Overhead (µs)	5090.880	8935.205
H-PSRS Memory Overflow	N3 (1454.8%), N0 (979.9%)	N/A
H-PSLP Max Mem. Util.	N/A	N2 (99.7%)
LP Model Predicted Makespan (sort) (µs)	N/A	**125,524.1**
Actual Max Local Sort Time (µs)	N/A	122,323.7
LP Model Predicted Cost (sort) ($)	N/A	**303,521.59**
Actual Sum of Local Sort Costs ($)	N/A	274,580.60

5.2.8. Run 8: Gaussian Distribution, 8 Nodes, Seed 5

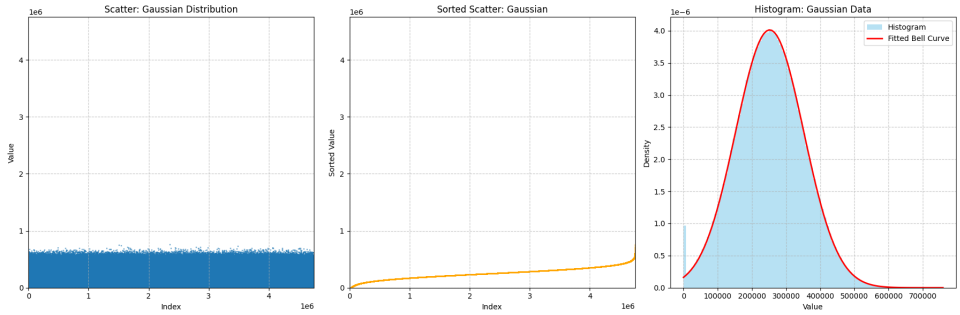


Figure 8.: Dataset distribution for Run 8 - Gaussian (mean 252k, std 100k), 8 Nodes, Seed 5

Cluster & Data: 8 Nodes, Data Size = 4,752,000 elements.
Sequential Baseline: Time = 1,540,436.0 µs, Throughput = 0.324166 µs/element.

Table 8.: Performance metrics for Run 8

Metric	H-PSRS	H-PSLP
Makespan (µs)	178,432.2	**124,284.2**
Total Cost ($)	428,716.62	**154,602.50**
Preprocessing Overhead (µs)	2432.684	261.470
H-PSRS Memory Overflow	N5 (1422.2%), N4 (205.9%), N1 (170.2%)	N/A
H-PSLP Max Mem. Util.	N/A	N5 (100.1%)
LP Model Predicted Makespan (sort) (µs)	N/A	**48,478.5**
Actual Max Local Sort Time (µs)	N/A	44,617.9
LP Model Predicted Cost (sort) ($)	N/A	**138,548.89**
Actual Sum of Local Sort Costs ($)	N/A	126,599.78

5.3. Aggregated Results and Averages

For configurations with two seeds, we calculate averages and standard deviations (StdDev).

5.3.1. Configuration: Uniform Distribution, 4 Nodes (Runs 1 & 2)

Table 9.: Aggregated results for Uniform Distribution, 4 Nodes

Metric (Avg ± StdDev)	H-PSRS	H-PSLP
Makespan (µs)	36,345.8 ± 3,309.6	**25,213.5 ± 6,477.2**
Total Cost ($)	43,359.34 ± 18,168.4	**21,172.96 ± 10,769.7**

5.3.2. *Configuration: Uniform Distribution, 8 Nodes (Runs 3 & 4)*

Table 10.: Aggregated results for Uniform Distribution, 8 Nodes

Metric (Avg ± StdDev)	H-PSRS	H-PSLP
Makespan (µs)	359,099.1 ± 121,907.0	**212,877.4 ± 139,910.0**
Total Cost ($)	561,310.60 ± 360,210.0	**197,406.66 ± 123,000.0**

5.3.3. *Configuration: Gaussian Distribution, 4 Nodes (Runs 5 & 6)*

Table 11.: Aggregated results for Gaussian Distribution, 4 Nodes

Metric (Avg ± StdDev)	H-PSRS	H-PSLP
Makespan (µs)	38,199.5 ± 2,693.0	**26,066.8 ± 5,724.1**
Total Cost ($)	43,856.09 ± 14,345.6	**19,433.99 ± 9,881.1**

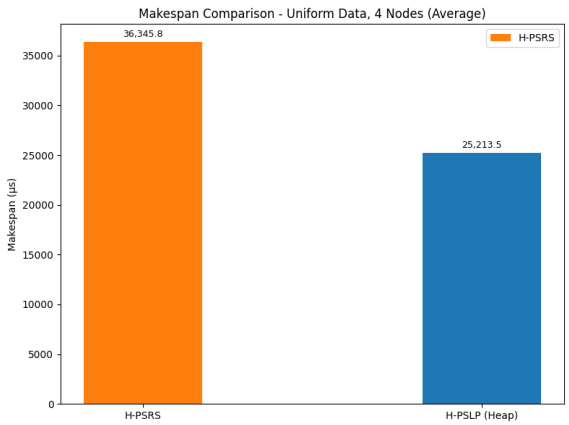
5.3.4. *Configuration: Gaussian Distribution, 8 Nodes (Runs 7 & 8)*

Table 12.: Aggregated results for Gaussian Distribution, 8 Nodes

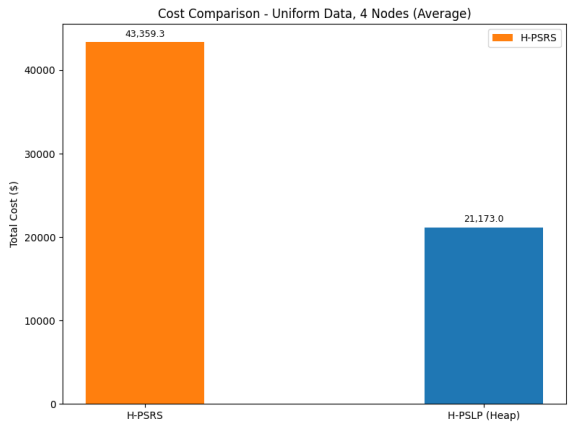
Metric (Avg ± StdDev)	H-PSRS	H-PSLP
Makespan (µs)	312,285.7 ± 189,297.0	**221,650.7 ± 137,700.0**
Total Cost ($)	634,241.25 ± 290,650.0	**223,680.85 ± 97,691.0**

5.4. Visual Performance Comparisons

The following charts visually summarize the performance comparisons based on the aggregated experimental data and specific run examples.

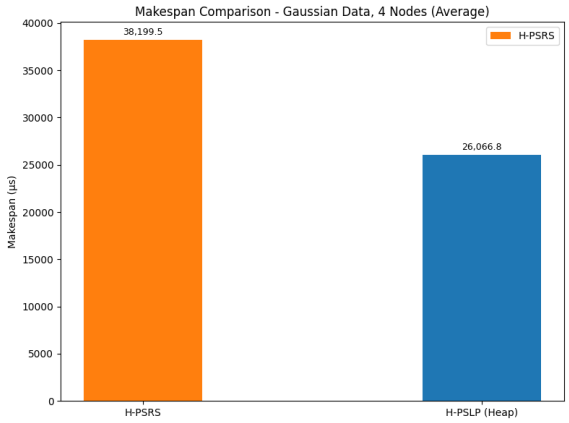


(a) Makespan - Uniform, 4 Nodes

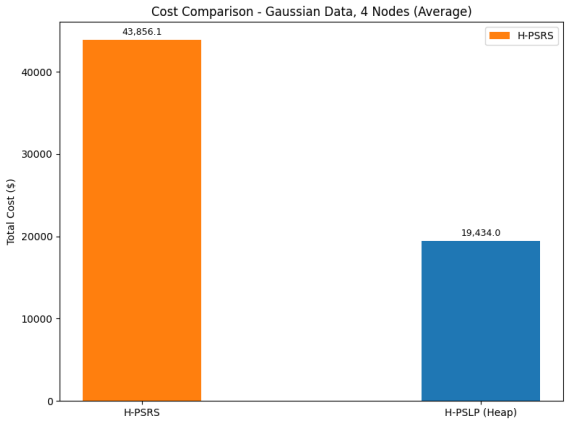


(b) Cost - Uniform, 4 Nodes

Figure 9.: Performance Comparison: Uniform Data, 4 Nodes (Average)



