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

Github Link

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

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

$T(P) = T_{compute}(P) + T_{comm}(P) + T_{overhead}(P)$

- $T_{compute}(P) = (N \times D \times T \times depth \times t_{op}) / P$
- $T_{comm}(P) = [(D \times B \times 4 \times depth \times T) / bandwidth + (depth \times T \times L)] \times \log_2(P)$
- $T_{overhead}(P) = \alpha \times P + \beta$

Assumptions: $t_{op} = 1.6 \times 10^{-9}$ s, bandwidth = 100 MB/s, latency L = 3 ms, $\alpha = 3$ s, $\beta = 10$ 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 (αP+β), 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.

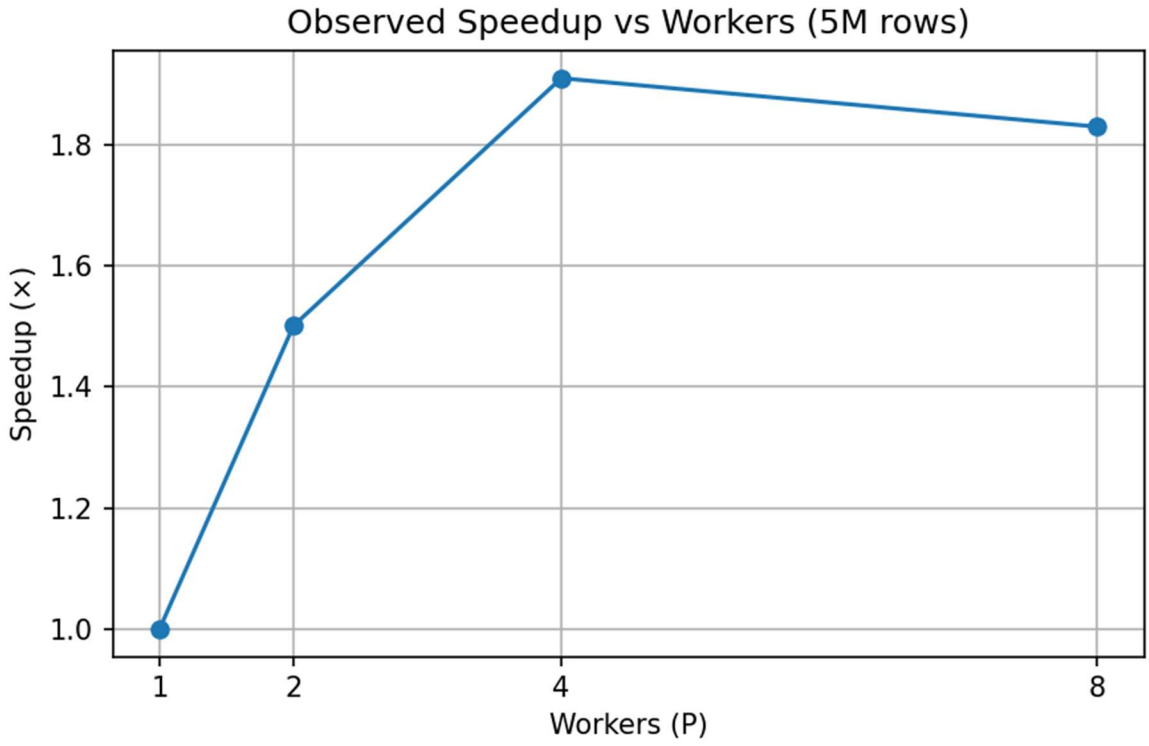
11. Summary Result (Observed)

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

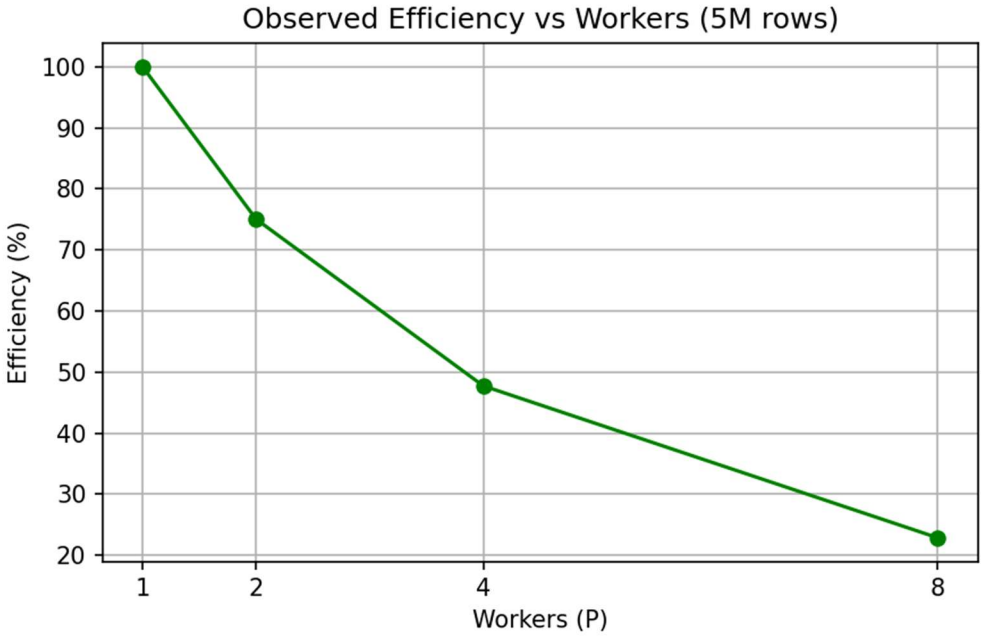
P	Mean Time (s)	Std Time (s)	Speedup (×)	Efficiency (%)	Mean AUC	Std AUC
1	239.67	1.42	1.000	100.00	0.843749	0.000000
2	159.70	0.15	1.501	75.04	0.843776	0.000000
4	125.51	1.03	1.910	47.74	0.844028	0.000099
8	131.00	0.34	1.829	22.87	0.843865	0.000152

Charts (Observed)

