CS60002 Distributed Systems

Assignment 2: Distributed Queue with Partitions and Broker Manager

Date: Feb 1st, 2023

Target Deadline (For sharing the git repo): Feb 28th, 2023

In the previous assignment, you implemented a single-process broker that handles queues for different topics. In this assignment, you will extend the basic broker implementation from Assignment 1 to a more sophisticated, multi-broker distributed queue system by dividing a topic queue into multiple partition queues. Partitioning is a common technique used in distributed queue systems to improve scalability and fault tolerance by allowing multiple brokers to handle different partitions of the same topic in parallel.

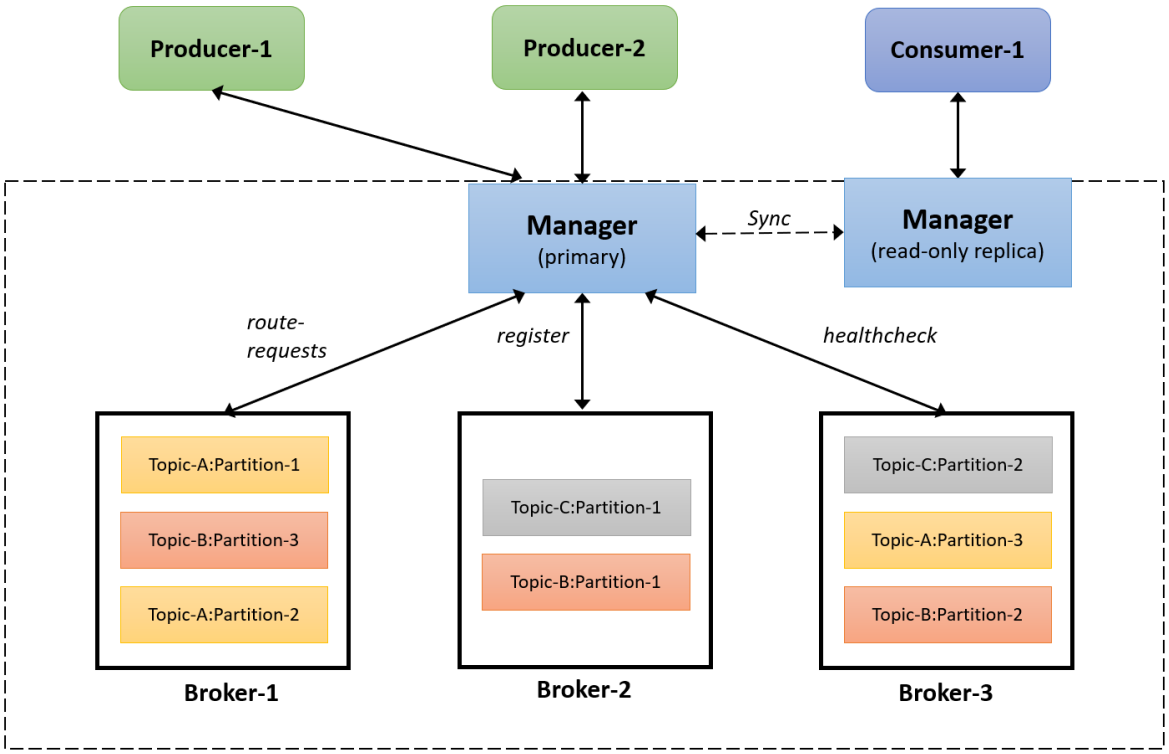

Fig: Architecture of distributed queue with partitions and Broker Manager

Your task is to implement partitioning in the distributed queue system you developed in the previous assignment. To do this, you'll need to:

1. Modify the broker to handle individual partitions of a topic.
2. Implement a new component called the *Broker Manager*, which will be responsible for managing the mapping between producers, topics, and partitions. The *Broker Manager* should also handle client metadata, such as the Consumers and Producers registered, and the Topics and their partitions registered.
3. The *Broker Manager* should provide an endpoint for producers to produce messages to a particular partition or allow for a round-robin algorithm to be used if no partition ID is specified.
4. The *Broker Manager* should also implement a health-check mechanism to monitor the state of the brokers and clients.
5. Implement a write-ahead logging mechanism to ensure data durability in the event of a broker failure.

The *Broker Manager* should be able to handle multiple concurrent requests from producers and consumers. You may make use of multi-threading or asynchronous I/O libraries in your implementation to improve performance. The Broker Manager should also be able to store its metadata in persistent storage such as a database.

Here is a guideline on how you can divide the assignments into sub tasks, note that it may require you to restructure your codebase from assignment 1. Also note that this is only a guideline and you can feel free to approach it in your own way while stating the relevant design choices you make along the way.

Task 1: Implement Partitions

- Introduce partitions, which divide Topics horizontally and each partition of a topic may reside on a different broker.
- In order to achieve horizontal scaling, you are required to implement a partitioning mechanism that will allow you to divide a topic's data into multiple partitions.

Task 2: Implement Broker Manager

- The manager is a new component that will act as an intermediary between producers and consumers and brokers.