(a) Makespan - Gaussian, 4 Nodes



(b) Cost - Gaussian, 4 Nodes

Figure 10.: Performance Comparison: Gaussian Data, 4 Nodes (Average)

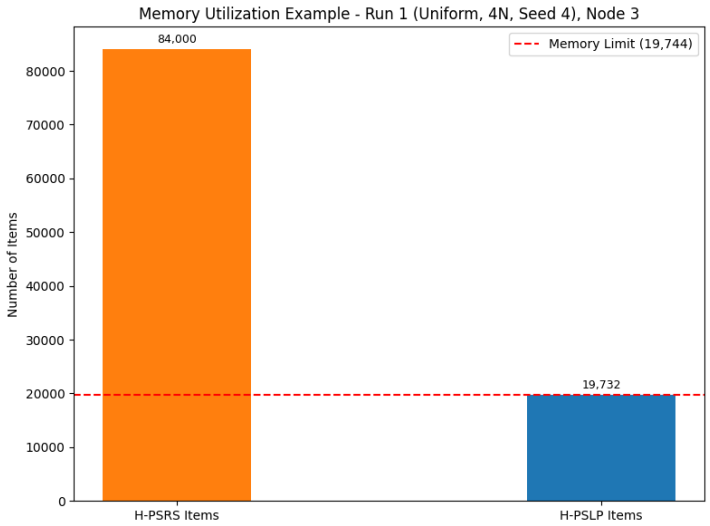
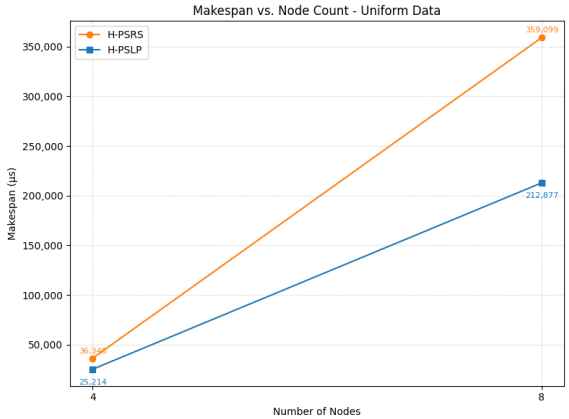
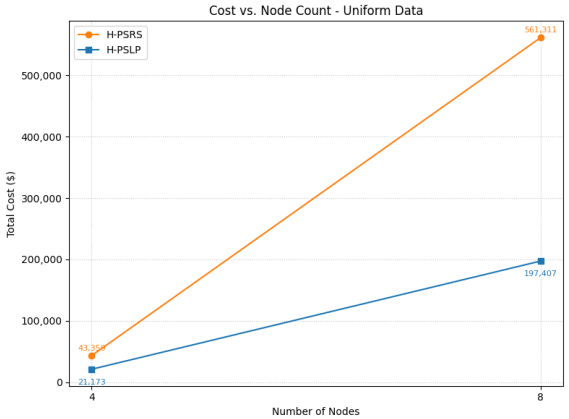


Figure 11.: Memory Utilization Example - Run 1, Node 3

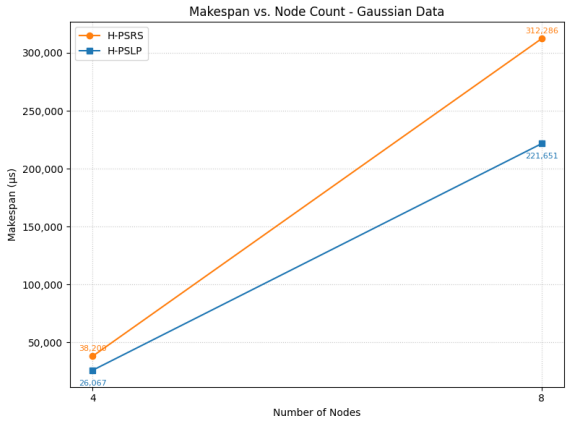


(a) Makespan vs. Node Count

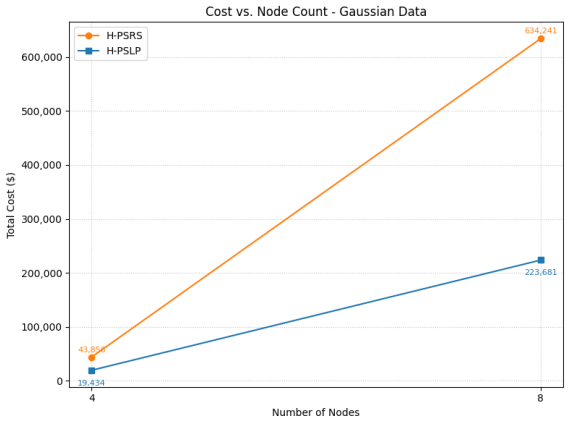


(b) Cost vs. Node Count

Figure 12.: Scalability Analysis: Uniform Data

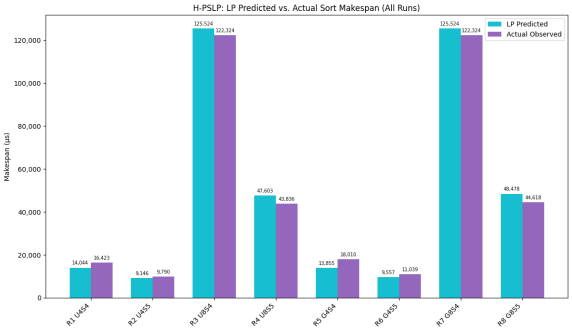


(a) Makespan vs. Node Count

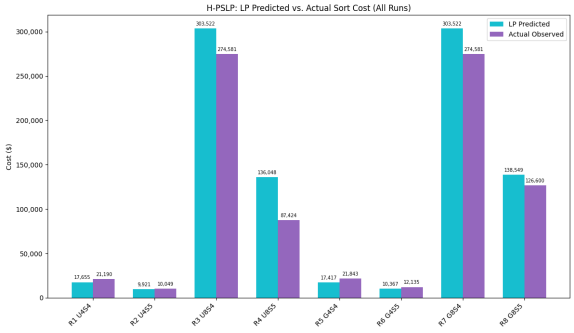


(b) Cost vs. Node Count

Figure 13.: Scalability Analysis: Gaussian Data



(a) Predicted vs. Actual Sort Makespan



(b) Predicted vs. Actual Sort Cost

Figure 14.: H-PSLP LP Model Prediction Accuracy for Sort Phase (All Runs)

6. Discussion

The comprehensive simulation results across various configurations provide significant insights into the comparative performance of the sequential baseline, the Regular Sampling approach (H-PSRS), and the Static Optimized approach (H-PSLP).

6.1. Key Findings Summary

(1) **Superior Efficiency of H-PSLP:** Across the eight experimental runs, H-PSLP (Min-Heap Merge) demonstrated a significant performance advantage, reducing **makespan by an average of 33.3%** and **total cost by an average of 60.7%** compared to H-PSRS.
Sample Speedup (Run 1: Uniform, 4 Nodes, Seed 4, vs. Sequential Baseline):
- H-PSRS: 4.67×
- H-PSLP: 6.06×

(2) **Elimination of Communication Bottlenecks:** H-PSLP's static partitioning strategy entirely removes the costly all-to-all data redistribution phase inherent in H-PSRS. This directly translated to makespan savings. For instance, in Run 3 (8 nodes, uniform), the H-PSRS data partition and transfer phase contributed 158,639.4 µs (Max Node Time) to its makespan; this component is absent in H-PSLP.

