**COMP41720 Distributed Systems: Architectural Principles** **LAB 2: Distributed Data Management and Consistency Models**

**Duration:** Week 5-6 (culminating in a lab submission by the end of Week 6)

**1. Lab Overview** This lab shifts our focus to the **fundamental architectural principles and challenges** inherent in storing and managing data across distributed systems. You will gain hands-on experience with **a distributed data store**, exploring **various replication strategies** and **consistency models**. The core objective is to understand the **trade-offs** involved in these choices, particularly concerning the CAP Theorem (Consistency, Availability, Partition Tolerance).

**2. Learning Objectives** By the end of this lab, you will be able to:

* **Understand the challenges** of storing and managing data across multiple nodes in a distributed environment.
* **Configure and compare different data replication strategies**, including primary-backup (leader-follower), multi-primary (multi-leader), and leaderless architectures.
* **Experiment with and differentiate between various consistency models**, such as strong consistency, eventual consistency, and (optionally) causal consistency.
* **Analyse the practical implications and architectural trade-offs** of choosing specific replication and consistency settings, particularly in the context of system availability, performance, and data integrity.
* **Apply architectural thinking** by justifying *why* certain data management decisions are made, rather than just *how* they are implemented.

**3. Introduction & Context** In distributed systems, managing data across multiple nodes introduces significant complexities, including concurrency, partial failures, and latency. The **CAP Theorem** serves as a foundational concept, stating that a distributed data store can only simultaneously guarantee two out of three properties: Consistency, Availability, and Partition Tolerance. Your choices in replication and consistency directly impact which of these properties your system prioritizes.

This lab provides an opportunity for **in-depth practical application** of these theoretical concepts. Remember, in software architecture, there are "no best practices" because every decision involves trade-offs. Your goal is to **strive for the least worst combination of trade-offs** for a given scenario.

**4. Tools & Environment** To avoid reliance on outdated technologies and allow for exploration, you are encouraged to **choose your own distributed NoSQL database**.

* **Recommended NoSQL Databases** (Choose ONE):
  + **Apache Cassandra:** Excellent for demonstrating tunable consistency and various replication factors.
  + **MongoDB (with Replica Sets):** Good for exploring primary-backup replication and different read/write concerns.
  + **Redis Cluster:** Useful for high-performance caching and eventual consistency patterns.
* **Client Application Language** (Choose ONE):
  + Python, Java (modern versions recommended).
* **Environment:**
  + Local setup (e.g., Docker for easy deployment of multiple database nodes).

**5. Lab Tasks & Experiments**

**Part A: Setup & Baseline**

1. **Database Cluster Setup:**
   * **Set up a distributed cluster** of your chosen NoSQL database. A minimum of **3 nodes** is highly recommended to properly demonstrate replication and partition tolerance.
   * **Document your setup process** (e.g., Docker commands, configuration files).
2. **Simple Data Model:**
   * Design a **simple data model** for a generic entity (e.g., UserProfile with fields like user\_id, username, email, last\_login\_time).
   * Insert an initial set of data into your cluster.

**Part B: Replication Strategies** For your chosen database, configure and demonstrate the following replication concepts:

1. **Replication Factor / Write Concern:**
   * **Configure your database to use a replication factor (RF) of at least 2 or 3.**
   * Demonstrate how **different write concerns/levels** (e.g., ONE, QUORUM, ALL in Cassandra; w:1, w:majority, w:all in MongoDB) affect write latency and durability across your cluster. Provide observations.
2. **Leader-Follower (Primary-Backup) Model:**
   * If your database inherently supports a leader/primary node (e.g., MongoDB replica set, Kafka leader partitions), **demonstrate writes and reads against the primary and how data propagates to followers**.
   * **Simulate a primary node failure** and observe how the system elects a new primary and handles ongoing operations. Note any downtime or data inconsistencies.
3. **Leaderless (Multi-Primary) Model (if applicable):**
   * If your database supports a leaderless or multi-primary setup (e.g., Cassandra), **demonstrate writes to any node** and observe how conflicts are resolved (or not) and how data eventually converges across the cluster.