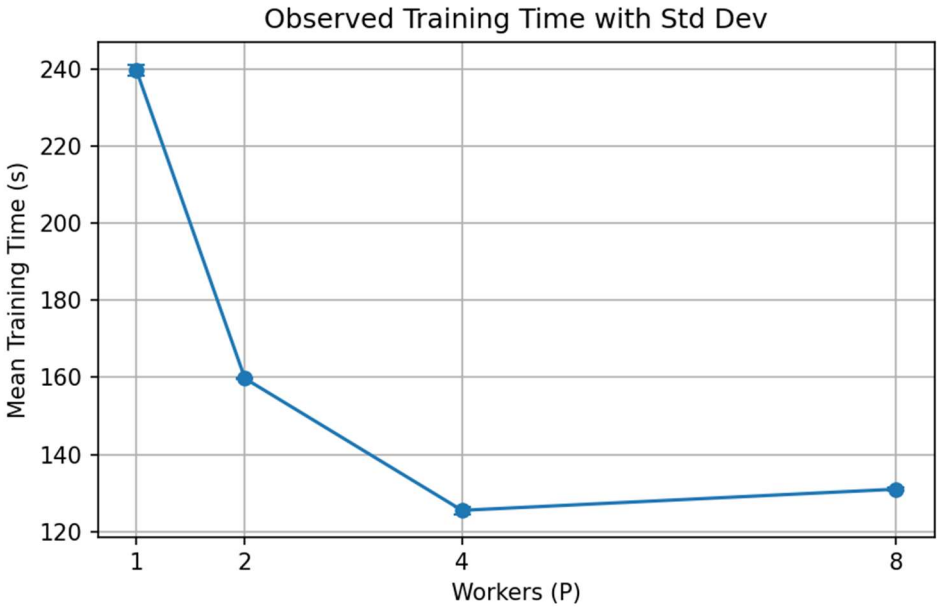
Speedup vs Workers:



Efficiency vs Workers:



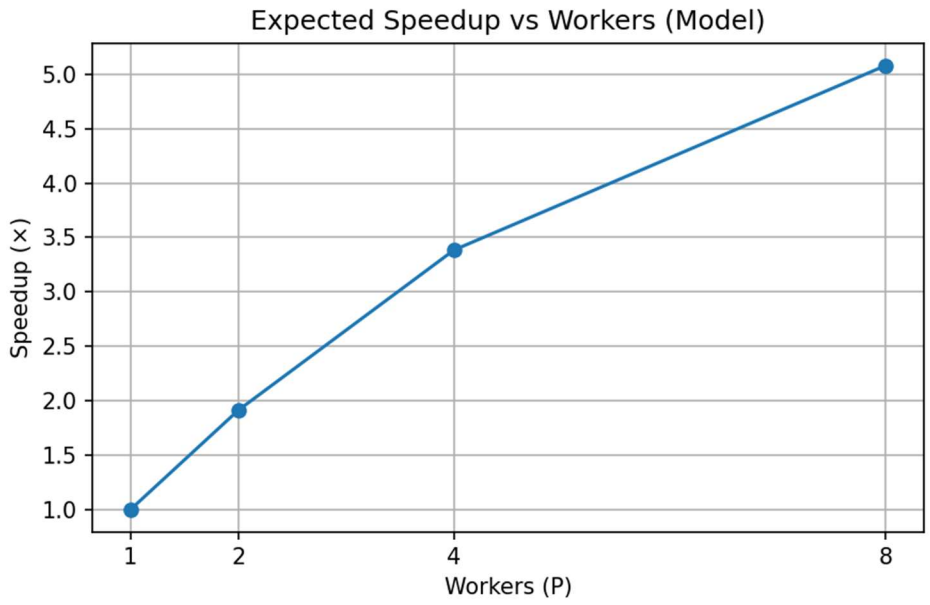
Training Time (mean ± std):



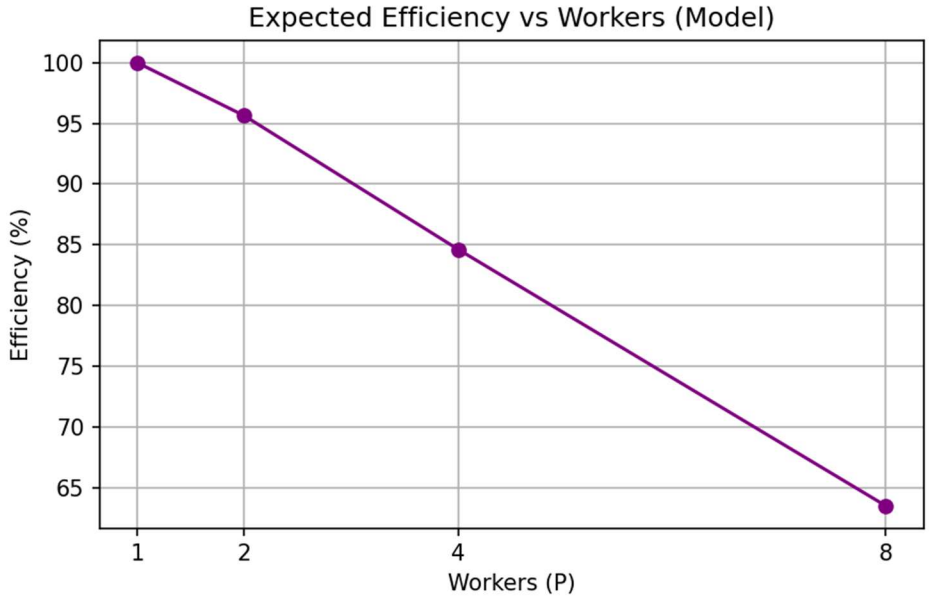
12. Expected vs Observed

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.

