Distributed System – Lab5 Report

Une image contenant Bleu électrique, bleu, logo, Graphique

Description générée automatiquement

Title: Distributed System Lab5 Report

Course: Distributed System CO3072

Professor: Dr. Thoai Nam

Authors:

Julien Holzer, 2360059

Lê Tến Phát, 2252600

Group: No12

Institution: Ho Chi Minh City University of Technology

Faculty: Computer Science and Engineering

Date: 05 January 2025

Project subject: HDFS

Table of contents

[Time Constraint Notice 3](#_Toc187240104)

[Technical Setup Requirements 3](#_Toc187240105)

[**Infrastructure Prerequisites** 3](#_Toc187240106)

[Implementation Details 4](#_Toc187240107)

[**Strategy 1: "Bring Compute to Data" Implementation** 4](#_Toc187240108)

[**Strategy 2: "Bring Data to Compute" Implementation** 5](#_Toc187240109)

[Data Processing Implementation 6](#_Toc187240110)

[Performance Measurement Implementation 7](#_Toc187240111)

[Expected Implementation Challenges 7](#_Toc187240112)

[Testing Strategy 7](#_Toc187240113)

[Verification Methods 8](#_Toc187240114)

[Future Improvements 8](#_Toc187240115)

# Time Constraint Notice

Due to time constraints, this report details the theoretical implementation approach rather than actual execution results. The following sections outline the complete technical setup and implementation process that would be required to conduct this laboratory experiment.

# Technical Setup Requirements

**Infrastructure Prerequisites**

To properly implement this lab, the following environment would need to be configured:

1. Hadoop Cluster Setup A minimal Hadoop cluster configuration would require:
   * 1 NameNode for metadata management
   * At least 3 DataNodes for distributed storage
   * HDFS properly configured and running
   * Minimum 16GB RAM per node
   * Sufficient storage capacity for environmental data (estimated 500GB+)
2. Network Configuration The network should be configured with:
   * Minimum 1Gbps network connectivity between nodes
   * Properly configured hostname resolution
   * Firewall rules allowing HDFS and RPC communication

# Implementation Details

**Strategy 1: "Bring Compute to Data" Implementation**

The distributed processing approach would be implemented in Java, utilizing Hadoop's native libraries. Here's the detailed technical approach:

public class DistributedEnvironmentalAnalysis {

private Configuration hadoopConfig;

private FileSystem fileSystem;

*// Initialize HDFS client configuration*

private void initializeHDFS() {

hadoopConfig = new Configuration();

hadoopConfig.set("fs.defaultFS", "hdfs://namenode:9000");

fileSystem = FileSystem.get(hadoopConfig);

}

*// Get block locations from NameNode*

private BlockLocation[] getBlockLocations(String filePath) {

LocatedFileStatus fileStatus = fileSystem.getFileStatus(new Path(filePath));

return fileStatus.getBlockLocations();

}

*// Execute RPC calls to DataNodes*

private void executeDistributedComputation(BlockLocation[] blocks) {

ExecutorService executor = Executors.newFixedThreadPool(blocks.length);

List<Future<ComputationResult>> results = new ArrayList<>();

for (BlockLocation block : blocks) {

results.add(executor.submit(new DataNodeProcessor(block)));

}

*// Aggregate results*

aggregateResults(results);

}

}

**Strategy 2: "Bring Data to Compute" Implementation**

The traditional approach would be implemented using the following structure:

public class CentralizedEnvironmentalAnalysis {

private Configuration hadoopConfig;

private FileSystem fileSystem;

*// Read complete file from HDFS*

private void readCompleteFile(String filePath) {

FSDataInputStream inputStream = fileSystem.open(new Path(filePath));

BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));

*// Read and process data centrally*

processEnvironmentalData(reader);

}

*// Central data processing*

private void processEnvironmentalData(BufferedReader reader) {

ExecutorService executor = Executors.newFixedThreadPool(THREAD\_COUNT);

List<Future<AnalysisResult>> results = new ArrayList<>();

*// Process chunks of data in memory*

while (reader.ready()) {

String chunk = readChunk(reader);

results.add(executor.submit(new DataProcessor(chunk)));

}

*// Aggregate results*

aggregateResults(results);

}

}

# Data Processing Implementation

The environmental data analysis would require implementations for:

1. Temperature Analysis Algorithm:

public class TemperatureAnalyzer {

private static final double MIN\_TEMP = 23.0;

private static final double MAX\_TEMP = 28.0;

public AnalysisResult analyzeTemperature(List<Double> temperatures) {

double outOfRangeTime = 0;

for (Double temp : temperatures) {

if (temp < MIN\_TEMP || temp > MAX\_TEMP) {

outOfRangeTime += MEASUREMENT\_INTERVAL;

}

}

return new AnalysisResult(outOfRangeTime);

}

}

1. Soil pH Analysis:

public class SoilAnalyzer {

private static final double MIN\_PH = 5.5;

private static final double MAX\_PH = 7.0;

public AnalysisResult analyzeSoilPH(List<Double> phReadings) {

double maxPH = Collections.max(phReadings);

double minPH = Collections.min(phReadings);

double outOfRangeTime = calculateOutOfRangeTime(phReadings);

return new AnalysisResult(minPH, maxPH, outOfRangeTime);

}

}

# Performance Measurement Implementation

To compare the strategies fairly, we would implement performance monitoring:

public class PerformanceMonitor {

private long startTime;

private Map<String, Long> metrics;

public void startMeasurement() {

startTime = System.currentTimeMillis();

}

public void recordMetric(String metricName, long value) {

metrics.put(metricName, value);

}

public PerformanceReport generateReport() {

long totalTime = System.currentTimeMillis() - startTime;

return new PerformanceReport(totalTime, metrics);

}

}

# Expected Implementation Challenges

1. Data Synchronization
   * Implementing proper synchronization mechanisms for distributed processing
   * Handling potential data inconsistencies across nodes
2. Error Handling
   * Implementing robust error handling for network failures
   * Managing node failures during processing
   * Handling data corruption scenarios
3. Performance Optimization
   * Tuning thread pool sizes for optimal performance
   * Optimizing memory usage for large datasets
   * Minimizing network overhead in distributed processing

# Testing Strategy

The implementation would require comprehensive testing:

1. Unit Tests
   * Individual component testing for data processors
   * Validation of analysis algorithms
   * Error handling verification
2. Integration Tests
   * End-to-end workflow testing
   * Network failure scenario testing
   * Performance benchmark testing
3. Load Tests
   * Large dataset processing tests
   * Concurrent processing tests
   * Network bandwidth utilization tests

# Verification Methods

To verify the implementation:

1. Data Accuracy
   * Cross-validation of results between both strategies
   * Statistical analysis of processed data
   * Verification against known sample datasets
2. Performance Metrics
   * Processing time measurements
   * Network utilization monitoring
   * Resource usage tracking

# Future Improvements

The implementation could be enhanced with:

1. Real-time Processing Capabilities
   * Stream processing implementation
   * Real-time alerting system
   * Dynamic scaling capabilities
2. Advanced Analytics
   * Machine learning integration
   * Predictive analytics
   * Trend analysis capabilities

This technical guide provides a comprehensive overview of how the lab should be implemented. While actual implementation was not possible due to time constraints, this document serves as a detailed blueprint for future execution of the experiment.