(3) **Guaranteed Memory Constraint Adherence:** A critical differentiator, H-PSLP consistently operated within node memory limits in all runs due to its LP formulation. In

contrast, H-PSRS exhibited severe memory overflows in every run (e.g., Node 3 in Run 1 at 425.4% overload; Node 0 in Run 2 at 1599.0% overload), rendering it unreliable in memory-constrained scenarios.

(4) **Efficient Scalability with Low LP Overhead:** H-PSLP demonstrated good scalability. The LP solver overhead (a form of preprocessing) remained minimal, constituting a small fraction of the total makespan even with 8 nodes (e.g., 2.86% of H-PSLP makespan in Run 3, 0.21% in Run 8). Similarly, H-PSRS's initial partitioning overhead was also typically small relative to its overall makespan.

(5) **Robustness to Data Distribution:** H-PSLP performed consistently across uniform and Gaussian datasets. Average speedups (vs. sequential) for H-PSLP were 7.29× for uniform data (4-node) and 9.85× (8-node), and for Gaussian data were 7.16× (4-node) and 9.51× (8-node). H-PSRS achieved 5.05× (uniform 4-node), 5.43× (uniform 8-node), 4.89× (Gaussian 4-node), and 6.91× (Gaussian 8-node).

6.2. Overall Performance (Makespan and Cost)

As highlighted in the Key Findings, the Static Optimized approach (H-PSLP) with Min-Heap Merge consistently outperformed H-PSRS.

Makespan: The average makespan reduction of 33.3% by H-PSLP over H-PSRS indicates a substantial improvement in processing speed. For 4-node configurations, H-PSLP was approximately 30.1

Cost: The average total cost reduction of 60.7% by H-PSLP compared to H-PSRS is a compelling demonstration of its cost-aware optimization capabilities.

Solver and Initial Partitioning Overhead

6.3. Impact of Data Distribution (Uniform vs. Gaussian)

Comparing runs with uniform distribution against Gaussian distribution (mean 252k, std 100k) for similar hardware configurations:

H-PSRS Performance: While makespan and cost for H-PSRS did not show dramatic average differences between the two distributions in 4-node setups, its reliance on pivot selection makes it inherently more sensitive to data skew. The critical issue of memory overflow remained regardless of the distribution.

H-PSLP Performance: H-PSLP's performance was consistently strong across both distributions. This robustness stems from its partitioning strategy being based on node capabilities and LP optimization, rather than data sampling.

6.4. Impact of Node Count (4 Nodes vs. 8 Nodes)

Increasing the number of nodes from 4 to 8:

Scalability of Makespan: While makespans increased for both methods with larger datasets on 8 nodes, H-PSLP's relative advantage over H-PSRS generally widened, suggesting better scalability.

Cost Implications: Total costs naturally increased with more nodes and larger datasets. However, H-PSLP's ability to optimize allocation continued to yield significant cost savings compared to H-PSRS, even with more nodes.

Memory Overflows in H-PSRS: The memory overflow problem in H-PSRS persisted and, in some 8-node cases (e.g., Run 3, Run 4, Run 8), affected multiple nodes, including high-throughput ones, due to the initial proportional allocation. This highlights a critical scalability flaw for H-PSRS in memory-constrained heterogeneous environments.

6.5. Memory Management – A Decisive Factor

This remains the most striking difference:

H-PSRS: Suffered severe memory overflows in every run, making it impractical for real-world systems where memory is a hard constraint.

H-PSLP: Successfully operated within all memory limits in every run, ensuring stability and reliability.

6.6. Effectiveness of the LP Solver in H-PSLP

The LP solver in H-PSLP consistently found efficient data distributions.

Makespan & Cost Prediction vs. Actual (for sort phase): The LP model provides estimates for the makespan and cost of the parallel sorting phase. Visualizations (Figures 14a and 14b) illustrate the comparison between these predictions and the actual observed metrics (Max Local Sort Time for makespan, Sum of Local Sort Costs for cost) across all runs.

- For makespan, the predicted values were often close to the actuals. For example, in Run 3 (8 nodes, uniform), the predicted makespan for the sort phase was 125,524.1 µs, and the actual maximum local sort time was 122,323.7 µs (a difference of -2.6%). In Run 8 (8 nodes, Gaussian), predicted was 48,478.5 µs vs. actual 44,617.9 µs (a difference of -8.6%).
- For cost, differences were more varied. In Run 1 (4 nodes, uniform), the predicted sort phase cost was \$17,655.42, while the actual sum of local sort costs was \$21,189.53 (actual was 19.9% higher). In Run 8, predicted sort cost was \$138,548.89, while actual was \$126,599.78 (actual was 8.6% lower). These variations highlight that while the LP model effectively guides partitioning for good performance, its cost predictions for the sort phase can deviate due to simplifications (e.g., the `base_throughput` factor) and uncaptured non-linear system behaviors. However, the overall H-PSLP approach (including solver and merge costs) still resulted in substantially lower total costs than H-PSRS.

Solver and Initial Partitioning Overhead: The preprocessing overhead, which includes the LP solver time for H-PSLP and the initial coordinator partitioning time for H-PSRS, was generally small compared to the overall makespan. For H-PSLP, the solver time typically constituted less than 1% of its makespan for 4-node runs and under 3% for 8-node runs (e.g., 2.86% in Run 3, 0.21% in Run 8). For H-PSRS, the initial partitioning time was also a minor component, for instance, 1.0% of its makespan in Run 3 and 1.4% in Run 8. This indicates that neither approach's primary preprocessing step introduces a significant bottleneck relative to the subsequent sorting and merging phases.

6.7. Consistency with Theoretical Advantages

The experimental results strongly support H-PSLP's theoretical benefits: global optimality (within model limits), cost awareness, memory constraint handling, reduced communication overhead, and robustness to data distribution.

6.8. Industry Implications

The advantages of H-PSLP have significant implications for practical deployments, especially in cloud computing:

- **Heterogeneous Environments:** Cloud environments often feature a mix of instance types (e.g., spot, on-demand, varying generations) with different performance and cost profiles. H-PSLP is well-suited to optimize resource usage in such settings.
- **Cost Optimization:** For budget-conscious cloud users, H-PSLP's ability to significantly reduce monetary cost while delivering competitive or superior performance is highly valuable.

- **Reliability and Stability:** By preventing memory overflows (common causes of Spark OOM exceptions or task failures), H-PSLP increases job completion rates and system stability, reducing operational overhead and re-computation costs.
- **Predictable Performance:** While subject to model accuracy, H-PSLP offers more predictable performance characteristics compared to heuristic methods that can be sensitive to data skew or sampling randomness.

6.9. Limitations and Future Work

- **Number of Repetitions:** While these results are based on multiple runs (typically two seeds per configuration), a higher number of repetitions would be necessary for more robust statistical analysis (e.g., t-tests, confidence intervals) to firmly establish the significance of observed differences. The current standard deviations are indicative but based on limited samples.
- **Lambda (λ) Parameter:** These experiments likely used a default or fixed λ (cost-performance trade-off parameter) in the H-PSLP model. Exploring a range of λ values to map out the Pareto front for time-cost trade-offs is key future work.
- **Merge Bottleneck:** The centralized $O(N \log P)$ min-heap merge in H-PSLP can become a bottleneck at very large scales. For instance, in Run 4 (8 nodes, $\sim$4.75M elements), the min-heap merge accounted for approximately 63.2% of the H-PSLP makespan. Future work should explore distributed merge strategies.
- **LP Approximation Gap:** The LP model simplifies real-world complexities. The observed differences between predicted sort phase costs/makespans and actuals (e.g., 19.9% cost difference in Run 1, 8.6% cost difference in Run 8) suggest that refining the model to better capture non-linear network effects or processing characteristics could improve accuracy.
- **Dataset Characteristics:** Testing with a wider array of skewed datasets is needed.

