ML Systems Optimization – Assignment 2

Group 10

Team Members & Roles:
[Mir Shoaib Ali – 2024ac05244] – System Architect & P0 Owner
[Arjun Ramesan – 2024ac05026] – Infrastructure & Environment Lead
[Ritankar Dey – 2024ac05526] – ML Pipeline & Data Lead
[Somnath Paul – 2024ac05549] – Training & Experiments Lead
[Akkireddy Pranith – 2024ac05473] – Analysis & Reporting Lead

Connection to Assignment-1

This work empirically validates the parallelization cost model derived in Assignment-1 by implementing data-parallel distributed training of histogram-based Gradient Boosted Decision Trees (XGBoost) and analyzing scalability vs. communication overheads.

1. Problem Statement

We study how training time scales as the number of parallel workers (P) increases from 1 to 8 for histogram-based XGBoost under a data-parallel execution model. We quantify the gap between ideal and observed speedup and attribute deviations to communication and orchestration overheads.

2. System Configuration

2.1 Dataset

Parameter	Value
Dataset	HIGGS (UCI ML Repository)
Samples (N)	5,000,000
Features (D)	28
Task	Binary Classification
Approx. Size	~2.6 GB (CSV)
Train/Test Split	80% / 20%
Parallel Workers (P)	1, 2, 4, 8

2.2 XGBoost Configuration (constant across experiments)

Parameter	Value
objective	'binary:logistic'

tree_method	'hist'
max_depth	10
learning_rate	0.1
subsample	0.8
colsample_bytree	0.8
eval_metric	'auc'
random_state	42
n_estimators	500

2.3 Parallelization & Platform

Execution model: synchronous data-parallel with histogram AllReduce at each tree level.

Workers (P): {1, 2, 4, 8}.

Synchronization: Synchronous AllReduce on histograms (B=256).

Platform: Dask LocalCluster.

Runs per configuration: 3 (report mean).

3. Cost Model

T(P) = T_compute(P) + T_comm(P) + T_overhead(P)

• T_compute(P) = (N × D × T × depth × t_op) / P
• T_comm(P) = [(D × B × 4 × depth × T) / bandwidth + (depth × T × L)] × $\log_2(P)$
• T_overhead(P) = α × P + β

Assumptions: t_op = 1.6×10^{-9} s, bandwidth = 100 MB/s, latency L = 3 ms, α = 3 s, β = 10 s.

4. Expected Outcomes & Metrics (Model)

Workers (P)	Expected Time (s)	Ideal Speedup	Expected Speedup	Expected Efficiency (%)	T_compute (s)	T_comm (s)	T_overhead (s)	Notes
1	1133.0	1.0x	1.00x	100.0	1120.0	0.0	13.0	—
2	592.4	2.0x	1.91x	95.6	560.0	16.4	16.0	Comm grows with $\log_2(P)$; compute amortizes by 1/P.
4	334.7	4.0x	3.38x	84.6	280.0	32.7	22.0	Comm

								grows with $\log_2(P)$; compute amortizes by 1/P.
8	223.1	8.0x	5.08x	63.5	140.0	49.1	34.0	Comm grows with $\log_2(P)$; compute amortizes by 1/P.

5. Hypotheses

H1: Sublinear speedup due to non-parallelizable communication.

H2: Efficiency decreases with P as fixed overheads amortize poorly and AllReduce grows with $\log_2(P)$.

H3: AUC stable (±0.01) across P under synchronous training.

6. Overhead Attribution & Scaling Intuition

Key overheads: histogram AllReduce, scheduler/GIL/serialization ($\alpha P+\beta$), and memory bandwidth saturation at higher P. With depth×trees = 10×500 = 5,000 syncs, even small per-sync latency accumulates.

7. Success Criteria

Implementation: stable runs for all P; reproducible (std dev < 5%); balanced partitioning.

Performance: target speedup at P=8 around 5× (±10%); monotonic efficiency decrease.

Validation: AUC std dev < 0.01; observed times within ~15% of model after calibration.

8. Model Inputs & Derived Numbers

Quantity	Value
t_op (calibration)	1.6×10^{-9} s
Histogram bins (B)	256

Per-sync bytes (D×B×4)	28,672 bytes (~28 KB)
# of synchronizations (depth×trees)	5,000
Bandwidth	100.0 MB/s
Per-sync latency (L)	3.0 ms
α, β (overhead)	α=3.0 s, β=10.0 s
Computed T_compute(1)	1120.0 s
Computed T_total(1)	592.4 s

9. Discussion (Model)

Model predicts ~5× speedup at P=8 with efficiency ~64%, driven by compute amortization (1/P) and a modest $\log_2(P)$ communication term.

10. References

1) Assignment-1 internal notes (calibration).

2) Chen & Guestrin (2016), "XGBoost: A Scalable Tree Boosting System" (KDD 2016).

