DS1 Project – Documentation

NAMES

1. The actors

Our simulation relies on 2 types of actors only: externalClient and Participant. The class Node from which Participant inherits is only a container made to extend the AbstractActor. However, it still contains some useful getters and setters, variables as well as a few generic methods regarding delays, multicast and logging.

Practically, only one client object is needed throughout the simulation since it only transmits requests to the different nodes of the system. N\_PARTICIPANT+1 (e.g. 6 in that particular case) Participant actors are created at the beginning. One is declared as coordinator and the others as simple participants. Each node has an integer id assigned, 0 for the initial coordinator and 1 to N\_PARTICIPANTS for the other nodes.

1. externalClient

This class only possesses a constructor, a Props method (necessary for the Akka actor creation) and one receive method for displaying the reception of read responses.

1. Participant

This class is the most important one: it contains all the methods for the coordinator and participant nodes. There are two constructors and two props methods for the initial actor creation, allowing to differentiate the creation of the first coordinator from the regular actors. Some useful getters and setters are declared below.

The bulk of the system relies on reception methods, divided into 3 behaviors:

1. Standard: This is the default operating mode, defining the way the system works during and epoch. It can process StartMessage, Update, WriteOK, UpdateRequest, ReadRequest, Timeout, CrashedNodeWarning, ElectionMessage, Acknowledgement, Heartbeat and CrashRequest classes.
2. Election: This receive behavior is triggered when the node in standard mode receives an election message. It is made to prevent the system from processing confusing updates during coordinator election. It is able to process: CrashedNodeWarning, ElectionMessage, ElectionAck, Synchronization, Update (under certain conditions) and Timeout messages.
3. Crashed: When a node enters crashed state: it switches to this behavior that prevents it from processing any new message and from broadcasting anything.

This class also contains a couple helper methods to factor some code, for instance concerning the timeout scheduling or checking the quorum.

1. The messages
2. Acknowledgement

Message sent by a participant after receiving an update from the coordinator. It contains the coordinates (epoch, sequence number) of a request and the sender id (to ease communication).

If the coordinator receives more than quorum acknowledgements, the update is enforced with the sending of a WriteOK message. The sequence number is then updated.

1. CrashedNodeWarning

Message sent by a node to all others when it detects that another node has crashed (crash detection *via* timeouts). It only contains the id of the crashed node.

1. CrashRequest

Message used for testing purposes only: switches the destinated node to the crashed state where the node can no longer receive nor send messages.

1. ElectionAck

This message is sent by a node to the previous node in the election ring topology, to make sure the election token has been passed along. It contains a reference to the relevant ElectionMessage as well as the sender id. The previous node cancels the timeout on reception of this message. If no ElectionAck is received in the timeout duration, the next node is considered crashed and the token passing is retried to the node after the next one.

1. ElectionMessage

The election message is a token circulating between nodes on a ring topology based on nodes id. It is first emitted by a node detecting the crash of the coordinator. It contains the epoch and the emitter id to identify the message. It also contains the list of candidate nodes in the election as well as a list containing the sequence number of the last stable update they implemented. Upon first reception of such a message, a node switches to the election behavior and adds its information to the token before passing it along. Upon second reception, the node can determine which node won the election and, if it won, assume leadership, and terminate the election process.

1. EndEpoch

This message is sent by the newly elected coordinator just before closing an epoch. This empty message triggers the standard receiving behavior and initializes the new epoch on all participating nodes.

1. Heartbeat

The heartbeat message has a double use: received by the coordinator after a heartbeart timeout (shorter that the participant heartbeat timeout), it means that it is time to send a new heartbeat to other participants. Received by a participant, it resets the timeout made to periodically check on the coordinator.

1. ReadRequest

This message is a simple read request, when received by a node, the node directly replies to the client the stored value with a ReadResponse message.

1. ReadResponse

Message sent by a node to reply to a client ReadRequest.

1. Request

Empty class from which inherit both ReadRequest and UpdateRequest

1. Synchronization

Message sent by a newly elected coordinator to terminate the election process. It contains the last stable update to implement by all other nodes. When received by the participants, they implement the last update contained inside and, if they have acknowledged a more recent one, they send it to the coordinator.

1. Timeout

A message scheduled to be sent by a node to itself after a certain time duration. Upon reception, depending on the type of timeout, different actions can be taken. For instance, the crash detection procedure. The types of timeout are: UPDATE, WRITEOK, HEARTBEAT, ELECTION, ELECTION\_TOTAL, ACK and SYNCH. Each type has a different time value that can be set independently.

1. Update

A message sent to all participants by the coordinator after it received or was forwarded an update request. It contains the epoch and a sequence number to identify the update, a value to implement, the sender id and two Booleans: one for the validation status of an update and another one to differentiate between the consolidation of an epoch before starting a new one, and a regular update.

Updates are stored by each node in a waiting list that also works as a version history.

Upon reception of an Update message, nodes are supposed to ACK the coordinator to warn the system they are ready to install the new value. At the same time, they set a timeout to detect if the coordinator crashed after the sending of the update.

In case of epoch consolidation, most recent acknowledged updates are forwarded by the participants to the coordinator that will decide to implement the latest if Quorum-1 participants share a same last sent Ack.

1. WriteOK

Message sent by the coordinator to tell the participants to implement an update. It contains the coordinates of the update (epoch and sequence numbers) as well as the sender id.

Upon reception, participant change the status of the update to validated in their history and install the said update. Then they update their value and the sequence number.

1. The main class

The class TwoPhaseBroadcast is the main file. The first part contains the creation of the actors and the settings of the system. The second part is composed of code blocks that trigger some test scenarios, to see the system operating and how it deals with failures.

Note that with such parallel systems, there is no guarantee of when exactly a crash request is executed when requested between standard requests.