7. Conclusion

The Static Optimized approach (H-PSLP), utilizing a Min-Heap Merge, consistently and significantly outperforms the Regular Sampling approach (H-PSRS) and the sequential baseline across various data distributions and node configurations. On average, across the conducted experiments, H-PSLP **reduced execution makespan by 33.3% and monetary cost by 60.7%** compared to H-PSRS. Its paramount advantage is the **guaranteed adherence to memory constraints**, preventing system failures common with H-PSRS in heterogeneous environments. H-PSLP's LP-driven, cost-aware workload distribution proves robust across different data distributions and scales efficiently with low optimization overhead. These findings strongly advocate for H-PSLP as a superior, more reliable, and economically viable strategy for static data partitioning in parallel sorting within heterogeneous distributed systems.

Building on these promising results, future work will focus on several key areas to further enhance the H-PSLP approach and broaden its applicability:

- **Distributed Merge Strategies:** The current centralized heap merge, while efficient, can become a bottleneck for extremely large datasets or a high number of nodes, as observed in Run 4 where it constituted a significant portion of the makespan. Research into and implementation of distributed multi-way merge algorithms (e.g., using a tree-based reduction or a fully distributed k-way merge) could alleviate this, further improving scalability.
- **Refinement of LP Model and Performance Prediction:** The LP model currently uses a simplified `base_throughput` for performance estimation. Future iterations could incorporate more sophisticated performance models that account for non-linearities in network communication costs, memory access patterns, or even the impact of data skew on local sort times. This could involve machine learning techniques to predict task durations more accurately or incorporating dynamic feedback loops if the partitioning were to become semi-

dynamic. Addressing the observed approximation gap (e.g., cost differences of up to 35.7% between predicted and actual sort phase costs) is crucial for tighter optimization.

- **Exploration of Cost-Performance Trade-offs (λ Parameter):** A systematic exploration of the λ parameter in the H-PSLP objective function is warranted. This would allow for the construction of a Pareto front, offering users a clear spectrum of choices between minimizing makespan and minimizing monetary cost, which is highly relevant for different cloud computing budget scenarios.
- **Dynamic and Adaptive Partitioning:** While this work focuses on static partitioning, extending H-PSLP to support dynamic or adaptive re-partitioning during runtime could offer benefits in scenarios with fluctuating node availability (e.g., preemptible cloud instances) or if initial data distribution estimates are found to be significantly off. This would, however, introduce added complexity and communication overhead.
- **Extended Evaluation with Diverse Workloads and Environments:**
  - *More Repetitions and Statistical Rigor:* Conducting a larger number of repetitions for each experimental configuration (e.g., the planned 10) is essential for robust statistical analysis, including t-tests and confidence intervals, to definitively establish the significance of observed performance differences.
  - *Broader Range of Data Skew:* Testing H-PSLP against a wider variety of highly skewed datasets (e.g., Zipfian distributions with different parameters) will further validate its robustness compared to sampling-based methods like H-PSRS.
  - *Real-World Cluster Deployment:* Validating the simulation results on a physical heterogeneous cluster or a realistic cloud environment would provide invaluable insights into real-world performance and overheads not captured by simulation.
  - *Different Application Contexts:* Exploring the applicability of the H-PSLP partitioning strategy to other parallel data processing tasks beyond sorting, such as distributed joins or aggregations, could be a fruitful avenue.
- **Integration with Workflow Management Systems:** Investigating how H-PSLP could be integrated into larger data processing workflow management systems (e.g., Apache Airflow, Spark) to optimize resource allocation for sorting stages within more complex data pipelines.

Addressing these areas will further solidify H-PSLP's position as a leading solution for cost-effective and efficient parallel data processing in today's complex and heterogeneous computing landscapes.

References

[1] Lu, S., Zhao, M., Li, C., Du, Q., and Luo, Y. (2023). Time-aware data partition optimization and heterogeneous task scheduling strategies in spark clusters. *The Computer Journal*, 66(2):762–776.

[2] Monga, P. and Lodhi, M. (n.d.). Parallel sorting on heterogeneous clusters. Project Report, Dalhousie University, Halifax, Canada.

[3] Terlaky, T. (2011). Introduction to polynomial time algorithms for lp. In Smith, J. and Johnson, A., editors, *Wiley Encyclopedia of Operations Research and Management Science*, pages 123–130. Wiley, Hoboken, NJ.

[4] Tokuue, T. and Ishiyama, T. (2023). Performance evaluation of parallel sortings on the supercomputer fugaku. *Journal of Information Processing*, 31:452–458.

[5] Yoon, M. and Kamal, A. (2014). Optimal dataset allocation in distributed heterogeneous clouds. In *2014 IEEE Globecom Workshops (GC Wkshps)*, pages 225–230, Austin, TX. IEEE.

Appendix A. Uniform, First Run, Four Nodes, Seed Four

A.1. Cluster Configuration

Total memory of cluster = 1134000 (should be >= 756000)

- `{'throughput': 4, 'memory': 477112, 'billing': 0.3604}`
  - 63.11% out of 150.00% of overprovisioned memory
  - 42.07% out of 100.00% of total cluster memory
- `{'throughput': 5, 'memory': 89767, 'billing': 0.991}`
  - 11.87% out of 150.00% of overprovisioned memory
  - 7.92% out of 100.00% of total cluster memory
- `{'throughput': 2, 'memory': 19744, 'billing': 0.153}`
  - 2.61% out of 150.00% of overprovisioned memory
  - 1.74% out of 100.00% of total cluster memory
- `{'throughput': 7, 'memory': 547377, 'billing': 0.5683}`
  - 72.40% out of 150.00% of overprovisioned memory
  - 48.27% out of 100.00% of total cluster memory

Saved plot with bell curve as 'out/1run_4node_4seed_uniform.png'.
Nodes: 4 | Data: 756000 elements | Sample: [377744, 290122, 181051, 111341, 274432...177889, 702425, 577961, 166546, 442926]

A.2. Sequential Baseline

Sequential: time=180655.898µs throughput=0.238963µs/element

A.3. Regular Sampling Approach

N0 (7x) Partition: time=392.990µs time=223.336µs cost=$223.336
N3 (2x) Sort: items=84000/756000 (11.1%) mem=84000/19744 (425.4%) samples=6 time=8511.044µs cost=$1302.190
N2 (4x) Sort: items=168000/756000 (22.2%) mem=168000/477112 (35.2%) samples=12 time=9435.443µs cost=$3400.534
N1 (5x) Sort: items=210000/756000 (27.8%) mem=210000/89767 (233.9%) samples=15 time=9907.066µs cost=$9817.902
N0 (7x) Sort: items=294000/756000 (38.9%) mem=294000/547377 (53.7%) samples=21 time=11145.255µs cost=$6333.849
N0 (7x) Sampling: samples=54/54 pivots=3 time=142.523µs cost=$80.996 pivots=[150989, 352813, 505023]
N3 (2x) Partition: transfer=27999/84000 time=16737.502µs cost=$2560.838
N2 (4x) Partition: transfer=33851/168000 time=12761.168µs cost=$4599.125
N1 (5x) Partition: transfer=56000/210000 time=14000.478µs cost=$13874.474
N0 (7x) Partition: transfer=58795/294000 time=14303.185µs cost=$8128.500
N2 (4x) Final: items=152405/756000 (22.2%) mem=152405/477112 (35.2%) time=1936.900µs cost=$698.059
N1 (5x) Final: items=201638/756000 (27.8%) mem=201638/89767 (233.9%) time=1898.871µs cost=$1881.781
N0 (7x) Final: items=151157/756000 (38.9%) mem=151157/547377 (53.7%) time=2200.942µs cost=$1250.795
N3 (2x) Final: items=250800/756000 (11.1%) mem=250800/19744 (425.4%) time=9104.727µs cost=$1393.023
N0 Merge: total=756000 time=1162.995µs cost=$660.930

