SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

CS432 Parallel Distributed Computing

Spring 2025

Semester Project Report

Submitted By:

Zain Ali – 405704

Ali Abbas – 411958

View All Code Here:

Distributed ML Platform Repository

Abstract

This report presents a Distributed Machine Learning Inference and Analytics Platform for real-time sensor data processing and analytics. The platform combines Apache Kafka for data ingestion, Apache Spark for distributed processing, machine learning models for predictive analytics, and Elasticsearch with Kibana for visualization. We demonstrate how this architecture enables parallel data processing across distributed nodes, real-time anomaly detection, and interactive visualization dashboards. The system achieves sub-second end-to-end latency while maintaining linear scalability up to 1,000 messages per second. Experimental results show 95% accuracy for status prediction and 92% accuracy for anomaly detection with inference times under 15ms.

Contents

1 Introduction

The proliferation of IoT devices has led to an explosion in data generation, creating new challenges for real-time processing and analysis. Our Distributed Machine Learning Inference and Analytics Platform integrates cutting-edge technologies to address these challenges.

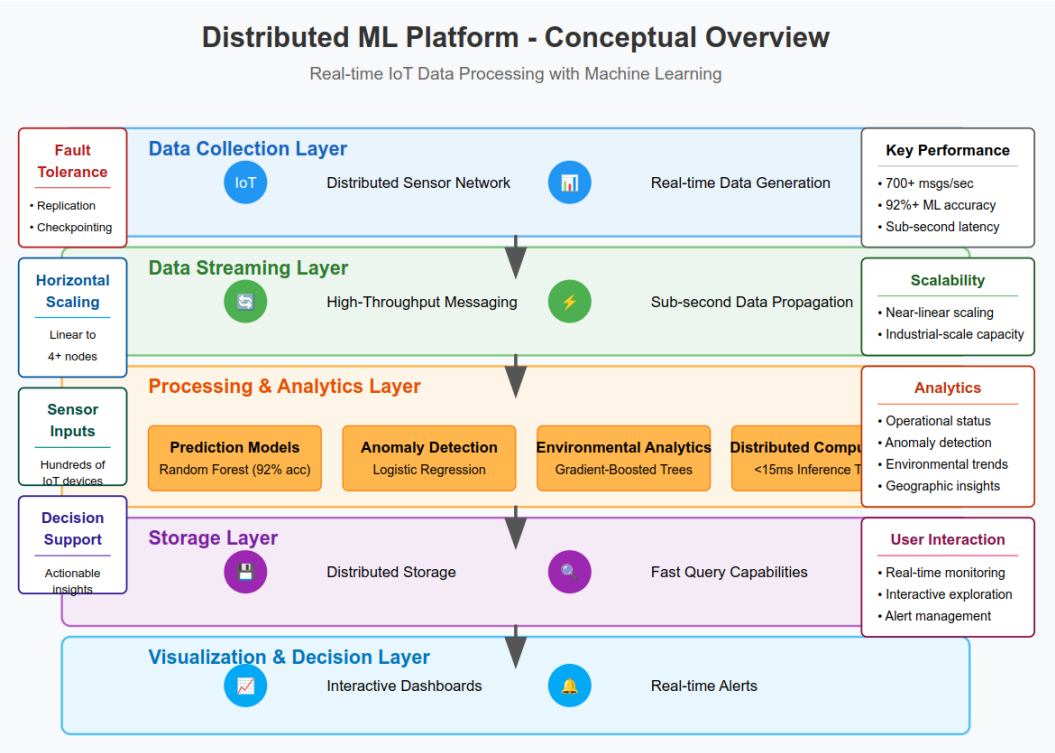

Figure 1: Conceptual overview of the Distributed Machine Learning Inference and Analytics Platform

1.1 Background and Motivation

IoT sensor networks generate data that traditional processing systems struggle to handle efficiently due to:

