3 Phase Commit

Submitted By: Akanksha Bansal, Prateek Agarwal

Notable Points about the Protocol

1. Need of storing PRE-COMMIT at Participant

Below scenario shows why it is necessary to store the PRE-COMMIT state in the logs.
a. Coordinator has not received ACK from any of the participants for its request to PRE-COMMIT. Thus all the Participants are in UNCERTAIN state
b. The Coordinator times-out and will decides on COMMIT, but fails before sending the message to any of the participants
c. The processes being UNCERTAIN will chose their own new coordinator. New coordinator on seeing that all the processes are in UNCERTAIN state, will decide on ABORT.
d. Now when the old coordinator comes up, based of the recovery protocol defined, he will see the decision made as COMMIT by him and will COMMIT, which leads to an inconsistent state of the System.

Basically, when the coordinator has decided on COMMIT, and has even sent this message to *k* out of the *N* processes, and all of these *k* processes along with *N* fail, and then the rest of the processes fail. When these *N-k* processes come back they will have the state UNCERTAIN instead of PRE-COMMIT. They will decide on ABORT after choosing a new Coordinator because they believe that no process can have COMMITED.

2. Need of storing PRECOMMIT at Coordinator

Suppose all the participants were in PRE-COMMIT phase. Then a total failure occurs, now the recovering processes cannot decide on COMMIT. They will have to wait for a process with decision to come back or ultimately the coordinator to be able to say for sure they can ABORT or COMMIT.
Logging PRECOMMIT allows them to make the decision to COMMIT without any kind of uncertainty in this scenario.

Thus from 1. and 2. "No need to record PRECOMMIT" in DTLog can lead to violation of 3PC validity.

NOTE:

We wanted to highlight that logging PRE-COMMIT also raises an anomaly. Take a look at the scenario below:
a. Coordinator has decided on PRE-COMMIT. Before it was able to send any kind of message to the rest of the processes, it failed.
b. Now the rest of the processes are in Uncertain State.
c. They choose a new coordinator, which will decide on ABORT based on the States of the participants.
d. Now the Old coordinator comes back. Based on his state being recorded as PRE-COMMIT, it should ideally expect the protocol to be only going to COMMIT state. But as we have mentioned it is possible for the processes to decide on ABORT, therefore the old coordinator should check from other processes as to what was their final decision.
We are not faced with this situation if PRE-COMMIT is not logged, but then other inconsistencies might arise.

Running instructions.

```
$ java -cp src/ pkg3pc.Main <>
```

Structure of 3PC:

```
├── config.properties
└── src
      ├── pkg3pc
      │   ├── Main.class
      │   ├── ...
      │
      └── ut
            └── distcomp
                  └── framework
            ├── Config.java
            ├── ...
```

Behavior of the System in a nutshell:

Each process is created by calling the Main class with some arguments which define the process behavior once it has started. Each process maintains:

a. **Log file**: Maintains track of all the details that happened during the last transaction. This file is used to recover to the state the process was before getting killed.

Sample:

```
Oct 07, 2013 11:13:36 PM pkg3pc.Process initTransaction
INFO: NEW TX - WAITING FOR VOTE REQ
Oct 07, 2013 11:13:38 PM pkg3pc.ProcessBackground run
INFO: ..........................................................CURRENT UP SET  [1]
```

b. **Playlist instructions file** which will contain the list of all the successful transaction which have taken place up till now. Based of this file the process is able to restore its state.

Sample:

```
1$ADD$dhinchak$songs.pk
2$EDIT$dhinchak$songs.pk$dhinchak2$songs2.pk
3$ADD$dhinchak$songs.pk
```

Significance of various properties in the config.properties.

clean: If this property is set to true then the log fies and the playlist files of the processes are cleared and the processes start from a fresh state.

NumProcesses: Total no of processes in the System

TxNo: The no of transaction the system is working on. Based of this the process is able to determine the meanings of various messages which are being sent across the channel.

CommadN: represents the Command that needs to be executed at Nth transaction No.

Command: is the default Command the Coordinator will try to execute when the CommandN is not present in the properties File. Currently this File is read only when the processes start.

Arguments Given to Process:

pid : The process Id for the process which is being started

vote(boolean true/false) : The vote which the process will be giving to all the transaction requests coming its way.

deathAfter: if set to N this argument will tell the process to die once it has received N no of Messages from the process p. if set to -1 then this feature will not be enabled

deathAfterFrom: If set to N implies the process should die once it has recieved N (set in previous argument) messages from the given process p.

partialCommit: If set other than -1 and less than Total no of processes, let us say N, the process will die if it sends a Commit message to this process N.

How to run the code Sample:

Compile:

```
$ cd 3PC/src
$ javac pkg3pc/Main.java
$ cd ..
```

To run coordinator:

```
java -cp src/ pkg3pc.Main 0 true
```

To run participant:

```
java -cp src/ pkg3pc.Main 1 true
java -cp src/ pkg3pc.Main 2 true
```