Assignment 1: Simulating a Distributed RDBMS

February 5, 2026

Overview

The objective of this assignment was to design and implement **SimuFragDB**, a system emulating a multi-node distributed database environment. The system utilizes multiple PostgreSQL instances on a local machine to simulate horizontal fragmentation. The focus of the project was on implementing deterministic query routing and understanding the complexities of data aggregation across distributed shards.

System Architecture

The implementation follows a client-server architecture where the `Driver` acts as the coordinator and the `FragmentClient` manages connections to N independent PostgreSQL nodes.

Horizontal Fragmentation & Routing

Data is partitioned based on the `student_id`.A deterministic routing function is implemented in `Router.java` that uses hash-based logic to map a specific primary key to a Fragment ID (0 to $N - 1$).

- **Point Operations:** `INSERT`, `UPDATE`, and `DELETE` operations are explicitly routed to a single node.
- **Consistency:** Point-reads (fetching a student profile) are consistently accurate because the system looks in the exact fragment where the data was hashed.

Data Aggregation Approach

For global queries, such as calculating the average score per department or finding students with the maximum number of courses, the current implementation selects a **random fragment** to execute the query.

- **Implementation Logic:** Instead of querying all nodes and performing a manual merge-sort or global aggregation in Java, the client delegates the query to a single node's SQL engine.
- **Trade-off:** This approach minimizes network overhead and client-side processing but results in partial data views.

Challenges Faced

1. **JDBC Connection Management:** Managing multiple simultaneous connections (`connectionPool`) to different databases required careful handling of the PostgreSQL `ADMIN` database to dynamically create and drop fragments.

2. **Global State Awareness:** Since each fragment is an independent RDBMS instance, they are unaware of each other's data. Achieving 100% accuracy for global aggregates would require a two-phase process (Map-Reduce style) which adds significant complexity to the client.

Experimental Results

The simulation was executed with $N = 3, 5, 10, 15, 20$ fragments. A baseline run was performed on a single fragment to generate the `expected_output.txt`, which was then compared against the distributed `actual_output.txt`.

Fragments (N)	**Baseline Time (ms)**	**Distributed Time (ms)**	**Accuracy (%)**
3	2953	2752	71.00 %
5	2541	2557	70.55 %
10	2776	2882	70.47 %
15	2583	2780	70.31 %
20	2922	3142	70.19 %

Table 1: Performance and Accuracy Scaling Metrics

Accuracy Analysis

As noted in the assignment objectives, the goal was not 100% correctness but gaining architectural insights. Because global queries (Average Scores, Read All) are only executed on a single random fragment, the system only reports a subset of the global state. To achieve 100% accuracy, the client would need to aggregate results from **all** nodes using a `UNION` or manual summation logic. Even though the overall accuracy is not 100%, the point-queries (profile reads) remain 100% accurate, validating that the routing function works correctly.

Contributions

- IMT2022021 Rutul Patel: Implemented the `getAvgScoreByDept()` and `getAllStudentsWithMostCourses()` functions, and part of the driver code.
- IMT2022049 Tanish Pathania: Implemented the `updateGrade()` and `deleteStudentFromCourse()` functions, and part of the driver code.
- IMT2022076 Mohit Naik: Implemented the `insertGrade()` and `updateGrade()` functions and wrote the report.
- IMT2022086 Ananthakrishna K: Implemented the `setupConnections()` and `insertStudent()` functions and wrote the readme.