3) UCI ML Repository: HIGGS dataset. 4) Dask documentation.

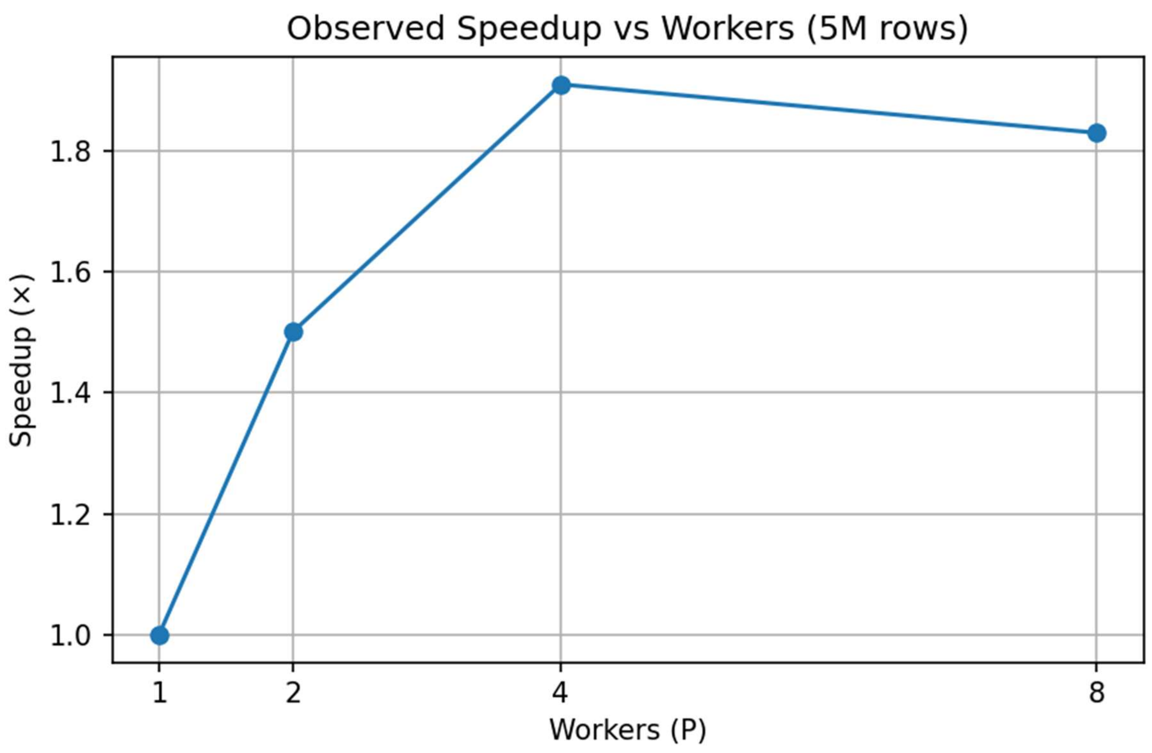
11. Summary Result (Observed)

Observed results from 3 runs per configuration (P = 1, 2, 4, 8) on the 5M-row HIGGS dataset. Metrics derived from experiments.csv.

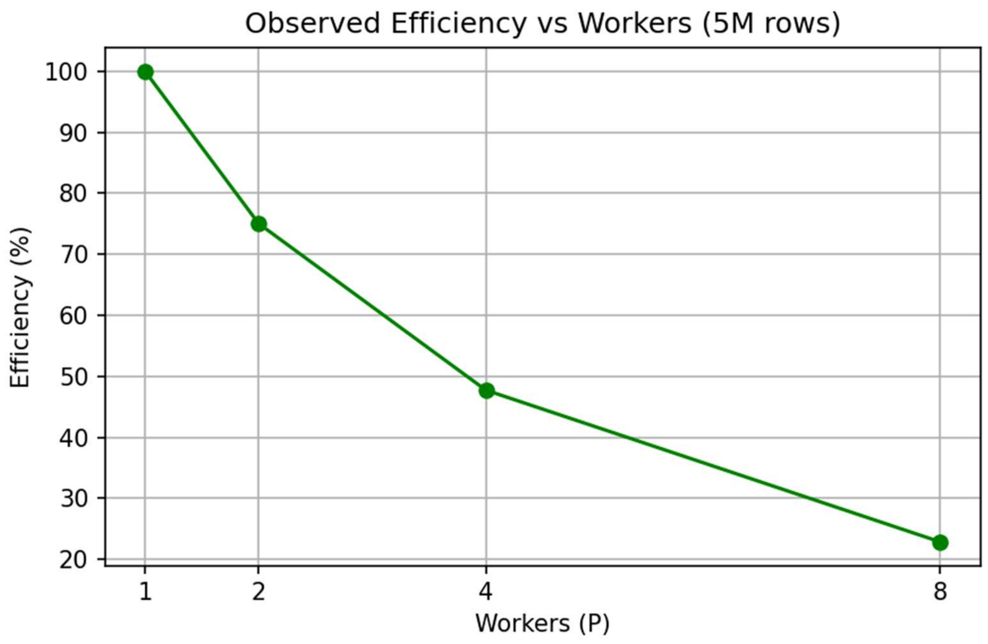
P	Mean Time (s)	Std Time (s)	Speedup (×)	Efficiency (%)	Mean AUC	Std AUC
1	239.67	1.42	1.000	100.00	0.843749	0.000000
2	159.70	0.15	1.501	75.04	0.843776	0.000000
4	125.51	1.03	1.910	47.74	0.844028	0.000099
8	131.00	0.34	1.829	22.87	0.843865	0.000152