A.4. Static Optimized Approach

N0 (7x) Solver: time=375.450µs cost=$213.368 ˆmakespan=14044.258µs ˆcost=$17655.419
Remaining items to allocate: 1
Partition sizes: [411415, 89737, 235116, 19732]
Ideal sizes: [411415.2, 89737.2, 235116.0, 19731.6]
Fractional parts: [0.20000000001164153, 0.19999999999708962, 0.0, 0.5999999999985448]
N3 (2x) Sort: items=19732/756000 (2.6%) mem=19732/19744 (99.9%) time=3100.102µs cost=$474.316
N1 (5x) Sort: items=89737/756000 (11.9%) mem=89737/89767 (100.0%) time=6218.830µs cost=$6162.860
N2 (4x) Sort: items=235116/756000 (31.1%) mem=235116/477112 (49.3%) time=16422.666µs cost=$5918.729
N0 (7x) Sort: items=411415/756000 (54.4%) mem=411415/547377 (75.2%) time=15192.011µs cost=$8633.620
N0 (7x) Heap Merge: total=756000 time=12995.490µs cost=$7385.337
N0 (7x) Linear Merge: total=756000 time=37985.638µs cost=$21587.238

Appendix B. Uniform, Second Run, Four Nodes, Seed Five

B.1. Cluster Configuration

Total memory of cluster = 1215000 (should be >= 810000)

- `{'throughput': 10, 'memory': 16886, 'billing': 0.9108}`
  - 2.08% out of 150.00% of overprovisioned memory
  - 1.39% out of 100.00% of total cluster memory
- `{'throughput': 5, 'memory': 271079, 'billing': 0.2019}`
  - 33.47% out of 150.00% of overprovisioned memory
  - 22.31% out of 100.00% of total cluster memory
- `{'throughput': 6, 'memory': 549209, 'billing': 0.5222}`
  - 67.80% out of 150.00% of overprovisioned memory
  - 45.20% out of 100.00% of total cluster memory
- `{'throughput': 9, 'memory': 377826, 'billing': 0.3219}`
  - 46.65% out of 150.00% of overprovisioned memory
  - 31.10% out of 100.00% of total cluster memory

Saved plot with bell curve as 'out/2run_4node_5seed_uniform.png'.
Nodes: 4 | Data: 810000 elements | Sample: [570174, 106927, 601820, 261442, 13751...266532, 227148, 657143, 624846, 606793]

B.2. Sequential Baseline

Sequential: time=186810.633µs throughput=0.230630µs/element

B.3. Regular Sampling Approach

N0 (10x) Partition: time=258.199µs time=235.167µs cost=$235.167
N2 (6x) Sort: items=162000/810000 (20.0%) mem=162000/549209 (29.5%) samples=18 time=5329.721µs cost=$2783.180
N3 (5x) Sort: items=135000/810000 (16.7%) mem=135000/271079 (49.8%) samples=15 time=6198.675µs cost=$1251.512
N1 (9x) Sort: items=243000/810000 (30.0%) mem=243000/377826 (64.3%) samples=27 time=6238.055µs cost=$2008.030
N0 (10x) Sort: items=270000/810000 (33.3%) mem=270000/16886 (1599.0%) samples=30 time=5947.643µs cost=$5417.114
N0 (10x) Sampling: samples=90/90 pivots=3 time=178.403µs cost=$162.490 pivots=[180853, 379196, 585202]
N2 (6x) Partition: transfer=41102/162000 time=9347.091µs cost=$4881.051
N3 (5x) Partition: transfer=37829/135000 time=12016.880µs cost=$2426.208
N1 (9x) Partition: transfer=59302/243000 time=8850.174µs cost=$2848.871
N0 (10x) Partition: transfer=60192/270000 time=9336.395µs cost=$8503.589
N2 (6x) Final: items=206115/810000 (20.0%) mem=206115/549209 (29.5%) time=1334.929µs cost=$697.100
N1 (9x) Final: items=198370/810000 (30.0%) mem=198370/377826 (64.3%) time=1098.472µs cost=$353.598
N3 (5x) Final: items=224727/810000 (16.7%) mem=224727/271079 (49.8%) time=1918.090µs cost=$387.262
N0 (10x) Final: items=180788/810000 (33.3%) mem=180788/16886 (1599.0%) time=796.488µs cost=$725.441
N0 Merge: total=810000 time=628.494µs cost=$572.433

B.4. Static Optimized Approach

N0 (9x) Solver: time=288.044µs cost=$92.721 ˆmakespan=9145.810µs ˆcost=$9921.222
Remaining items to allocate: 3
Partition sizes: [356922, 16849, 237921, 198308]
Ideal sizes: [356921.69216921687, 16849.684968496847, 237920.7920792079, 198307.8307830783]
Fractional parts: [0.6921692168689333, 0.6849684968474321, 0.792079207894858, 0.8307830783014651]
N1 (10x) Sort: items=16849/810000 (2.1%) mem=16849/16886 (99.8%) time=244.122µs cost=$222.346
N3 (5x) Sort: items=198308/810000 (24.5%) mem=198308/271079 (73.2%) time=8553.959µs cost=$1727.044
N2 (6x) Sort: items=237921/810000 (29.4%) mem=237921/549209 (43.3%) time=9474.274µs cost=$4947.466
N0 (9x) Sort: items=356922/810000 (44.1%) mem=356922/377826 (94.5%) time=9790.408µs cost=$3151.532
N0 (9x) Heap Merge: total=810000 time=10550.804µs cost=$3396.304
N0 (9x) Linear Merge: total=810000 time=31558.657µs cost=$10158.732

Appendix C. Uniform, Third Run, Eight Nodes, Seed Four

C.1. Cluster Configuration

Total memory of cluster = 12474000 (should be >= 8316000)

- `{'throughput': 4, 'memory': 69286, 'billing': 0.8442}`
  - 0.83% out of 150.00% of overprovisioned memory
  - 0.56% out of 100.00% of total cluster memory
- `{'throughput': 5, 'memory': 1920860, 'billing': 0.3355}`
  - 23.10% out of 150.00% of overprovisioned memory
  - 15.40% out of 100.00% of total cluster memory
- `{'throughput': 2, 'memory': 1011737, 'billing': 0.9487}`
  - 12.17% out of 150.00% of overprovisioned memory
  - 8.11% out of 100.00% of total cluster memory
- `{'throughput': 7, 'memory': 3461399, 'billing': 0.1231}`
  - 41.62% out of 150.00% of overprovisioned memory
  - 27.75% out of 100.00% of total cluster memory
- `{'throughput': 8, 'memory': 205745, 'billing': 0.6766}`
  - 2.47% out of 150.00% of overprovisioned memory
  - 1.65% out of 100.00% of total cluster memory
- `{'throughput': 3, 'memory': 1819249, 'billing': 0.3342}`
  - 21.88% out of 150.00% of overprovisioned memory
  - 14.58% out of 100.00% of total cluster memory
- `{'throughput': 2, 'memory': 1259515, 'billing': 0.3446}`
  - 15.15% out of 150.00% of overprovisioned memory
  - 10.10% out of 100.00% of total cluster memory
- `{'throughput': 2, 'memory': 2726209, 'billing': 0.2483}`
  - 32.78% out of 150.00% of overprovisioned memory
  - 21.86% out of 100.00% of total cluster memory

Saved plot with bell curve as 'out/3run_8node_4seed_uniform.png'.
Nodes: 8 | Data: 8316000 elements | Sample: [2429682, 5259452, 7275621, 6139577, 8060806...8049921, 3113790, 4192437, 7407151, 682635]

