Distributed LMS with LLM-based Tutoring and Raft-based Data Consistency

In this assignment, you will develop a simplified distributed Learning Management System (LMS) that supports
basic student and instructor interactions. The system will include real-time tutoring assistance powered by a
lightweight LLM (that can run on a CPU). To ensure consistency in critical data (like grades and progress), a
basic implementation of the Raft consensus protocol will be used. The focus of the assignment is on understanding
RPC communication, integrating a lightweight LLM, and implementing a simplified version of Raft for consistency.

Note: This is a team assignment. The recommended size is THREE. A team size smaller than this is NOT recommended.
Any exceptions to this team size will require prior permission of the IC.

System Components

The following is a general expectation of the components that your system should have:

1. Client Nodes:
   - Represent students and instructors who interact with the LMS.
   - Students can submit assignments, ask questions, and receive scores/grades for their assignments.
   - Instructors can grade assignments, answer student queries / provide feedback to students.
2. Tutoring Server:
   - Hosts a lightweight LLM (e.g., a small instance of GPT-2 or a lightweight NLP model such as spaCy or NLTK) that provides simple tutoring responses based on student queries.
3. LMS Server:
   - Manages course content, basic student records, and interactions between students and instructors.
   - Includes a simplified Raft implementation for ensuring consistency in critical data (like grades).
4. Raft Consensus Protocol:
   - A basic version of Raft, primarily focusing on leader election and simple log replication, to ensure that update

     to critical data are consistent across nodes.

Functional Requirements

1. Student Interaction:
   - Students can log in, view course materials, submit assignments, and receive scores/grades.
   - Students can ask the LLM tutoring server a set of simple questions related to the course, receiving simple,
context-aware responses.
2. Instructor Interaction:
   - Instructors can upload course materials, grade assignments, and provide brief feedback.
   - Instructors’ grading and feedback are synchronized across all nodes using the Raft protocol.
3. Data Consistency:
   - Implement a simplified Raft protocol to ensure that grades and student progress are consistently recorded
across all servers.
4. Fault Tolerance:
   - Basic fault tolerance using Raft to handle node fai
 ensuring that the system remains operational and
consistent.

Assumptions/scope

1. User Session: User sessions are handled by the client via a token generated by the LMS server, valid until logout.
2. Data Management: No deletions or modifications of data are required.
3. Query Handling: Students posting queries can state whether they want responses from the LLM or the instructor.
4. Course Management: Only one course is available; addition or deletion of courses is not allowed.

5. Anonymity: It is not required to know which student wrote which query.
Demo Setup
For a good working demo, your system should be designed to run on 4-5 nodes. Here is the breakdown:
1. Node 1: Tutoring Server
 - Runs the lightweight LLM that handles student queries and provides tutoring responses.
 - This node focuses on handling the AI component of the system, ensuring it operates independently of the other
components.
2. Node 2: LMS Server with Raft Leader
 - Manages course content, student interactions, and handles critical data (like grades and progress).
 - This node can serve as the leader in the Raft consensus protocol, coordinating updates tonsure consistency
across the distributed system.
3. Node 3: LMS Server with Raft Followers
 - Another instance of the LMS server that replicates data from the Raft leader to ensure fault tolerance.
 - This node demonstrates how the system maintains consistency and continues to operate even if one server
goes down.
4. Node 4: Additional Raft Follower
 - An extra node to further demonstrate the fault tolerance and redundancy capabilities of the system.
 - Useful to showcase how the Raft protocol handles leader election and ensures data consistency with more than
two nodes.
5. Node 5: Extra/optional Client Node
 - Could be used to simulate multiple student or instructor interactions, testing the system's scalability and load
management.
Function Signatures
The following is a general expectation of the functions you should implement in order to achieve the desired
functionality. Minor deviations or modifications to the below list are allowed. However, you MUST use RPC
wherever RPC is desirable or expected. Client (Student/Instructor):
1. login(username, password):
o Description: Authenticates a user with the provided username and password.
o Returns: loginResponse(status, Optional(token))
2. logout(token):
o Description: Logs out the user associated with the provided token.
o Returns: status
3. post(token, type, data):
o Description: Create/Upload the given type of data to the LMS Server. The contents of the data
variable depend on the type provided (e.g., queries, assignments).
o Returns: status
4. get(token, type, Optional(data)):
o Description: Retrieves a list of the given type of data (queries, course materials, assignments,
students). Additional optional data may be sent, such as studentId for retrieving assignments of
a particular student.
o Returns: getResponse(status, List[(typeId, data)])
LMS Server:
1. loginResponse(status, Optional(token)):
o Description: Returns the result of a login attempt. If successful, incl2. getResponse(status, List[(typeId, data)]):
o Description: Returns a list of the requested type of data to the user.
3. getLLMAnswer(queryId, query, context):
o Description: Requests the Tutoring Server for an answer for the given query based on the given