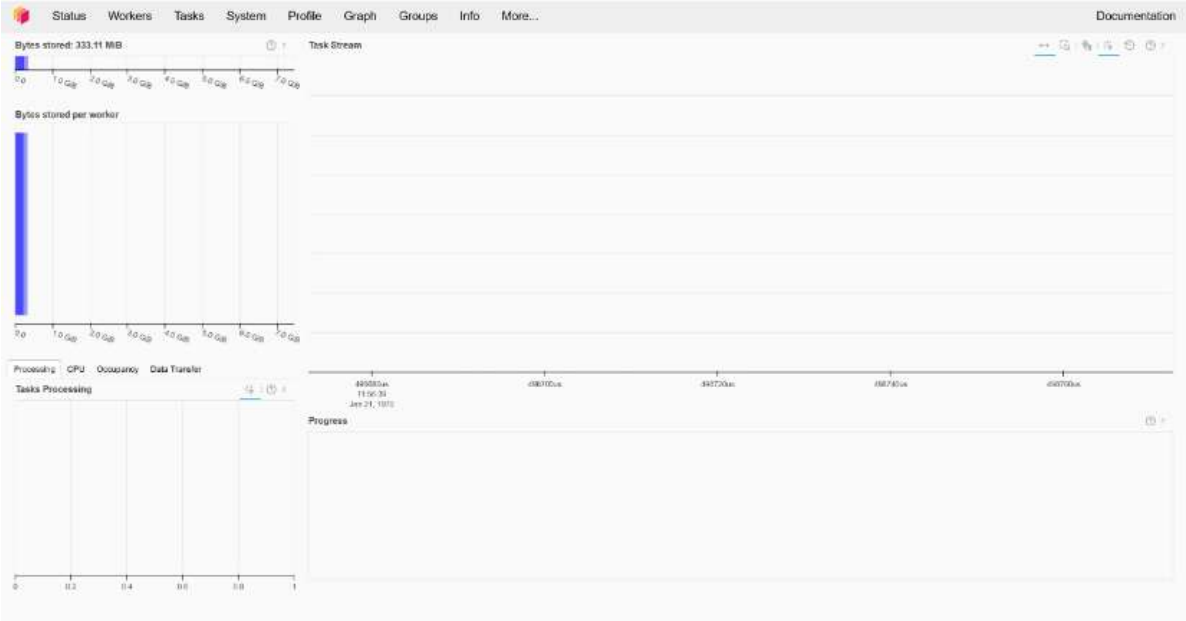


Figure 13.2 – P=2 Workers: At two workers, parallel execution becomes apparent with alternating task stripes across both workers. The overhead introduced by Dask's scheduler remains low, and compute tasks dominate the timeline.

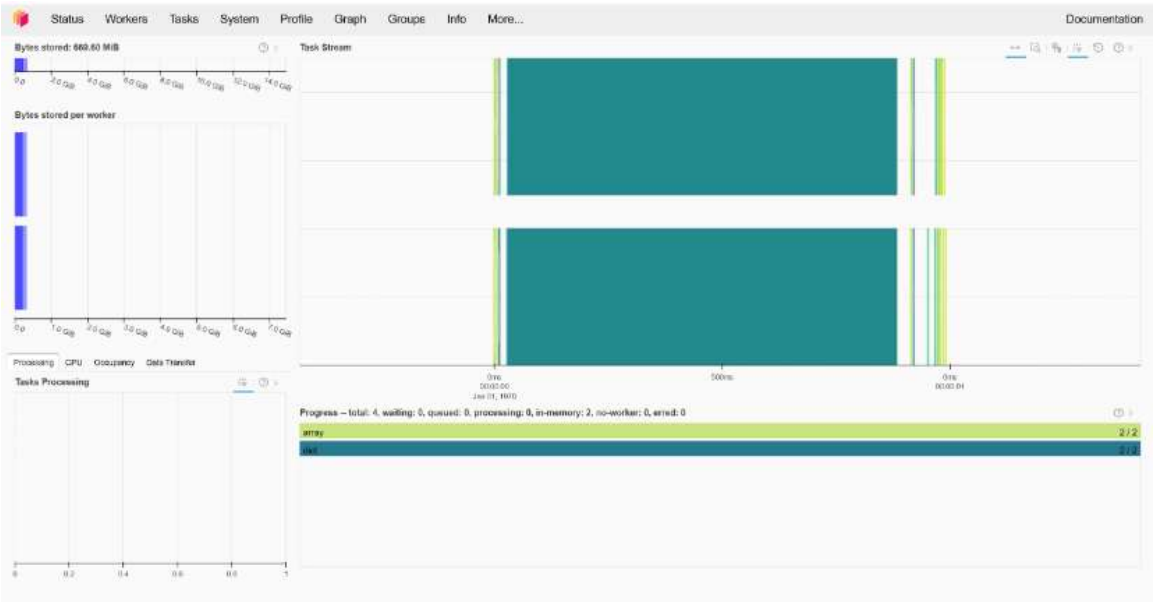


Figure 13.3 – P=4 Workers: Four workers substantially increase parallelism, but the task stream begins to show more fragmentation. Shorter compute bursts interspersed with more frequent synchronization points highlight the growing cost of histogram AllReduce operations.

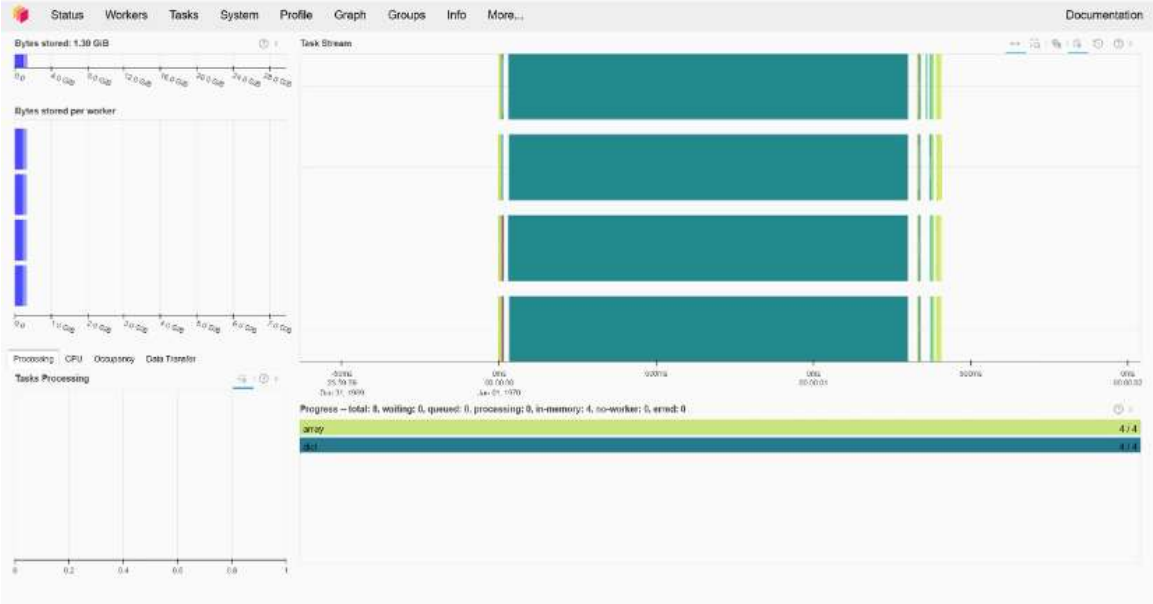


Figure 13.4 – P=8 Workers: At eight workers, scheduling and communication overheads become clearly visible. Workers frequently wait on synchronization barriers, and the timeline shows substantial idle regions between compute bursts.

