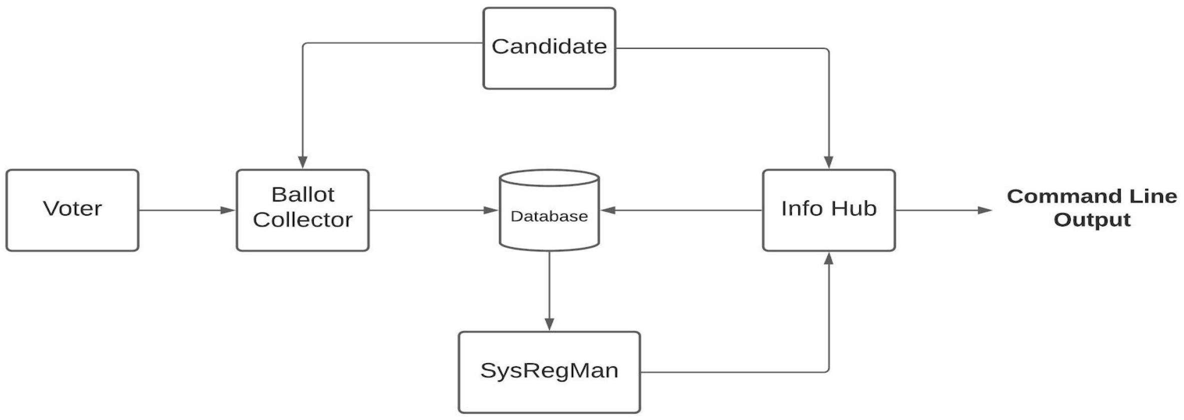
Final Report

Our project is a model of an electoral system. It consists of a voter module, which sends votes to a ballot collector. This ballot collector is also connected to candidates, in order to verify both the candidates and the votes made. The ballot collector then passes the verified votes to the Cassandra database, which is then obtained by the system registration manager. This then sums up the votes from Cassandra and sends the final result to the Information Hub where it is displayed. Adding data to a Cassandra database during the overall process ensures fault tolerance and keeps data persistent. The Information Hub is also connected to candidates in order to display their information. The 2 main technologies used for connecting this distributed system were Netflix Eureka and Apache Cassandra.



Voter

The starting point of the distributed system. Voter takes in a predefined array of voters stored locally, consisting of voter ID, name, region and the candidate they've voted for. It is created as a Netflix Eureka service and is registered to the previously created Eureka service registry.
It then uses a POST mapping REST endpoint to send the predefined array of voters to the ballot collector which has also been registered on the Eureka service registry. Once all the votes have been sent, the module sleeps for a period of time before then calling the System Registration Manager in order for it tally the final votes .

Ballot Collector

The Ballot Collector module is used to count the votes from different regions and store the partial results in a cassandra db. The Candidate module sends a POST request to Ballot Collector with the list of candidates running so that votes can be checked to see if the vote is for a valid candidate. The Voter module sends a POST request to Ballot Collector for every vote being cast.

Candidate

The candidate module is responsible for the retrieval of the candidates stored in the cassandra database. These candidates are stored locally first, a POST mapping is then used to send the candidate data to the information hub for registration. Once they are registered with the information hub, the candidate data is then stored in Cassandra. Candidates are also sent to the ballot collector for vote verification to make sure votes are only counted if they are for candidates that are registered.

System Registration Manager

Once the votes have been stored in cassandra by Ballot Collector the System Registration Manager will read the partial sum results and calculate the total number of votes for each candidate. The total votes for each candidate is then sent to the Information Hub using a POST mapping.

Information Hub

The Information Hub's main functionality is to display information to the users such as candidate names, party alignment and their manifestos. This allows a voter to align their vote with the correct candidate. The Information Hub also has APIs there to check if a candidate has already been registered and to register a candidate using a RESTful architecture. The candidate information is stored in a Cassandra database which is queried for the above functions.

Pros and Cons

By using the Cassandra database, there is a lot of potential for scaling (i.e. we can run hundreds of elections and still store our historical data) and Cassandra offers fault tolerance and replication for data availability and providing no single points of failure. One of the main issues faced with this electoral system was the reliability of storing all our data and getting the candidate data. This was solved through the use of our two distributed technologies - Apache Cassandra and Netflix Eureka. The use of a distributed system for our original problem model (i.e. an election system) makes sense intuitively, as in the real world an electoral system uses multiple different components (e.g. voting booths, people tallying the votes, the voters, etc.) in order to run an election.

One of the main issues with this project was the learning curve for Apache Cassandra. It was a distributed technology that none of the team were familiar with and it had quite a tricky setup. Once it was set up, we then had to learn it's querying language and the Java client drivers being used (i.e. Datastax). With Eureka, implementing the load balancing was also quite a difficult hurdle as it didn't seem like an important requirement when we started the project. However, even though the implementation and familiarisation with the technologies was difficult, the benefits were evident since it was a system with potential to scale, emphasized data availability and has potential of strong load balancing.