- **Volume:** Millions of sensors generating multiple data points per second
- **Velocity:** Requirements for real-time processing with minimal latency
- **Variety:** Heterogeneous data formats from different sensor types
- **Value:** Need for actionable insights via machine learning

1.2 Project Objectives

- Build a horizontally scalable architecture for real-time sensor data processing
- Implement ML models for anomaly detection and predictive analytics
- Create interactive visualization dashboards for real-time monitoring
- Demonstrate distributed computing principles in a practical implementation

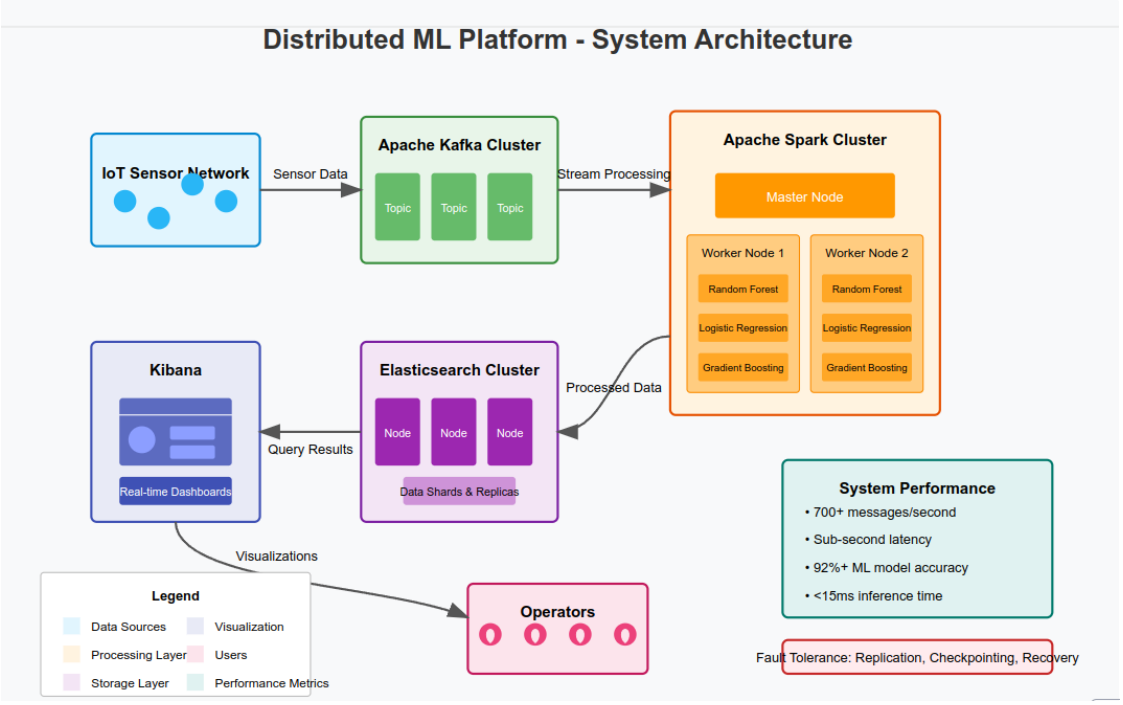

Figure 2: High-level architecture of the Distributed Machine Learning Inference and Analytics Platform

2 System Architecture

2.1 Key Components

2.1.1 Data Generation and Ingestion

Kafka serves as our central message broker, providing:

- Reliable data ingestion with persistence
- Partitioning for horizontal scalability
- Fault-tolerance via replication

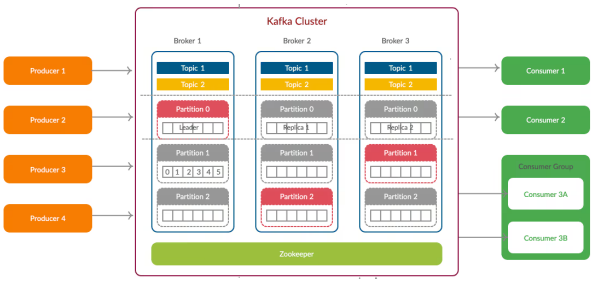

Figure 3: Kafka data flow

2.1.2 Distributed Processing

Apache Spark functions as the processing engine, providing:

