A brief report on CPSC – 551 Project\_02

~ by Harshit Singh Rathod & Rahul Chauhan

**Step\_01: Creating configuration files.**

Created unique tuplespace/adapter pairs as below:

|  |  |  |
| --- | --- | --- |
| **ts\_name** | **ts\_uri** | **adapter\_uri** |
| Alice | localhost:12345 | 8000 |
| Bob | localhost:12346 | 8001 |
| Chuck | localhost:12347 | 8002 |
| Diana | localhost:12348 | 8003 |
| naming\_server | localhost:12349 | 8004 |

with below multicast address/port:

|  |  |  |
| --- | --- | --- |
| **Function/program** | **Address** | **Port** |
| subscribe | 224.0.0.1 | 54321 |
| recovery | 224.0.0.1 | 54322 |
| naming\_server | 224.0.0.1 | 54323 |
| Tsmanager(Alice) | 224.0.0.1 | 54324 |
| Tsmanager(Bob) | 224.0.0.1 | 54325 |
| Tsmanager(Chuck) | 224.0.0.1 | 54326 |
| Tsmanager(Diana) | 224.0.0.1 | 54327 |

**Step\_02: Creating logging and recovery server**

Created a recovery server **recovery.py** to log all write and take events and recover when a tuplespace starts. It listens and splits notification as **tsname, event, message/tuple** and pass the variable to **logrec.py**, which handles logging and recovery. It uses UDP protocol for communication.

**Observations:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test\_no.** | **Test case** | **Expected result** | **Actual result/Observation** |
| T01\_01 | A ts/adapter pair is started for the very first time. | A file(yaml) is created and  details are logged in the file. | Same as expected. |
| T01\_02 | A few write and take events happen. | Events are logged into the file. | Same as expected. |
| T01\_03 | A ts/adapter pair is restarted. | Tuplespace is recovered. | Tuplespace is recovered but log file has now **duplicates**. |

In the last test **T01\_03** the whole concept of having a recovery server is fulfilled but with a very

serious glitch i.e. the messages in the log file are duplicated. This is because all the tuplespace notify events through multicast and our server listen to the same message and creates a log for it. We need some new protocol/function to ask tuplespace to stop notifying the recovered messages.

**Step\_03: Naming and delivery**

Created tuplespace/adapter pair for naming server (as in Step\_01). Created a naming server **name\_server.py** which listen for adapter start notifications and logs all the tsname and adapter pairs in tuplespace as [tsname,adapter\_uri]. It also logs the name of other tuplespace as [“users”,[names of all the tuplespace]].

Created a command line client **name\_client.py** which reads the list of users from tuplespace, gets their respective adapter\_uri`s and delivers the message. If a tuplespace is down, it skips it.

**Observations:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test\_no.** | **Test case** | **Expected result** | **Actual result/Obesrvation** |
| T02\_01 | A ts/adapter pair is started for the very first time. | Naming server will write naming\_tuplespace with tsname and adapter\_uri, and the users list | Same as expected. |
| T02\_02 | A write command is written on the command line client. Consider 3 ts/adapter pairs have subscribed to naming server and all are up. | Naming client will deliver the message to all the tuplespace who have subscribed. | Same as expected. |
| T02\_03 | A write command is written on the command line client. Consider 3 ts/adapter pairs have subscribed to naming server and **one of the adapters is down**. | Naming client should deliver the message to all the tuplespace who have subscribed. If a pair is down, then deliver the message when it restarts. | Naming client delivers the messages **only to those tuplespaces whose adapter is up**. It leads to **inconsistency** as the data in all the tuplespaces is not same. |
| T02\_04 | A write command is written on the command line client. Consider 3 ts/adapter pairs have subscribed to naming server and **one of the tuplespace is down**. | Naming client should deliver the message to all the tuplespace who have subscribed. If a pair is down, then deliver the message when it restarts. | Naming client delivers the messages **to all the active adapter**. Since the tuplespace is down adapter won`t be able to pass message to tuplespace. It leads to **inconsistency** as the data in all the tuplespaces is not same. |
| T02\_05 | A write command is written on the command line client. Consider 3 ts/adapter pairs have subscribed to naming server and **one of the tuplespace restarts**. | The recovery mechanism will recover the state of tuplespace. Naming client will deliver the message to all the tuplespace who have subscribed. | Same as expected, only if messages are not lost when tuplespace was down. **Inconsistency may arise in this case**. |
| T02\_06 | A write command is written on the command line client. Consider 3 ts/adapter pairs have subscribed to naming server and **a new user subscribes the naming server.** | Naming server will add the adapter\_uri to naming\_tuplespace and update user list. Naming client will deliver the message to all the tuplespace who have subscribed. | Same as expected, but the **new user will not receive the previous messages posted to blog**. It leads to i**nconsistency.** |

