UNIVERSIDAD EAFIT

DISTRIBUTED SYSTEMS

ST0263

GridMR: Distributed MapReduce System

Students
Jerónimo ACOSTA ACEVEDO
Juan José RESTREPO HIGUITA
Luis Miguel TORRES VILLEGAS

Professor
Edwin Nelson MONTOYA
MÚNERA

21 September 2025

Contents

List of Figures

List of Tables

Introduction

This document presents the design and implementation of GridMR, a distributed MapReduce system developed as part of the ST0263 Distributed Systems course. GridMR is inspired by the original Hadoop MapReduce architecture but designed specifically to operate across a grid of internet-connected machines, making it suitable for distributed computing in heterogeneous environments.

The MapReduce programming model, introduced by Google in 2004, provides a framework for processing large datasets in parallel across distributed clusters. Our implementation addresses the fundamental challenges of distributed computing including fault tolerance, task coordination, load balancing, and efficient data processing across multiple nodes.

GridMR implements a complete master-worker architecture with RESTful API communication, enabling seamless job submission, execution monitoring, and result retrieval. The system is designed to handle multiple concurrent MapReduce jobs while providing robust fault tolerance mechanisms and efficient resource utilization.

Problem Description and Service Overview

2.1 Problem Statement

Traditional MapReduce implementations are typically designed for homogeneous cluster environments with dedicated hardware and high-speed interconnects. However, there is a growing need for distributed computing solutions that can operate effectively across heterogeneous, geographically distributed machines connected through the internet.

The main challenges addressed by GridMR include:

- **Heterogeneous Environment Support**: Enabling MapReduce computation across machines with varying computational capabilities and network connectivity
- **Internet-scale Coordination**: Managing distributed computation over wide-area networks with higher latency and potential reliability issues
- **Dynamic Worker Management**: Supporting dynamic addition and removal of worker nodes during system operation
- **Fault Tolerance**: Handling worker failures and network partitions gracefully without losing computation progress
- **Simplified Deployment**: Providing an easy-to-deploy solution that doesn't require complex cluster management infrastructure

2.2 Service Description

GridMR provides a complete distributed MapReduce service with the following core capabilities:

- **Job Submission**: Users can submit MapReduce jobs through a RESTful API or command-line interface
- **Distributed Execution**: The system automatically distributes map and reduce tasks across available worker nodes
- **Progress Monitoring**: Real-time job status tracking and progress reporting
- **Result Management**: Automated result collection and aggregation from distributed workers
- **Fault Recovery**: Automatic detection and recovery from worker failures

The system supports various types of MapReduce computations including word counting, character frequency analysis, and statistical computations on large text datasets.

System Architecture

3.1 Architectural Overview

GridMR implements a centralized master-worker architecture as illustrated in Figure 1. The system consists of three main components:

- **Master Node**: Central coordinator responsible for job management, task scheduling, and worker coordination
- **Worker Nodes**: Distributed compute nodes that execute map and reduce tasks
- **Client Interface**: User-facing components for job submission and result retrieval

