Specification for Paxos Protocol

Goal

Replicate an application robustly across multiple nodes using Paxos. Present a clean API to clients and client applications.

High Level Description

Paxos is a protocol for reaching consensus on a single value. This Paxos application is used for reaching consensus on multiple values, a replicated-log. This Paxos implementation uses leader election as an optimization to increase performance and reduce network latency effects. This Paxos implementation also collapses the roles of Proposer, Acceptor, and Learner, into a single role. It is commonly accepted that the distinction between these roles only becomes important when there are many instances of paxos running. Most applications of Paxos do not need this distinction, therefore, this Paxos implementation does not incur the cost of this abstraction.

This Paxos protocol specifically implements multi-paxos with collapsed-roles and leader-election. The high level description of such a protocol can be found on the Paxos wikipedia page. The terms that will be used here are the standard and can also be found on that site.

APIs:

There are three different APIs that must be implemented to conform with this Paxos specification. First there is the Paxos API which is used internally, within the Paxos Cluster itself in order to communicate with each other, clients, and applications. Then there is the Client API which is used by clients to connect and make requests to the Paxos Cluster. Finally, there is the Application API. The application API is responsible for brokering communication between the registered client application that is replicated across the Paxos Cluster, and the Paxos Cluster itself.

Background: Messages

There is a single, consistent message structure that is used throughout all APIs. The messages encoded using JSON and unused and zero-valued entries are omitted from the JSON structures to reduce packet size. Recognized zero-values are 0, "", and false.

Msg is the message type. The message structure contains all of the fields that could be possibly used by any communication. It always contains the type of the message, which is also described below. Messages always record the address and port from which it came, and sometimes, record that of the current leader, if there is one. Messages used also can contain a request field, which holds the information for the current client request. This allows the Paxos cluster to keep track of specific requests as they travel throughout the cluster. The request field should be considered read only and should not be altered after creation by the Proposer. Entry is used to keep track of what entry in the replicated log this message is attempting to fill. Round is used to keep track of what Paxos round we are on for this Entry. Each Paxos round starts off at 0 for each unique log entry. This allows us to think of each log entry as an instance of Paxos, while not incurring the overhead of actually creating multiple instances of Paxos. Value is used to carry the value that should be set for this round. Round value is used in Nack responses to send back the value that has already been accepted for this log entry, and the round that it was accepted for. Finally error allows the Paxos Agents to send error information to other Paxos Agents and clients.

Message:

```
Msg{
    Number: type,           // type is a member of
MsgType
    string: formaddress,    // ipv4 address of the
sender
    string: fromport        // port of the node that
sent it
```

```
    string: leaderaddress   // ipv4 address of the
leader
    string: leaderport      // port of the leader
    Object: request         // request is a RequestInfo
for
                            // this client transaction
    Number: entry           // log entry for this
request
    Number: round           // paxos round for this
entry
    string: value           // value to assign to this
entry
    Object: roundvalue      // roundvalue is a
“roundValue”
                            // previous round & value
accepted
    string: error           // error description
}
```

RequestInfo:

RequestInfo is the associated information for this client transaction. Since each Paxos transaction is initiated by a client, it has associated information regarding the transaction. RequestInfo logs the clients unique identification number which is given to it by the Paxos node is has connected to. It also logs the request number for this client. This way the Paxos node it has connected to can keep track of requests on a per client basis, while giving clients the ability to request to see their old requests (by giving an old request number).

The RequestInfo structure also contains the value that the client is trying to set. After the client has connected with the Paxos cluster, the entry number is assigned to be the next uncommitted log entry. The leader is then in charge of

incrementing this entry number, and retrying the commit process until it does get committed to the replicated log.

Noset is a flag that tells the Paxos cluster that this is merely a query for a certain entry and that no value should be committed to the log. This is useful during the "catch up" phase of Paxos nodes as they must request old values, and is also useful during client application recovery. If they were in the middle of a "transaction" with the Paxos cluster, they will be able to see which entries were in fact committed to the replicated log.

```
RequestInfo{
    Number: id    // uid of the client for the request
    Number: no    // request number for this client
                  // must be monotonically increasing
    String: val   // value the client is attempting to
log
    Number: entry // entry in the log for this request
    Bool: noset   // true iff only response should be
Nack
}
```

Message Type

Message Types are used to specify the type of the Msg. Though they have strings associated with them, they are represented as integers in an enum. They are zero-indexed and are in the order as follows.

Note:

A quorum is defined as the number of paxos agents divided by 2 plus 1: a majority.

Paxos API:

The first 6 message types are used for internal Paxos communication between agents.

1. Empty: An empty message is an invalid message and should be ignored.
   Required Fields: None
2. Prepare: Prepare is the initial Prepare request sent by the Proposer.
   Required Fields:
   fromaddress - The sender's (Proposer's) address
   fromoport - The sender's (Proposer's) address
   request - The RequestInfo associated with this request
   round - The round this message is for
3. Promise: Promise is one of the possible responses an Acceptor can send to the Proposer after receiving a Prepare request. If a proposer has not promised the specified entry, with a round greater than or equal to the round specified in the Prepare request to anyone else and there is no current leader, then the Acceptor should send back a Promise to the Proposer. After sending this Promise, this Acceptor should never send a Promise for this entry with a round less than or equal to the round it promised for. The sender of a Promise, must set his accepted leader to be the Proposer they are promising to.
   Required Fields:
   fromaddress, fromport, request, round (same as before)
   roundvalue - The last accepted value for this entry and when it what round it was accepted. If no such value exists, the roundvalue{round: –1, value: ""} is used.
4. Nack: The response by an Acceptor saying that it has already promised a higher round number to a different Proposer.
   Required Fields:
   fromaddress, fromport, request, round, roundvalue (same as above).
5. AcceptRequest: Accept request is the response sent by the Proposer after receiving a quorum of promises, or if the Proposer has already been declared leader. It is sent to all Paxos agents, demanding that they accept the specified for this log entry and round. Upon receiving a quorum of Promises, the leader declares himself leader and starts sending out heartbeat messages.
   Required Fields:

fromaddress, fromport, request, round (same as before)
value - the value the Learners should accept

