Distributed Agreement

December 14, 2019

Javier Palomares, Matt Molter

Introduction

- Project was to implement a weighted distributed agreement algorithm based on Paxos and one of Dr. Garg's papers on weighted Byzantine agreement
- Due to a misunderstanding, we also implemented a Byzantine Paxos and a weighted version of that algorithm
- Structure
    - Recap of Paxos
    - Weighted Paxos
    - Byzantine Paxos
    - Weighted Byzantine Paxos
    - Results
    - Future Work
    - Demo

Paxos

- Paxos is a non-Byzantine fault tolerant consensus algorithm for asynchronous distributed systems
- Safety Requirements
  a. Only a value that has been proposed may be chosen
  b. Only a single value is chosen
  c. A process never learns that a value has been chosen unless it actually has been
- Liveness Requirements
  a. Some value is eventually chosen
  b. If a value is chosen, all learners eventually learn of it
     - Omitted in this paper since it does not affect correctness of the algorithm

Paxos

- Three classes of processes
    a. Proposers
    b. Acceptors
    c. Learners
- Proposers and acceptors are the primary actors in the consensus portion of the algorithm
- If N is number of servers and f is number of faults tolerated, quorum size is $N - f$
    a. Simple way to guarantee non-empty intersection of any two quorums is to require $f < N/2$

Paxos

- Required properties
  - [P1.] An acceptor can vote for a value *v* in ballot *b* only if *v* is safe at *b*
  - [P2.] Different acceptors cannot vote for different values in the same ballot.
  - [P3a.] If no acceptor in the quorum has voted in a ballot numbered less than *b*, then all values are safe at *b*
  - [P3b.] If some acceptor in the quorum has voted, let *c* be the highest-numbered ballot less than *b* in which such a vote was cast. The value voted for in ballot *c* is safe at *b*. (By P2, there is only one such value.)

Paxos

- Algorithm
  - Phase 1
    - 1*a* - The ballot-b leader sends a 1*a* message to the acceptors.
    - 1*b* - An acceptor responds to the leader's ballot-*b* 1*a* message with a 1*b* message containing the number of the highest-numbered ballot in which it has voted and the value it voted for in that ballot, or saying that it has cast no votes.
  - Phase 2
    - 2*a* - Using the 1*b* messages sent by a quorum of acceptors, the leader chooses a value *v* that is safe at *b* and sends a 2*a* message containing *v* to the acceptors.
    - 2*b* - Upon receipt of the leader's ballot-*b* 2*a* message, an acceptor votes for *v* in ballot *b* by sending a 2*b* message.
  - Phase 3
    - Learning; omitted from our project

Introduction to Weights

- Used to normalize influence of each server on quorum so that sum is always 1
- For plain Paxos, all acceptors have equal weight.
  - Divide single server by number of servers to get each weight
  - Weights are trivially all the same, and sum to 1
  - Instead of tolerating f faults, we tolerate a weighted ρ of faults
  - In plain Paxos, tolerate $f < N/2$ faults, in weighted Paxos, tolerate $\rho < ½$ fault
- Special cases:
  - Centralized algorithm - Weight of 1 server is 1, weight of all others is 0

Why Introduce Weights?

- There is no obligation that weights are the same across all servers
- Can use it to establish a quantified level of trust or reliability
  - Increase weight of more reliable servers, decrease weight of less reliable servers
- Increases fault tolerance in certain cases
  - Number of failed servers can exceed N/2 as long as the remaining servers have a combined weight of >= 1/2
- Example use:
  - Buy mostly cheap commodity machines and assign them a low weight, and spend more on several enterprise grade machines and assign them a high weight
  - Failure of many commodity machines can be tolerated and recovered from as long as reliable machines remain up

Weighted Paxos

- Based on applying Garg's paper on weighted Byzantine agreements to Paxos
- Reduces to plain Paxos when all server weights are equivalent
- Same safety and liveness requirements as plain Paxos
- Same required maintenance properties as plain Paxos
- Two modifications
- First modification required is quorum properties
  - Instead of requiring $N - f$ for quorum, where $f < N/2$, we now require $1 - \rho$ weight, where $\rho < 0.5$
  - Instead of incrementing counter, add weights to counter
- Second modification is all servers need to keep a vector of all server weights

Weighted Paxos - Example

- Assume you have 4 servers, {P1, P2, P3, P4} with weights {0.25, 0.25, 0.25, 0.25} respectively in order to simulate plain Paxos
  - In this case, the loss of 2 servers makes consensus impossible
- Now assume they have weights of {0.3, 0.3, 0.2, 0.2} respectively
  - We can now lose P3 and P4 with no affect on our ability to achieve consensus
  - However, loss of any combination of P1 or P2 and any other process makes consensus impossible

PCon (modified Paxos)

- Used as basis for Byzantized Paxos algorithm
- Requires two modifications to required properties in order to determine safe values
    - [P3a.] If no acceptor in the quorum has voted in a ballot numbered less than *b*, then all values are safe at *b*.
    - [P3c.] If a ballot-*c* message with value *v* has been sent, for some $c < b$, and (i) no acceptor in the quorum has voted in any ballot greater than cand less than *b*, and (ii) any acceptor in the quorum that has voted in ballot *c* voted for *v* in that ballot, then *v* is safe at *b*.