context. The query identifier is passed to ensure the correct query is updated when a response is received.
Tutoring Server:
1. getLLMAnswerResponse(queryId, answer):
o Description: Generates an answer for the given query using the LLM engine based on the given context.
Raft (LMS):
1. requestVote(from, to, term, lastLogIndex, lastLogTerm):
o Description: Request to another node to vote for the sender as leader in the specified term, including information about the sender's log.
o Returns: requestVoteReply(from, to, term, voteGranted)
2. requestVoteReply(from, to, term, voteGranted):
o Description: Response to a vote request, indicating whether the recipient node grants its vote to the sender.
3. appendEntries(from, to, term, prevIndex, prevTerm, commitIndex, entries):
o Description: Batch of new log entries to a follower node, along with informationprevious log entry and the current commit index, to replicate the leader's log.
o Returns: appendEntriesReply(from, to, term, entryAppended, matchIndex)
4. appendEntriesReply(from, to, term, entryAppended, matchIndex):
o Description: Response to an append entries request, indicating whether the new log entries were successfully appended and reporting the index up to which the recipient node's log matches the sender's log.
Code and Deliverables
Implement your code in Python or C/C++. For RPC, use gRPC framework (only). You are free to use any existing libraries in Python or C/C++. You are also free to use any existing code from Github or take help of
ChatGPT. However, you must attribute such sources properly.
Milestone 0: Decide your teams. DEADLINE: 31st August 2024
Milestone 1: Basic RPC Application. DEADLINE: 14th September 2024
 Objective: Implement the basic client-server communication using gRPC. The focus on core LMS functionalities where students and instructors can log in, post, and retrieve data.
 Tasks:
o Implement login, logout, post, and get RPC functions on both the client and LMS server sides.
o Handle session management with tokens.
o Create a basic command-line interface for students and instructors to interact with the system.
 Expected Outcome: A working prototype of the LMS where users can authenticate, upload assignments, post queries, and retrieve course materials.
Milestone 2: LLM Integration. DEADLINE: 28th September 2024
 Objective: Integrate the lightweight LLM with the tutoring server to provide responses to student queries.
 Tasks:
o Implement the getLLMAnswer RPC on the LMS server and getLLMAnswerResponse on the Tutoring Server.
o Ensure that the system can handle queries marked for LLM responses and return them to the requesting student.
o Test the LLM responses with a few predefined contexts.
 Expected Outcome: The LMS should be able to route student queries to the LLM, and the LLM should generate relevant answers, which are then displayed to the student.
Milestone 3: Raft Consensus Protocol Implementation. DEADLINE: 19th October 2024
 Objective: Implement the Raft consensus protocol to ensure consistency of critical data (such as grades and
student progress) across distributed LMS nodes.
 Tasks:
o Implement requestVote, requestVoteReply, appendEntries, and appendEntriesReply RPCs on the LMS servers.
o Set up multiple LMS nodes to replicate data and handle leader election and log replication.

o Test the fault tolerance of the system by simulating node failures and ensuring data consistency is maintained.

Expected Outcome: A fault-tolerant, distributed LMS where all critical data is consistently replicated across multiple nodes, ensuring that the system remains operational and consistent even during node failures.