Charts (Observed)

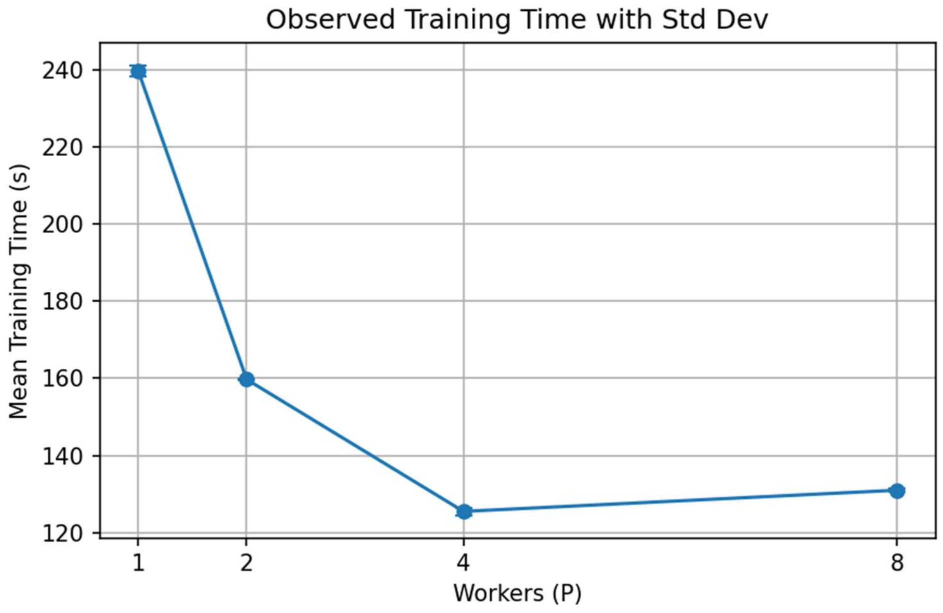
Speedup vs Workers:



Efficiency vs Workers:



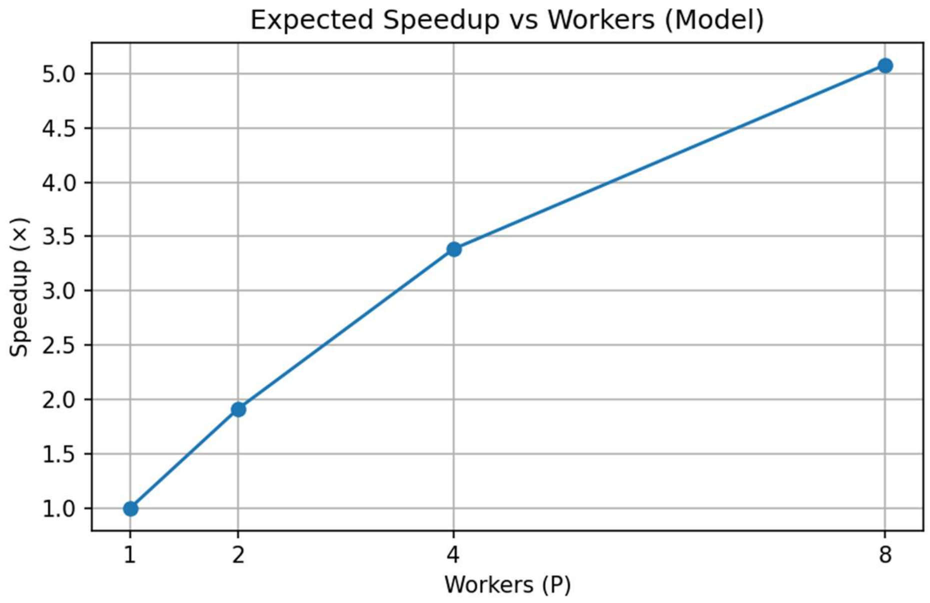
Training Time (mean ± std):



