Assignment 1

Distributed Systems — MapReduce and MPI

Course ID: CS3.401
Compute Platform: RCE Cluster
Submission Deadline: 2 February 2026

Course Infrastructure

Cluster Configuration

- **MaxCPUsPerUser:** 8
- **MaxJobsPerUser:** 1
- **MaxSubmitJobsPerUser:** 1
- **Walltime:** 10 minutes
- **Account expiration date:** 15 March 2026

Team Formation

- Students must work in teams of **exactly two members**.
- Each team will submit **one joint submission**.
- Both team members must be clearly mentioned in the submission.
- Submissions will be checked for plagiarism and cross-team similarity.

Submission & Evaluation Policy

- Submissions **may be evaluated automatically**.
- Submissions must **strictly adhere** to the specified input/output formats.

- Any deviation from formats, filenames, or execution instructions may result in rejection.
- Jobs exceeding the wall-time limit will be terminated and treated as incorrect.

Problem 1 — MapReduce: Inverse Document Frequency (IDF)

Weight: 30% of total marks

Objective

Compute the Inverse Document Frequency (IDF) for each term in a large document corpus using MapReduce.

$$\text{IDF}(w) = \log\left(\frac{N}{df(w)}\right)$$

where:

- N = total number of documents
- $df(w)$ = number of documents containing word w

Input Format

```
doc_id<TAB>document_text
```

Each line represents one document with a document ID followed by a tab character and the document text.

Output Format

```
word<TAB>idf_value
```

- `idf_value` must be printed with **4 decimal places**.
- Output order is not constrained.

Sample Test Case

Input:

```
doc1     the cat sat on the mat
doc2     the dog sat on the log
doc3     the cats and dogs are pets
```

Expected Output:

```
and      1.0986
are      1.0986
cat      1.0986
cats     1.0986
dog      1.0986
dogs     1.0986
log      1.0986
mat      1.0986
on       0.4055
pets     1.0986
sat      0.4055
the      0.0000
```

(Note: Order may vary)

Constraints

- Corpus size may not fit in memory of a single machine.
- The solution **must use MapReduce**.
- Combiners are allowed and encouraged.
- Use natural logarithm (base e).

Performance Analysis Requirements

IMPORTANT: In addition to your code, you must submit a performance analysis document that includes:

1. **Data Processing Estimation:**
    - Calculate the maximum amount of data your solution can process within the 10-minute walltime limit

- Show your computation based on:
  - Task initialization time
  - Map operation throughput (MB/s or records/s)
  - Reduce operation throughput
  - Disk I/O time for intermediate writes
  - Network transfer overhead (if applicable)

2. **Profiling Results:**
   - Measure and report actual execution times for:
     - Task initialization time
     - Mapper execution time
     - Shuffle/sort time
     - Reducer execution time
     - Total job completion time
   - Profile with different input sizes (e.g., 1MB, 10MB, 100MB)
   - Compare theoretical estimates with actual measured times
3. **FLOPs Analysis:**
   - Estimate computational complexity in terms of floating-point operations
   - Calculate expected FLOPs based on input size
   - Measure actual computational performance

Submit this analysis as a file named `performance_analysis_p1.pdf` in your `problem1/` directory.

Problem 2 — MapReduce: PageRank

Weight: 40% of total marks

Objective

Compute PageRank scores for a directed graph using iterative MapReduce.

Input Format

```
node_id<TAB>neighbor_1,neighbor_2,...
```

Each line represents a node and its outgoing edges. If a node has no outgoing edges, the neighbor list is empty.

Parameters

- **Initial PageRank:** $\frac{1}{N}$ (where N = total number of nodes)
- **Damping factor:** 0.85
- **Number of iterations:** 10 (fixed)

Output Format

```
node_id<TAB>pagerank_value
```

- PageRank values must be printed with **6 decimal places**.
- Output order is not constrained.

Sample Test Case

Input:

```
0	1,2
1	2
2	0
```

Expected Output (after 10 iterations):

```
0	0.388913
1	0.214416
2	0.396670
```

(Note: Order may vary)

Constraints

- Each iteration must be implemented as a **separate MapReduce job**.
- Intermediate outputs must be written to disk.
- In-memory iteration or Spark-style solutions are **not allowed**.

PageRank Formula

For each iteration:

$$\mathrm{PR}(v) = \frac{1-d}{N} + d \sum_{u \in \mathrm{incoming}(v)} \frac{\mathrm{PR}(u)}{\text{out-degree}(u)}$$

where $d = 0.85$ is the damping factor.

Performance Analysis Requirements

IMPORTANT: Submit a performance analysis document that includes:

1. **Iteration Time Analysis:**
   - Profile time taken for each of the 10 iterations
   - Measure MapReduce job initialization overhead per iteration
   - Calculate time for reading/writing intermediate results
   - Estimate maximum graph size processable within 10-minute limit
2. **Scalability Analysis:**
   - Show how execution time scales with:
     - Number of nodes in the graph
     - Number of edges (graph density)
   - Provide timing measurements for different graph sizes
   - Compare theoretical complexity with actual performance
3. **I/O Performance:**
   - Measure disk I/O overhead for intermediate data between iterations
   - Calculate data volume written/read per iteration
   - Estimate network overhead in distributed setup

Submit this analysis as `performance_analysis_p2.pdf` in your `problem2/` directory.