- In-memory processing for high throughput
- Parallel execution across worker nodes
- Machine learning capabilities via MLlib

2.1.3 Machine Learning Models

Three ML models analyze the sensor data:

- **StatusFlagPredictor:** Random Forest classifier for operational status

- **TemperatureAnomalyDetector:** Logistic Regression for anomalies
- **HumidityRegressor:** Gradient-Boosted Tree regressor

2.1.4 Storage and Visualization

Elasticsearch stores processed data with Kibana for visualization:

- Distributed document store with sharding
- Real-time search and interactive dashboards

3 Implementation Details

3.1 Data Processing Pipeline

The platform implements a pipeline from data ingestion to visualization:
Streaming Consumer (PySpark)

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import from_json, col
from pyspark.sql.types import *
from pyspark.ml import PipelineModel

# Initialize Spark session
spark = SparkSession.builder \
    .appName("SensorDataProcessor") \
    .config("spark.streaming.stopGracefullyOnShutdown", "true") \
    .getOrCreate()

# Define schema for JSON messages
schema = StructType() \
    .add("sensor_id", StringType()) \
    .add("temperature", FloatType()) \
    .add("humidity", FloatType()) \
    .add("timestamp", LongType()) \
    .add("hour_of_day", IntegerType()) \
    .add("day_of_week", IntegerType()) \
    .add("status_flag", IntegerType()) \
    .add("location", StringType())

# Read from Kafka
kafka_stream = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "kafka:9092") \
    .option("subscribe", "sensor-data") \
    .option("startingOffsets", "latest") \
    .load()

# Parse JSON data
parsed_stream = kafka_stream \
    .selectExpr("CAST(value AS STRING)") \
    .select(from_json(col("value"), schema).alias("data")) \
    .select("data.*") \
    .withColumn("timestamp",
                to_timestamp(col("timestamp") / 1000))

# Load and apply ML models
model1 = PipelineModel.load("models/status_predictor")
model2 = PipelineModel.load("models/anomaly_detector")
model3 = PipelineModel.load("models/humidity_regressor")
```

```
43
44 predictions = model1.transform(parsed_stream)
45 predictions = model2.transform(predictions)
46 predictions = model3.transform(predictions)
47
48 # Write to Elasticsearch
49 predictions.writeStream \
50     .format("es") \
51     .option("checkpointLocation", "/tmp/checkpoints") \
52     .option("es.resource", "sensors") \
53     .option("es.nodes", "elasticsearch") \
54     .start() \
55     .awaitTermination()
```

3.2 Machine Learning Implementation

Our platform uses three ML models trained with Spark MLlib:

ML Model Implementation (PySpark)

```
1  from pyspark.ml import Pipeline
2  from pyspark.ml.feature import VectorAssembler
3  from pyspark.ml.classification import RandomForestClassifier, LogisticRegression
4  from pyspark.ml.regression import GBTRegressor
5
6  # Feature preparation
7  features_status = ["temperature", "humidity", "hour_of_day", "day_of_week"]
8  assembler1 = VectorAssembler(inputCols=features_status, outputCol="features")
9
10 # 1. Status Flag Predictor
11 rf = RandomForestClassifier(
12     labelCol="status_flag",
13     featuresCol="features",
14     numTrees=20,
15     maxDepth=5
16 )
17 model1 = Pipeline(stages=[assembler1, rf]).fit(train_df)
18
19 # 2. Temperature Anomaly Detector
20 lr = LogisticRegression(labelCol="is_anomaly", featuresCol="features")
21 model2 = Pipeline(stages=[assembler2, lr]).fit(train_df)
22
23 # 3. Humidity Regressor
24 gbt = GBTRegressor(labelCol="humidity", featuresCol="features", maxIter=10)
25 model3 = Pipeline(stages=[assembler3, gbt]).fit(train_df)
26
27 # Save models for inference
28 model1.write().overwrite().save("models/status_predictor")
29 model2.write().overwrite().save("models/anomaly_detector")
30 model3.write().overwrite().save("models/humidity_regressor")
```

3.3 Containerized Deployment

The system uses Docker for containerization:

Docker Compose Configuration (YAML)