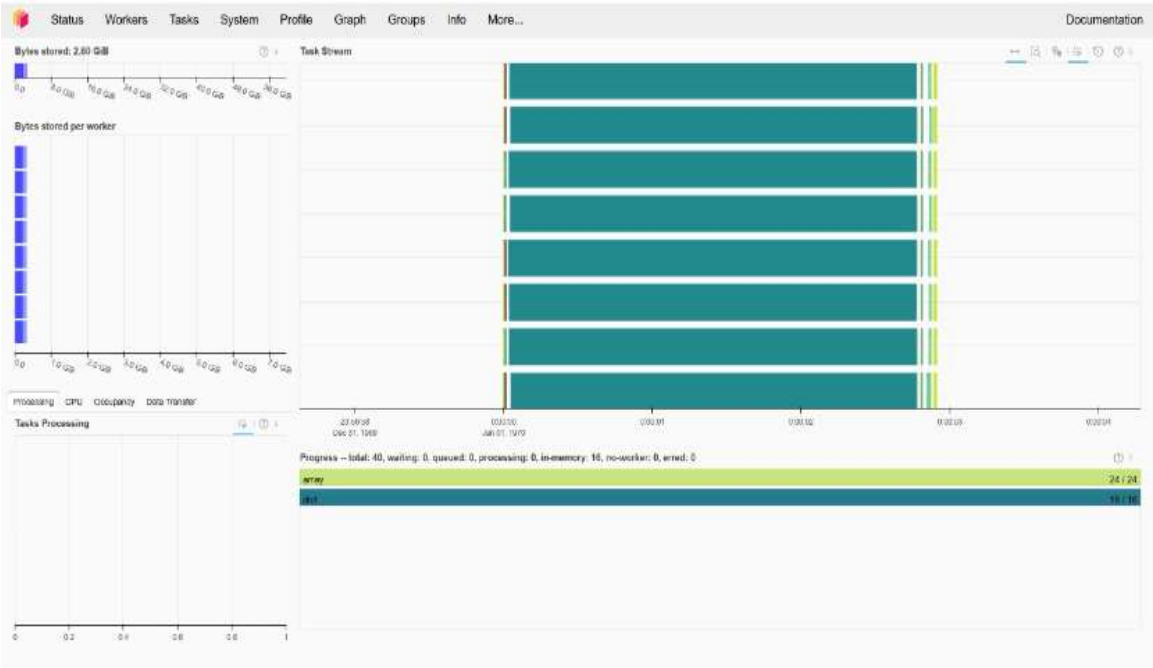
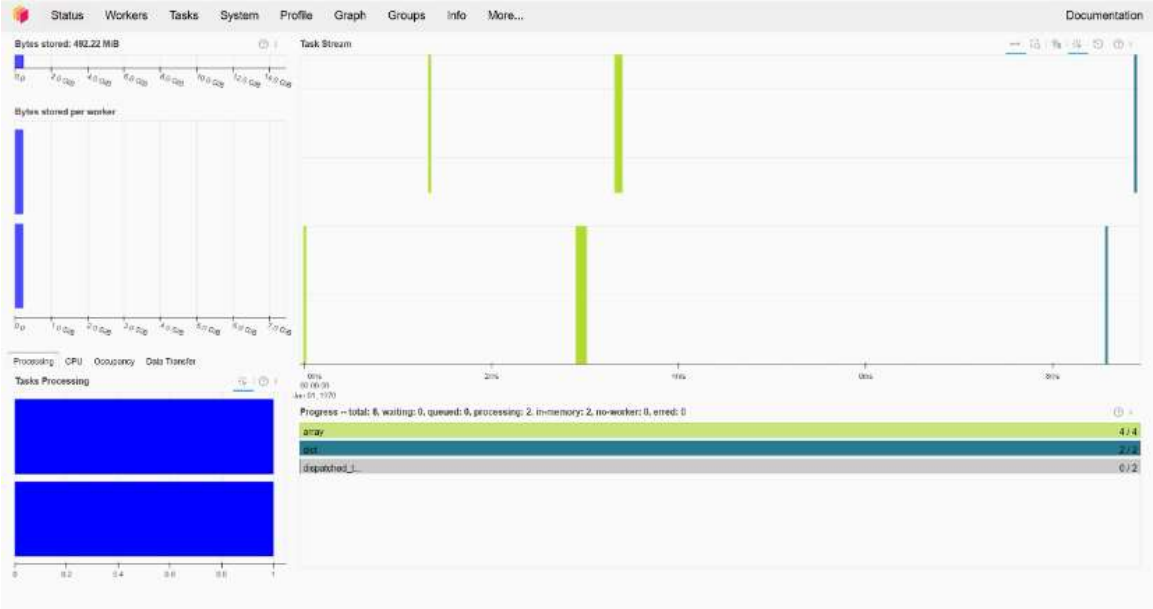


Figure 13.5 – P=2 on 11M Rows (Transition Zone): Doubling the dataset size shifts the balance between compute and communication. Compute time increases, improving worker utilization and making overhead proportionally smaller.



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.

