

COMP30220: Distributed Systems

Dr. Rem Collier

Group Project: A Fault Tolerant Voting Service

Due on We're not even sure

Group Name: Lamp + Gary

Joe Duffin - 13738019

Niamh Kavanagh - 12495522

Edwin Keville - 13718661

Gary Mac Elhinney - XXXXXXXXX

Contents

Project Description	2
Technologies Used	3
Jax-WS	3
Java RMI	3
SQLite	3
The Raft Algorithm	4
Basic Principle	4
Electing a leader	4
Heart Beats	4
Our Implementation	5
jgUDDI	5
Clients	5
Servers	5
Conclusion	7

Project Description

This will pretty much be a re-iteration of the proposal, but we're no longer using uddi

Technologies Used

so info on the technologies and a sentence on how we used them/what we used them for

Jax-WS

To publish server end points

Java RMI

to maintain a list of active servers

SQLite

each server has it's own sql db, used to keep track of votes.

The Raft Algorithm

Basic Principle

It is an algorithm for coming to data consensus between multiple servers.

Each server has 3 states (follower, candidate, leader)

The algorithm plays out as follower

1. Every server starts as as follower
2. A leader is elected
3. That leader sends out heart beat messages with data
4. If no heart beat is sent out a new leader is elected

Electing a leader

Explicit Steps

1. If no heartbeat is detected by a follower, who ever's election count down timer runs out first puts him self up for election
2. That server becomes a candidate and a new election term is started and they request votes from other servers
3. Follower servers vote once per term to the candidate server that first requests a vote from them
4. When a candidate server receives a majority of votes it is now the leader and starts heart beats
5. if a tie is reached (no majority) then no action is taken and election count down timers restart

Heart Beats

1. With each heart beat from the leader data is distributed
2. Follower servers acknowledge receipt of the data but do not commit it to memory yet
3. Only when the leader knows that a majority of follower servers have received the data does the leader instruct the followers to commit the data
4. A leader will continue sending heart beat messages indefinitely (until it crashes)

Our Implementation

Consists of the following 3 parts

kgUDDI

For our distributed voting service we needed a means of publishing the endpoints of a service which is detectable and reachable by the other systems, whether they be servers or clients. UDDI (Universal Description, Discovery, and Integration) is an XML-based standard for describing, publishing, and finding web services. This standard defines a methodology for publishing services in a way in which we need but we found the java implementation (jUDDI) to be way too heavy weight.

We implemented our own UDDI service called kgUDDI (Joe and Gary UDDI). We used JAVA RMI to host an instance of our kguddi service. The service is responsible for yielding the published endpoint of current leader at any one time as well as the published endpoints of all other servers. When ever a new server comes online it publishes its endpoint to kgUDDI. When a new leader is elected kgUDDI is updated with the endpoint of the new leader.

The kgUDDI service is owned by one single server but is not a single point of failure. Should the hosting server crash, thus killing the kgUDDI service, the next online server to query kgUDDI will realise it is down, re-start the service and re-populate it's endpoint list with a local copy. This server is now the owner of kgUDDI. The owner of kgUDDI is fully independent of the notion of the current Raft leader.

Clients

The clients are a simple part of our project. It is a command line based implementation where the user is prompted to either vote for a candidate or print the current results.

For every interaction from the client, the client first looks to kgUDDI, gets the endpoint of the current leader, and then either votes to or queries that server for results. It is the servers responsibility to replicate the data to all other raft servers. The client is ignorant as to which server it is dealing with at any time.

Servers

The server is the core of our project. Any number of servers can be brought online at any one time. Any server can be killed at any time (including the current leader) without an interruption to the distributed voting service. The client only sees an interface where it can vote or request results. This interface could be an instance of any one of the servers at any given time. Every server is exactly the same. Each server intermittently checks the kgUDDI service to establish if more servers have come online, or if a known server has gone offline.

The server has three main components.

The Raft Element

Each server is controlled by 2 threads in charge of the two countdown timers (the election count down timer and the heartbeat count down timer). When a server is first brought online it publishes its self and it's election countdown timer is started. During this time period it waits to receive heartbeats from the current leader. If no heart beat is detected, it will put its self up for election.

If the server is elected as leader its election countdown timer stops and its heart beat countdown timer starts. When each "pulse" time period has elapsed the server will call receiveHeartBeat on every server

known to it at that time. With this heart beat will travel any voting data recieved since the last heart beat. The leader looks for acknowledgement of heart beats from a majority of servers and only once this majority is reached, the leader instructs all servers to commit the data to their data base.

The Coordinator

It is the responsibility of the coordinator to be the middle man between the database and the server. When a server is informed of a new vote via a heart beat, it is the coordinator that holds this vote until the instruction to commit it to the data base is recieved.

When a client requests the current state of the database from a server, the server puts that responsibility on the coordinator to request this information from the database and returns it to the server. The server then returns the data back to the client.

The Database

Each server uses an SQLite database to keep track of the current vote counts. It is only when data consensus has been reached between the servers (a majority of servers have acknowledged receipt of the new data) that the new data is committed to the data base. We used an SQLite database so that our solution is scalable past the toy example that we implemented.

Conclusion

it was fab