CSCI 520: Distributed Log Programming Project

Introduction

While the specification of distributed algorithms can be elegant and straightforward, the implementation is made complicated by many details. It requires a certain skill to solve practical implementation challenges while staying true to algorithm specification. The goal of this assignment is to give you experience with implementation of a physical distributed system.

Assignment

Your task is to 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 Wuu 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`: Start Time
- `end_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 Wuu 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 half 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 Wuu 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 Wuu 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 Wuu 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 user interface (UI) to view, insert, and delete appointments. A text-based UI is fine. The contents of the log must also be viewable, for example by reading a file, so long as the file is human readable. Please make sure the action of your nodes, such as receiving events, are triggered by these events arriving over the network, rather than through a UI.

You will deploy your project on Amazon EC2, on four machines in four different regions. You will use micro-instances. You should sign up for AWS Educate to get your $35 credit for use of Amazon's cloud infrastructure. This should be enough for you to deploy and demo the project. You should not do your development in AWS, but push your code from a repository (GitHub) to your AWS instances to test and demo.

Grading

Your grade will be based on a screencast that demonstrates the functionality of your code. I expect the demonstration to show the different types of events that users may insert, event deletions, event conflicts, and the results of your resolution mechanism. I will be looking for functionality and correctness of your mechanisms.

Specifically, you will be graded according to the following rubric:

4 points - Log and dictionary implementation
2 - log is truncated correctly
2 - nodes use permanent storage

8 points - Calendar implementation
1 - can check if calendar free (show calendar)
1 - can schedule event with itself
2 - replicates event to other nodes' logs
1 - schedule with others
2 - can delete appointments in the distributed log
1 - deleted events removed from calendar

5 points - Calendar event conflict resolution
2 - scheduling conflicts detected
2 - scheduling conflict resolution algorithm correct
1 - scheduling resolution implemented

3 points - AWS
2 - implemented on different AWS nodes
1 - multithreaded implementation

What to submit?

1. A report containing:
    a. A link to a YouTube video showing your code running on AWS instances.
    b. Description of your mechanism for handling scheduling conflicts.
2. Your source code (unzipped) for D2L plagiarism check