**Step\_04: Replication**

Created tuplespace managers for respective tuplespaces **tsm\_alice.py, tsm\_bob.py, tsm\_chuck.py** It(ts\_managers) listens to the notifications and pass it to **logrec.py** which handle logging and recovery. It(tsm\_...) also writes the messages of others to their own tuplespaces.

**Observations:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test\_no.** | **Test case** | **Expected result** | **Actual result/Observations** |
| T03\_01 | Three ts/adapter pairs start for the very first time. | Respective ts\_managers will create a yaml file to log all the events. | Same as expected. |
| T03\_02 | A write event happens at one of the ts/adapter pairs(say Bob). | 1. Message will be written in Bobs tuplespace(by the client). 2. Message will be replicated and logged to other tuplespaces(by ts\_managers). 3. Message will be only logged in file by ts\_manager of Bob. | Same as expected. |
| T03\_03 | A write event happens at one of the ts/adapter pairs(say Bob) and one of the **tuplespace is down**(say Alice). | 1. Message will be written in Bobs tuplespace(by the client). 2. **Except Alice tuplespace** message will be replicated and logged to other tuplespaces(by ts\_managers). 3. Message will be only logged in file by ts\_manager of Bob. 4. **The ts\_manager of Alice will log this event**. | Same as expected. |
| T03\_04 | One of the **tuplespace restarts**(say Alice). | The ts\_manager of alice will listen to this event and recover alice tuplespace. | Same as expected, but it leads to **duplication** as well. Duplication of messages to other tuplespaces(not in log file). |
| T03\_05 | A write event happens at one of the ts/adapter pairs(say Bob) and one of the **adapters is down(**say Alice). | 1. Message will be written in Bobs tuplespace(by the client). 2. **Except Alice tuplespace** message will be replicated and logged to other tuplespaces(by ts\_managers). 3. Message will be only logged in file by ts\_manager of Bob. 4. **The ts\_manager of Alice will log this event**. | Same as expected. Although there will be **inconsistency,** when one of the ts/adapter pair restarts those message can be recovered from log file. |
| T03\_06 | A write event happens at one of the ts/adapter pairs(say Bob) and a new user joins(say Diana). | 1. Message will be written in Bobs tuplespace(by the client). 2. Message will be replicated and logged to other tuplespaces(by ts\_managers). 3. Message will be only logged in file by ts\_manager of Bob. | Same as expected. Diana will **not receive previous message**. If other users restarts then Diana might get previous messages. |

**Step\_05: Comparing two approaches**

As we have seen in previous observations that both the approaches have few complications and they fulfill the requirements partially. Here is the summary of both the processes.

* Complexity in implementation: The first approach is more complex than second in term of implementation. The first approach involves a lot of process(maintaining the adapter\_uri and users list,etc.) whereas the second approach have a simple ts\_managers and recovery mechanism.
* Reliability: Since the first approach stores the adapter\_uri in tuplespace and tuplespace`s are vulnerable itself, it is not very reliable. If the tuplespace of server goes down and restarts, then messages won`t be delivered to subscribed user`s even if they are running. The second approach is better than first in terms of reliability as it does not have to depend on tuplespace for anything. It can log all the events to a file, which is way better.
* Consistency: Both methods have consistency issues, but the second approach is slightly better than first. If a tuplespace is down and message in sent then, in first approach the message is lost forever whereas in second approach at least, the message will be logged in the yaml file and when the tuplespace restarts those messages will be written to tuplespace leading to better consistency.
* Fault tolerance: The fault tolerance of first approach is bad than second approach. The second approach has ts\_managers for each tuplespace which can look after for their own tuplespaces. Even if the respective tuplespaces are down ts\_managers will log the events and messages won`t be lost.