Problem 3 — MPI: Distributed Graph Algorithm

Weight: 30% of total marks

Objective

Implement a distributed graph algorithm using MPI. Choose **one** of the following:

1. **Parallel Borůvka-style Minimum Spanning Tree**
2. **Parallel Connected Components**

3. **Cut-edge detection**

Input Format

```
n m
u1 v1 w1
u2 v2 w2
...
um vm wm
```

where:

- `n` = number of nodes
- `m` = number of edges
- Each subsequent line contains an edge: `u v w` (nodes u and v connected with weight w)
- Graph is **undirected** and **weighted**
- Nodes are labeled from **0 to n-1**

Output Format

For Minimum Spanning Tree:

```
total_mst_weight
```

Output a single integer: the total weight of the MST.

For Connected Components:

```
node_0 component_id
node_1 component_id
...
node_n-1 component_id
```

Output each node and its component ID (one per line).

For Cut Edge:

```
u v
```

Output any valid cut edge (bridge) as two node IDs.

Sample Test Case

Input:

```
4 5
0 1 1
0 2 4
1 2 2
1 3 5
2 3 3
```

Expected Output (for MST):

```
6
```

Explanation: The MST includes edges (0,1), (1,2), (2,3) with total weight = 1 + 2 + 3 = 6.

Constraints

- Implementation must use **MPI primitives** (MPI_Send, MPI_Recv, MPI_Bcast, MPI_Allreduce, etc.).
- Shared-memory or single-process solutions are **not allowed**.
- The program must run correctly with multiple MPI ranks.

Allowed Languages

- C / C++ / Java / Python
- MPI implementations: OpenMPI or MPICH

Performance Analysis Requirements

IMPORTANT: Submit a performance analysis document that includes:

1. **Parallel Efficiency Analysis:**
   - Measure execution time with different numbers of MPI processes (1, 2, 4, 8)
   - Calculate speedup and parallel efficiency
   - Identify communication bottlenecks
   - Estimate maximum graph size processable within 10-minute limit

2. **Communication Profiling:**
    - Measure time spent in MPI communication primitives
    - Calculate communication-to-computation ratio
    - Analyze load balancing across processes
    - Profile message sizes and frequency
3. **Computational Complexity:**
    - Provide FLOPs analysis for your algorithm
    - Compare theoretical complexity with measured performance
    - Show how performance scales with graph size

Submit this analysis as `performance_analysis_p3.pdf` in your `problem3/` directory.

Marks Distribution Summary

Problem	Description	Weight
Problem 1	IDF MapReduce	30%
Problem 2	PageRank MapReduce	40%
Problem 3	MPI Graph Algorithm	30%
Total		**100%**

Submission Format

Your submission must follow this **exact directory structure**:

```
TeamXX/
├── team_info.txt
├── problem1/
│   ├── mapper.py
│   ├── reducer.py
│   ├── combiner.py (optional)
│   ├── run_mapreduce.sh
│   └── performance_analysis_p1.pdf
├── problem2/
│   ├── mapper.py
│   ├── reducer.py
│   ├── run_pagerank.sh
│   └── performance_analysis_p2.pdf
└── problem3/
    ├── mpi_graph.cpp (or .py, .java, .c)
    ├── compile.sh (if compilation needed)
    ├── run_mpi.sh
    └── performance_analysis_p3.pdf
```

File Descriptions

team_info.txt

Must contain:

```
Team ID: TeamXX
Member 1: Full Name, Roll Number
Member 2: Full Name, Roll Number
```

Problem 1 Files

- **mapper.py** - Your mapper implementation
- **reducer.py** - Your reducer implementation
- **combiner.py** - (Optional) Combiner implementation
- **run_mapreduce.sh** - Shell script that will be executed as:

```
bash run_mapreduce.sh <input_file> <output_dir>
```

Problem 2 Files

- **mapper.py** - Your mapper implementation

- **reducer.py** - Your reducer implementation
- **run_pagerank.sh** - Shell script that will be executed as:

```
bash run_pagerank.sh <input_file> <output_dir>
```

Problem 3 Files

- **mpi_graph.cpp** (or .py, .java, .c) - Your MPI implementation
- **compile.sh** - (If needed) Script to compile your code:

```
bash compile.sh
```

- **run_mpi.sh** - Shell script that will be executed as:

```
bash run_mpi.sh <input_file> <num_processes>
```

Packaging Your Submission

1. Ensure your directory structure matches exactly as shown above
2. All shell scripts must be executable (`chmod +x *.sh`)
3. Create a compressed archive:

```
zip -r TeamXX.zip TeamXX/
```

4. Submit the `TeamXX.zip` file

Important Notes

1. **File Naming:** Use exact file names as specified (case-sensitive)
2. **Input/Output Formats:** Must match exactly, including decimal places
3. **No Hard-coded Values:** Do not assume input sizes or structure
4. **Robustness:** Your programs must handle various input sizes
5. **Testing:** Thoroughly test your code before submission
6. **Plagiarism:** All submissions will be checked for similarity

Evaluation Criteria

Your submission will be evaluated based on:

1. **Correctness:** Does your solution produce correct output? (40%)
2. **Format Compliance:** Does output match required format exactly? (10%)
3. **Implementation:** Did you use required frameworks (MapReduce/MPI)? (20%)
4. **Efficiency:** Does your solution complete within time limits? (10%)
5. **Performance Analysis:** Quality and accuracy of profiling, timing measurements, and data processing estimates (15%)
6. **Robustness:** Does it handle various input sizes and edge cases? (5%)

Support & Resources

- **Hadoop Documentation:** For MapReduce implementation
- **MPI Documentation:** For Message Passing Interface
- **Course Lectures:** Review MPC/MST lecture notes for Problem 3
- **Office Hours:** Contact course staff for clarifications

Academic Integrity

- Work must be original and done by your team
- Cite any external resources or references used
- Do not share code with other teams
- Violations will result in severe penalties

Good luck with your assignment!

This assignment is part of CS3.401 Distributed Systems course. All submissions must be made through the course portal by the specified deadline. Late submissions will not be accepted.