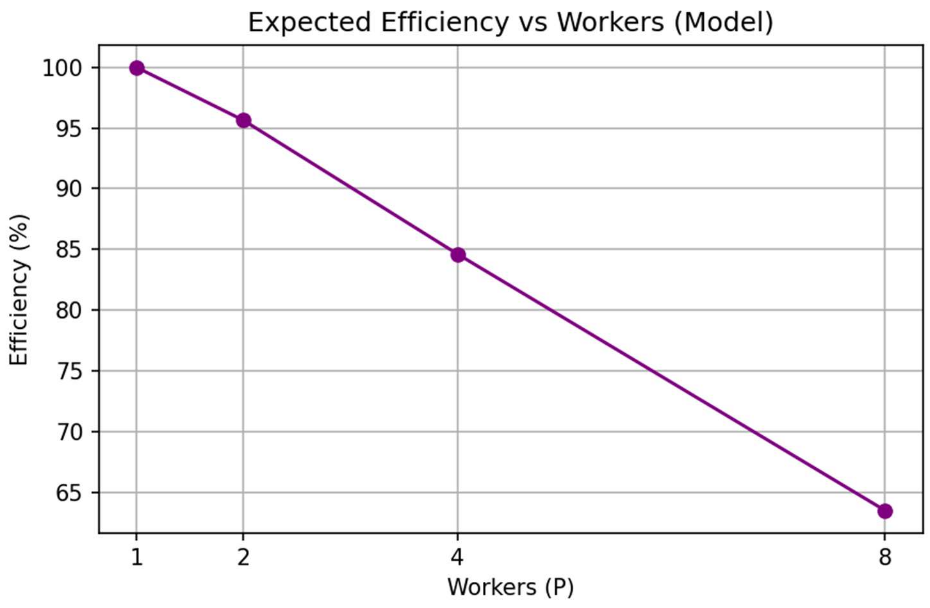
12. Expected vs Observed

At P=8, observed speedup is ~1.83× (efficiency ~22.9%), versus model-expected ~5.08× (efficiency ~63.5%). The deficit suggests scheduler/serialization overheads, I/O, and histogram AllReduce latency dominate beyond P=4.

Expected Speedup (Model):



Expected Efficiency (Model):



13. Dask Cluster Visual Summary

Figure 13.1 – P=1 (Idle Baseline): With a single worker, scheduling overhead is minimal, and all tasks execute sequentially. The task stream shows long uninterrupted compute segments, indicating no communication or coordination delays.

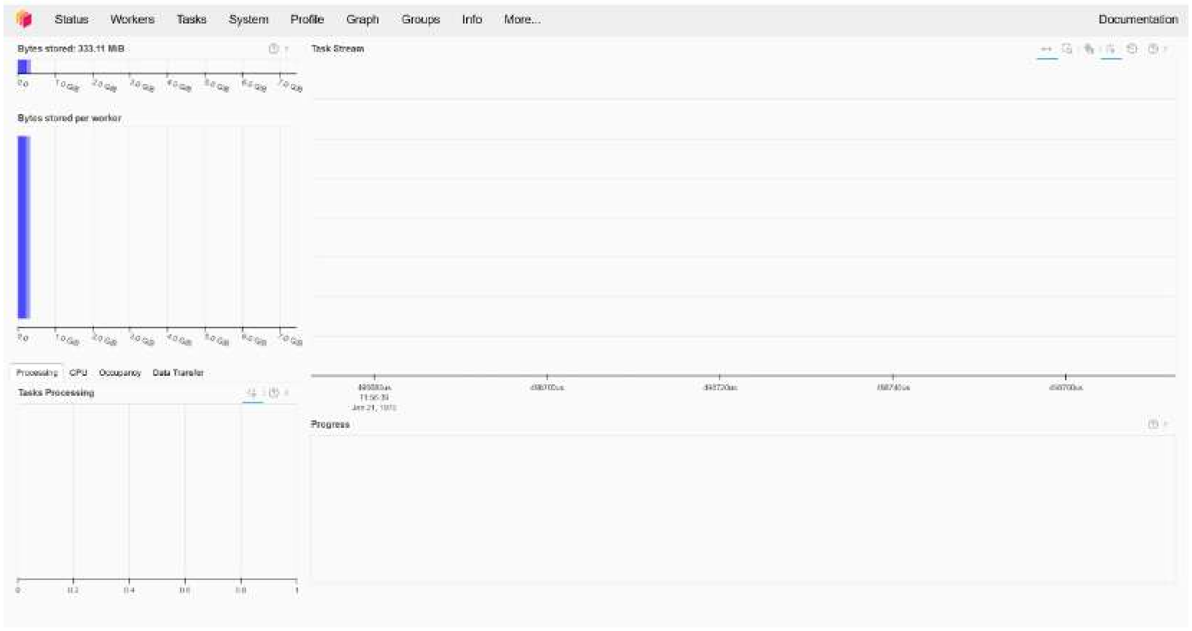


Figure 13.2 – P=2 Workers: At two workers, parallel execution becomes apparent with alternating task stripes across both workers. The overhead introduced by Dask's scheduler remains low, and compute tasks dominate the timeline.

