Code Location: `https://github.com/Josh-Larson/AFIT-CSCE689-HW4-S`

1 Server Architecture

The general server architecture is easy to visualize as a triangle: the antenna simulator feeds the replication servers as well as the local database, the replication server feeds the local database, and the database effectively deduplicates at some point before writing to the file. This is managed using database flags on each plot, where new plots are marked as such, and fixed time skews have adjusted node IDs.

2 Communication Methods

2.1 Implementation

For this project we used TCP for short-lived data transmission, with AES as the authentication mechanism. The other replica server locations are stored in a local CSV file. Every 20 seconds, new data is sent to the other replica servers if they are online.

2.2 Theory

Since this communication method is effectively UDP over TCP, it would be more efficient to switch to UDP. If the replication data happens to overflow the maximum size of a UDP packet, we can run multiple replication transactions. Utilizing only AES with a distributed shared key is also a non-ideal security solution, and while the possible security solutions could easily fill two pages, it's sufficient to say that it is more secure to use TLSv1.3 over TCP instead.

3 Consistency Model

3.1 Implementation

For this project, we didn't use any consistency models. While we have to guarantee a specific data ordering, there is nothing depending on the result of that ordering–leading to my argument that there is no consistency models in use. To an extent there are data consistency guarantees.Since there is only one sequential write to the file, it could be argued that this technically has sequential consistency.

3.2 Theory

There are several ways to guarantee consistency and global ordering in a distributed system. Some of them are based on a unique naming of each consistency unit (conit), and some are based on an algorithm for agreement on ordering. Examples of unique naming are enumerated in the naming section, and we will soon see why that is the only way to guarantee a strict global ordering. First we will start with coordination techniques based on the information we currently have in the system.

4 Coordination Techniques

There are many algorithms for guaranteeing a global ordering of data in distributed systems. One of the best is vector clocks, which provides a great balance between fault tolerance and speed. In this case, since nodes do not replicate non-local data, vector clocks are equivalent to lamport clocks. Other methods are largely based on mutual exclusion in one way or another. Using just mutual exclusion, an option is to elect a leader to provide the global ordering of all the data. Extending that model slightly to provide higher read/write throughput, we could also use a distributed hash table based on a key associated with each drone plot. Finally, we could use Paxos, which is effectively a leadership election for every data point to guarantee a global ordering.

Starting with lamport clocks, we begin to see the recurring theme for the issue that plagues each algorithmic method listed above. The only guaranteed unique key we have is time-based, where we either increment the ID for every plot we find or we use the timestamp listed within each plot. Assuming that each drone plot timestamp is unique (separated by 5) and will always increase in value from the antenna, we can use that for our vector clock. This is therefore equivalent to using an incrementing counter for each plot, and storing it with each plot.

Applying lamport clocks to our consistency problem, a duplicate plot come in: they have different lamport clock timestamps, because they're from different replicas. The best we can do now is sort the duplicate according to the other values we have received from that other replica. If that same plot comes in again, we will see that it has a duplicate lamport clock value and we can discard it; which does not solve our core problem of removing all duplicates regardless of node.

Instead of using clocks we can try another coordination technique to provide the global ordering we're looking for: paxos. Since choosing a leader and utilizing a distributed hash table are equivalent to using paxos assuming no faults, I will focus on paxos to prove that the others won't work either.

First, a review: paxos is effectively a multi-layered agreement algorithm. If a node wants data to be written to the global list, it starts the algorithm as the leader and begins collecting the agreement of a majority of writers. After each writer agrees on the next plot to add to the global list, the readers agree on the new value to be added. Once the readers acknowledge the new plot, they commit the value to their local databases.

As we can see, paxos will guarantee a global ordering, but what ordering does it decide on? An ordering. Nothing to do with timestamps or duplicates, just a global ordering. Surely we can add a caveat to paxos to only accept new timestamps, though. If we do add logic to sort by timestamps, that does not fix the relative skew by each node and the drone may still jump back and forth in our final global ordering. This is the reason why we fundamentally need a unique name for each drone plot to reach a consensus about the actual flight path of the drone.

5 Naming

5.1 Implementation

Before jumping into a suggested naming scheme for the drone plots, a note on the current implementation of the server. There are three parts of the system that are named: each drone, each node, and each replica server. The drone is identified by a unique ID in our antenna simulation input, and each node is identified in a similar way. In a separate list, each replica server has its own name based on the IP/port it is bound to. The two most important names are the drone ID and the node ID, because they are critical to the deduplication algorithm.

5.2 Theory

Now that we have established that algorithmic methods for removing duplicates are not sufficient, we can explore ways to create a logical global ordering for the drone's flight path. There are two straightforward ways to ensure no duplicates:

1. Sending an incrementing ID from the drone with each new plot
2. Sending the current GPS time from the drone with the lat/lon

We can use the incrementing ID or the GPS time as a unique key. If any replication server tries to add a plot that conflicts with an existing value on another replication server, the operation would fail immediately if the plot already exists. These methods may seem like cheating, because it's obvious how to remove duplicates. That's a good thing. The correctness of a software system should be obvious.

6 Deduplication Algorithm

The deduplication algorithm is split into three parts: searching for time skews, updating the time skews, and removing duplicates. I assume that the time skew stays constant throughout the lifetime of the servers, there is at most one duplicate within a 15s window (guaranteed by sysadmin), and each lat/lon are within 1e-5 of each other.

The first step is searching for time skews. This is an $O(N^2)$ algorithm where for every data point, I search for duplicates that are within 15s of each other. If found, the time skew between the nodes is recorded.

The second step is updating all of the time skews to match the lowest numbered node's relative time. For each plot, I search for the skew to the leader's relative time. If I did not observe a direct skew between the two, then I chain relative skews until one matches. If no match is found, I don't update the skew.

Finally, I sort by time and remove all duplicates that are now next to each other with identical times. This is run every time a new data point comes in to provide the most up-to-date information to whoever may want to query this in the future.

7 Acknowledgements

I, like many others in this class, owe a large part of my success on this assignment to Lt Vincent Bownes spending so much time and effort on this. Without his lessons learned on state machines, architecture of the replication servers, and the data plots, I may not have been able to complete this assignment.