Weekly Report – Raft

Tasks

- Learn more about Raft
- Implement a demo of Raft

Main Activities

1 Raft Algorithm

1.1 Differences between raft and paxos

Paxos is very famous consensus algorithm. However, it has two fatal flaws.

- The first is that paxos is too hard to understand.
- The second drawback of Paxos is that it is difficult to implement in a real environment.

Raft is a consistent algorithm for managing log replication. It's the same as the performance and functionality of the Paxos, but it's not the same structure as Paxos which makes the Raft easier to understand and easier to build the actual system.

The Raft has several new features:

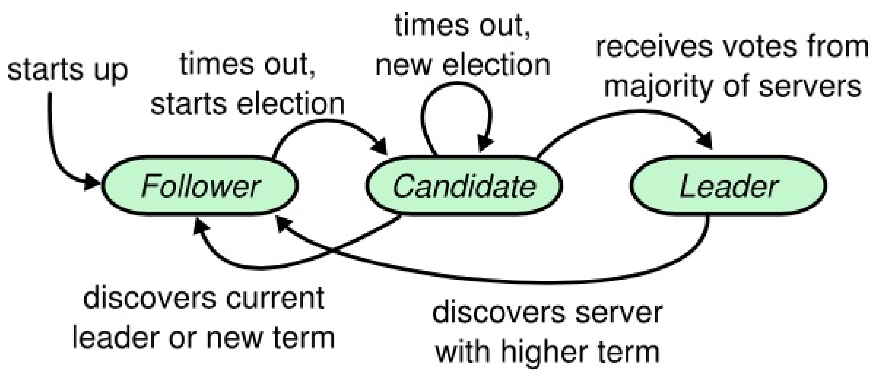
- Strong Leader
- Leader Selection
- Membership Change

1.2 Introduction of raft

The Raft was made because the Paxos was too difficult to implement, with the goal of making it as easy to understand as possible without losing it to Paxos.

The Raft achieves consistency by selecting a leader first, and then gives the leader the responsibility to fully manage the replication log.

Each server has three states: leader, follower and candidate. In normal times, only one server is the leader and the rest of the servers are followers. Followers do not send any requests, but respond to requests from leader and candidates. The transformation of three states is shown in the diagram below.


starts up
times out, starts election
times out, new election
receives votes from majority of servers
Follower
Candidate
Leader
discovers current leader or new term
discovers server with higher term

The algorithm requires two types of RPC *(Remote Procedure Call Protocol)*

- RequestVote RPC

  Initiated by the candidates in the election process, after another server receives this RPC, only when the other term and log are at least as well as their new would vote for received the most votes the candidate gets elected leader.

- AppendEntries RPC

  The leader initiates the distribution of logs, forcing the log of the follwer to agree with himself to achieve consistency.

The Raft decomposes the consistency problem into three small problems:

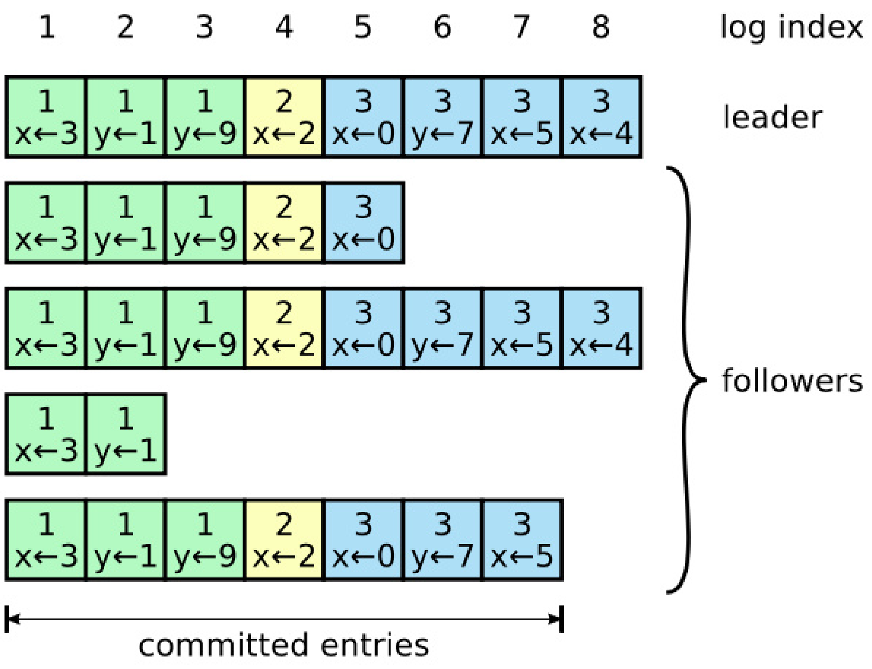
- leader election

  If a follower does not receive a message from the leader in the time of election timeout, enter the new term, turn to candidate, vote for himself, and initiate the election RequestVote RPC.

