CS 675 – Distributed Systems
Project Description
A Distributed Highly Available Calendar Tool for Workgroups

1 Introduction

In this project, you will extend the calendar tool you designed and implemented in Project 1. The goal is to design a calendar service with the following requirements:

1. All the requirements of the calendar tool from the previous project also apply to this project (i.e., group events, callbacks, persistence, etc.)
2. Distributed architecture: Unlike the calendar tool of the previous project where all the calendar objects were resident in the same server process, the calendar objects must be distributed among multiple server processes running on different computers.
3. High Availability: The calendar and calendar manager service must be highly available, i.e. clients should be able to access the **latest version** of their calendars even if one or more hosts that are running the calendar service become unavailable.
4. Replication transparency: Replication of calendars and calendar managers must be transparent to users. Thus the same user interface program used in the previous project can be used for this project.

2 Design Guidelines

The need for high availability suggests a design that uses replication. The requirement for a distributed architecture will necessitate a scheme for being able to map calendars objects (and their replicas) to servers, i.e., a distributed naming scheme.

The requirement that users be able to access the latest version of their calendars suggests a design that provides relatively strict consistency, e.g., sequential consistency.

Moreover, it will be necessary for replicas to coordinate with each other and keep track of each other, i.e., to implement failure detectors.

Handling group events is particularly challenging for a distributed architecture. Ideally, you need to ensure (i) that group events are handled correctly despite the possibility of partial failure and (ii) that your scheme avoids deadlocks. While avoiding deadlocks is relatively straightforward, the first requirement means that you will need to provide support for inserting (or updating) group events in an atomic fashion. This is quite challenging, and it will be an extra credit component to **implement** this feature.

However, you are required to explain how you would implement this feature in the project report.

3 Suggested Design

I recommend using a design with the following features:

1. Distributed architecture and naming service: The server processes providing the calendar service are organized in a ring a la Chord. The Chord scheme for mapping resources to hosts can be used to map calendar objects (and their replicas) to hosts.
2. Primary-backup scheme for replication: This is the simplest scheme and it provides sequential consistency. For this application, a single backup replica should suffice, which further reduces the complexity of the scheme and implementation.
3. Failure Detectors: You will need to implement failure detectors for each process.
4. Deadlock avoidance: To prevent deadlocks when entering group events, all clients should always access calendar objects in the same order.
5. Distributed commit protocol: To ensure insertion (or updates) of group events are handled atomically, it is necessary to use a protocol such as Two-Phase Commit.

You are free to use an alternative design as long as you meet all the project requirements. Note that the design above brings together several concepts discussed in class.

4 Submission

Your grade will depend upon (i) 70% - Implementation via a demo in which you demonstrate that your implementation meets the design requirements, (ii) 30% - A document that describes your design in detail (including the optional component of this project, i.e., your approach for handling group events correctly in the face of partial failures). The extra credit component is worth an additional 30%.

Your design document should describe in detail how your calendar service handles partial failures and discuss the limitations of your design, i.e., what kind of partial failures are NOT handled and what you would need to do to handle those failures.

In addition to your design document, please submit an archive including all your source code, instructions for building and running your client and servers, etc. in a tar or zip archive file and submit via email. Your code should be well commented so that I can understand what it is doing.