

Distributed Calendar

CS271 – Fall 2008
Introduction to Distributed Systems
Specification for Project 1
Due Date:s 12th November

October 27, 2008

Description

This project is to build a calendar application in a distributed environment using a distributed log. The distributed system consists of entities or nodes which want to maintain a local calendar as well as have a view of the global calendar. Let us consider a system where nodes are numbered $1 \dots N$, and each node has an associated user (e.g., $User_i$ is associated with $Node_i$). $Node_i$ has the latest and most up to date view of the calendar for $User_i$, and a view of the calendars of the other users in the system (but this view is not guaranteed to be up-to-date). If $User_i$ wants to set up an appointment with $User_j$ ($i \neq j$), then $User_i$ looks at the calendars of both users at $Node_i$ (note that $Node_i$ might have a stale copy of the calendar for $User_j$), finds out an empty slot, and then puts the appointment in the *log*. Now, $User_j$ at $Node_j$ has to be notified about this new event, and this is done by shipping the log to $Node_j$. Please refer to the paper by Wu and Benstein covered in the lectures for ideas for maintaining and shipping of logs. The logs should be truncated to reduce the communication overhead. Again, refer to the paper, for ideas.

The same idea can be extended for setting up meetings with more than 2 users. After a common time is selected, the log is shipped to all the concerned parties. On the other hand, if $User_i$ wants to insert an event only in his calendar, he is not required to ship the log to any other node. Again, you can consider the assumptions made in the referred paper for the model of the consistency and persistent storage.

Note that since a node might have a stale view of the calendar, there might be conflicts ($User_i$'s view of $User_j$'s calendar showed a vacant slot, but since the view was stale, the slot might have been taken in the mean time). You can come up with a conflict resolution protocol of your choice. After resolution, all parties should have a consistent view of the event, i.e. if there is a conflict when event e (involving $User_i$ and $User_j$) is to be inserted into the calendar at $Node_j$, and the conflict resolution protocol decides that event e cannot be inserted, then $User_i$ should be notified about this so that both users have a mutually consistent view.

Implementation Details

To start with, implement this system with a system of five (5) distributed nodes. The *log* should be persistent (stored in disk to guarantee that node crashes can be tolerated). Your implementation need not have a good GUI, but a minimal UI to effectively view, insert, delete or modify events should be provided. There is no preference for language used for implementation. Your implementation should be demo-able in GSL or CSIL machines.

For starters, Java provides a good networking library for implementing communication over the network (java.net). You can rely on that library for network communication. This is just a guideline, not a mandate.

Deliverables

Due date for the project is 12th November. On 12th November, turn-in a 2 page specification of the project providing detailed design and implementation. The specification is due in class. Along with the specification,

you need to provide experimental evaluation of your implementation in terms of the amount of bandwidth consumed by this protocol for a specified workload (the workload would be provided on the course web-page at a later date). The experimental details should include the number of messages sent from each site, and the size of the log shipped with these messages (thereby giving an estimate of the bandwidth consumed). You should be ready to demo by 12th November. The date and times for the demo will be notified at a later date. At the time of demonstration, you should have a UI for displaying the calendar of all 5 nodes, as well as the bandwidth consumed at each node.