Project Proposal - Pub-Sub Streams for Viewing Traffic
proj2_g4w8_g6y9a_i6y8_o5z8

Introduction

Kafka-Basics:

Kafka is a distributed streaming platform that allows clients to read and write data and stores this data in a fault-tolerant fashion. Kafka decouples the logic between those producing data and those consuming data, through a publish-subscribe protocol. Producers write to a topic (a named stream) and Consumers, only knowing the topic name, can pull data from this stream without having to know who the Producers are. Kafka guarantees successful writes are not lost even if there is a server failure.

Problem

Using our Kafka-inspired system, we want to generate heat maps representing traffic based on a bounding location represented by a list of coordinates that generate a closed shape. For someone to read the traffic data, they must also participate by sending their GPS coordinates. We assume that these clients accurately send their coordinates and are not adversarial in that they would spoof their location.

This is a distributed problem because we don't want all the connections sending GPS data to a single server due to the heavy load and single point of failure. Instead we want to implement a system where data lives in replicated nodes that form a cluster and can be retrieved from these clusters when a consumer client reads from them.

Approach

Our Proposed System
There are 4 (or 5 - depending how you count) types of nodes in our system:

1) Producers
2) Consumers
3) Cluster
    a) Leaders
    b) Followers
4) Server

Producers and Consumers are both client endpoints with their own API. Producers can create topics or send to any already-existing topic once they are connected to a Leader. Consumers can read any existing topic it knows the name to.

Topics in our system will be bounded areas on the map, roughly in the bounds of a city. Clients can only produce data to one topic at a time but can consume data from multiple topics.

Leader and Follower nodes act as a single cluster. They are separate from the Producer/Consumer relationship and are the implementation of topics. The data held by these nodes represent the state of a particular topic. Producer/Consumer API uses the server to find the connection to a particular Leader/Follower cluster in charge of a topic.

Implementation

Promotion of Follower to Leader:

In order to be eligible for a Follower to become a Leader, a Follower must have all the data for which a Producer observed a successful write.

```
                                               Leader1
     P1    -------------------W1----------------->
                                               ----------> Disseminate W1 to its N Followers
                                               <---------- Received N/2 (May tweak this #)
           <----------------ACK W1------------
     P1    -------------------W2----------------->
                                               ----------> Disseminate W1 to its N Followers
                                               <---------- Received N/2 (May tweak this #)
           <----------------ACK W2------------

     P1    -------------------W2----------------->    Leader1 dies
```

At this point, any of the followers that has all of P1s data up until W2 is eligible to be a Leader since the previous Leader (that died) did not return an ACK. If there are no eligible leaders (ex. one follower only has W1, another only has W2 and there is no path connecting each other to send their respective missing data), a leader cannot be elected and a Follower (picked at ‘random’) must notify the Producer.

Server

There is a single Server node that provides a Producer node with a cluster (1 Leader + X Followers) when it first creates a topic. Topics are per cluster (Should we change this? - Maybe the scope is too big to have multiple topics ….). On start-up of the Server node, it is responsible for creating a pool of nameless clusters and providing a nameless cluster to a Producer when a topic is created. The only time a Producer should communicate with the Server is when creating a new topic when it does not already have a corresponding cluster.

Sparsely-Populated/Empty Regions

In areas where there are not enough nodes to create a viable cluster to maintain traffic data we will need to implement strategies for sharing the state to nearby clusters. Let's say one topic is being heavily written to - thereby overloading LeaderA in ClusterA. We have 2 somewhat similar ideas we're toying with:

1) **Merging clusters**.

   ClusterA has neighbouring clusters, we offload the writes so now a Producer writes to an additional cluster and when one of them returns successfully (meaning it has disseminated the write to at least N/2 Followers), then it is considered successful. Previously a Producer would write to a single cluster (to LeaderA for example), but to minimize overloading LeaderA, we also write to LeaderB:

```
                |------ Producer ------|
             LeaderA                 LeaderB
           /         \              /       \
         F1          F5, F6, F7, F8         F2
```

   Diagram 1

   In this diagram, we will have some number of Followers that overlap. So we are essentially doing a semi-merge on the clusters - allowing some (not-yet-decided overlap).

2) **Splitting clusters.**
ClusterA's Leader was being hammered and is the single point between the Producer and the Cluster. In ClusterA, elect a secondary Leader (LeaderB), and this ends up following the **Diagram 1**. The main difference is that we are using our own cluster rather than borrowing another underused cluster.

Data Consistency

We need to make sure that the followers have data that is consistent with the leader state so that the integrity of the cluster is maintained even during leader failure.

Failures and Rejoining

Follower nodes are replicas of the Leader and any of these nodes can fail. If a Follower fails, a Leader checks if it meets the minimum number of followers. If it does, it can continue to operate as if the follower did not fail. If the number of followers dips below the minimum, the LeaderA makes a request to other Leaders to give up their own follower so it can join LeaderA. If a Leader fails, all the Followers of the Leader elect a leader based on the consensus protocol.

Once a new Leader node is elected, it must notify the Producer and from then on, any data from the Producer is routed to this new leader.

When a Leader node has failed and a Producer attempts to write to a topic, this will be a blocking operation until either a new Leader has been elected or there are no nodes that are eligible to be a Leader node.

Consensus Protocol

Not decided yet