Design

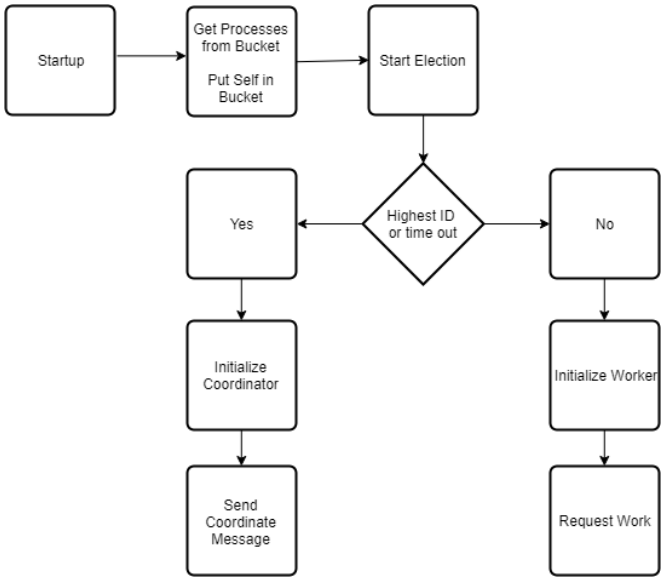
I chose to use the Bully algorithm for distributed election. The algorithm is relatively more straightforward to implement than the alternatives. Although it is not a scalable as some of the other options, the low requirement on the maximum number of processes means it is still suitable for this assignment. The ID in this case is a UUID generated by each process on startup and uses lexicographical comparison to determine which ID is greater.

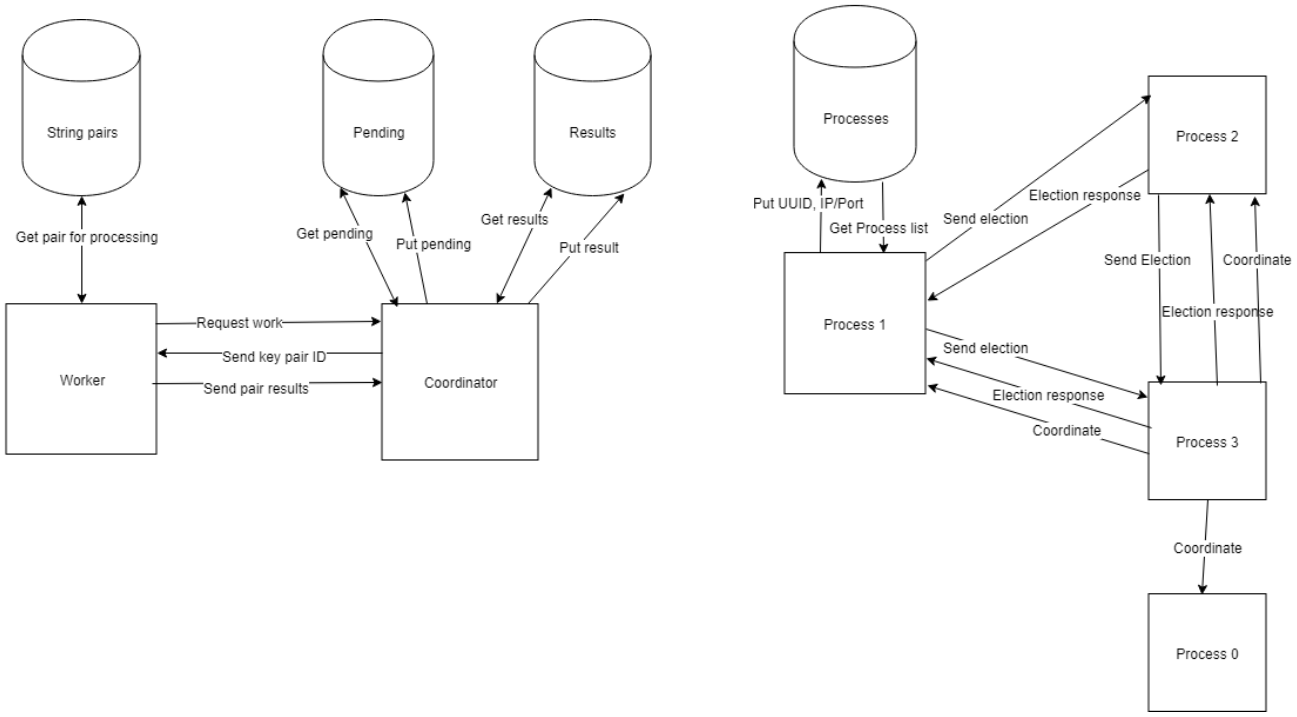
I created a total of 4 buckets on S3 to hold parts of each stage of the system. The **String Pair** bucket holds the complete list of string pairs. This bucket is never modified and is only read from. Workers read from this bucket when given a string pair ID to work on and the Coordinator lists the object keys in this bucket to get the initial set of string pairs that need to be worked on.

The **Pending** bucket holds the list of string pair IDs that are currently being worked on. These IDs match the object names from the String Pair bucket. The current Coordinator writes an ID to this bucket whenever it responds to a work request from a Worker and deletes from it when the result for a pending job is received. Newly elected Coordinators also read from this bucket to get the list of pending jobs. Each entry in this list is assigned a timeout and removed from the bucket to be returned to the work pool if a submission for this pending string pair is not received in time.

Similar to the Pending bucket, the **Results** bucket holds the completed list of string pairs and their distances in the form <ID#, distance>. The Coordinator writes the result to this bucket when receiving a submit work request from a Worker, at the same time it removes the string pair from the Pending bucket. Newly elected Coordinators also get the list of completed jobs from the Result bucket and remove those jobs from its list of available jobs retrieved from the String Pair bucket.

Finally, the **Processes** bucket is used to keep track of the currently active processes. The key of each entry is the unique ID of the process, which is used for election. The body contains the IP and port of the process. Processes immediately write their own generated UUID and IP/port binding to the Processes bucket on startup. They also retrieve the list of other processes in order to send election requests to all current processes with a greater ID.





Observations

I noticed that the coordinator node tends to stabilize as more processes are created using the Bully algorithm. Initially, each new process has a good chance of becoming the coordinator as its randomly generated ID may be equally likely to be above or below the coordinator's ID. However, as more processes are created, the coordinator ID gets gradually higher until it reaches a fairly high percentile and new nodes are no longer likely to have a greater ID.

The Bully algorithm also requires sending a lot of messages each time a process starts up or is downed. There is also a delay when the coordinator becomes unresponsive as each process must wait until a timeout expires on whatever its current request is before initiating an election. Having each active process simultaneously initiate an election means a lot of messages must be sent before a new coordinator is chosen. The new node with the highest ID must also wait for its election request to the downed coordinator to timeout before continuing on.