Fault Tolerant Distributed Systems

Group No-3

Orcun Yildiz 77178

Pushparaj Motamri 77160

Protocol Stack:

We implemented the Throughtput Optimal Total Order Broadcast using following Appia Stack.

SampleApplLayer
LCRLayer
FairNessLayer
TcpBasedPFDLayer
BasicBroadCastLayer
KillMeLayer
TcpCompleteLayer

A Brief Explanation of the Layers are below.

1. **SamplApplLayer**: This is the application layer , from which user can give commands to broadcast messages and start perfect failure detector.
2. **LCRLayer:** This layer includes the logic of LCR, uniform total-order protocol.
3. **FairNessLayer:** This layer implements the mechanism which ensures fairness with respect to broadcasting of messages.
4. **TcpBasedPFDLayer:** This layer implements the perfect failure detection mechanism.
5. **BasicBroadCastLayer:** This layer based on our request, it will send to the successor on the ring or to every process on the ring.
6. **KillMeLayer:** This layer is used to simulate the failure of the processes.
7. **TcpCompleteLayer:** This layer implements the mechanism of sending the messages to the processes using tcp.

Explanation Of Code:

We created ACKRecoveryEvent, RecoveryEvent, EndRecoveryEvent classes, which help in handling the crash of the process. We followed the same algorithm in the paper. The implementation can be found in LCRSession.java. We implemented pending list as LCRStack class. This class implements comparator interface, hence the comparator method in it implemented as per the comparison specified in the paper. LCRStack stores entries which are instances of LCRElement Class in an arraylist, which is sorted using the comparator specified previously. We implemented the predecessor method in the LCRSession.java which will give the correct predecessor even in case of failures of other processes on the ring. When an event has to be sent to only successor, we push a Boolean value true .This value is popped by the BasicBroadCast Layer .If the value is true it will send to the successor else to all the processes on the ring. The logic of fairness can be found in FairnessSession.java. In that we created a PiggyBackEvent which is used to piggy Back burst size along with other events. We implemented the fairness algorithm as specified in the paper.

Instruction to run the Code:

1.Start three consoles with the following command:

```
java -cp .:build:lib/appia-core-4.1.2.jar:lib/log4j-1.2.14.jar tfsd.SampleAppl -f
conf/process1.conf -n 0 -qos lcr &>test0.txt
java -cp .:build:lib/appia-core-4.1.2.jar:lib/log4j-1.2.14.jar tfsd.SampleAppl -f
conf/process1.conf -n 1 -qos lcr &>test1.txt
java -cp .:build:lib/appia-core-4.1.2.jar:lib/log4j-1.2.14.jar tfsd.SampleAppl -f
conf/process1.conf -n 2 -qos lcr &>test2.txt
```

2. *After running the commands,at the end of run give the following command to grep the delivered events to the application:*

```
for i in 0 1 2; do grep deL test$i.txt > $i.txt ; done
```

3. *Now , make the diff to check whether every process delivered the same sequence of the messages or not:*

```
diff 0.txt 1.txt …
```

TestCase: We set as Thread.sleep(50) so every process will send 20 messages in each second and also we set to total number of messages that will be sent as 3000.We evaluated with 3000 messages with and without failures.
TestCase2: We set different Thread.sleep(50 *(myRank+1)) so every process will generate messages in different rate and also we set to total number of messages that will be sent as (3000/myRank+1).We evaluated for fairness and the order was same at the end.

P.S-1: In the KillMe Layer,we are killing process 2 for simulating failures.

P.S-2: Logging level can be set via file Logger.java. (which is in src package).