- Implement a new broker manager that handles client metadata, topic metadata, and mappings between consumers, topics, and partitions.
- The broker manager should store client metadata such as the list of registered producers and consumers, the list of registered topics, the list of partitions of each topic, and the mapping between consumers, topics, and partitions.
- The broker manager should allow producers to produce to a particular partition or run a round-robin algorithm to decide the partition within a topic if no partition ID is specified.
- The broker manager should also handle broker cluster configuration management, such as adding and removing brokers.
- Implement a healthcheck mechanism that records the timestamp of the last heartbeat from brokers, producers, and consumers. If there is no item left to consume, the response should be empty.
- Implement a service discovery mechanism for client requests, either by redirecting consumers or handling the requests internally.
- Additionally, the manager should implement a write-ahead logging (WAL) mechanism to persist the metadata it maintains, allowing it to restore the metadata in the event of a failure. You are free to choose the type of storage for WAL (e.g. relational database, NoSQL database, etc.)
- Store metadata in persistent storage, such as a database, and ensure the manager has read/write replicas.

Task 3: Implement Broker

- The brokers are responsible for implementing the endpoints that interact with the manager.
- Each broker should manage its assigned partitions of a topic and persist the partition data, allowing it to restore the partition in the event of a failure.
- Service client requests: The brokers should also service requests from producers and consumers as directed by the manager.

Guidance:

- The broker manager should be able to handle multiple concurrent requests from producers and consumers. You are free to make a design choice on how to achieve this, such as using multi-threading, async io, or similar libraries in other languages.
- The manager should implement read/write replicas for high availability. This means that there should be multiple instances of the manager running at any given time, with one instance designated as the primary (or "write") manager and

the others designated as secondary (or "read") managers. The primary manager should be responsible for accepting write requests and updating the WAL, while the secondary managers should be responsible for serving read requests.
- The WAL mechanism should ensure that any changes made to the metadata by the primary manager are persisted to disk before being acknowledged as complete. This will help ensure the durability of the metadata in the event of a failure.
- You are free to make design choices where appropriate (e.g. the method of partitioning, the type of storage for WAL, the method of handling concurrent requests, etc.), but be prepared to justify your choices in your final submission.
- You may simulate a cluster of hosts using VMs locally or deploy brokers and the manager on different machines.

Grading Scheme:

- Implementation of Partitions (20%)
- Implementation of the Broker Manager (30%)
- Implementation of the Broker (20%)
- Performance optimization (10%)
- Documentation and Presentation (20%)

Submission:

- You are expected to write clean, well-documented code and to include a detailed report explaining your design choices and implementation details.
- Submit the complete code, along with a detailed documentation explaining the design choices made and the testing process carried out using the same procedure as earlier (using link to a git repo) . Demo/Presentation instructions will be conveyed via Piazza in due time.

Glossary, Explanations and References:

Partitions:

Partitions are used in a distributed queue system for the following reasons:

1. Scalability: Partitions allow for horizontal scaling by dividing a topic into multiple smaller, manageable parts that can be processed in parallel.
2. Fault tolerance: Partitions provide a way to distribute the data across multiple brokers, so that the system can continue to function even if one or more brokers fail.
3. Load balancing: Partitions can help to balance the load among the brokers by distributing the data in a way that ensures an even distribution of work.

Write Ahead Logging (WAL):

In the context of this assignment, Write Ahead Logging (WAL) is a mechanism used to ensure data durability and consistency in the event of a system failure. The idea behind WAL is to write transaction logs to a persistent storage before committing the changes to the database. In case of a system crash or failure, the logs can be used to restore the system to a consistent state.

Here's a guideline on how to implement WAL in the Broker Manager:

1. For each transaction, write the transaction logs to a persistent storage before making changes to the database.
2. Store the logs in a way that they can be easily retrieved and replayed in case of a system failure. This can be done by using a file system or a database that supports WAL.
3. Implement a recovery mechanism that reads the logs and re-applies the transactions in case of a system failure. This mechanism should be able to detect the last committed transaction and only replay the logs that were not committed.
4. Ensure that the logs are written to the persistent storage in a way that ensures their durability, even in the event of a system failure. This can be done by using a file system that supports journaling or by using a database that supports WAL.
5. Test the WAL implementation thoroughly to ensure that it works as expected in case of a system failure. This can be done by simulating a system failure and verifying that the system is able to recover to a consistent state using the logs.

By implementing WAL, you can ensure that the Broker Manager will be able to maintain data consistency and durability, even in the event of a system failure.

Virtual Machine setup and management (VM):

A Virtual Machine (VM) is an emulation of a physical computer. You can run a few VMs locally, deploy your packages and test the cluster setup locally.

You can use Oracle's Virtualbox which provides GUI as well as CLI called `VBoxManage`.

Resources on VirtualBox:

- Managing Oracle VM VirtualBox from the Command Line
- VBoxManage - An Introduction to VirtualBox CLI with Examples | CyberITHub
- A Complete Guide to Using VirtualBox on Your Computer (nakivo.com)

Note: You might need to enable port-forwarding (create a tunnel between local port and VM's port) to enable inter-VM communication.