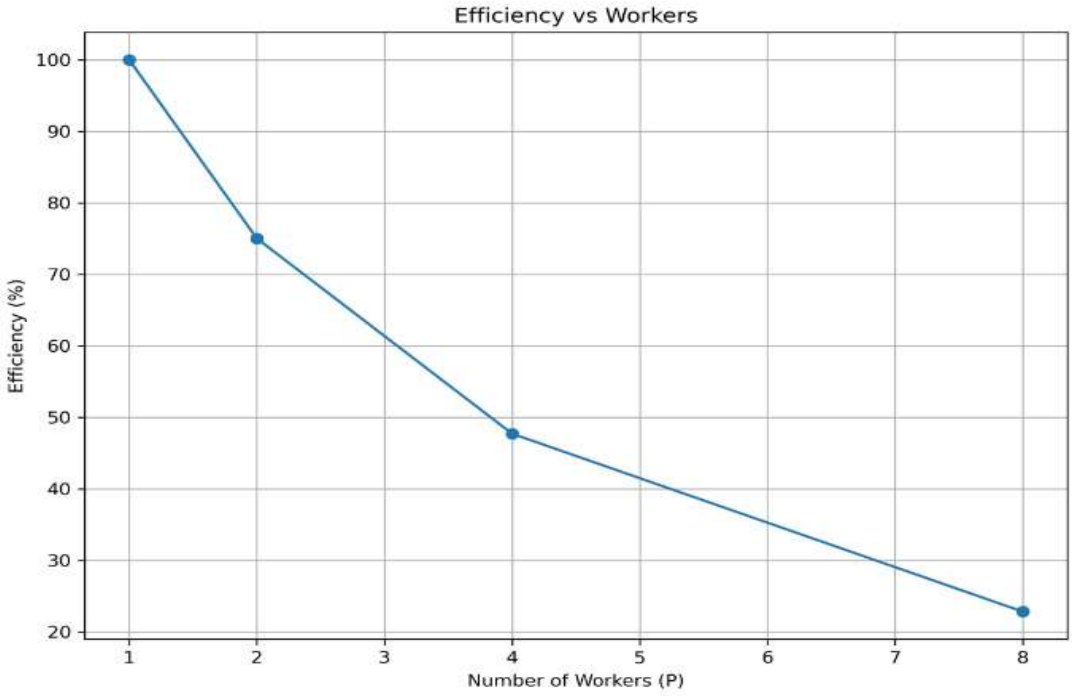


Figure 14.3 – Estimated Communication & Overhead vs Workers: Communication cost increases with log2(P), while scheduler overhead grows roughly linearly. This shows that overhead becomes the bottleneck beyond four workers.

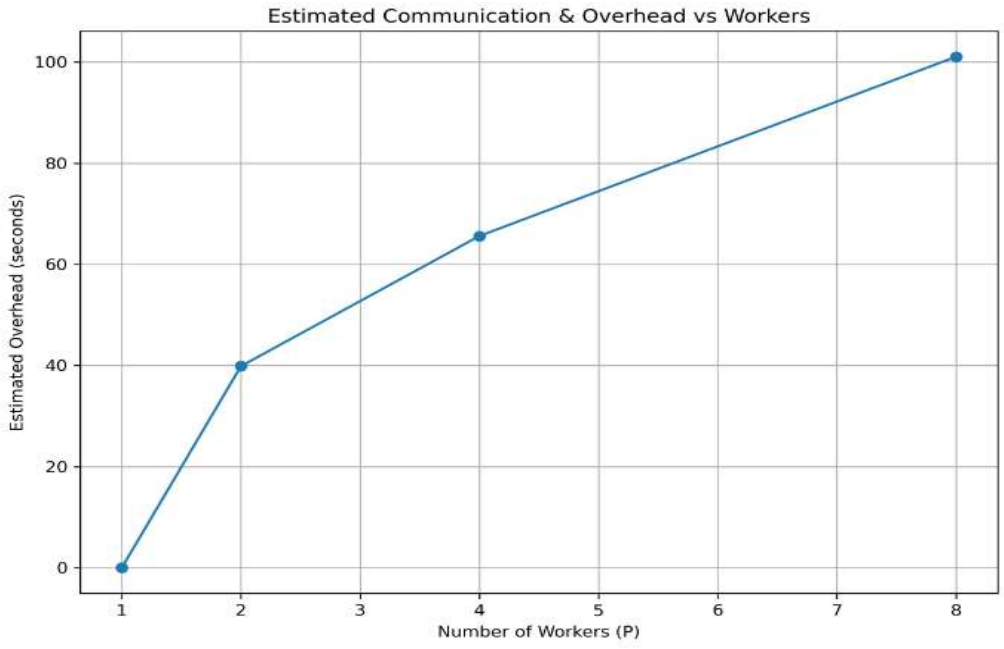
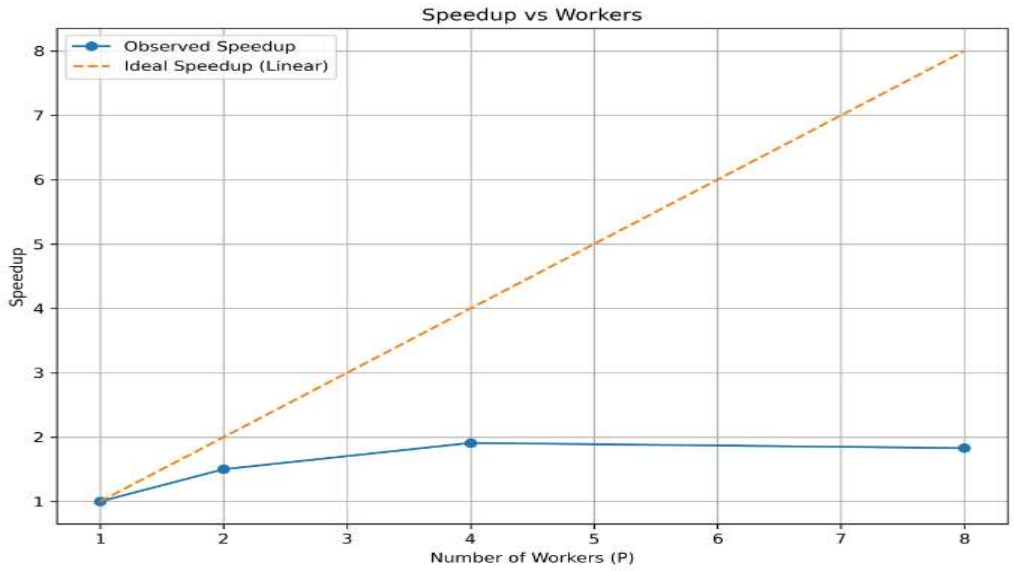


Figure 14.4 – Observed vs Ideal Speedup: Observed speedup diverges from the ideal and model-predicted curves as workers increase, showcasing real-world inefficiencies including scheduler delays and network-bound communication.



