Virtual Synchrony - Project Report

Distributed Systems 1, 2017 - 2018

Daniele Bissoli - 197810, Andrea Zampieri - 198762

1 Introduction

In this project, we have implemented a simple group communication mechanism that permits different members of a group to communicate through messages exchange. The management of the group is done in a centralized fashion through a special node, named *Group Manager*. As design requirement the system provide the notion of Virtual Synchrony, guaranteeing the synchronization between members.

1.1 Virtual Synchrony

Under the assumptions of reliable links and FIFO channels, *Virtual Synchrony* is a mechanism which makes all the member of a group agree on who is still operational within the group. It is based on the notion of View, which represent all the nodes of the group which are currently alive. A view can be seen as an epoch of the system, so that each epoch represent different state of the system.

2 Project Architecture

The project is divided into three main classes (*Node*, *Participant* and *Group Manager*). In addition, it contains the definition of all exchanged messages and the class in charge to manage the creation of the system. In the following paragraphs will be explained the three main classes.

2.1 Node

This is the main class, which represent a common group member (called also *participant*). It defines the main points of a group communication service working under the Virtual Synchrony. These are:

- reception of a message
- stability of a received message
- management of views
- view change mechanism
- management of flush messages

2.2 Participant Class

It a member of the group which continuously send and receive message from others. It extend the *Node* class providing more functionalities such as send of data messages and mechanisms to set the participant as crashed. In addition, it provides a finer implementation of some functionalities as flush messages management and view change.

The so called *Participant* is the implementation of an active node in the system: it sends messages to the other nodes and delivers the received ones according to the Virtual Synchrony logic.
It inherits all the attributes of *Node* (e.g. the list of received messages, the unstable ones (with respect to each view) and the queue for messages received too early), and adds some of its own in order to be able to control the flow and the behaviour of the actors. Other than trivial ones (messageID, MIN_DELAY, MAX_DELAY for handling the transmission of new messages) there are some status variables to enforce the wanted behaviour.

- ***justEntered***: it's used just once to trigger a new *SendDatamessage* that makes the process send a new multicast, and reissues itself with a random delay $d \in$ [MIN_DELAY,MAX_DELAY]

- ***allowSending***: as its name tells, when its value is **false**, the process won't send any new message to the others in the system

- ***crashed***: tells wether the process has failed or not (it's used to simulate the crash of a given process)

Stuff to discuss: - A2A message (state that we decided to distinguish it from common data message) - explain how flush management works (e.g receivedFlush.isEmpty()) - slightly different protocol from the one given by Timofei - how timeouts after flush timeouts are avoided

2.3 Group Manager

It is a *reliable* group participant responsible to coordinate view updates between group members. It is build on top of the *Node* class and therefore it share most of the characteristics of a group participant. In fact, it is only able to receive messages, but not to send. Moreover, it is in charge of detecting crashes, which is done through timeout mechanism based on:

- **missing heartbeat message** - when group manager does not receive a heartbeat message (message sent continuously by a cohort to notify that it is still operational) within predefined timeout

- **missing flush message** - when group manager expect to receive a flush message from a participant but it does not arrive within the expected predefined timeout

In both cases, the group manager will trigger a view change, creating a new view and sending it to every group member through multicast exchange.

Apart from previous tasks, Group manager also deals with requests from new participants to join the group. To respect the Virtual Synchrony, a new view is generated to keep track of new member acquisition.

3 Crashes and Joining

The system is able to tolerate silent cohort crashes by means of the flush protocol. To control cohorts state, it is possible to send them special messages that make them crash according to predefined behaviour. Crashing modalities are specified below:

- **crash at any time** - immediately set the chosen node as crashed
- **crash on sending** - set the node crashed when it will perform next multicast to the group. As a result, only half of the group members will receive the sent message. It is up to the *all-to-all* exchange, performed during view change, to make all the other nodes delivering the sent message.
- **crash on receiving** - set the node crashed when it will receive next message. In this way the node is not able to perform the delivery of received message before crashing.
- **crash on view change** - set the node crashed when next view change is triggered. Therefore, the chosen node will not be able to perform the all-to-all message exchange and to send flush messages.

The system also allows to add further members. To join a new participant, a request must be sent to the group manager, which will to fulfill it. The group manager will generate a new *id* for the new member and it will introduce the participant into the group, triggering a view change.