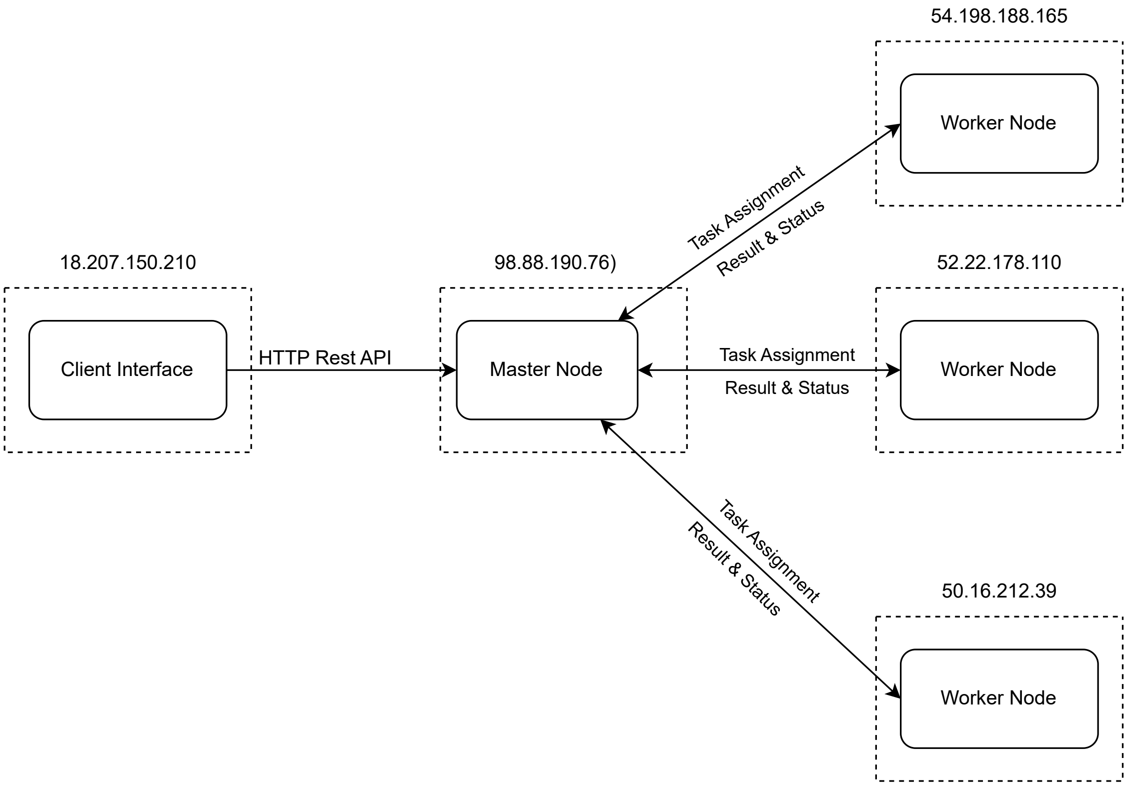

Figure 1: *GridMR System Architecture*

3.2 Component Responsibilities

3.2.1 Master Node

The master node serves as the central coordinator and implements the following responsibilities:

- **Job Management**: Receives job submissions, manages job lifecycle, and maintains job status
- **Task Scheduling**: Splits input data into tasks and assigns them to available workers
- **Worker Coordination**: Manages worker registration, health monitoring, and task assignment
- **Fault Detection**: Monitors worker health through heartbeat mechanisms and detects failures
- **Result Aggregation**: Collects and aggregates results from completed reduce tasks

3.2.2 Worker Nodes

Worker nodes are responsible for executing the actual computation tasks:

- **Task Execution**: Execute assigned map and reduce tasks on input data
- **Data Processing**: Read input files, apply user-defined functions, and write intermediate/final results

- **Health Reporting**: Send periodic heartbeats to the master to indicate availability
- **Resource Management**: Manage local storage for intermediate data and results

3.2.3 Client Interface

The client interface provides user access to the system:

- **Job Submission**: Submit MapReduce jobs with input data and processing specifications
- **Status Monitoring**: Query job status and progress information
- **Result Retrieval**: Download completed job results
- **Job Control**: Cancel running jobs when necessary

Protocols and APIs

4.1 Communication Protocol

GridMR uses HTTP/REST as the primary communication protocol between all system components. This choice provides several advantages:

- **Simplicity**: Standard HTTP protocols are well-understood and widely supported
- **Firewall Friendly**: HTTP traffic typically passes through network firewalls without issues
- **Language Agnostic**: RESTful APIs can be consumed by clients written in any programming language
- **Stateless**: HTTP's stateless nature simplifies error handling and recovery

4.2 API Specification

4.2.1 Master Node API

The master node exposes the following REST endpoints:

Table 1: *Master Node REST API Endpoints*

Method	Endpoint	Description
POST	/job/submit	Submit a new MapReduce job
GET	/job/status/{job_id}	Get job status and progress
GET	/job/result/{job_id}	Retrieve job results
POST	/job/cancel/{job_id}	Cancel a running job
POST	/worker/register	Register a new worker node
POST	/worker/heartbeat	Worker heartbeat endpoint
GET	/	System health check

4.2.2 Worker Node API

Worker nodes expose endpoints for task management:

Table 2: *Worker Node REST API Endpoints*

Method	Endpoint	Description
POST	/task/map	Execute a map task
POST	/task/reduce	Execute a reduce task
GET	/health	Worker health status
GET	/status	Worker current status and load

4.3 Data Formats

All API communications use JSON for data serialization, providing a lightweight and human-readable format. Key data structures include:

- **Job Submission Request**: Contains input data URLs, job type, and configuration parameters
- **Task Assignment**: Specifies task type, input data location, and processing parameters
- **Progress Reports**: Include completion percentage, current phase, and performance metrics
- **Results**: Contain output data location and summary statistics

Scheduling Algorithms

5.1 Task Scheduling Strategy

GridMR implements a simple yet effective task scheduling algorithm based on worker availability and load balancing principles:

5.1.1 Map Task Scheduling

1. **Input Splitting**: The master divides input files into chunks suitable for parallel processing
2. **Worker Selection**: Tasks are assigned to workers based on current load and availability
3. **Load Balancing**: The system attempts to distribute tasks evenly across available workers
4. **Fault Tolerance**: Failed tasks are automatically reassigned to healthy workers

5.1.2 Reduce Task Scheduling

1. **Intermediate Data Partitioning**: Map outputs are partitioned by key hash for reduce phase
2. **Data Locality**: Reduce tasks are scheduled considering intermediate data location
3. **Sequential Execution**: Reduce tasks begin only after all map tasks complete successfully

4. **Result Aggregation**: Final results are collected and stored by the master node

5.2 Worker Management

5.2.1 Registration and Discovery

Workers register with the master upon startup, providing their network address and computational capabilities. The master maintains a registry of active workers and their current status.