15. Code Snippets

```python
# src/config.py
class ExperimentConfig:
    # Dataset
    DATASET_PATH = "data/HIGGS.csv"
    # N_SAMPLES = 11_000_000
    N_SAMPLES = 5000000
    N_FEATURES = 28
    TEST_SIZE = 0.2

    # XGBoost params (MUST be constant across all experiments)
    XGBOOST_PARAMS = {
        'objective': 'binary:logistic',
        'tree_method': 'hist',
        'max_depth': 10,
        'learning_rate': 0.1,
        'subsample': 0.8,
        'colsample_bytree': 0.8,
        'random_state': 42,
        'eval_metric': 'auc'
    }
    N_ESTIMATORS = 500


    # Experiment settings
    WORKER_COUNTS = [1, 2, 4, 8]
    N_RUNS_PER_CONFIG = 3  # For averaging

    # Output
    RESULTS_FILE = "results/experiments.csv"
    PLOTS_DIR = "results/plots/"
```

```python
from dask.distributed import Client, LocalCluster
from typing import Tuple
import logging
import os

logger = logging.getLogger(__name__)


def setup_dask_cluster(
    n_workers: int,
    threads_per_worker: int = 1,
    memory_limit: str = "8GB",
    dashboard: bool = False,
) -> Tuple[LocalCluster, Client]:
    """
    Initialize a Dask LocalCluster.

    Dashboard is optional and should be enabled only for debugging.
    """

    dashboard_address = ":8787" if dashboard else None

    logger.info(
        f"Starting Dask cluster | workers={n_workers}, "
        f"threads/worker={threads_per_worker}, memory={memory_limit}, "
        f"dashboard={dashboard}"
    )

    cluster = LocalCluster(
        n_workers=n_workers,
        threads_per_worker=threads_per_worker,
        processes=True,              # REQUIRED: avoid GIL
        memory_limit=memory_limit,
        dashboard_address=dashboard_address,
        silence_logs=logging.ERROR,
    )

    client = Client(cluster)
    client.wait_for_workers(n_workers)

    if dashboard:
        logger.info(f"Dask dashboard available at: {client.dashboard_link}")

    return cluster, client


def shutdown_dask_cluster(cluster: LocalCluster, client: Client):
    logger.info("Shutting down Dask cluster")
    client.close()
    cluster.close()
```

```python
# src/data_loader.py

import os
import urllib.request
import gzip
import shutil
from typing import Tuple

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from dask import array as da


HIGGS_URL = (
    "https://archive.ics.uci.edu/ml/machine-learning-databases/00280/HIGGS.csv.gz"
)


class DataLoader:
    def __init__(self, config):
        self.config = config

    # ------------------------------------------------
    # Ensure Dataset Exists (Download if Missing)
    # ------------------------------------------------
    def _ensure_dataset_exists(self):
        dataset_path = self.config.DATASET_PATH
        os.makedirs(os.path.dirname(dataset_path), exist_ok=True)

        gz_path = dataset_path + ".gz"


        if os.path.exists(dataset_path):
            print("Higgs dataset found locally.")
            return

        if os.path.exists(gz_path):
            print("Compressed Higgs dataset found locally. Extracting...")
            # Extract
            with gzip.open(gz_path, "rb") as f_in:
                with open(dataset_path, "wb") as f_out:
                    shutil.copyfileobj(f_in, f_out)
            return



        print("Higgs dataset not found. Downloading 2.6 GB file...")
        print("You could also go to " + HIGGS_URL + " and download it manually if you prefer. Place it
inside data folder and name it higgs.csv.gz")

        # Download compressed file
        urllib.request.urlretrieve(HIGGS_URL, gz_path)
        # Extract
        with gzip.open(gz_path, "rb") as f_in:
            with open(dataset_path, "wb") as f_out:
                shutil.copyfileobj(f_in, f_out)

        print("Download complete.")
```

```python
    # ------------------------------------------------
    # Load Dataset
    # ------------------------------------------------
    def load_dataset(self) -> Tuple[np.ndarray, np.ndarray]:
        """
        Load Higgs dataset.

        Format:
        Column 0 = label
        Columns 1-28 = features
        """

        self._ensure_dataset_exists()

        print("Loading dataset...")

        df = pd.read_csv(
            self.config.DATASET_PATH,
            header=None,
            dtype=np.float32,
            nrows=self.config.N_SAMPLES
        )

        y = df.iloc[:, 0].values.astype(np.int32, copy=False)
        X = df.iloc[:, 1:].values  # already float32

        print(f"Dataset loaded: {X.shape}")

        return X, y
```

```python
    # ------------------------------------------------
    # Preprocess
    # ------------------------------------------------
    def preprocess(
        self,
        X: np.ndarray,
        y: np.ndarray
    ) -> Tuple[np.ndarray, np.ndarray]:
        """
        Minimal preprocessing:
