

Distributed Systems CSCI-4963/6963 – Fall 2015

Project 1

Assigned Sep. 25, 2015

Work in groups of two

Description

For this project, you will implement a distributed calendar application using a replicated log and dictionary. The system consists of N nodes, each node corresponding to a single user. Each node stores its own local calendar, and you must implement a distributed algorithm to share calendar information among users. You will use Wu and Bernstein's algorithm to share this information.

The calendar keeps track of appointment events, where each appointment is a tuple consisting of the following fields:

- Name: Name of the appointment
- Day: Date of the appointment
- Start Time
- End Time
- Participants: List of node ids

Events should also be stored in a log, so that they can be shared with other nodes using the Wu and Bernstein algorithm. For simplicity, you can make the following assumptions:

- Appointment names are always unique.
- The calendar spans exactly 7 days, from Sunday 12:00am until the following Saturday at 11:59pm.
- Appointments can be scheduled only in $\frac{1}{2}$ hour increments, e.g. an appointment can last from 10:00am – 11:30am, but not 10:15am – 11:30am.
- Appointments do not span multiple days.

If User i wants to set up an appointment with itself as the only attendee, User i should be able to check the local calendar to see when it is free, and then schedule an appointment. The application (running on node i) should add the appointment event to the calendar and insert the corresponding event record into the log.

If User i wants to schedule a meeting with both Users i and j ($i \neq j$), then User i should be able to check its local calendar to see when both i and j are free and schedule an event (note that node i 's view of node j 's calendar may be stale). The application (running on node i) should add the event to the calendar and insert the corresponding event record into the log. User j then needs to be notified of the appointment, so node i should send a message, with node i 's log, to node j . Node j should then update its log and calendar according to algorithm. This same idea can be extended to meetings with more than two users.

In the Wu and Bernstein algorithm, nodes may not always have the most up-to-date information about the other nodes' logs and dictionaries. Therefore, this algorithm may result in the scheduling of conflicting appointments. You must design and implement your own conflict resolution protocol. When the conflict is resolved, all participants involved in the conflict should have a consistent view of the conflicting appointment(s). For example, if there is a conflict with appointment e where e 's

participants are User i and User j , and your conflict resolution protocol determines that e should be cancelled, then both node i and node j should be notified of this. *Note: Do not modify the Wu and Bernstein algorithm to avoid schedule conflicts, but rather add your own algorithm to resolve them.*

A user should also be able to cancel an appointment (if that user is a participant). If an appointment is cancelled, whether by a user explicitly or by your conflict resolution protocol, this cancellation should be recorded as a delete event in the log, and the event should be removed from the local calendar. Any other participants in the cancelled meeting should be notified of this deletion as well (by sending the log).

Your application should truncate logs and reduce message sizes using the Wu and Bernstein algorithm.

Implementation Details

You must implement a system with four nodes. Each node will be run on a different machine. The log and calendar should be stored on disk so that it will survive node crashes. You need to provide, at least, a minimal UI to view, insert, and delete appointments. A text-based UI is fine. The contents of the log must also be viewable. This can be done by reading a file, so long as the file is human-readable.

You will demo your project on Amazon EC2, on four machines in four different regions. You will use micro-instances. You can use any programming language supported by this platform that supports the message-passing model and does not mask failures (e.g., C, C++, Java, Python). Details about setting up accounts on EC2 are on the course web site.

Deliverables and Due Dates

Project Due Sunday, Oct 25, 2015 at 10:00pm.

Turn in your source code and a 2-page project report. The report should provide details of your design and implementation. The code and report will be submitted using the RPI Homework Server. Details will be provided closer to the due date. **No late turn-ins will be accepted.**

We will schedule project demonstrations for the week of Oct. 26 – Oct. 30, 2015

Based on a project by S. Das and A. El Abbadi