Project Proposal: NebulaGrid

NebulaGrid: A Browser-Based Distributed Computing System for Parallel AI Processing

Course:
Parallel & Distributed Computing

Team Members:

Muhammad Junaid	SP23-BCS-091
Sajjad Afzaal	SP23-BCS-115
Waleed Waris	SP23-BCS-125

December 11, 2025

1. Introduction

In the era of Big Data and Artificial Intelligence, the need for computational power often exceeds the capabilities of a single machine. Traditional high-performance computing (HPC) solutions, such as Beowulf clusters or cloud-based Kubernetes setups, require complex configuration, static IP addresses, and significant technical expertise (SSH keys, firewall configurations, etc.).

NebulaGrid is a lightweight, zero-setup distributed computing framework that utilizes the ubiquity of modern web browsers to create an ad-hoc computing cluster. By leveraging **WebRTC** (Web Real-Time Communication), NebulaGrid allows disparate devices—laptops, smartphones, and desktops—to join a network instantly and contribute to a shared parallel processing task.

2. Problem Statement

Students and researchers often face barriers when attempting to implement distributed systems:

- **High Barrier to Entry:** Setting up MPI (Message Passing Interface) or Hadoop clusters requires OS-level configuration.
- **Resource Inaccessibility:** Utilizing idle computing power across different devices (e.g., a phone and a laptop) is difficult due to network NAT/firewall issues.
- **Visualization Gap:** It is often difficult to visualize how data is partitioned and processed in real-time in a command-line environment.

NebulaGrid solves these problems by moving the distributed environment entirely into the browser, making it platform-agnostic and instantly accessible.

3. Project Objectives

The primary objectives of this semester project are:

1. **To demonstrate the Master-Slave Architecture:** Implement a central Host node that manages state and Worker nodes that execute tasks.
2. **To implement Data Parallelism:** Show how a large dataset (text narrative) can be partitioned into smaller chunks for simultaneous processing.
3. **To utilize Peer-to-Peer Networking:** Establish direct data channels between browsers without an intermediary processing server.
4. **To perform Distributed AI Analysis:** Use the cluster to perform parallel Sentiment Analysis and Entity Extraction on large text datasets.

4. System Architecture

The system follows a **Star Topology** using a Master-Slave paradigm:

4.1. The Host (Master Node)

Responsibility: Job creation, data partitioning, task scheduling, and result aggregation.
Logic:

- Accepts a large text input.
- Splits input into chunks (Data Partitioning).
- Maintains a queue of tasks.
- Assigns tasks to connected Workers via a Round-Robin or On-Demand mechanism.
- Aggregates results into a final report.

4.2. The Workers (Slave Nodes)

Responsibility: Execution of assigned tasks.
Logic:

- Connects to the Host via a unique Peer ID.
- Receives a data chunk and an instruction set.
- Processes the data (using Google Gemini API).
- Returns the result (Sentiment/Keywords) to the Host.
- Signals ”Idle” status to request more work.

5. Technology Stack

- **Frontend Framework:** React (TypeScript) – For a responsive, component-based UI.
- **Networking Layer:** PeerJS (WebRTC Wrapper) – Enables decentralized, low-latency P2P data transfer.
- **Computation Engine:** Google Gemini AI Model (gemini-2.5-flash) – Performs the actual natural language processing.
- **Visualization:** Recharts – Renders real-time latency and sentiment distribution charts.
- **Styling:** Tailwind CSS – Provides a modern, ”Cyberpunk” aesthetic.

6. Implementation Methodology

Phase 1: Connection Handshake
The Host initializes a P2P session and generates a **Cluster ID**. Workers input this ID to establish a **DataConnection**. Upon connection, the Host securely transfers necessary configuration (API Keys) to the Worker (Simulating secure environment variable syncing).

Phase 2: Task Dispatch (Map Phase)
The Host implements a ”Map” function. The large text file is tokenized and split into N segments. These segments are wrapped in *Task* objects and dispatched to available workers over the network.

Phase 3: Parallel Execution
All connected workers process their specific chunks simultaneously. This demonstrates **Task Parallelism** (if tasks differed) and **Data Parallelism** (same task, different data).

Phase 4: Aggregation (Reduce Phase)
As *TaskResult* messages arrive back at the Host, they are reduced into a single state array. The Host updates the UI progress bars and charts in real-time.

7. Expected Outcomes

By the end of the demonstration, we will show:

- **Speedup:** A task that takes T seconds on one machine will take approximately T/N seconds (plus network overhead) on N machines.
- **Scalability:** The ability to add new devices to the cluster mid-computation dynamically.
- **Fault Tolerance:** If a worker disconnects, the Host detects the socket closure (though full task-retry logic is a future scope).
- **Cross-Device Compatibility:** A cluster composed of a Laptop (Host) and a Smartphone (Worker) functioning seamlessly.

8. Conclusion

NebulaGrid successfully abstracts the complexities of distributed computing. It provides a visual and interactive platform to understand how large jobs can be decomposed, distributed, executed in parallel, and reassembled, fulfilling the core learning outcomes of the Parallel and Distributed Computing course.