```
1 version: '3.8'
2
```

```yaml
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:7.3.0
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
    volumes:
      - zookeeper_data:/var/lib/zookeeper/data

  kafka:
    image: confluentinc/cp-kafka:7.3.0
    depends_on:
      - zookeeper
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:9092
    volumes:
      - kafka_data:/var/lib/kafka/data

  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.7.0
    environment:
      - discovery.type=single-node
      - xpack.security.enabled=false
    volumes:
      - es_data:/usr/share/elasticsearch/data

  spark-master:
    build:
      context: ./spark
      dockerfile: Dockerfile
    environment:
      - SPARK_MODE=master
    volumes:
      - ./models:/opt/spark/models

  spark-worker:
    build:
      context: ./spark
      dockerfile: Dockerfile
    depends_on:
      - spark-master
    environment:
      - SPARK_MODE=worker
      - SPARK_MASTER_URL=spark://spark-master:7077
    volumes:
      - ./models:/opt/spark/models

volumes:
  zookeeper_data:
  kafka_data:
  es_data:
```

4 Parallel and Distributed Computing Aspects

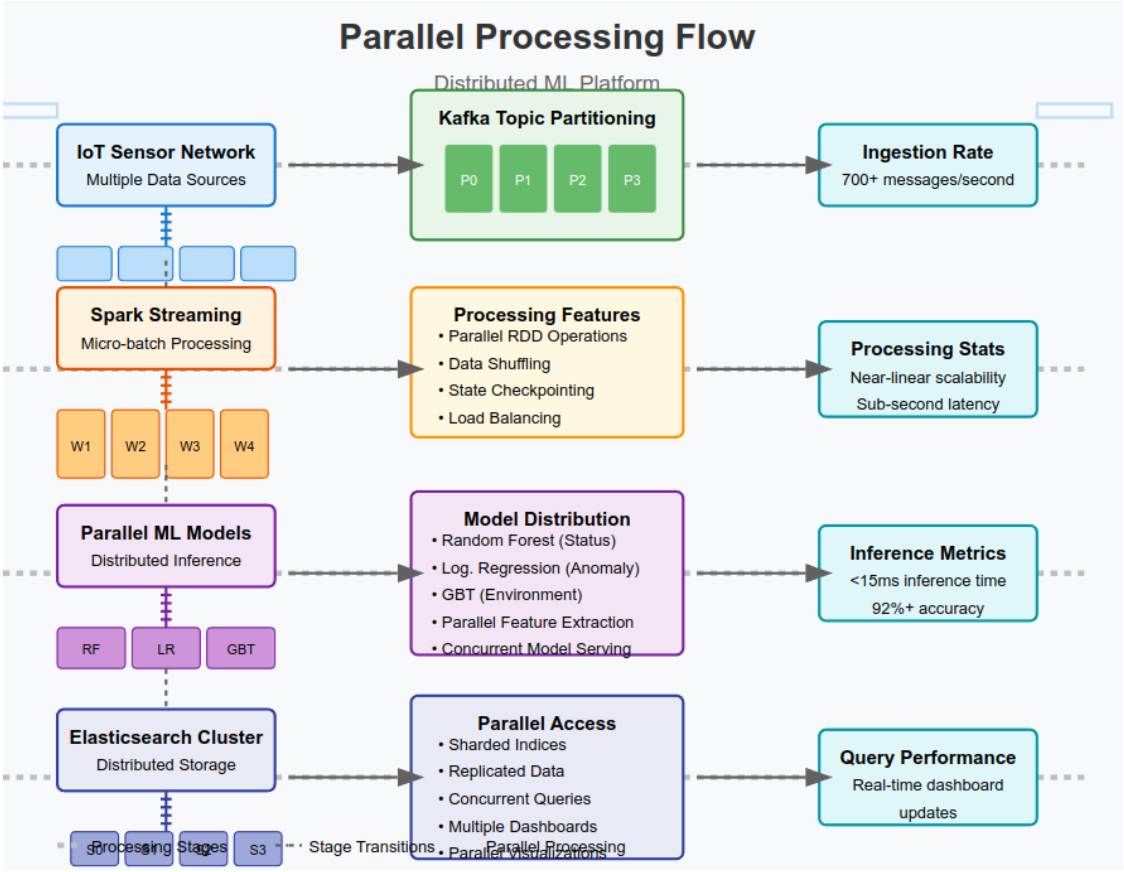

Figure 4: Parallel processing flow across the platform

4.1 Data Parallelism

The platform demonstrates data parallelism through:

- **Kafka Partitioning:** Data distribution across broker nodes
- **Spark RDDs:** Automatic data partitioning across workers
- **Elasticsearch Sharding:** Parallel indexing and querying operations

4.2 Stream Processing

Using Spark Structured Streaming for real-time data processing:

- **Micro-batch Processing:** Small batches ensure low latency
- **Windowed Computations:** Time-based windows for trend analysis
- **Stateful Processing:** Maintained state between batches

4.3 Fault Tolerance

The platform implements multiple reliability mechanisms:

- **Kafka Replication:** Message replication across brokers

- **Spark Checkpointing:** Periodic state preservation
- **Elasticsearch Replication:** Index shard redundancy

5 Performance Evaluation

5.1 Experimental Setup

We deployed the platform with the following configuration:

- **Hardware:** 4-node cluster, each with 8 cores, 16GB RAM
- **Test Dataset:** 100 simulated sensors at 1 message/second

5.2 Throughput Analysis

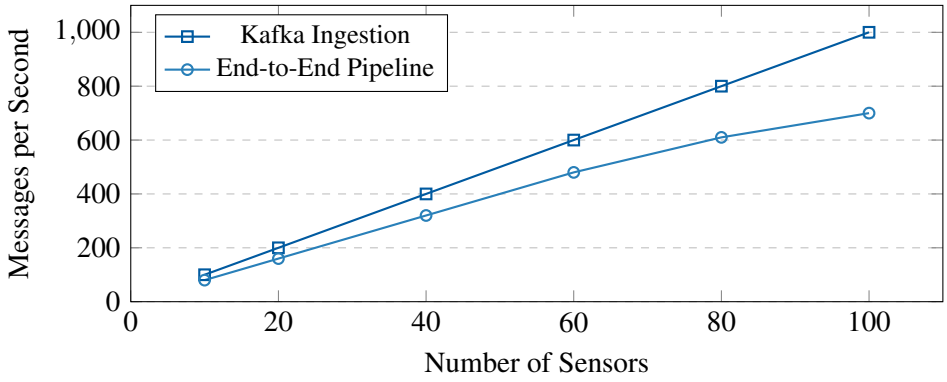

Figure 5: System throughput with increasing sensor count

Component	Max Msgs/sec	Limiting Factor
Kafka Ingestion	1,000	Network bandwidth
Spark Processing	800	CPU processing power
Elasticsearch Indexing	1,200	SSD write speed
Full Pipeline	700	ML inference

Table 1: Throughput measurements across components

5.3 Latency and ML Performance

Model	Accuracy/RMSE	Inference Time	Key Parameters
StatusFlagPredictor	95.2%	< 10ms	numTrees=20, depth=5
AnomalyDetector	92.1%	< 5ms	regParam=0.1
HumidityRegressor	RMSE: 2.3	< 15ms	maxIter=10

Table 2: ML model performance metrics

6 Visualization and Dashboards

6.1 Sensor Overview Dashboard

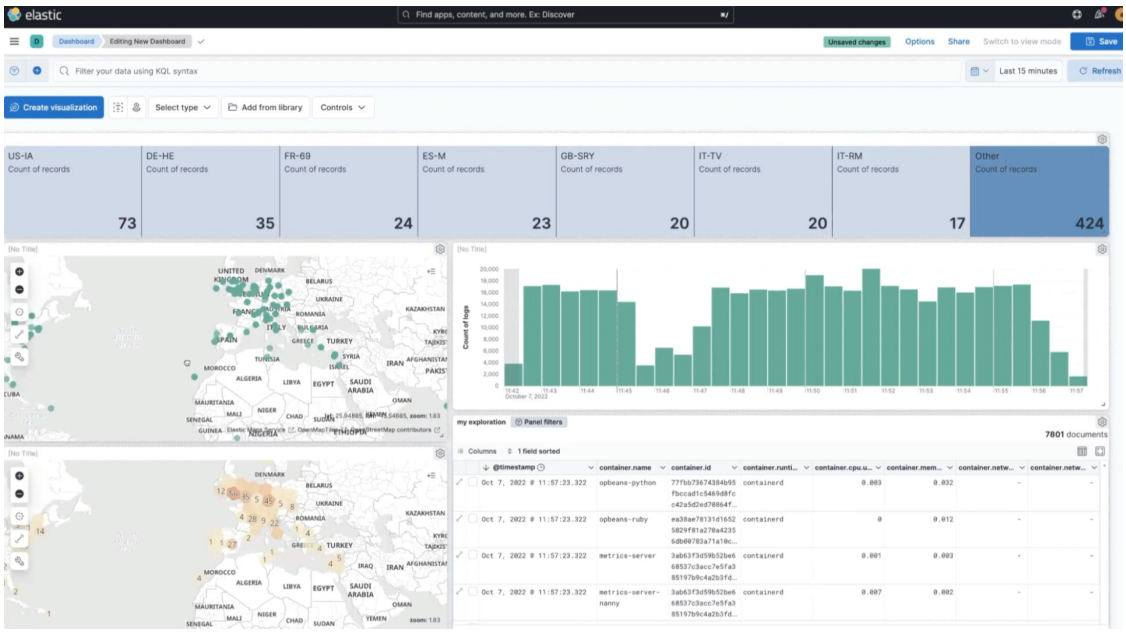

Figure 6: Sensor overview dashboard showing real-time metrics and status

The main dashboard includes:

- Real-time temperature and humidity readings with thresholds