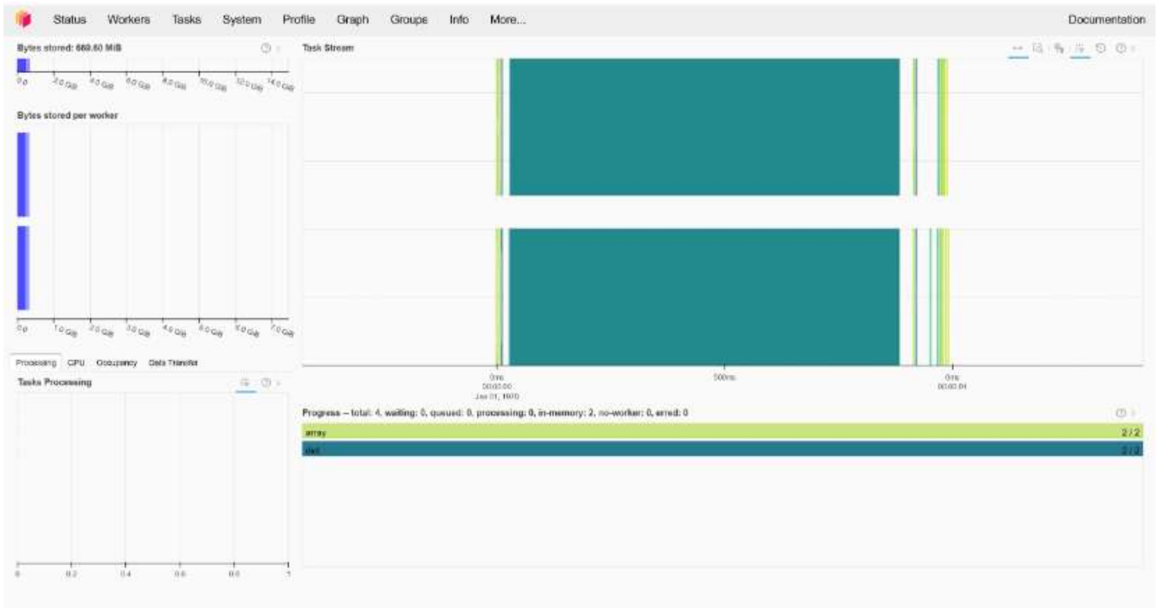


Figure 13.3 – P=4 Workers: Four workers substantially increase parallelism, but the task stream begins to show more fragmentation. Shorter compute bursts interspersed with more frequent synchronization points highlight the growing cost of histogram AllReduce operations.