        - Ensure contiguous arrays
        - Ensure correct dtype
        """

        X = np.ascontiguousarray(X, dtype=np.float32)
        y = np.ascontiguousarray(y, dtype=np.int32)

        return X, y

    # ------------------------------------------------
    # Train/Test Split (Stratified)
    # ------------------------------------------------
    def split_data(
        self,
        X: np.ndarray,
        y: np.ndarray
    ) -> Tuple[np.ndarray, np.ndarray, np.ndarray, np.ndarray]:

        return train_test_split(
            X,
            y,
            test_size=self.config.TEST_SIZE,
            random_state=42,
            stratify=y
        )
```

```python
    # ------------------------------------------------
    # Partition for Dask (Horizontal Split)
    # ------------------------------------------------
    def partition_for_dask(
        self,
        X: np.ndarray,
        y: np.ndarray,
        n_workers: int
    ) -> Tuple[da.Array, da.Array]:

        if n_workers <= 0:
            raise ValueError("n_workers must be > 0")

        n_samples = X.shape[0]

        if n_workers > n_samples:
            raise ValueError("Number of workers exceeds number of samples.")

        # Even chunk distribution
        base_chunk = n_samples // n_workers
        remainder = n_samples % n_workers

        chunk_sizes = [
            base_chunk + 1 if i < remainder else base_chunk
            for i in range(n_workers)
        ]

        X_dask = da.from_array(
            X,
            chunks=(tuple(chunk_sizes), X.shape[1])
        )

        y_dask = da.from_array(
            y,
            chunks=(tuple(chunk_sizes),)
        )

        print(
            f"Dask partitioned into {n_workers} chunks "
            f"(~{base_chunk} rows per worker)"
        )

        return X_dask, y_dask
```

```python
import time
from typing import Tuple, Dict

import os
os.environ["OMP_NUM_THREADS"] = "1"
os.environ["MKL_NUM_THREADS"] = "1"
os.environ["OPENBLAS_NUM_THREADS"] = "1"

import xgboost as xgb
from dask import array as da
from sklearn.metrics import accuracy_score, roc_auc_score

from src.dask_cluster import setup_dask_cluster, shutdown_dask_cluster


class DistributedXGBoostTrainer:
    def __init__(self, n_workers: int, config):
        self.n_workers = n_workers
        self.config = config
        self.cluster = None
        self.client = None

    # ------------------------------------------------
    # Setup Dask Cluster
    # ------------------------------------------------
    def setup_cluster(self, show_dashboard=False):
        self.cluster, self.client = setup_dask_cluster(
            n_workers=self.n_workers,
            threads_per_worker=1,
            memory_limit="8GB",
            dashboard=show_dashboard
        )

    # ------------------------------------------------
    # Train Distributed XGBoost
    # ------------------------------------------------
    def train(self, X, y) -> Tuple[xgb.Booster, float]:

        if self.client is None:
            raise RuntimeError("Dask cluster not initialized. Call setup_cluster() first.")

        # Chunk size per worker
        chunk_size = len(X) // self.n_workers

        X_dask = da.from_array(X, chunks=(chunk_size, X.shape[1]))
        y_dask = da.from_array(y, chunks=(chunk_size,))

        dtrain = xgb.dask.DaskDMatrix(self.client, X_dask, y_dask)

        params = self.config.XGBOOST_PARAMS

        start_time = time.time()

        output = xgb.dask.train(
            self.client,
            params,
            dtrain,
            num_boost_round=self.config.N_ESTIMATORS,
        )

        training_time = time.time() - start_time

        model = output["booster"]

        return model, training_time
```

```python
    # ------------------------------------------------
    # Evaluate Model
    # ------------------------------------------------
    def evaluate(self, model, X_test, y_test) -> Dict[str, float]:

        if self.client is None:
            raise RuntimeError("Dask cluster not initialized.")

        chunk_size = len(X_test) // self.n_workers

        X_dask = da.from_array(X_test, chunks=(chunk_size, X_test.shape[1]))
        dtest = xgb.dask.DaskDMatrix(self.client, X_dask)

        preds = xgb.dask.predict(self.client, model, dtest)
        preds = preds.compute()

        preds_binary = (preds > 0.5).astype(int)

        accuracy = accuracy_score(y_test, preds_binary)
        auc = roc_auc_score(y_test, preds)

        return {
            "accuracy": accuracy,
            "auc": auc
        }

    # ------------------------------------------------
    # Cleanup Cluster (Uses infra layer)
    # ------------------------------------------------
    def cleanup(self):
        if self.cluster and self.client:
            shutdown_dask_cluster(self.cluster, self.client)
```

