Fault tolerant consensus protocol for distributed database transactions

MOHD ASIF KHAN
202IT013
UNDER GUIDANCE OF
PROF. ANANTHANARAYANA V. S.

ABSTRACT

Defining a protocol for commit consensus problem in distributed database transactions is not an easy task because we have to think about a fault tolerant system. In this paper, Author presents a fault tolerant and simple mechanism designed for solving commit consensus problem, based on replicated validation of messages sent between transaction participants.

PROBLEM

In the consensus problem, a collection of processes called participants cooperate to choose a value. We have make sure that the value is same at given time at all the participants.

ENVIRONMENT DETAILS

- Ubuntu 20.04
- Console application running on different application ports (for distributed environment).

TECHNOLOGY

- Go is an open source programming language that makes it easy to build simple, reliable, and efficient software.

- Noise is an opinionated, easy-to-use P2P network stack for decentralized applications, and cryptographic protocols written in Go.

- PostgreSQL is a free and open-source relational database management system emphasizing extensibility and SQL compliance.

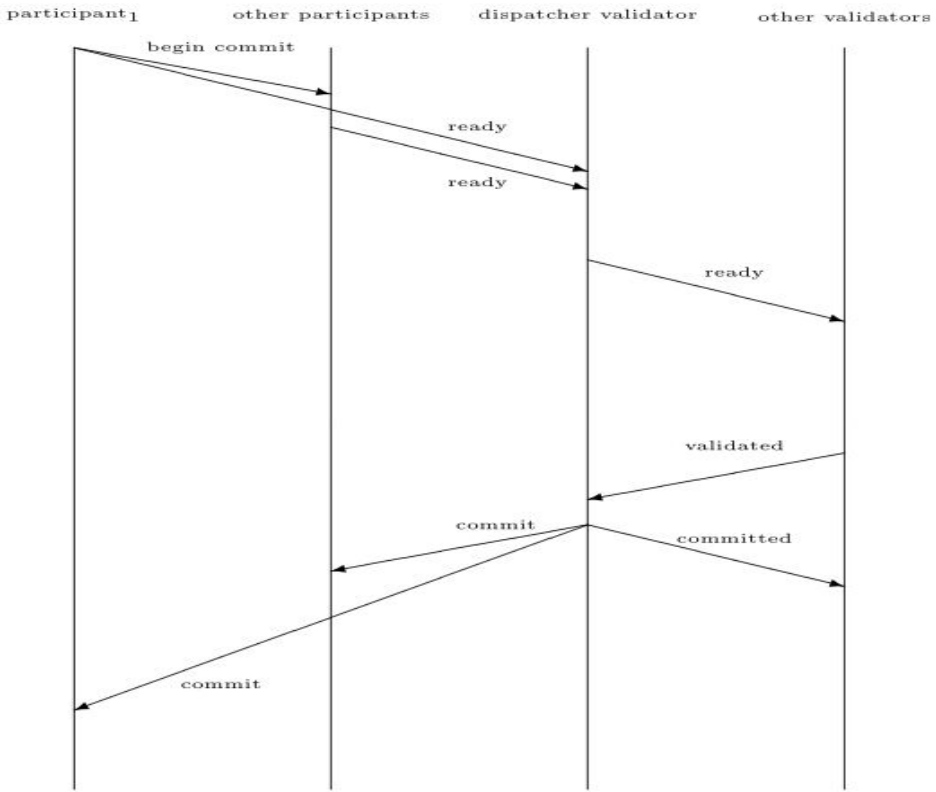
ASSUMPTIONS

In this paper, the algorithm do not consider Byzantine failures, assuming that all the messages are delivered correctly and exactly once. Byzantine failures refer some issues initially presented under the name of “Byzantine Generals Problem”. This describes the situation when a group of generals, each controlling some troops, must agree on the next action they should take. This is done by sending messengers that can fail their goal (similar to distributed nodes failures). Moreover, another problem is that some generals may be traitors, in group or individually, falsifying messages in order to confuse the other loyal generals.

METHODOLOGY

- n−1 “begin commit” messages, sent by the node which initiates the transaction to the other participant nodes
- n “ready” messages, received by the dispatcher node from all participants
- 2n(m − 1) validation messages; for every “ready” message, the validation process uses 2(m−1) messages sent between dispatcher node and the other validator nodes
- n “commit” messages, sent by dispatcher node to all transaction participants
- m − 1 “committed” messages, also sent by dispatcher node, but for all validator nodes

ALGORITHM ILLUSTRATION



Validator node problems

A blocked validator will be identified by the dispatcher validator, based on the heartbeat messages that the latter will regularly send; when this happens, the validator is taken out of the network automatically. Whenever a validator stops its execution, the system continues to work correctly if the initial majority can be achieved.

After the problem is solved, the validator can re-enter in the network and first of all it has to find out which node is the dispatcher (it will start a communication with other node and the latter will respond with the information regarding the dispatcher). After the dispatcher has been found, the node has to get the list of current states of transactions that are about commit. After this list is saved in the local memory, the new validator sends a confirmation message and after that it will be accepted in the network.

Participant node problems

When the participation node stops and then come alive ,its responsibility is to ask dispatcher for REDO logs of transactions which happened while it was dead.

If any of the participant node is not alive at the time of the transaction, Transaction will be rollbacked.

Dispatcher node problems

When the dispatcher has a technical problem and stops from working, based on the heartbeat messages, the other validators will start an election for a new dispatcher. The system will be blocked for some moments, until the election ends. After that, the transactions will continue to run in the normal way.

DISPATCHER ELECTION

- Each node generates random numbers in (0, 1) interval. If generated numbers is greater than the chosen threshold, then it will send to other nodes the generated number and it votes for itself;
- When one node receives the number generated by other node, it has to vote for sender node if it has not voted before for a bigger value in current round of vote; moreover,it sends its greatest random generated number and the ID of the last message received from the broken dispatcher in its response;
- The node which receives all the positive votes will be the coordinator in order to run the random roulette wheel selection using all the numbers received from other nodes; the winner will be the new leader;
- When one node receives a negative vote response, it will invalidate its state;
- After the leader is chosen, the coordinator will send it a message and after receiving it, the leader will broadcast a special/heartbeat message and will apply the logic regarding validated messages consistency.

IMPLEMENTATION

RESULTS

BASE PAPER

Performance test was made using 5 nodes running on distinct virtual machines and the consensus for a transaction finished in 235 milliseconds in average, with a minimum of 140 milliseconds and a maximum of 313 milliseconds. In 90% of cases, consensus was reached in at most 289 milliseconds.

My Results :

I used 3 nodes as validators and 2 as participants. For 10 transactions average time taken to commit in ms.

Transaction	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
Time taken	609	39	51	42	44	49	41	43	42	81

Minimum : 39ms , Maximum : 609ms , Average : 104ms

CONTRIBUTION

- Instead of REDO Logs which is stored in disk for participant nodes I used memory buffer for same also.
- Here a transaction list of all the transactions which is stored in memory is present at all the nodes.
- Whenever a node(participant,validator) dies and come back alive it will ask for transaction list from dispatcher.
- Dispatcher also sends the list to all the validators. So when dispatcher dies then all the validators have list and they can become new dispatcher since they have transactions list.

REFERENCES

- [Marius Rafailescu, Mircea Stelian Petrescu] Fault tolerant consensus protocol for distributed database transactions
- [Marius Rafailescu] FAULT TOLERANT CONSENSUS AGREEMENT ALGORITHM