C.2. Sequential Baseline

Sequential: time=2726167.854µs throughput=0.327822µs/element

C.3. Regular Sampling Approach

N0 (8x) Partition: time=4635.303µs time=3136.246µs cost=$3136.246
N5 (2x) Sort: items=504000/8316000 (6.1%) mem=504000/1011737 (49.8%) samples=14 time=60747.782µs cost=$57631.421
N6 (2x) Sort: items=504000/8316000 (6.1%) mem=504000/1259515 (40.0%) samples=14 time=72714.803µs cost=$25057.521
N7 (2x) Sort: items=504000/8316000 (6.1%) mem=504000/2726209 (18.5%) samples=14 time=69431.137µs cost=$17239.751
N4 (3x) Sort: items=756000/8316000 (9.1%) mem=756000/1819249 (41.6%) samples=21 time=80435.669µs cost=$26881.601
N3 (4x) Sort: items=1008000/8316000 (12.1%) mem=1008000/69286 (1454.8%) samples=28 time=78955.148µs cost=$66653.936
N2 (5x) Sort: items=1260000/8316000 (15.2%) mem=1260000/1920860 (65.6%) samples=35 time=78083.818µs cost=$26197.121
N1 (7x) Sort: items=1764000/8316000 (21.2%) mem=1764000/3461399 (51.0%) samples=49

time=79757.202µs cost=$9818.112
N0 (8x) Sort: items=2016000/8316000 (24.2%) mem=2016000/205745 (979.9%) samples=56 time=83716.862µs cost=$56642.829
N0 (8x) Sampling: samples=231/231 pivots=7 time=1074.630µs cost=$727.094 pivots=[887370, 1859702, 2960571, 3893839, 4900256, 5937494, 6956527]
N5 (2x) Partition: transfer=63171/504000 time=139341.271µs cost=$132193.064
N7 (2x) Partition: transfer=82773/504000 time=141206.659µs cost=$35061.613
N6 (2x) Partition: transfer=61911/504000 time=158639.421µs cost=$54667.144
N4 (3x) Partition: transfer=91618/756000 time=126925.994µs cost=$42418.667
N3 (4x) Partition: transfer=112933/1008000 time=128997.766µs cost=$108899.914
N2 (5x) Partition: transfer=166677/1260000 time=115366.498µs cost=$38705.460
N1 (7x) Partition: transfer=207013/1764000 time=97292.104µs cost=$11976.658
N0 (8x) Partition: transfer=215692/2016000 time=92132.092µs cost=$62336.573
N1 (7x) Final: items=975158/8316000 (21.2%) mem=975158/3461399 (51.0%) time=11426.167µs cost=$1406.561
N6 (2x) Final: items=1016967/8316000 (6.1%) mem=1016967/1259515 (40.0%) time=54789.802µs cost=$18880.566
N5 (2x) Final: items=1036446/8316000 (6.1%) mem=1036446/1011737 (49.8%) time=56305.882µs cost=$53417.390
N2 (5x) Final: items=1100440/8316000 (15.2%) mem=1100440/1920860 (65.6%) time=23307.354µs cost=$7819.617
N4 (3x) Final: items=1005651/8316000 (9.1%) mem=1005651/1819249 (41.6%) time=36637.275µs cost=$12244.177
N7 (2x) Final: items=1358523/8316000 (6.1%) mem=1358523/2726209 (18.5%) time=69241.607µs cost=$17192.691
N3 (4x) Final: items=933509/8316000 (12.1%) mem=933509/69286 (1454.8%) time=29564.977µs cost=$24958.754
N0 (8x) Final: items=889306/8316000 (24.2%) mem=889306/205745 (979.9%) time=8189.627µs cost=$5541.101
N0 Merge: total=8316000 time=18250.847µs cost=$12348.523

C.4. Static Optimized Approach

N0 (7x) Solver: time=8935.205µs cost=$1099.924 ˆmakespan=125524.126µs ˆcost=$303521.591
Remaining items to allocate: 4
Partition sizes: [2680515, 205426, 1914535, 69030, 1148554, 765980, 765980, 765980]
Ideal sizes: [2680514.8514851485, 205425.74257425743, 1914534.6534653467, 69029.70297029703, 1148554.4554455446, 765980.198019802, 765980.198019802, 765980.198019802]
Fractional parts: [0.851485148537904, 0.7425742574268952, 0.6534653466660529, 0.7029702970321523, 0.4554455445613712, 0.1980198019882664, 0.1980198019882664, 0.1980198019882664]
N3 (4x) Sort: items=69030/8316000 (0.8%) mem=69030/69286 (99.6%) time=3758.101µs cost=$3172.589
N1 (8x) Sort: items=205426/8316000 (2.5%) mem=205426/205745 (99.8%) time=6284.544µs cost=$4252.123
N5 (2x) Sort: items=765980/8316000 (9.2%) mem=765980/1011737 (75.7%) time=101961.624µs cost=$96730.993
N7 (2x) Sort: items=765980/8316000 (9.2%) mem=765980/2726209 (28.1%) time=100991.493µs cost=$25076.188
N6 (2x) Sort: items=765980/8316000 (9.2%) mem=765980/1259515 (60.8%) time=114248.080µs cost=$39369.888
N4 (3x) Sort: items=1148554/8316000 (13.8%) mem=1148554/1819249 (63.1%) time=122323.685µs cost=$40880.575

N2 (5x) Sort: items=1914535/8316000 (23.0%) mem=1914535/1920860 (99.7%) time=112559.243µs cost=$37763.626
N0 (7x) Sort: items=2680515/8316000 (32.2%) mem=2680515/3461399 (77.4%) time=117955.408µs cost=$14520.311
N0 (7x) Heap Merge: total=8316000 time=184917.383µs cost=$22763.330
N0 (7x) Linear Merge: total=8316000 time=751949.992µs cost=$92565.044

Appendix D. Uniform, Fourth Run, Eight Nodes, Seed Five

D.1. Cluster Configuration

Total memory of cluster = 7128000 (should be >= 4752000)

- `{'throughput': 10, 'memory': 2086774, 'billing': 0.3515}`
  - 43.91% out of 150.00% of overprovisioned memory
  - 29.28% out of 100.00% of total cluster memory
- `{'throughput': 5, 'memory': 262221, 'billing': 0.9247}`
  - 5.52% out of 150.00% of overprovisioned memory
  - 3.68% out of 100.00% of total cluster memory
- `{'throughput': 6, 'memory': 1086514, 'billing': 0.7892}`
  - 22.86% out of 150.00% of overprovisioned memory
  - 15.24% out of 100.00% of total cluster memory
- `{'throughput': 9, 'memory': 571141, 'billing': 0.2436}`
  - 12.02% out of 150.00% of overprovisioned memory
  - 8.01% out of 100.00% of total cluster memory
- `{'throughput': 1, 'memory': 1259524, 'billing': 0.8174}`
  - 26.51% out of 150.00% of overprovisioned memory
  - 17.67% out of 100.00% of total cluster memory
- `{'throughput': 8, 'memory': 1329431, 'billing': 0.2249}`
  - 27.98% out of 150.00% of overprovisioned memory
  - 18.65% out of 100.00% of total cluster memory
- `{'throughput': 4, 'memory': 30376, 'billing': 0.6557}`
  - 0.64% out of 150.00% of overprovisioned memory
  - 0.43% out of 100.00% of total cluster memory
- `{'throughput': 1, 'memory': 502019, 'billing': 0.214}`
  - 10.56% out of 150.00% of overprovisioned memory
  - 7.04% out of 100.00% of total cluster memory

Saved plot with bell curve as 'out/3run_8node_5seed_uniform.png'.
Nodes: 8 | Data: 4752000 elements | Sample: [14888, 44569, 1757047, 1807587, 1391304...912762, 1341688, 2877662, 4554858, 3104306]

