Distributed Consensus in Content Centric Networking

Marc Mosko[1]*

Abstract
We describe a method to achieve distributed consensus in a Content Centric Network using the PAXOS algorithm. Consensus is necessary, for example, if multiple writers wish to agree on the current version number of a CCNx name or if multiple distributed systems wish to elect a leader for fast transaction processing. We describe two forms of protocols, one using standard CCNx Interest request and Content Object response, and the second using a CCNx Push request and response. We further divide the protocols in to those using the CCNx 0.x protocol where Content Object name may continue Interest names and the CCNx 1.0 protocol where Content Object names exactly match Interest names.

Keywords
Content Centric Networks – Named Data Networks

[1] *Palo Alto Research Center*
*__Corresponding author__: marc.mosko@parc.com

Contents

Introduction

Distributed consensus is vital for today's networks to provide fast, reliable, and lively services. The PAXOS algorithm [1, 2] The present work proceeds along the lines of Multi-Paxos [3]. Distributed consensus is expensive, so it is common to use the distributed consensus to elect a leader who can quickly process transactions. Like the Chubby [4] distributed lock system, a single system with a renewing lease handles the fast transactions.

1. PAXOS Overview

Following Lamport [2], we describe the Multi-Paxos protocol, which is built on top of the Basic Paxos protocol.

In Basic Paxos, a Proposer issues requests to a set of Acceptors in two rounds. In the first round, the Proposer sends a *prepare* request with a counter N, often taken as a natural number though any type with a total order works. It sends the *prepare* to a least a majority of acceptors. When an acceptor receives a *prepare(N)*, with will respond with an acknowledgement that N is the current maximum. If the acceptor had accepted any value (N_v, V) in the past, it will include that information to the Proposer. The Proposer, if it receives acknowledgements from a majority of the Acceptors, will proceed to the second round and send an Accept request for (N, V) to the Acceptors. If there were no previous (N_v, V), then the Proposer can select any V of its choice. If the accept the request, they will send an accept response. If the Proposer receives a majority of Accept responses, then it knows (N, V) was accepted as the consensus value. When an Acceptor accepts a value, it tells a Learner about it. The Learner will tell other interested systems about the consensus value.

Multi-Paxos uses a series of iterations of Basic Paxos such that a consensus value V can evolve over time, as $\{V_0, \ldots, V_i\}$. Using Basic Paxos, one can select a single master Proposer. After it has succeeded in Phase 1, it can submit as many values as it wants in Phase 2, so it can begin submitting pairs $\{i, V_i\}$.

The distinction between Proposer, Acceptor, and Learner, following [2], is not exclusive. In fact, we could call them just Servers and each is a potential Proposer, and Acceptor, and a Learner. They contend for the Proposer role and all act as Acceptors and Learners.

In multi-server settings, one ofter constructs the number N as the tuple (n, id), where n is a natural number and id is the unique identity of a server. The id could be administratively assigned, or it could be a cryptographic principal identity such as the hash of a public key. Using this system ensures that no two servers will ever issue the same number and that ties are broken, in this case based on the sort order of the server identities. One could also use the tuple $(n, priority, id)$, where $priority$ is a server-selected value of its willingness to become the master and could change iteration to iteration.

2. CCNx Distributed Consensus

Because cached responses are of little use in an on-line consensus protocol, each content producer should set the MaxAge of a ContentObject [cite the cache control document] to a small value or even zero. A small value would allow some retransmissions within those few milli-seconds. A zero value would prevent all cache responses.

We will identify a specific program protecting a specific variable using a consensus group as the tuple $\{grp, prg, var\}$, where grp is a group name, prg is a program name, and var is a protected variable (which could be considered an advisory lock for a set of variables). In some cases, we need to identify a specific version of a group such that we know the number of acceptors that constitute a majority. We use the notation $grpver$ for that version of a grp.

2.1 Proposers, Acceptors, and Learners

The current Proposer, or master, of a consensus group is elected using distributed consensus where each contending proposer bids to have its value accepted. The accepted value determines the master proposer. The actual Value is the name of a CCNx content object that describes the proposer.

The set of Acceptors is maintained as a consensus value. For a new system to enter as an Acceptor or be removed by the Proposer if it is non-responsive, is done as a protected variable. This allows the Proposer to know what constitutes a majority.

The identity of the Listener(s) (the function could be spread over multiple systems responsible for different notifications) is maintained as a protected value. The Acceptors know the identity of the current Listener and will inform it of accept choices. The Listener will use the identities of the acceptor group associated with the given value and notify all Acceptors and Proposers.

2.2 Individual request/response model

In the individual request/response model, the Proposer must know the identity of a majority of acceptors. It can determine this by reading the current value of the Acceptor group. The Proposer will send a unicast Interest message to a majority with a Prepare or Accept request. The Acceptor will response with a Content Object response either acknowledging the request or denying the request. Fig. 1 depicts this model. A proposer P sends individual Interest messages with payload to a majority of the acceptors, A_0, A_1, A_2. Each accept responds with a Content Object that follows the Interest reverse path.

Any system may read the current consensus value using Eq. 1. By asking the Proposer for a `read` request, it will return the current consensus value, being the tuple $(N, tier, V_{tier})$. A client may also specify a specific N or $(N, iter)$ pair. If the proposer does not know the consensus value for a read request, it responds with a NACK. Following [2], a proposer may also fill in unknown values with no-ops once it achieves consensus on that value. A NACK means indeterminate state, while a no-op means the consensus is there is no value for that state.