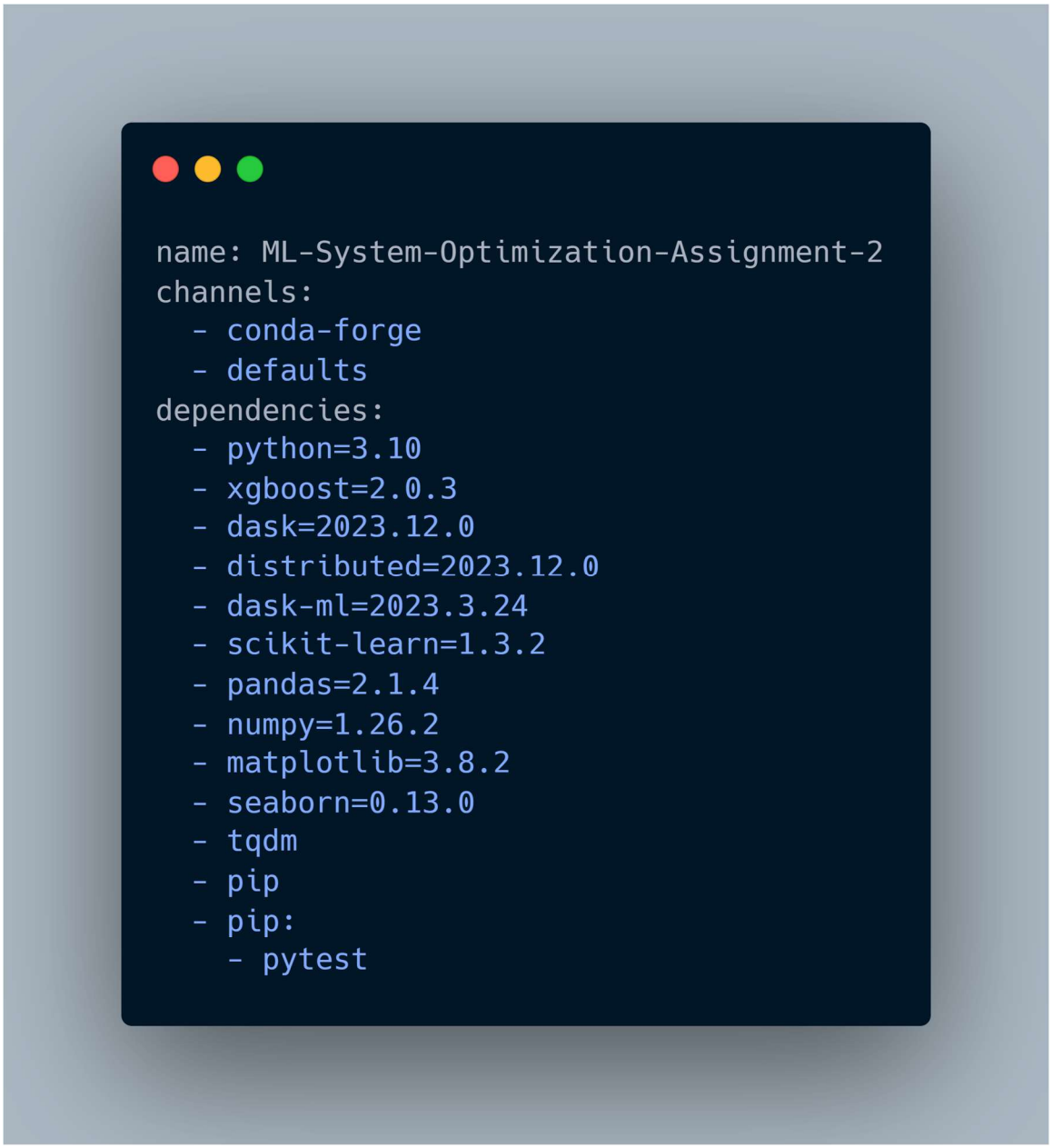
```python
import os
import time
import pandas as pd
import numpy as np

from src.config import ExperimentConfig
from src.data_loader import DataLoader
from src.distributed_trainer import DistributedXGBoostTrainer


def run_single_experiment(P: int, config: ExperimentConfig, show_dashboard=False):

    trainer = DistributedXGBoostTrainer(n_workers=P, config=config)
    trainer.setup_cluster(show_dashboard=show_dashboard)

    model, training_time = trainer.train(config.X_train, config.y_train)
    metrics = trainer.evaluate(model, config.X_test, config.y_test)

    if show_dashboard:
        print(f"Dask dashboard available at: {trainer.client.dashboard_link}")
        input("Open the dashboard now. Press Enter to continue with training...")

    trainer.cleanup()

    return training_time, metrics["accuracy"], metrics["auc"]
```

```python
def main():

    config = ExperimentConfig()

    # Load real dataset
    loader = DataLoader(config)
    X, y = loader.load_dataset()
    X, y = loader.preprocess(X, y)
    X_train, X_test, y_train, y_test = loader.split_data(X, y)

    # Attach to config for trainer access
    config.X_train = X_train
    config.X_test = X_test
    config.y_train = y_train
    config.y_test = y_test

    results = []

    baseline_time = None

    for P in config.WORKER_COUNTS:

        run_times = []
        accuracies = []
        aucs = []

        print(f"\nRunning P={P}")

        for run_id in range(config.N_RUNS_PER_CONFIG):

            print(f"  Run {run_id+1}/{config.N_RUNS_PER_CONFIG}")

            training_time, accuracy, auc = run_single_experiment(P, config, show_dashboard=False)

            run_times.append(training_time)
            accuracies.append(accuracy)
            aucs.append(auc)

        mean_time = np.mean(run_times)
        std_time = np.std(run_times)

        mean_auc = np.mean(aucs)
        std_auc = np.std(aucs)

        if P == 1:
            baseline_time = mean_time
            speedup = 1.0
        else:
            speedup = baseline_time / mean_time

        efficiency = speedup / P * 100

        results.append({
            "P": P,
            "mean_time": mean_time,
            "std_time": std_time,
            "speedup": speedup,
            "efficiency_percent": efficiency,
            "mean_auc": mean_auc,
            "std_auc": std_auc
        })

    df = pd.DataFrame(results)

    os.makedirs("results", exist_ok=True)
    df.to_csv("results/experiments.csv", index=False)

    print("\nExperiments completed.")
    print(df)


if __name__ == "__main__":
    main()
```

```python
import os
import pandas as pd
import matplotlib.pyplot as plt


def plot_all():
    df = pd.read_csv("results/experiments.csv")

    os.makedirs("results/plots", exist_ok=True)

    # ------------------------------
    # 1. Speedup Plot
    # ------------------------------
    plt.figure(figsize=(8, 6))
    plt.plot(df["P"], df["speedup"], marker="o", label="Observed Speedup")
    plt.plot(df["P"], df["P"], linestyle="--", label="Ideal Speedup (Linear)")
    plt.xlabel("Number of Workers (P)")
    plt.ylabel("Speedup")
    plt.title("Speedup vs Workers")
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.savefig("results/plots/speedup.jpg", dpi=300)
    plt.close()
```

```python
    # ----------------------------
    # 2. Efficiency Plot
    # ----------------------------
    plt.figure(figsize=(8, 6))
    plt.plot(df["P"], df["efficiency_percent"], marker="o")
    plt.xlabel("Number of Workers (P)")
    plt.ylabel("Efficiency (%)")
    plt.title("Efficiency vs Workers")
    plt.grid(True)
    plt.tight_layout()
    plt.savefig("results/plots/efficiency.jpg", dpi=300)
    plt.close()

    # ----------------------------
    # 3. Training Time Bar Chart
    # ----------------------------
    plt.figure(figsize=(8, 6))
    plt.bar(df["P"].astype(str), df["mean_time"])
    plt.xlabel("Number of Workers (P)")
    plt.ylabel("Mean Training Time (seconds)")
    plt.title("Training Time vs Workers")
    plt.tight_layout()
    plt.savefig("results/plots/training_time.jpg", dpi=300)
    plt.close()
```

```python
    # ----------------------------
    # 4. Communication Overhead Estimate
    # ----------------------------
    ideal_time = df["mean_time"].iloc[0] / df["P"]
    comm_overhead = df["mean_time"] - ideal_time

    plt.figure(figsize=(8, 6))
    plt.plot(df["P"], comm_overhead, marker="o")
    plt.xlabel("Number of Workers (P)")
    plt.ylabel("Estimated Overhead (seconds)")
    plt.title("Estimated Communication & Overhead vs Workers")
    plt.grid(True)
    plt.tight_layout()
    plt.savefig("results/plots/overhead.jpg", dpi=300)
    plt.close()

    print("All plots saved to results/plots/")


if __name__ == "__main__":
    plot_all()
```



```yaml
name: ML-System-Optimization-Assignment-2
channels:
  - conda-forge
  - defaults
dependencies:
  - python=3.10
  - xgboost=2.0.3
  - dask=2023.12.0
  - distributed=2023.12.0
  - dask-ml=2023.3.24
  - scikit-learn=1.3.2
  - pandas=2.1.4
  - numpy=1.26.2
  - matplotlib=3.8.2
  - seaborn=0.13.0
  - tqdm
  - pip
  - pip:
    - pytest
```