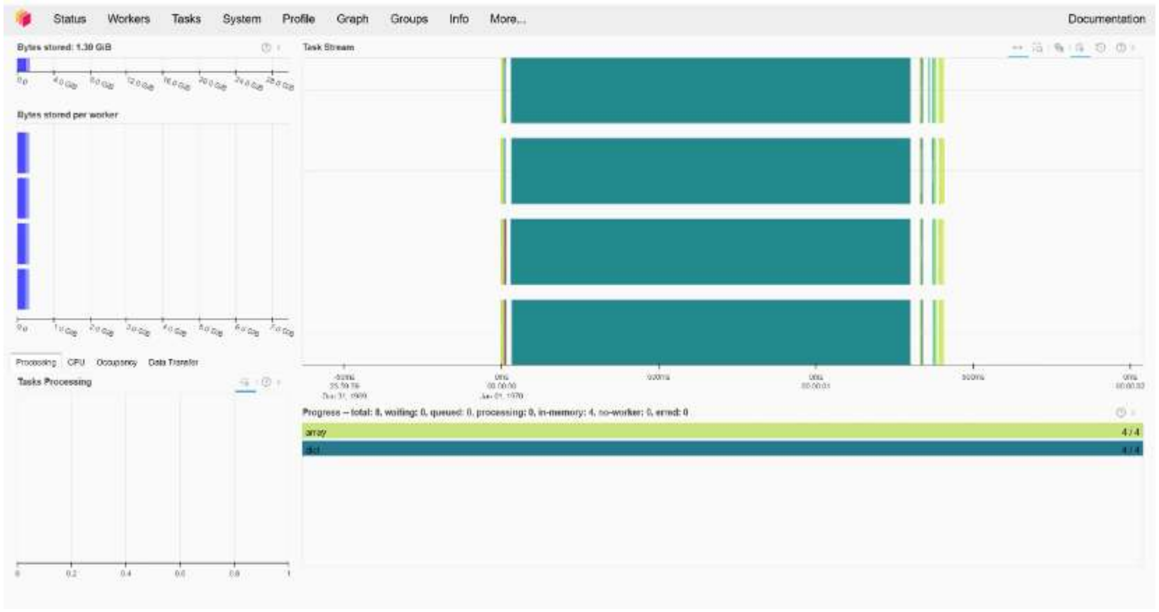
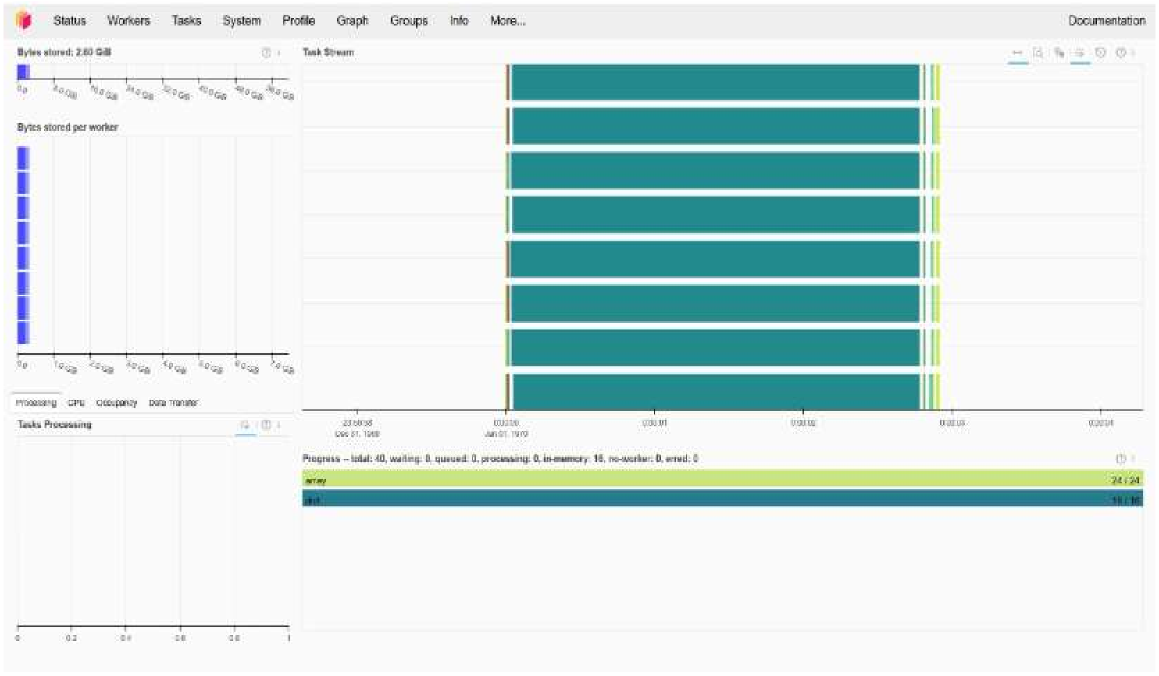
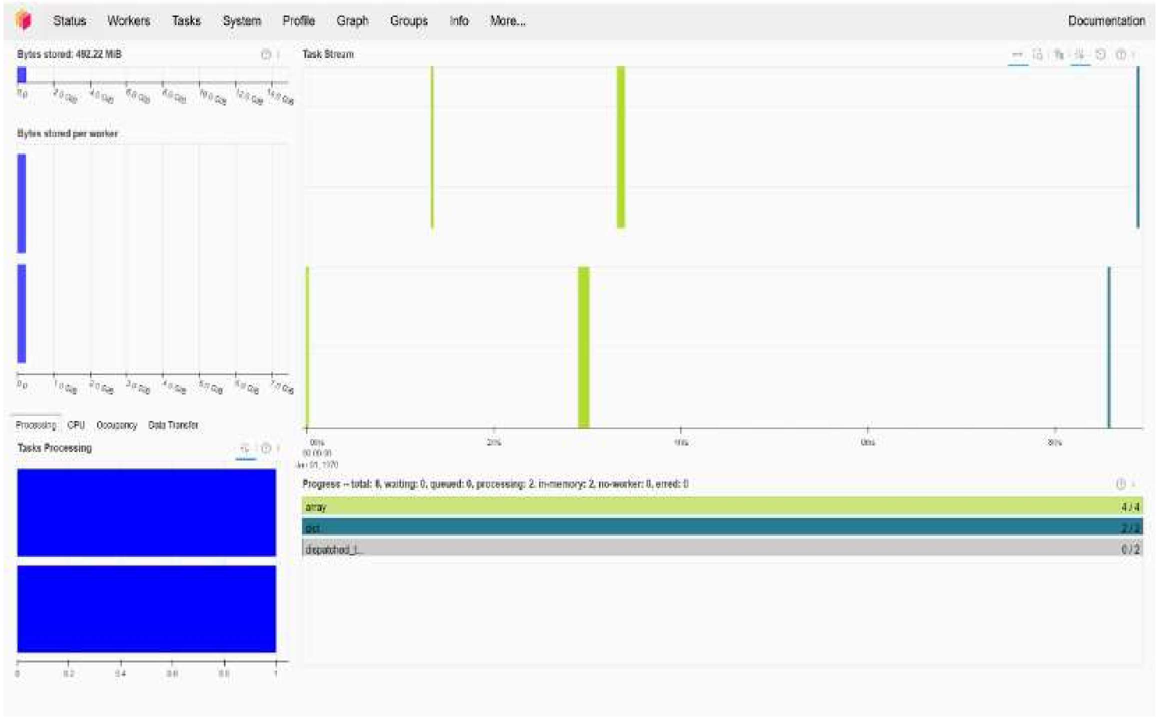