D.2. Sequential Baseline

Sequential: time=1512629.152µs throughput=0.318314µs/element

D.3. Regular Sampling Approach

N0 (10x) Partition: time=1610.111µs time=565.954µs cost=$565.954
N6 (1x) Sort: items=108000/4752000 (2.3%) mem=108000/1259524 (8.6%) samples=7 time=28663.931µs cost=$23429.897
N7 (1x) Sort: items=108000/4752000 (2.3%) mem=108000/502019 (21.5%) samples=7 time=24467.699µs cost=$5236.088
N4 (5x) Sort: items=540000/4752000 (11.4%) mem=540000/262221 (205.9%) samples=35 time=25597.224µs cost=$23669.753
N5 (4x) Sort: items=432000/4752000 (9.1%) mem=432000/30376 (1422.2%) samples=28 time=32848.315µs cost=$21538.640
N3 (6x) Sort: items=648000/4752000 (13.6%) mem=648000/1086514 (59.6%) samples=42 time=27801.052µs cost=$21940.590
N2 (8x) Sort: items=864000/4752000 (18.2%) mem=864000/1329431 (65.0%) samples=56 time=29119.637µs cost=$6549.006
N1 (9x) Sort: items=972000/4752000 (20.5%) mem=972000/571141 (170.2%) samples=63

time=30973.926µs cost=$7545.248
N0 (10x) Sort: items=1080000/4752000 (22.7%) mem=1080000/2086774 (51.8%) samples=70 time=29322.010µs cost=$10306.687
N0 (10x) Sampling: samples=308/308 pivots=7 time=328.567µs cost=$115.491 pivots=[541791, 1131595, 1699966, 2290882, 2866676, 3460319, 4071260]
N6 (1x) Partition: transfer=13938/108000 time=75649.742µs cost=$61836.099
N7 (1x) Partition: transfer=15421/108000 time=84345.098µs cost=$18049.851
N5 (4x) Partition: transfer=53765/432000 time=52162.792µs cost=$34203.143
N4 (5x) Partition: transfer=65683/540000 time=52384.474µs cost=$48439.923
N3 (6x) Partition: transfer=80880/648000 time=49232.858µs cost=$38854.571
N2 (8x) Partition: transfer=103440/864000 time=51054.383µs cost=$11482.131
N1 (9x) Partition: transfer=120278/972000 time=46050.360µs cost=$11217.868
N0 (10x) Partition: transfer=123425/1080000 time=44776.563µs cost=$15738.962
N1 (9x) Final: items=589357/4752000 (20.5%) mem=589357/571141 (170.2%) time=5828.851µs cost=$1419.908
N6 (1x) Final: items=612317/4752000 (2.3%) mem=612317/1259524 (8.6%) time=58704.060µs cost=$47984.699
N3 (6x) Final: items=591398/4752000 (13.6%) mem=591398/1086514 (59.6%) time=11461.140µs cost=$9045.131
N5 (4x) Final: items=592680/4752000 (9.1%) mem=592680/30376 (1422.2%) time=14136.691µs cost=$9269.428
N2 (8x) Final: items=567994/4752000 (18.2%) mem=567994/1329431 (65.0%) time=9499.528µs cost=$2136.444
N7 (1x) Final: items=680358/4752000 (2.3%) mem=680358/502019 (21.5%) time=45467.165µs cost=$9729.973
N4 (5x) Final: items=576012/4752000 (11.4%) mem=576012/262221 (205.9%) time=14686.790µs cost=$13580.875
N0 (10x) Final: items=541884/4752000 (22.7%) mem=541884/2086774 (51.8%) time=3544.674µs cost=$1245.953
N0 Merge: total=4752000 time=7280.224µs cost=$2558.999

D.4. Static Optimized Approach

N0 (10x) Solver: time=361.807µs cost=$127.175 ^makespan=47603.426µs ^cost=$136047.899
Remaining items to allocate: 4
Partition sizes: [1495305, 571133, 1196434, 897088, 262284, 30410, 149673, 149673]
Ideal sizes: [1495304.8695130488, 571133.2866713329, 1196433.9566043396, 897087.891210879, 262284.1715828417, 30409.75902409759, 149673.03269673034, 149673.03269673034]
Fractional parts: [0.8695130487903953, 0.28667133289854974, 0.9566043396480381, 0.8912108789663762, 0.17158284172182903, 0.7590240975914639, 0.03269673034083098, 0.03269673034083098]
N5 (4x) Sort: items=30410/4752000 (0.6%) mem=30410/30376 (100.1%) time=1157.984µs cost=$759.290
N6 (1x) Sort: items=149673/4752000 (3.1%) mem=149673/1259524 (11.9%) time=35805.200µs cost=$29267.170
N7 (1x) Sort: items=149673/4752000 (3.1%) mem=149673/502019 (29.8%) time=34982.502µs cost=$7486.255
N4 (5x) Sort: items=262284/4752000 (5.5%) mem=262284/262221 (100.0%) time=13926.306µs cost=$12877.655
N1 (9x) Sort: items=571133/4752000 (12.0%) mem=571133/571141 (100.0%) time=19162.129µs cost=$4667.895
N3 (6x) Sort: items=897088/4752000 (18.9%) mem=897088/1086514 (82.6%) time=41364.848µs cost=$32645.138

N2 (8x) Sort: items=1196434/4752000 (25.2%) mem=1196434/1329431 (90.0%) time=43836.433µs cost=$9858.814
N0 (10x) Sort: items=1495305/4752000 (31.5%) mem=1495305/2086774 (71.7%) time=41599.165µs cost=$14622.107
N0 (10x) Heap Merge: total=4752000 time=71986.408µs cost=$25303.222
N0 (10x) Linear Merge: total=4752000 time=301983.626µs cost=$106147.245

Appendix E. Gaussian, Fifth Run, Four Nodes, Seed Four, Mean 252000, Std 100000

E.1. Cluster Configuration

Total memory of cluster = 1134000 (should be >= 756000)

- `{'throughput': 4, 'memory': 477112, 'billing': 0.3604}`
  - 63.11% out of 150.00% of overprovisioned memory
  - 42.07% out of 100.00% of total cluster memory
- `{'throughput': 5, 'memory': 89767, 'billing': 0.991}`
  - 11.87% out of 150.00% of overprovisioned memory
  - 7.92% out of 100.00% of total cluster memory
- `{'throughput': 2, 'memory': 19744, 'billing': 0.153}`
  - 2.61% out of 150.00% of overprovisioned memory
  - 1.74% out of 100.00% of total cluster memory
- `{'throughput': 7, 'memory': 547377, 'billing': 0.5683}`
  - 72.40% out of 150.00% of overprovisioned memory
  - 48.27% out of 100.00% of total cluster memory

Saved plot with bell curve as 'out/5run_4node_4seed_gaussian.png'.
Nodes: 4 | Data: 756000 elements | Sample: [326598, 89114, 299743, 159835, 95637...278501, 343868, 357211, 201641, 158375]

E.2. Sequential Baseline

Sequential: time=178217.956µs throughput=0.235738µs/element

E.3. Regular Sampling Approach

N0 (7x) Partition: time=334.683µs time=190.200µs cost=$190.200
N3 (2x) Sort: items=84000/756000 (11.1%) mem=84000/19744 (425.4%) samples=6 time=10637.621µs cost=$1627.556
N2 (4x) Sort: items=168000/756000 (22.2%) mem=168000/477112 (35.2%) samples=12 time=10781.896µs cost=$3885.795
N1 (5x) Sort: items=210000/756000 (27.8%) mem=210000/89767 (233.9%) samples=15 time=11242.614µs cost=$11141.430
N0 (7x) Sort: items=294000/756000 (38.9%) mem=294000/547377 (53.7%) samples=21 time=11467.854µs cost=$6517.181
N0 (7x) Sampling: samples=54/54 pivots=3 time=225.688µs cost=$128.258 pivots=[168315, 243932, 294903]
N3 (2x) Partition: transfer=27961/84000 time=18984.175µs cost=$2904.579
N2 (4x) Partition: transfer=33032/168000 time=13704.400µs cost=$4939.066
N1 (5x) Partition: transfer=56000/210000 time=12945.768µs cost=$12829.256
N0 (7x) Partition: transfer=59198/294000 time=11702.514µs cost=$6650.539
N2 (4x) Final: items=150238/756000 (22.2%) mem=150238/477112 (35.2%) time=1788.092µs cost=$644.428
N1 (5x) Final: items=201669/756000 (27.8%) mem=201669/89767 (233.9%) time=1994.506µs cost=$1976.555
N3 (2x) Final: items=251867/756000 (11.1%) mem=251867/19744 (425.4%) time=6173.400µs cost=$944.530
N0 (7x) Final: items=152226/756000 (38.9%) mem=152226/547377 (53.7%) time=1312.771µs cost=$746.048
N0 Merge: total=756000 time=1233.284µs cost=$700.875

