Distributed Calendar Tool Using WebServices

CS 675 Distributed Systems Project Report

Kshitiz Bhattarai
George Mason University

I. Introduction

For this project, we implemented a Distributed calendar tool for workgroup. The project was implemented in Java using Web Services as a method of communication. We decided to use Rest Web Services, for which each server needed to be equipped with Tomcat Server with Jersey libraries to allow Rest-based communication. Hence, the calendar uses Tomcat web server as a server while browser is our primary client.

II. Chord Implementation

In this part, we will discuss about out our implementation of Chord protocol for this project. While it closely relates to Chord protocol, but for simplicity some part have changed.

User and Node ID:

For each Node its IP address is used to get the ID. We used Message Digest SHA-1 method to convert it into a number and then mod the result with 67. Similarly for each user the username is used to get the ID in a similar manner.

Network Topology

The Network is setup as a ring where each node has a successor (S) and a predecessor (P). There are special cases like a single node in which, the successor and predecessor are both set as the same node. For the two node case, each node points both of its successor and predecessor to the other. Each node also keeps tracks of successor of successor (SOS) and predecessor of predecessor (POP) in its finger table, which is required in case of node failure.

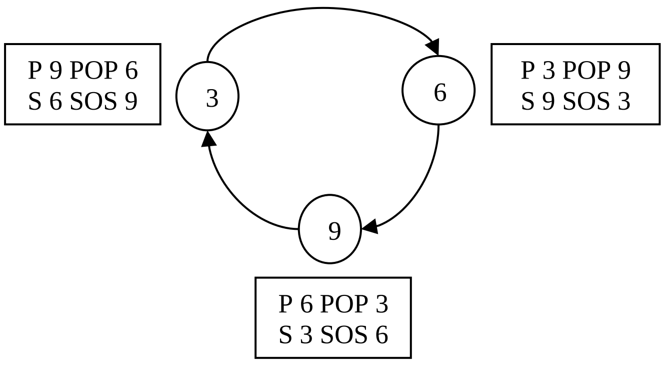

Figure 1 Network topology with their neighbors

Node Join/Rejoin

A node can join at any point in the ring. For join/rejoin the node that is joining the ring goes through following phases.

1. Node (N) gets the IP of one of the node in the ring from the user.
2. N then asks that node to get the IP of the head (the node with the smallest ID in the ring)
3. It asks the head to where it should join. Head then finds the correct successor where this node should join linearly going through the ring and returns it.
4. N makes it place between provided successor(S) and predecessor (P).
5. N gets all the appropriate users from P and set them as its primary. N then makes S as backup of these users.
6. N also makes sure it will backup of all the P's primary users and remove these from S's backup users. Thereby, doing the node balancing.

Figure 2 and 3 shows the condition of the ring before node 6 joins and after 6 joins. The side boxes display the primary and backup keys associated with each node.

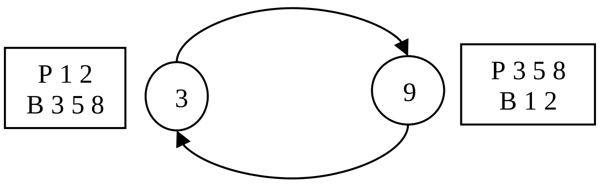

Figure 2 Ring with node 2 and 6

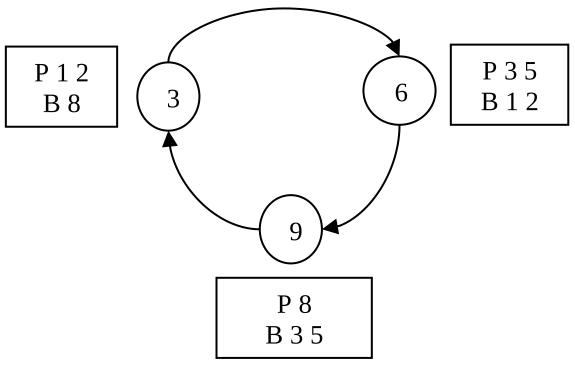

Figure 3 Ring after node 6 joins

Node Failure

Each node has P, S, POP and SOS in its finger table. Each node talks to P and S every 3 seconds and asks them to set the P, S, POP or SOS, in case of node failure the set request crashes thus allowing the node to check if they are alive or not. In case of failure of S, it sets S = SOS and SOS = null. In case of failure of P, it sets P = POP and POP = null and also goes through following phases to stabilize the ring.