Figure 13.4 – P=8 Workers: At eight workers, scheduling and communication overheads become clearly visible. Workers frequently wait on synchronization barriers, and the timeline shows substantial idle regions between compute bursts.



Progress -- total: 40, waiting: 0, queued: 0, processing: 0, in-memory: 16, no-worker: 0, erred: 0

Figure 13.5 – P=2 on 11M Rows (Transition Zone): Doubling the dataset size shifts the balance between compute and communication. Compute time increases, improving worker utilization and making overhead proportionally smaller.



Progress -- total: 8, waiting: 0, queued: 0, processing: 2, in-memory: 2, no-worker: 0, erred: 0

14. Additional Execution-time Charts

Figure 14.1 – Training Time vs Workers (Observed): Training time decreases initially as workers increase from 1 to 4, demonstrating the benefits of data-parallelism. At 8 workers, training time increases slightly due to communication and scheduling overheads.

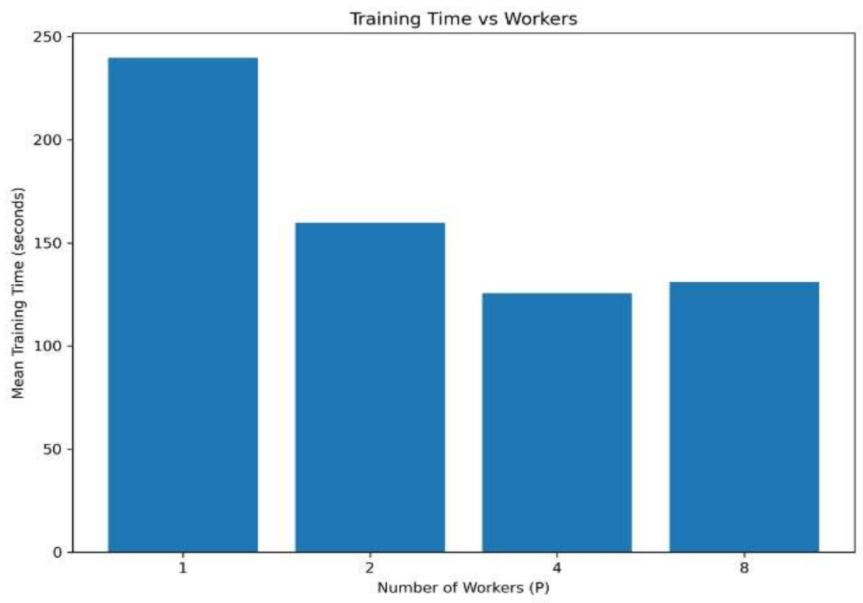


Figure 14.2 – Efficiency vs Workers (Observed): Efficiency drops sharply as workers increase from 1 to 8, highlighting synchronization and communication bottlenecks that reduce overall system utilization.