  If not the Server will be able to get the most votes, then have to go into the next term, this time to choose again. To avoid this situation continues, each Server election time is set to different values, random timeout first so you can launch the next election.

- log replication

  Once the leader is elected, it begins to receive requests from the client. Each client request contains a command that needs to be executed by the replicated state machine. The leader adds this command as a new log entry to its log, and then initiates AppendEntries RPC to the other servers in parallel.



- safety

  But so far still can guarantee security. For example, when a leader in the commit log, some followers went offline, then the follower was later selected as the leader, it will override follwer now those who have committed the log, because these log has been carried out, so the result of different machine can perform different instructions.

  In the process of the election, one more restriction can prevent this from happening:
  For any term, the leader should contain logs that were committed in the previous term.

2 Implementation Of raft

2.1 The open source implementation of raft

There is not only the details of the algorithm, but also many open source implementations at https://raft.github.io/

Name	Primary Authors	Language	License	Leader Election + Log Replication?	Membership Changes?	Log Compaction?	Row Last Updated
bspolley/raft	Alex Kaiser, Brennan Polley, Helen Weng (Berkeley)	Bloom		Some			2013-05-18
noeleo/raft	Noel Moldvai, Rohit Turumella, Josh Muhlfelder, James Butkovic (Berkeley)	Bloom	Simplified BSD	Lacking persistence	No	No	2013-05-25
whitewater	Adam Midvidy, Anh Mai, Karoun Kasraie, Sanketh Katta (Berkeley)	Bloom	MIT	Some correctness issues	No	No	2013-05-18
willemt/raft	Willem-Hendrik Thiart	C	BSD	Yes	Yes	No	2016-01-28
NRaft	Bert Willems (Premotion)	C#	MIT				2013-09-02
jraft	Andy Chen	Java	Apache2	Yes	Yes	Yes	2016-04-14
dupdob/RAFTiNG	Cyrille Dupuydauby	C#	Apache2				2013-09-02
cppa-raft	Zhang Yichao	C++	MIT	Partial	No	No	2014-02-04
Kudu	David Alves, Todd Lipcon, Mike Percy	C++	Apache2	Yes	Yes	Yes	2016-06-02

Copycat is a feature complete, fully asynchronous implementation of the Raft consensus algorithm in Java which is my main reference template.
https://github.com/atomix/copycat

2.2 My implementation of raft

According to figure2 in paper *In Search of an understandable Consensus Algorithm*
We can build the basic model of the algorithm.

Persistent and volatile state

```
// 所有服务器都有的状态
public class Server {
    // 服务器知道的最新的term
    private int currentTerm;

    // 记录当前分期内给那个Candidate投过票
    private int votedFor;

    // 日志条目数组， 每条包含状态机执行的命令，和受到时的term
    private Object[] log;

    /*  下面是volatile state */

    // 已知的最大的committed的index (初始化为0， 单调递增)
    private int commitIndex;

    // 最后被应用到状态机的条目的索引值 (初始化为0， 单调递增)
    private int lastAppliedIndex;

    // 3种状态 (follower, leader, candidate)
    private int state;

    // 计时器, 周期函数每次执行增加
    private int timeoutElapsed;
}
```

RequestVote

```
public class RequestVote {
    // 竞选人的term
    private int term;

    // 竞选人的ID
    private int candidateId;

    // 竞选人最新一条日志的index
    private int lastLogIndex;

    // 竞选人最新的日志的term
    private int lastLogTerm;

    public RequestVote() {
    }

    public RequestVote(int term, int candidateId, int lastLogIndex, int lastLogTerm) {
        this.term = term;
        this.candidateId = candidateId;
        this.lastLogIndex = lastLogIndex;
        this.lastLogTerm = lastLogTerm;
    }

    public int getTerm() {
        return term;
    }

    public void setTerm(int term) {
        this.term = term;
    }

    public int getCandidateId() {
        return candidateId;
    }

    public void setCandidateId(int candidateId) {
        this.candidateId = candidateId;
    }

    public int getLastLogIndex() {
        return lastLogIndex;
    }

    public void setLastLogIndex(int lastLogIndex) {
        this.lastLogIndex = lastLogIndex;
    }

    public int getLastLogTerm() {
        return lastLogTerm;
    }

    public void setLastLogTerm(int lastLogTerm) {
        this.lastLogTerm = lastLogTerm;
    }

    @Override
    public String toString() {
        return "RequestVote{" +
                "term=" + term +
                ", candidateId=" + candidateId +
                ", lastLogIndex=" + lastLogIndex +
                ", lastLogTerm=" + lastLogTerm +
                '}';
    }
}
```

