**CS60002 Distributed Systems**

Assignment 2: Distributed Queue with Partitions and Broker Manager

Date : March 7, 2023

*Contributed by:*

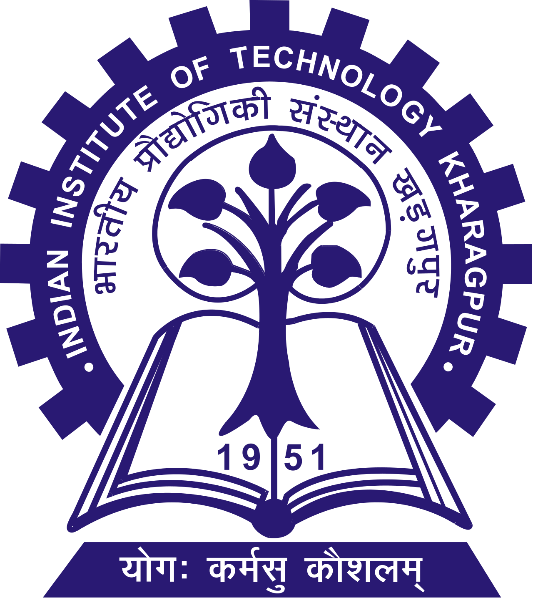
Esha Manideep Dinne 19CS10030

Amartya Mandal 19CS10009

Divyansh Bhatia 19CS10027

Rupinder Goyal 19CS10050

Anish Sofat 19CS10011



Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Spring Semester, 2022-23

# Assignment 1 Modifications

In Assignment 1, we worked with a single broker. There was a single database that contained all data that include information about all producers, consumers and topics, as well as all messages in the system.

A modified version of this same system is used to implement each broker for this assignment. Here, instead of grouping messages by topic as earlier, we are grouping messages according to partitions. Each broker has a database that contains all partitions assigned to it (a broker might have multiple partitions of the same topic). The broker manager redirects requests to a broker as required, and the broker perform all required functionalities that include enqueue, dequeue, registering consumers, getting size and probing.

Probing is a new functionality implemented to check the first message in the queue without marking it as viewed. This functionality is required to compare the time stamps of all messages that are at the front of the queue for all brokers to obtain the actual first message.

# Design Decisions

## Partitions

* + Each topic is broken down into partitions horizontally.
  + Each partition of a topic might be on different brokers.
  + Multiple partitions of the same topic might pe on the same broker.
  + Creating a new topic automatically creates one partition for the same.
  + Every new partition created is assigned to the broker that handles the least number of partitions.
  + Any log message produced by the producer is assigned to a partition in a round robin fashion, unless a specific partition number is provided by the producer.

## Broker Manager

* + The Broker manager handles the metadata of producers, consumers, topics, brokers and partitions. It also stores the mapping between consumers and topics & partitions.
  + Producers can choose a specific partition to produce, in absence of which a round robin algorithm is used to choose partition.
  + The broker manager can add or remove brokers. Each broker is run using a docker, and the broker manager spawns a new broker in the docker port provided.
  + Current algorithm assigns every new partition to the broker with least partitions for uniform load.
  + Also, we send a request to launch a new broker whenever the number of partitions per broker passes a limit for load balancing.
  + Services like List Topics, Register Producer are handled by the broker manager; the rest including Create Topic, Create Partition, Register Consumer, Enqueue, Dequeue, Size are redirected to the respective broker (might be multiple brokers).
  + Health check is implemented using separate function in all brokers, that is called periodically on a separate thread to obtain a heartbeat from all brokers.
  + The PostgreSQL database management service used for the persistent storage management is already implemented with an in-built write-ahead-logging mechanism that is automatically enabled.

[ Refer: [https://www.postgresql.org/docs/current/wal-internals.html](https://www.postgresql.org/docs/current/wal-internals.html%20) ]

## Broker

We carry forward the design decisions we had implemented in Assignment 1 (Part B i.e., after implementing the persistent storage for handling server crashes), while implementing the brokers.

