CPSC 551 Project II

Justin Chin
Daniel Miranda

Tuplespace goes down:

Reliability:

In the event that a tuplespace fails, both approaches should maintain their reliability. Because the logic for getting name:address bindings and recovering tuplespaces is not contained within the tuplespace itself, all services needed by users (aside from those needed by the user whose tuplespace failed) are still available.

Fault Tolerance:

Both systems are fault tolerant in the event that tuplespace goes down. However, for the Event Logging approach, the set-up has a single point-of-failure, the recovery server and/or the log file itself. If either one of those fail, our system will continue to operate, but will lose the recovery functionality. Since the Replication approach does not utilize a separate recovery server to replay actions from a log file it is not susceptible to this same issue. Rather all of the tuplespaces are replicas of each other (ideally), therefore any of them can be used to recover, avoiding a single point-of-failure.

Correctness:

In both cases, tuplespaces correctness will not be affected. Each users tuplespace will continue to house all of the microblogs posted to it, aside from the one that has gone down, of course.

Adapter goes down:

Reliability:

In both approaches, the adapter is the intermediary in order to take action on a tuplespace. Therefore, if an adapter goes down, the system remains reliable if accessible without the adapter (e.g using a ruby client instead of python). In the Event Logging approach, when actions are attempted on a tuplespace who's adapter is down, the actions will be unsuccessful, and because nothing is written to the tuplespace, no events are multicast and logged. Similarly, in the Replication approach, if action is attempted on the tuplespace who's adapter has gone down, it

will not be successful and therefore there will be no replication necessary by the others.

Fault Tolerance:

In both scenarios, the set-up is fault tolerant. In both cases, the system can afford to have adapters go down without affecting the overall functionality of the users as a whole. Only the individual users will be affected, but the others will not be impacted.

Correctness:

Correctness for the Event Logging approach will persist. Since in this approach, all action taken on a tuplespace is logged, once the adapter goes down, no further action can be taken on the tuplespace, and therefore no further logging of that tuplespace will be necessary. This is in contrast to the Replication approach. In the Replication approach, an adapter going down inhibits the ability to perform action on the tuplespace, which is vital since it needs to be able to reproduce actions performed at other tuplespaces in the group. This would yield an outdated or stale tuplespace which would be incorrect and inconsistent with the other tuplespaces in the group.

Tuplespace is restarted:

Reliability:

A tuplespace being restarted in the Event Logging approach will yield reliability, assuming that it connects to the same adapter. If not, and a new tuplespace and adapter pair are started, then the Nameserver would need to be updated appropriately. However, this would also result in a 'dangling' adapter that was doing nothing (wasted resources). If enough of these situations occurred, it could lead to a system resource exhaustion and eventually crash. In contrast, in the Replication approach, a tuplespace restarting would render the system useless. This is because upon starting/restarting a tuplespace, replication will occur, but actions taken on the restarted tuplespace will be communicated to the others on the multicast group, causing them to take action, which in turn will be performed by the others in the group, so on and so forth. Essentially this causes a circular reference that will render the system in perpetual operation and useless to the users.

Fault Tolerance:

N/A

Correctness:

In both approaches 'correctness' is limited, if at all attainable. In the Event Logging approach, when a tuplespace is restarted, a recovery will take place. That recovery will not be correct even if it is the first recovery that the system has experienced. This results from the fact that each tuplespace is multicasting their own events, so for a single message, there will be N events logged, one for each tuplespace in the group. Once this recovery occurs, we get a duplicate record of the history as the recovered tuplespace multicasts its own write and take events.

In approach 1, the tuplespaces that are online from the beginning of the microblogging platform will be correct, with regards to content and its multiplicity. However, as soon as one recovery event is played, that tuplespace will receive N copies of the messages posted to the platform.

In approach 2, we have no correctness with respect to the quantity of messages. While the content is correct, a single message written to the platform results in a flurry of messages being bounced between all the tuplespaces participating in the multicast group. We are able to play events to a restarting tuplespace, however, this only exacerbates the message bouncing problem.

New tuplespace joins group:

Reliability:

Assuming that the new tuplespace properly reports and persist its information to the name server, the system will be reliable. As for the Replication approach, the same situation as a tuplespace restarting (above) would arise and render the system useless to the user.

Fault Tolerance:

N/A

Correctness:

Same situation as above, where we try to restart a tuplespace that has failed. Because tuplespaces are transient data stores, restarting one is almost equivalent to a new join. Neither approach offers much in the way of correctness. The first approach results in N copies of all messages being replayed to the joining tuplespace, whereas the second approach just adds another node to bounce messages around indefinitely (or until memory limits are exceeded).