6. Accepted: After a Learner has received an AcceptRequest, if they have not already accepted something else, and if the sender is the leader, then they accept the given message, otherwise they send back a Nack response with the previous value that they accepted. If the recipient accepted the value then they commit it to their log. After the Proposer has received a quorum of Accepted responses, they commit the message to their history and send back the ClientResponse to the client who initiated the request.
   Required Fields:
   fromaddress, fromport, request, round, value (same as before)
7. Heartbeat: If this agent is the current leader, then it sends out a Heartbeat in regular intervals of 200 milliseconds. If the followers do not receive a Heartbeat from the leader in 400 milliseconds, then they declare the leader dead.
   Required Fields:
   fromaddress, fromport (same as above)
   leaderaddress, leaderport - the address and port of the leader

Client API:

8. ClientRequest: A client makes requests to the Paxos Cluster using ClientRequest messages.
   Required Fields:
   request.id - client id assigned by the ClientConn message
   request.no - for new requests a unique number to this request

```
        for old requests the number corresponding to
it
```

   request.value - the value the client is attempting to log
9. ClientResponse: The response to the request received from the Paxos Cluster for the given Client Request. If a client application is registered to the Paxos Cluster, then the ClientResponse will return the application's return value for the ClientRequest's request value. For instance if a client sends a "Get(name)" request to a key-value store which currently contains

the pair (“name”, “bob”), the response would contain the return value from the application, “bob”. If no application is registered to the Paxos Cluster, then the Paxos Cluster sends back a Response with the value that it committed to its log.

Required Fields:

request (same as above)

value - the response value passed back from the application, if there is an application registered, otherwise the value of the entry that was committed.

10. ClientRedirect: After receiving a ClientConnectRequest, the server can send back a ClientRedirect if they are not the Paxos leader, this ensures that message latencies on average are lower.

    Required Fields:

    leaderaddress, leaderport (same as above)
11. ClientConn: ClientConn messages contain the Client Connection information. After sending a connect request and being accepted, the Client is sent its connection information. It is specifically sent back its Client Id and starting Request Number. These are stored in their respective fields in the message’s “request” field.

    Required Fields:

    request.id - the unique identifier assigned to the client

    request.no - the request number for the first client message
12. ClientConnectRequest: When a client first connects to the Paxos cluster, it sends a message of type ClientConnectRequest. This initializes a connection with the Paxos Cluster. If the node contacted is not the Paxos leader, it will send back a ClientRedirect with the leader address and port to redirect to. If the connection is successful the client will receive back a ClientConn message detailing its connection info.

    Required Fields: None

Application API:

13. LogResponse: After establishing a connection with a client application, the paxos leader node, which it is connected to, sends out a stream of LogResponses using the established TCP connection. These log entries come in order, and if the noset field is false, then the application response

is sent back in an AppResponse, otherwise no response is sent back to the Paxos Cluster. These LogResponses must be processed in order, and are not allowed to be processed concurrently unless the results and all future requests will be unaffected by processing them concurrently or out of order.

Required Fields:

request (same as before)

value - the return value of the application for the request

14. ClientApp: When a client application desires to connect with the Paxos Cluster, after sending the ClientConnectRequest, the client application should send a ClientApp message. Once the client application sends this message, the server starts streaming log entries to the client application using the established TCP connection.
    Required Fields: None
15. AppResponse: After processing a LogResponse the application sends back an AppResponse with the return value.

    Required Fields:

    request (same as before)

    value - the application's return value

    entry - the log entry that this is a result for
16. Done: If an agent receives a Done request, it kills itself, stops listening to clients and other agents. If it was leader it also kills the client application. This is used for testing purposes.
17. Error: Details about the error value, i.e. if there was a server error and what it was.

Paxos API

Paxos Agents communicate with other Agents through UDP. They communicate with clients and applications through TCP.

All messages each Paxos agent receives must be stored and flushed into a local log before sending the response. This ensures that the Paxos agents will never go back on their promises even if they crash and recover.

Whenever a Paxos node starts up, it must read from the local log file that it has accumulated and recover from that. This is to ensure that the Paxos agent always "remembers" values that it has previously accepted as

well. This way the accepted values are persisted, and the fact that they were accepted by a majority of nodes is not forgotten.

In regards to leader election, when there is no leader elected, the paxos protocol goes through all the steps Propose, Promise, Accept, Accepted. When a Proposer has received a quorum of Promises, it becomes leader and starts sending out heartbeat messages. When other agents send a Promise to a Proposer, they must record that he is the current leader. However, they will time him out if they don't receive a heartbeat every 400 milliseconds. All client connections are redirected to the leader, and the leader skips the Promise and Propose step, just doing the Accept, Accepted cycle until he receives a Nack, which would indicate that he is no longer leader, or when he dies.

Go API

The Go API's are described on godoc.org/github.com/dyv/paxos. The API's allows easy creation of paxos agents as well as the ability to register client applications, by just pointing the API to the binary. The Application API is similarly simple where you just connect to an address and port and stream the log as it is dynamically generated, Committing each value after it is done being computed. The Client API connects to the port and address of a Paxos node and then calls NewRequest to makes new requests to the Paxos Cluster.