* All the log messages will be available to the new users as well(give some reasonable explanation).
* Database is implemented using PostGRESQL.
* Django handles all the low level detail communication with the Database.
* There are multiple models(tables), they are Topic, Producer, Consumer,
* LogMessage mainly.
* There are two other linking tables which link consumers with their
* subscriptions (Consumer Subscriptions) and another table which links log
* messages to the consumers who viewed the messages (consumerViews).
* There are various fields like TextField, DateTimeField etc.
* Consumer and LogMessages are linked using a ManyToManyField .
* LogMessages are sorted based on their creation time order(by using the
* ordering = [‘created’] in the Meta Class of LogMessage).
* All the functions implemented in Part A have been implemented again in the
* file queue\_funcs.py using the databases.

## Load Balancer

* The main library behind all the async request handling is the requests\_futures which uses futures library to handle requests asynchronously
* We have redirected all the read requests to one of the read managers and the write requests to the main broker\_manager
* The assigning of read requests to the read managers is done in a round robin fashion
* We have utilised flask and python async to handle the redirection

# Model Designs

## Broker Manager

The following database models are implemented to store the metadata for the broker managers.

1. Persistent Storage
   1. Producer Model

Stores the producer id (int) and topic (string) subscribed

* 1. Consumer Model

Stores the consumer id (int) and topic (string) subscribed

* 1. Topic Model

Stores all topics (string) present in the system

* 1. Broker Model

Stores the port (int) and database name (string) for each broker (Each broker has its own database that is assigned some unique name)

* 1. Partition Model

Stores the id (int), topic (string) of the partition, partition number (int) and the broker it is assigned to.

1. In Memory Storage
   1. Lock

To implement a threading to prevent deadlocks and handle multiple asynchronous accesses.

* 1. Topic

In memory storage for faster access. Contains details of all topics that include the name, produces and consumers subscribed, partitions and an index for round robin allocation to partitions.

* 1. Metadata

Maps all topic names to the topic

* 1. Broker

Maps every broker to the number of partitions it contains. Used for load balancing.

* 1. Ids

Store three integer ids for brokers, producers and consumers to assign ids to them sequentially.

## Broker

The following database models are implemented to store the metadata for the broker managers.

1. Persistent Storage
   1. Log Messages

Stores all the log messages for each partition in the broker. Contains the message (string), creation time (DateTimeField) and partition (foreign key)

* 1. Consumers

Stores the consumer ids (int) of the consumers, and two many to many fields for mapping to their subscriptions and the messages viewed. The consumer model required for this assignment and the previous assignment is slightly different because, in this assignment, a specific consumer can be subscribed to multiple partitions in the same broker(which was not allowed as new consumer\_ids would be given in assignment 1)

c. Producers

Redundant model as the check whether a producer is registered to the topic or not is made at broker manager itself.

d. Topic

This model is modified to store the partitions present in the broker rather than just the topics. In the broker we modeled each partition of assignment 2 as a separate topic in terms of assignment 1. Instead of topic\_name being the only key, we made both topic\_name and partition\_no be the keys of this model

# Testing

## Setting up Dependencies

Run the following commands in the terminal.

- To setup and activate the virtual environment:

*assign2-venv\Scripts\activate*

- Install dependencies:

*pip install -r requirements.txt*

## Running the Project

- Run the app

*python loadbalancer.py*

- Run the write manager

*flask run -p 5001*

- Run read managers

*flask run -p %port*

\* Replace %port with port number for the manager replica

Setting up dockers for multiple brokers

- Setup docker

*python create\_broker.py %n*

\* Replace %n with the id number for the broker

## Running tests

- For testing all functionalities

*python -i tester.py*

All functionalities are now available to be used by the user very conviniently

* *createTopic(topic\_name) :*

Create a new Topic

* *listTopics() :*

List all topics present

* *registerProducer(topic\_name) :*

Register a producer to this topic

* *registerConsumer(topic\_name) :*

Register a consumer to this topic

* *enqueue(topic\_name, producer\_id, message, partition\_no : default) :*

Add a log message

* *dequeue(topic\_name, consumer\_id) :*

View a log message

* *size(topic\_name, consumer\_id, partition\_no : default) :*

Get the number of messages not yet viewed by the consumer

* *getPartitions(topic\_name) :*

Get the number of partitions for this topic

* *partition(topic\_name, producer\_id) :*

Create a new partition for this topic

* *add\_broker(port) :*

Add a broker using this port