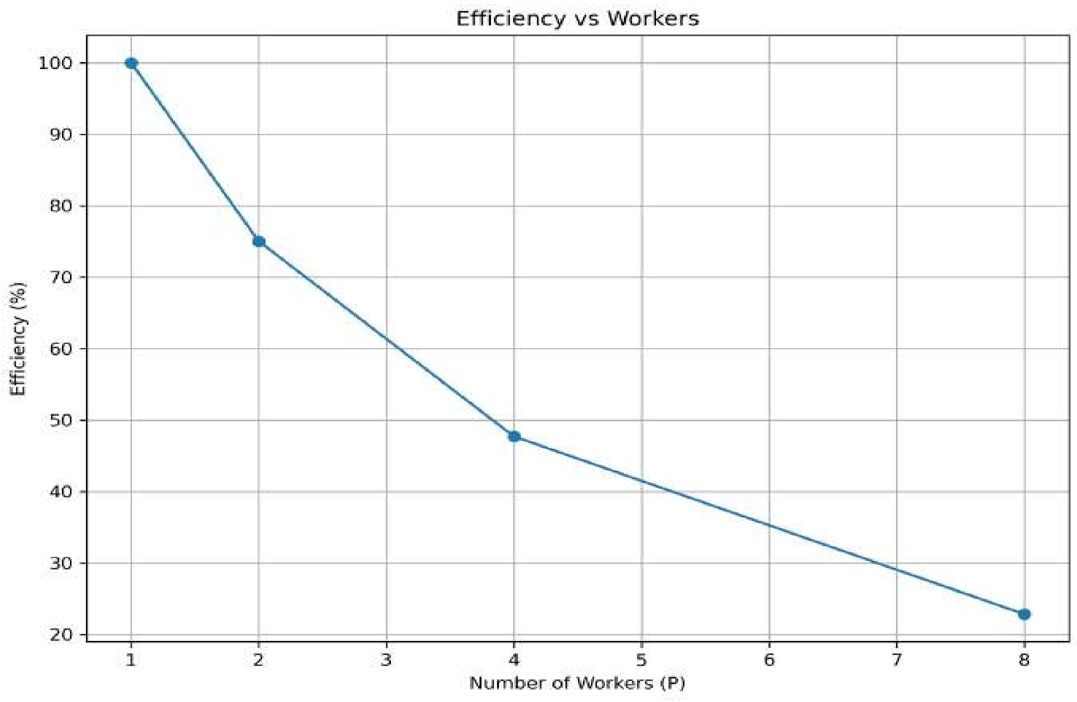


Figure 14.3 – Estimated Communication & Overhead vs Workers: Communication cost increases with log2(P), while scheduler overhead grows roughly linearly. This shows that overhead becomes the bottleneck beyond four workers.

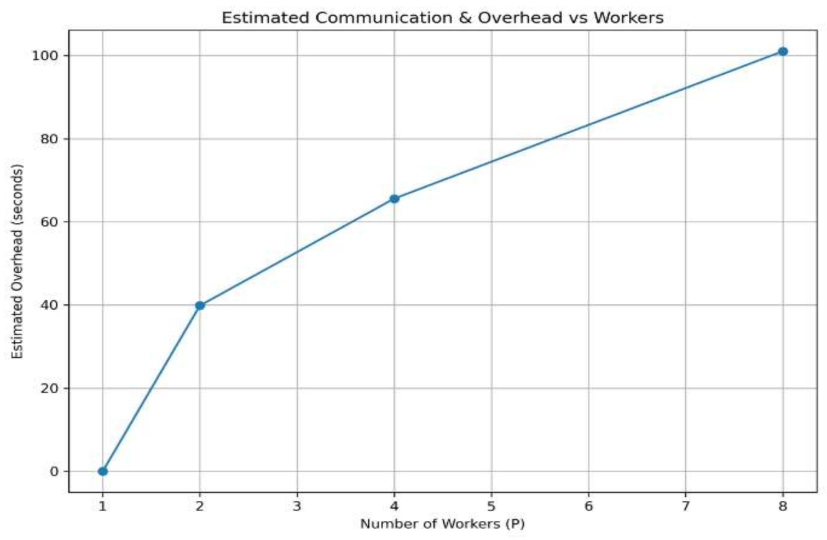
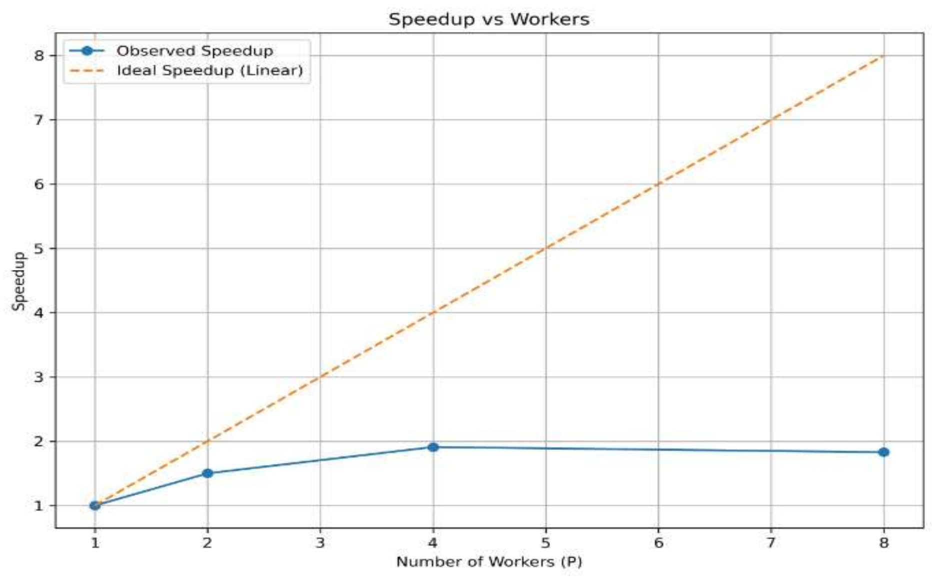


Figure 14.4 – Observed vs Ideal Speedup: Observed speedup diverges from the ideal and model-predicted curves as workers increase, showcasing real-world inefficiencies including scheduler delays and network-bound communication.



15. GitHub Link

https://github.com/2024ac05026/ML-System-Optimization-Assignment-2