Consensus Protocol Algorithm

Suyash Datt Dubey   Student ID: sdd1n17

I implemented the inter process communication using Sockets and ServerSockets. The coordinator uses ServerSocket and Socket to establish a TCP connection with the participants. It contains a nested ServerThread class, which includes a thread that handles communication with participants. It also processes the “JOIN”, “OUTCOME” and “TIE” messages sent by the participants. Once all the participants have joined, the coordinator calls appropriate methods to send the “DETAILS” and “VOTE_OPTIONS” to each participant.

Participants also use Sockets to communicate with the coordinator. The Participant class contains a nested CoordinatorThread class, which includes a thread that handles communication with the coordinator. The thread sends out the “JOIN”, “OUTCOME” and “TIE” messages to the coordinator and processes the “DETAILS” and “VOTE_OPTIONS” messages sent by the coordinator. The Participant class also contains a nested ListenerThread class, which includes a thread that handles communication with other participants. The thread consists of a multithreaded TCP listener. It processes the “VOTE” messages sent by other participants. Once vote options have been received from the coordinator, the ListenerThread is started, and participants send out their votes (via TCP send) to other participants in the first round of voting (provided the socket hasn’t timed out). This is followed by a mandatory second round of voting in which participants send out the new votes they received in the first round. Next, each participant computes the outcome. If consensus is achieved, it sends the “OUTCOME” message to the coordinator, which the coordinator displays. If consensus is not achieved, the participant sends a “TIE” message to the coordinator. The coordinator, in turn, sends each participant a reduced set of vote options. In the case of consecutive ties, the coordinator keeps sending reduced vote options until there is only one vote option left and the participants reach consensus.

The solution I have implemented for the consensus protocol follows the specifications. Since two rounds of voting have been implemented, the coordinator reports the correct outcome when there are no failures as well as when one participant has failed due to either type of failure. Ties are resolved by the coordinator restarting the vote with fewer options.

The participant clears the vote options and votes it received once the outcome has been sent so that in case of a tie it can receive new vote options from the coordinator and use those for voting. Two rounds of voting take place by default, regardless of whether a participant fails. The CoordinatorThread sleeps for a second before starting the first and second round of voting and before sending the outcome to the coordinator so that all participants have time to get up to speed. If a participant fails, it sends its port number to the coordinator, after which the coordinator displays a message stating that the participant has failed. Participants choose votes at random. In case of a tie, the participants send out a “TIE” message to the coordinator so that it can send them the reduced vote options and restart voting. If the participant has a non-zero failure condition, it fails and the Socket it uses to communicate with the coordinator closes. If there is a tie and only one participant is running, a random vote option is chosen and displayed by the coordinator. The coordinator exits once all the outcomes have been received and the results have been displayed. Participants exit once consensus has been achieved.