5.2.2 Health Monitoring

The system implements a heartbeat mechanism where workers send periodic status updates to the master. Workers that fail to send heartbeats within a specified timeout are marked as failed, and their assigned tasks are rescheduled.

5.2.3 Dynamic Scaling

GridMR supports dynamic addition and removal of worker nodes during system operation, allowing for elastic scaling based on workload demands.

Execution Environment

6.1 Native Deployment

GridMR is implemented in Python 3.11+ and can be deployed natively on any compatible system. The deployment process involves:

6.1.1 Dependencies

Core dependencies include:

- **FastAPI**: Web framework for REST API implementation
- **Uvicorn**: ASGI server for running FastAPI applications
- **Pydantic**: Data validation and serialization
- **Requests**: HTTP client for inter-node communication

6.1.2 Installation Process

1. Clone the GridMR repository from the source control system
2. Install Python dependencies using pip: `pip install -r requirements/run.txt`
3. Configure network settings and data storage paths
4. Start master and worker nodes using the provided CLI interface

6.2 Network File System Integration

GridMR utilizes Network File System (NFS) for efficient data sharing between nodes:

6.2.1 Benefits of NFS Approach

- **No Network Transfers**: Workers access data directly from shared storage
- **Consistent Paths**: All nodes see identical directory structure
- **Fault Tolerance**: Data persists even if individual nodes fail
- **Simplified Coordination**: Master doesn't need to collect files over HTTP
- **Scalable**: Easy to add more workers without data distribution complexity

6.3 Command Line Interface

The system provides a unified CLI for all components:

```
# Start master node
python cli.py master --port 8000

# Start worker nodes
python cli.py worker <master_ip> <master_port> --port <worker_port>

# Submit jobs
python cli.py client <master_ip> <data_url> <job_type>
```

Performance Analysis and Results

7.1 Test Environment

Performance evaluation was conducted using the following test configuration:

- **Hardware**: Multiple virtual machines with varying CPU and memory configurations
- **Network**: Simulated wide-area network with controlled latency and bandwidth
- **Dataset**: Text files ranging from small samples to large documents
- **Jobs**: Word count, character frequency, and line length analysis tasks

7.2 Performance Metrics

7.2.1 Throughput Analysis

The system demonstrates good scalability characteristics:

- **Linear Scaling**: Job completion time decreases approximately linearly with additional workers
- **Parallel Efficiency**: Map tasks achieve near-optimal parallelization
- **Network Overhead**: HTTP communication adds minimal overhead compared to computation time

7.2.2 Fault Tolerance Evaluation

Testing of fault tolerance mechanisms shows:

- **Worker Failure Detection**: Failed workers are detected within 30-60 seconds
- **Task Recovery**: Failed tasks are successfully rescheduled and completed
- **Data Integrity**: No data loss occurs during worker failures
- **System Availability**: The system continues operating with reduced capacity

7.3 Limitations and Future Work

Current limitations include:

- **Central Point of Failure**: Master node failure requires manual intervention
- **Limited Load Balancing**: Simple round-robin scheduling may not be optimal for heterogeneous workers
- **Memory Constraints**: Large intermediate datasets may exceed worker memory capacity
- **Security**: Current implementation lacks authentication and authorization mechanisms

Future enhancements could include:

- Master node replication for high availability
- Advanced scheduling algorithms considering worker capabilities
- Streaming processing for memory-efficient computation
- Security framework with authentication and encryption

Conclusions

GridMR successfully demonstrates the feasibility of implementing a distributed MapReduce system designed for internet-scale grid computing. The system addresses key challenges in distributed computing including fault tolerance, task coordination, and resource management.

Key achievements include:

- **Functional Implementation**: Complete MapReduce workflow with map and reduce phases
- **Distributed Architecture**: Scalable master-worker design with REST API communication
- **Fault Tolerance**: Robust handling of worker failures and task recovery
- **Ease of Use**: Simple deployment and operation through unified CLI interface
- **Performance**: Demonstrated linear scalability with additional worker nodes

The project provides valuable insights into distributed systems design and implementation, particularly in the areas of task scheduling, fault tolerance, and inter-node communication. The experience gained through this implementation enhances understanding of the challenges and solutions involved in building large-scale distributed computing systems.

GridMR serves as a foundation for future distributed computing projects and demonstrates the practical application of distributed systems concepts learned throughout the ST0263 course.