**Part C: Consistency Models** Design and execute experiments to illustrate different consistency models:

1. **Strong Consistency:**
   * **Configure both writes and reads to demand strong consistency** (e.g., QUORUM or ALL for both in Cassandra; w:majority for writes and readConcern:majority for reads in MongoDB).
   * Perform a write operation on one node and immediately attempt to read it from *another* node. **Verify that the data is immediately consistent**.
   * **Introduce a network partition or node failure** during this experiment. Observe the impact on write/read operations (e.g., does it block, throw an error, become unavailable?). **Relate this observation directly to the CAP theorem.**
2. **Eventual Consistency:**
   * **Configure writes and reads for eventual consistency** (e.g., ONE for both in Cassandra; w:1 for writes and default read concern in MongoDB).
   * Perform a write operation on one node. Immediately attempt to read it from *another* node. **Observe if you can read stale (old) data before it propagates.**
   * Implement a **simple loop that repeatedly reads the data** until the latest value is observed, demonstrating the "eventual" nature.
   * **Discuss scenarios where eventual consistency is acceptable or even desirable** (e.g., social media likes, sensor data) and *why* this choice is beneficial in those contexts (e.g., availability, performance).
3. **Causal Consistency (Optional / Bonus):**
   * If your chosen database supports mechanisms for causal consistency (e.g., some distributed key-value stores with version vectors), **design an experiment to demonstrate that causally related operations are observed in order**, even if other concurrent operations are not.

**Part D: Distributed Transactions (Conceptual / Optional Coding)**

* **Review the challenges of distributed transactions** (e.g., Saga pattern) as discussed in the course materials.
* **Conceptual Exercise:** Choose a simple multi-service workflow (e.g., placing an e-commerce order involving OrderService, PaymentService, InventoryService).
  + **Describe how this workflow would be managed** using:
    - **ACID transactions** (and why it's problematic in a truly distributed system).
    - **Sagas (Orchestrated or Choreographed)**.
  + **Analyse the trade-offs** between these approaches in terms of consistency, complexity, fault tolerance, and performance. You do not need to implement this part; a detailed conceptual explanation is sufficient.

**6. Deliverables**

1. **Lab Report (PDF format) or a 15-minute Video:**
   * **Introduction:** Briefly state the lab's purpose and your chosen database/tools.
   * **Setup & Configuration:** Detail your database cluster setup (e.g., node count, Docker Compose file, configuration settings for replication). Include relevant diagrams for clarity.
   * **Replication & Consistency Experiments:**
     + For each experiment in Part B and Part C:
       - Describe the **specific configuration** (e.g., replication factor, write/read concern).
       - **Document your observations** (e.g., latency, data visibility, behaviour during failures). Use screenshots or console output snippets as evidence.
       - **Crucially, provide a detailed analysis of the architectural trade-offs.** Justify *why* you would choose this specific configuration for a given business requirement, linking back to the CAP theorem and the course's emphasis on "why" over "how". For example, when would strong consistency be paramount, and what are its costs? When would eventual consistency be a better fit?
   * **Distributed Transactions (Part D - Conceptual):** Your detailed conceptual analysis of the e-commerce workflow, contrasting ACID with Saga patterns and their trade-offs.
   * **Conclusion:** Summarise your key learnings and any unexpected observations.
2. **Source Code Repository Link:** Provide a link to your code repository (e.g., GitHub, GitLab) containing all client application code used for your experiments.

**7. Assessment Criteria** Your lab will be assessed based on the following:

* **Correctness and Completeness of Implementation (40%):**
  + Successful setup and configuration of the distributed database cluster.
  + Accurate implementation of experiments demonstrating replication and consistency.
* **Depth of Analysis and Justification (40%):**
  + **Clear and insightful analysis of architectural trade-offs** for each configuration tested.
  + Strong justifications for design choices, linking observations to distributed systems principles (e.g., CAP Theorem, performance, availability, fault tolerance).
  + Coherent conceptual explanation of distributed transactions.
* **Clarity and Organisation of Report (10%):**
  + Well-structured, concise, and easy-to-understand report.
  + Effective use of diagrams and visual aids.
* **Code Quality and Readability (10%):**
  + Clean, well-commented code that is easy to understand and reproduce.

**8. Tips for Success**

* **Start Early:** Setting up distributed systems can be complex. Give yourself ample time for troubleshooting.
* **Read Documentation:** Familiarise yourself with the documentation for your chosen database and its client libraries.
* **Focus on the "Why":** Beyond getting the code to work, ensure you deeply understand *why* certain behaviours occur and *why* different architectural choices yield different trade-offs.
* **Incremental Builds:** Implement and test each part of the lab incrementally.
* **Collaborate, but Deliver Independently:** Discuss concepts and issues with peers, but ensure your submitted code and report reflect your own understanding and analysis.
* **Utilise Support:** Teaching Assistants (TAs) and instructors are available for questions during lab sessions. Do not hesitate to ask for clarification on vague instructions or technical challenges.