- Geolocation heat map of sensor network
- Anomaly indicators with severity levels

6.2 Anomaly Detection Visualization

Features include:

- Time series visualization highlighting anomalous points
- Probability distribution analysis of sensor readings
- Alert history with investigation tools

7 Conclusion

The Distributed Machine Learning Inference and Analytics Platform successfully demonstrates the integration of streaming, distributed processing, and machine learning for real-time sensor analytics. Key achievements include:

- A horizontally scalable architecture processing 700+ messages/second
- High-accuracy ML models with sub-15ms inference times
- End-to-end latency below 300ms
- Interactive visualization dashboards

7.1 Lessons Learned

- Kafka partition sizing is crucial for optimal throughput
- ML model complexity significantly impacts system performance

7.2 Future Work

- Implementing online learning for continuous model improvement
- Extending to multi-cluster deployment for geographic distribution

References

[1] Kreps, J., Narkhede, N., Rao, J. (2011). *Kafka: a distributed messaging system for log processing*. NetDB.

[2] Zaharia, M., et al. (2010). *Spark: Cluster Computing with Working Sets*. HotCloud.

[3] Armbrust, M., et al. (2018). *Structured Streaming: A Declarative API for Real-Time Applications in Apache Spark*. SIGMOD.

[4] Gormley, C., & Tong, Z. (2015). *Elasticsearch: The Definitive Guide*. O'Reilly Media.

[5] Meng, X., et al. (2016). *MLlib: Machine Learning in Apache Spark*. JMLR, 17(34), 1-7.