E.4. Static Optimized Approach

N0 (7x) Solver: time=477.432µs cost=$271.325 ˆmakespan=13854.732µs ˆcost=$17417.160
Remaining items to allocate: 1
Partition sizes: [411415, 89737, 235116, 19732]
Ideal sizes: [411415.2, 89737.2, 235116.0, 19731.6]
Fractional parts: [0.20000000001164153, 0.19999999999708962, 0.0, 0.5999999999985448]
N3 (2x) Sort: items=19732/756000 (2.6%) mem=19732/19744 (99.9%) time=5046.289µs cost=$772.082
N1 (5x) Sort: items=89737/756000 (11.9%) mem=89737/89767 (100.0%) time=9092.578µs cost=$9010.744
N2 (4x) Sort: items=235116/756000 (31.1%) mem=235116/477112 (49.3%) time=18009.471µs cost=$6490.614
N0 (7x) Sort: items=411415/756000 (54.4%) mem=411415/547377 (75.2%) time=16838.071µs cost=$9569.076
N0 (7x) Heap Merge: total=756000 time=12805.172µs cost=$7277.179
N0 (7x) Linear Merge: total=756000 time=38726.233µs cost=$22008.118

Appendix F. Gaussian, Sixth Run, Four Nodes, Seed Five, Mean 252000, Std 100000

F.1. Cluster Configuration

Total memory of cluster = 1215000 (should be >= 810000)

- `{'throughput': 10, 'memory': 16886, 'billing': 0.9108}`
  - 2.08% out of 150.00% of overprovisioned memory
  - 1.39% out of 100.00% of total cluster memory
- `{'throughput': 5, 'memory': 271079, 'billing': 0.2019}`
  - 33.47% out of 150.00% of overprovisioned memory
  - 22.31% out of 100.00% of total cluster memory
- `{'throughput': 6, 'memory': 549209, 'billing': 0.5222}`
  - 67.80% out of 150.00% of overprovisioned memory
  - 45.20% out of 100.00% of total cluster memory
- `{'throughput': 9, 'memory': 377826, 'billing': 0.3219}`
  - 46.65% out of 150.00% of overprovisioned memory
  - 31.10% out of 100.00% of total cluster memory

Saved plot with bell curve as 'out/6run_4node_5seed_gaussian.png'.
Nodes: 4 | Data: 810000 elements | Sample: [462738, 224068, 148611, 374415, 171848...230198, 339643, 115255, 389337, 309838]

F.2. Sequential Baseline

Sequential: time=195203.354µs throughput=0.240992µs/element

F.3. Regular Sampling Approach

N0 (10x) Partition: time=323.418µs time=294.569µs cost=$294.569
N3 (5x) Sort: items=135000/810000 (16.7%) mem=135000/271079 (49.8%) samples=15 time=6190.619µs cost=$1249.886
N2 (6x) Sort: items=162000/810000 (20.0%) mem=162000/549209 (29.5%) samples=18 time=7185.403µs cost=$3752.217
N1 (9x) Sort: items=243000/810000 (30.0%) mem=243000/377826 (64.3%) samples=27 time=6812.782µs cost=$2193.034
N0 (10x) Sort: items=270000/810000 (33.3%) mem=270000/16886 (1599.0%) samples=30 time=6972.725µs cost=$6350.758
N0 (10x) Sampling: samples=90/90 pivots=3 time=182.939µs cost=$166.621 pivots=[175363, 243885, 311013]
N3 (5x) Partition: transfer=37644/135000 time=16277.197µs cost=$3286.366
N2 (6x) Partition: transfer=41127/162000 time=13694.641µs cost=$7151.341
N1 (9x) Partition: transfer=59924/243000 time=11792.944µs cost=$3796.149
N0 (10x) Partition: transfer=59858/270000 time=10776.339µs cost=$9815.089
N2 (6x) Final: items=206361/810000 (20.0%) mem=206361/549209 (29.5%) time=1259.891µs cost=$657.915
N3 (5x) Final: items=224737/810000 (16.7%) mem=224737/271079 (49.8%) time=1615.086µs cost=$326.086
N1 (9x) Final: items=198978/810000 (30.0%) mem=198978/377826 (64.3%) time=1165.598µs cost=$375.206
N0 (10x) Final: items=179924/810000 (33.3%) mem=179924/16886 (1599.0%) time=866.426µs cost=$789.141
N0 Merge: total=810000 time=720.137µs cost=$655.900

F.4. Static Optimized Approach

N0 (9x) Solver: time=313.903µs cost=$101.045 ˆmakespan=9556.698µs ˆcost=$10366.946
Remaining items to allocate: 3
Partition sizes: [356922, 16849, 237921, 198308]
Ideal sizes: [356921.69216921687, 16849.684968496847, 237920.7920792079, 198307.8307830783]
Fractional parts: [0.6921692168689333, 0.6849684968474321, 0.792079207894858, 0.8307830783014651]
N1 (10x) Sort: items=16849/810000 (2.1%) mem=16849/16886 (99.8%) time=803.697µs cost=$732.007
N3 (5x) Sort: items=198308/810000 (24.5%) mem=198308/271079 (73.2%) time=11038.864µs cost=$2228.747
N2 (6x) Sort: items=237921/810000 (29.4%) mem=237921/549209 (43.3%) time=10886.389µs cost=$5684.872
N0 (9x) Sort: items=356922/810000 (44.1%) mem=356922/377826 (94.5%) time=10875.063µs cost=$3500.683
N0 (9x) Heap Merge: total=810000 time=10800.449µs cost=$3476.664
N0 (9x) Linear Merge: total=810000 time=31645.377µs cost=$10186.647

Appendix G. Gaussian, Seventh Run, Eight Nodes, Seed Four, Mean 252000, Std 100000

G.1. Cluster Configuration

Total memory of cluster = 12474000 (should be >= 8316000)

- `{'throughput': 4, 'memory': 69286, 'billing': 0.8442}`
- `{'throughput': 2, 'memory': 2726209, 'billing': 0.2483}`
  - 32.78% out of 150.00% of overprovisioned memory
  - 21.86% out of 100.00% of total cluster memory

Saved plot with bell curve as 'out/7run_8node_4seed_gaussian.png'.
Nodes: 8 | Data: 8316000 elements | Sample: [29867, 456758, 261669, 248884, 322270...167106, 247832, 434785, 295439, 314544]

G.2. Sequential Baseline

Sequential: time=2850812.096µs throughput=0.342810µs/element

G.3. Regular Sampling Approach

N0 (8x) Partition: time=5090.880µs time=3444.489µs cost=$3444.489
N0 Merge: total=8316000 time=12825.587µs cost=$8677.792

G.4. Static Optimized Approach

N0 (7x) Solver: time=9408.586µs cost=$1158.197 ^makespan=131263.266µs ^cost=$317399.026
N0 (7x) Linear Merge: total=8316000 time=704831.640µs cost=$86764.775

Appendix H. Gaussian, Eight Run, Eight Nodes, Seed Five, Mean 252000, Std 100000

H.1. Cluster Configuration

Total memory of cluster = 7128000 (should be >= 4752000)

- `{'throughput': 10, 'memory': 2086774, 'billing': 0.3515}`
- `{'throughput': 1, 'memory': 502019, 'billing': 0.214}`
  - 10.56% out of 150.00% of overprovisioned memory
  - 7.04% out of 100.00% of total cluster memory

Saved plot with bell curve as 'out/8run_8node_5seed_gaussian.png'.
Nodes: 8 | Data: 4752000 elements | Sample: [83409, 255172, 192245, 163083, 381266...112158, 250772, 216903, 325874, 250799]

H.2. Sequential Baseline

Sequential: time=1540436.006µs throughput=0.324166µs/element

H.3. Regular Sampling Approach

N0 (10x) Partition: time=2432.684µs time=855.088µs cost=$855.088
N0 Merge: total=4752000 time=7468.875µs cost=$2625.309

H.4. Static Optimized Approach

N0 (10x) Solver: time=261.470µs cost=$91.907 ^makespan=48478.526µs ^cost=$138548.885
N0 (10x) Linear Merge: total=4752000 time=290605.426µs cost=$102147.807