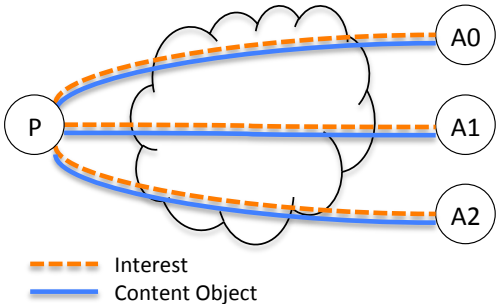

Figure 1. Individual Requests

```
/proposer/grp/prg/var/read[/N[/iter]]          (1)
/acceptor/grp/prg/var/prepare/N[/iter]         (2)
/acceptor/grp/prg/var/accept/N[/iter]          (3)
/target/grp/prg/var/learn/N[/iter]             (4)
```

A Prepare request message uses the name format shown in Eq. 2. The `/acceptor` prefix identifies the specific acceptor for routing purposes. The `/grp/prg/var` substring identifies the consensus group the acceptor participates in, the logical program and protected variable. The substring `prepare` identifies it as a Prepare request. The suffix `/N[/iter]` identifies the ordering N and the optional iteration $iter$. If using CCNx 1.0 labeled names, the suffix could take the form of `App:prepare=N`, and `App:iter=iter`. The accept request message is similar to the prepare, except for the different identifier of `accept`.

The payload of the request carries the state of the request. In particular it carries the value V. Often, the value will be a CCNx 1.0 Link to a particular piece of content that describes the current state of an algorithm. In some cases, it links to a CCnx 1.0 Manifest or is, in fact, that manifest imbedded in the request.

The response of an Acceptor is a CCNx Content Object. The Content Object follows the reverse path of the request back to the Proposer. It carries the consensus state for the current round or iteration. Generally, the Content Object should have a short or zero MaxAge to prevent excessive caching. Requests for state should go to the Proposer or the Acceptors to determine what is accepted state. Fetching a single Content Object response cannot determine the consensus.

Finally, Eq. 4 is used by a Learner. Acceptors will send an Interest to the `target` of the Learner with an accepted value. Once the Learner receives a majority of requests for that state, it can notify other Acceptors and Proposers of the value by sending the state to them using the same name but with a different `target`. The payload of the message is the consensus value tuple $(N, [iter,] V_{iter})$. The Acceptors and Learners may use semi-reliable communications with Interest out and Content Object ACK back. If available, the Learner could use Interest Multicast to Push the learned value, as described in the next section.

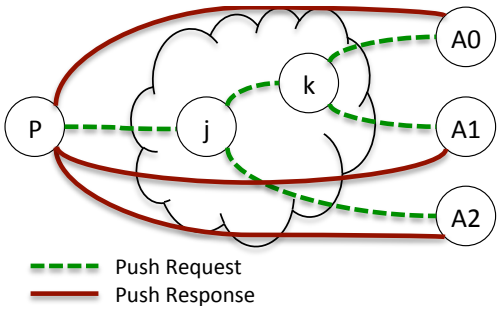

Figure 2. Multicast

Acceptors and Learners may aggregate state in a Learn message. *N* would be the value of the maximum state enclosed in the message, which would then iterate the learning tuples.

2.3 Interest multicast model

Using Interest multicast, a Proposer can send a single Push message to an Interest multicast group and have all listening acceptors receive the single message. Because the Proposer needs to know when it has received at least majority of responses, the group of Acceptors listening to the group name must be identified by a specific group version with a known number of Acceptors. Each acceptor needs to send an individual Push response message back to the Proposer, who identified itself in the Push payload. Fig. 2 illustrates this mode. A proposer sends a single Push request to a group name that uses multicast delivery. Each acceptor, $A_0 \ldots A_2$, responds with an individual Push response back to the Proposer. The responses do not necessarily follow the request reverse path.

The multicast model uses similar signaling to the previous model, except the routable prefix is now the group name `grp` rather than an individual member. In the Learning stage, the learner may use a mixture of interest/content object exchanges or of multicast push learns. If using semi-reliable signaling, a node would response with a Push ACK to a learn message.

`/grp/grpver/prg/var/prepare/N[/iter]` (5)

`/grp/grpver/prg/var/accept/N[/iter]` (6)

`/grp/grpver/prg/var/learn/N[/iter]` (7)

One difference to the individual model is that in some implementation the payload of a request must carry the target name to use in the response. This is not strictly necessary because all systems should be aware of the current system state and know the identity of the Proposer (always the response target for a `prepare` or `accept` request) and Learner (always the ACK destination for a Learn message) from the consensus state and the group version *grpver*.

3. Conclusion

We have presented a method to encode and execute the Basic Paxos and Multi-Paxos algorithms over CCNx 1.0 signaling and messages. We presented one variation that uses individual Interest requests and Content Object responses. We presented a second variation that uses Push multicast requests and individual Push responses.

References

[1] Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2):133–169, 1998.

[2] Leslie Lamport. Paxos made simple, fast, and byzantine. In *OPODIS*, volume 4, pages 7–9, 2002.

[3] Tushar Deepak Chandra, Robert Griesemer, and Joshua Redstone. Paxos made live - an engineering perspective (2006 invited talk). In *Proceedings of the 26th Annual ACM Symposium on Principles of Distributed Computing*, 2007.

[4] Mike Burrows. The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 335–350, Berkeley, CA, USA, 2006. USENIX Association.

[5] Jim Gray et al. The transaction concept: Virtues and limitations. In *VLDB*, volume 81, pages 144–154, 1981.