RequestVoteResponse

```
public class RequestVoteResponse {
    // 当前的term, 用于竞选者根据投票结果自己更新
    private int term;

    // 投票结果, 如果竞选者接受投票就为true
    private boolean voteGranted;

    public RequestVoteResponse() {
    }

    public RequestVoteResponse(int term, boolean voteGranted) {
        this.term = term;
        this.voteGranted = voteGranted;
    }

    public int getTerm() {
        return term;
    }

    public void setTerm(int term) {
        this.term = term;
    }

    public boolean isVoteGranted() {
        return voteGranted;
    }

    public void setVoteGranted(boolean voteGranted) {
        this.voteGranted = voteGranted;
    }

    @Override
    public String toString() {
        return "RequestVoteResponse{" +
                "term=" + term +
                ", voteGranted=" + voteGranted +
                '}';
    }
}
```

AppendEntries

```
public class AppendEntries {
    // leader的term
    private int term;

    // 最新日志的前一条日志的Index，用于follower确认与leader的日志完全一致
    private int prevLogIndex;

    // 最新日志的前一条日志的term
    private int prevLogTerm;

    // 消息中携带的日志数组
    private Object[] entries;

    // leader中已经确定提交到状态机的log的index，也就是说follower也可以将该index之前的log提交
    private int leaderCommit;
```

```java
    public AppendEntries() {
    }

    public AppendEntries(int term, int prevLogIndex, int prevLogTerm, Object[] entries, int leaderCommit) {
        this.term = term;
        this.prevLogIndex = prevLogIndex;
        this.prevLogTerm = prevLogTerm;
        this.entries = entries;
        this.leaderCommit = leaderCommit;
    }

    public int getTerm() {
        return term;
    }

    public void setTerm(int term) {
        this.term = term;
    }

    public int getPrevLogIndex() {
        return prevLogIndex;
    }

    public void setPrevLogIndex(int prevLogIndex) {
        this.prevLogIndex = prevLogIndex;
    }

    public int getPrevLogTerm() {
        return prevLogTerm;
    }

    public void setPrevLogTerm(int prevLogTerm) {
        this.prevLogTerm = prevLogTerm;
    }

    public Object[] getEntries() {
        return entries;
    }

    public void setEntries(Object[] entries) {
        this.entries = entries;
    }

    public int getLeaderCommit() {
        return leaderCommit;
    }

    public void setLeaderCommit(int leaderCommit) {
        this.leaderCommit = leaderCommit;
    }

    @Override
    public String toString() {
        return "AppendEntries{" +
                "term=" + term +
                ", prevLogIndex=" + prevLogIndex +
                ", prevLogTerm=" + prevLogTerm +
                ", entries=" + Arrays.toString(entries) +
                ", leaderCommit=" + leaderCommit +
                '}';
    }
}
```

AppendEntriesResponse

```java
public class AppendEntriesResponse {
    // 当前的term
    private int term;

    // 如果prevLogIndex和prevLogTerm都比对成功，那么日志添加成功，返回true；否则返回false
    private boolean success;

    // 处理添加日志请求后本地的最大日志索引
    private int currentIndex;

    // 从添加日志请求中接受的第一条日志索引
    private int firstIndex;

    public AppendEntriesResponse() {
    }

    public AppendEntriesResponse(int term, boolean success, int currentIndex, int firstIndex) {
        this.term = term;
        this.success = success;
```

```
            this.currentIndex = currentIndex;
            this.firstIndex = firstIndex;
        }

        public int getTerm() {
            return term;
        }

        public void setTerm(int term) {
            this.term = term;
        }

        public boolean isSuccess() {
            return success;
        }

        public void setSuccess(boolean success) {
            this.success = success;
        }

        public int getCurrentIndex() {
            return currentIndex;
        }

        public void setCurrentIndex(int currentIndex) {
            this.currentIndex = currentIndex;
        }

        public int getFirstIndex() {
            return firstIndex;
        }

        public void setFirstIndex(int firstIndex) {
            this.firstIndex = firstIndex;
        }

        @Override
        public String toString() {
            return "AppendEntriesResponse{" +
                     "term=" + term +
                     ", success=" + success +
                     ", currentIndex=" + currentIndex +
                     ", firstIndex=" + firstIndex +
                     '}';
        }
    }
```

However, when I want to create some class to handle the communication between different server, I suddenly realized RPC seemed a bit difficult. Open source code is also hard to understand here. Now I am learning netty which is a nio framework of java. Can you give me some advice on implementing RPC.

Period: 2017.4.26-2017.5.2
Submitted by: jiacheng huang 2017.5.2