1. N moves all the backup to its primary
2. N removes all the users from backup
3. N asks S to set backup of its primary users
4. N asks P to send copy of P's primary users to save in backup.

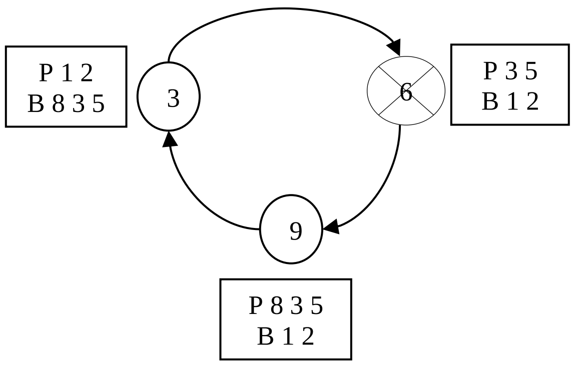

Figure 4 Chord Stabilizing after Node 6 crashes

III. CALENDAR

The Distributed Calendar tool is a web based application which uses Tomcat as the web-app container, with Jersey libraries to allow Rest-based communication. Each user has a persistent calendar object stored in a node. In case of our implementation, for a user to login and view his calendar, s/he can navigate to any node via browser http://NodeIP:8080/DistCalendar, and login using his/her username and password. Upon receiving the request, each node checks if it contain the user or not, if not it will forward the request to its successor. Once a node finds the user it will then display the calendar via browser, where s/he can further make changes. In case of node failure, user is navigated to successor where s/he will have to re-login.

Our calendar provides schedule, update, delete, view other calendar, notification services. Each service works in the distributed way as well.

There are three types of events that user can schedule, namely, public, private and group. Public and private scheduling works on the correct node that has the current user in its primary using 2-phase commit. So, for public and private, it will lock, schedule and save on backup, unlock, if it fail the entire calendar is restored from temporary backup that was created before the commit. The two-phase commit will ensure consistency on the replicated database.

For Group event, we follow the following principle,

1. Lock each member calendar
2. Make sure the event does not conflict with any user
3. If, all the users were able to lock and available, create a temporary backup and proceed with schedule and backup for all the members.
4. Unlock each member calendar
5. If in any phase anything goes wrong, like node failure, the initiator first rollback to previous state from saved temporary backup which was recorded before changes and the initiator asks all the rest of nodes to rollback.

Due to time limitation, the last step of rollback was not implemented in this project.

IV. PRIMARY BACKUP SCHEME

Each node keeps a backup of its primary users on the successor. Thus, in case of node failure, the backup will take place over primary allowing high availability. Since we have assumed that at any point only one node can crash, hence two nodes cannot simultaneously crash thereby allowing this implementation to provide high availability.

V. CONCLUSION

The Distributed Calendar tools works perfectly for the above mentioned cases but there may me other cases in which it may fail. One of the biggest problems that still exist with this system is that during stabilization either while node join or leave, the system should be allowed around 30 seconds to stabilize, for file IO to complete. For client side distributed transparency, we have JavaScript threads monitoring the availability of the server.

VI. INSTALLATION INSTRUCTION

1. Download the DistCalendar.war file.
2. Navigate to Tomcat manager and upload the file.
3. Repeat for each node.
4. Once all nodes are up open http://nodeip:8080/DistCalendar/rest/admin page for all the nodes.
5. On one of the nodes click the "Load the database"
6. For each subsequent node, type in the above node (that has the database loaded) ip address and click the "Join Chord". Allow 10-15secs for the ring to stabilize.
7. Once done navigate to http://nodeip:8080/DistCalendar/ and login.

The "shutdown node" button on the node closes the Tomcat server so press it with caution.