PCon (modified Paxos)

- And two modifications to the actions
  - Addition to phase 1
    - 1*c* - Using the 1*b* messages from a quorum of acceptors, the leader chooses a set of values that are safe at *b* and sends a 1*c* message for each of those values.
  - Modification to phase 2
    - New 2*a* - The leader sends a 2*a* message for some value for which it has sent a 1*c* message.

Byzantine Paxos

- Lamport’s original algorithm for Paxos tolerates only benign failures, and can not handle Byzantine failures
- Original Byzantine Paxos algorithm introduced by Castro and Liskov
  - Our project uses a simplified version by Lamport
- There are several ways to handle Byzantine failures, which result in varying levels of fault tolerance
- The two that Lamport presents are equivalent to the Queen and King’s algorithms for Byzantine consensus, tolerating $f < N/4$ and $f < N/3$ respectively
- The algorithm for N/3 relies on cryptographic verification of message source, so we did not implement this algorithm due to time and complexity restraints

Byzantine Paxos

- Modifications need to be made to the required properties
- [P3a'.] If there is no ballot numbered less than *b* in which $f + 1$ acceptors have voted, then all values are safe at *b*.
- [P3b'.] If there is some ballot *c* in which acceptors have voted and there is no higher-numbered ballot less than *b* in which $f + 1$ acceptors have voted, then the value *v* voted for in *c* is safe at *b*.

Byzantizing Paxos

- The algorithm that we implemented required one primary change to tolerate Byzantine acceptors, with assumptions that $f < N/4$
- Increase quorum size from $2f + 1$ to $3f + 1$
  - Referred to as a byzquorum, which is the union of our original quorum and the number of Byzantine processes
- As mentioned, additional change can be made to reduce number of processes required to $> 3f$
  - Send 1*b* information with 2*a* request
    - Requires cryptographic validation of the 1*b* info at acceptors receiving 2*a* request

Byzantizing Paxos Cont'd

- Also needs to tolerate malicious leaders; actions so far have been only to tolerate malicious acceptors
- Primary method is for acceptors to broadcast 1*a* and 2*a* requests received to all other acceptors
  - Referred to as the 2*av* action, which emulates the original 2a action
- Once an acceptor has received a byzquorum of 2*av* broadcasts, it can respond to the leader with a 2*b* message

Byzantine Paxos Tradeoffs

- Increases message complexity since acceptors have to broadcast in response to 1*a* and 2*a* messages
- Significantly increases implementation complexity
- Reduces tolerance of non-Byzantine faults since quorum size has been increased

Byzantine Paxos Algorithm; N > 4f

- Phase 1
  - 1*a* - The ballot-*b* leader sends a 1*a* message to the acceptors.
  - 1*b* - An acceptor performs a 2*av* process. Once an acceptor receives a byzquorum of 2*av* messages, it responds to the leader's ballot-*b* 1*a* message with a 1*b* message containing the number of the highest-numbered ballot in which it has voted and the value it voted for in that ballot, or saying that it has cast no votes.
  - 1*c* - Using the 1*b* messages from a quorum of acceptors, the leader chooses a set of values that are safe at *b* and sends a 1*c* message for each of those values. The acceptors compare these values with their own safe values and accept if it matches.
- Phase 2
  - 2a - For a safe value *v* already sent in a 1*c* message, the ballot-*b* leader sends a 2*a* message containing *v* to the acceptors.
  - 2*av* - An acceptor broadcasts a 2*a* request to all other acceptors in a 2*av* message
  - 2b - Upon receipt of a byzquorum of 2*av* messages, an acceptor votes for *v* in ballot *b* by sending a 2*b* message to the acceptor.

Weighted Byzantine Paxos

- We add weights to Byzantine Paxos in a similar method to plain Paxos
- Byzquorum is changed to a weighted byzquorum, requiring total weight > ¾ for consensus and $\rho < ¼$
- Each process keeps a vector of all process weights
- Rest of algorithm remains the same
- Again, there is no obligation that weights are the same across all servers
    - Can use it to establish a quantified level of trust
    - Increase weight of trusted servers, decrease weight of less trusted servers
- Special cases:
    - Centralized algorithm - Weight of 1 server is 1, weight of all others is 0
        - Guarantees Byzantine acceptor tolerance as long as 1 server is trusted by eliminating all others from the system

Weighted Byzantine Paxos Properties

- [WBP1.] An acceptor can vote for a value *v* in ballot *b* only if *v* is safe at *b*
- [WBP2.] Different acceptors cannot vote for different values in the same ballot.
- [WBP3a.] Each message in S asserts that its sender has not voted.
- [WBP3b.] If there is some ballot *c* in which acceptors have voted and there is no higher-numbered ballot less than *b* in which acceptors with combined weight ≥ ρ have voted, then the value v voted for in c is safe at *b*
- [WBP3c.] For some *c* < *b* and some value *v*, (a) each message in S asserts that (i) its sender has not voted in any ballot greater than *c* and (ii) if it voted in *c* then that vote was for *v*, and (b) there are ≥ ρ weighted 1*b* messages from byzacceptors saying that they sent a 2*av* message with value *v* in a ballot ≥ *c*.

Weighted Byzantine Paxos Algorithm

- Phase 1
    - 1*a* - The ballot-*b* leader sends a 1*a* message to the acceptors.
    - 1*b* - An acceptor performs a 2*av* process. Once an acceptor receives a weighted byzquorum > 1 - ρ of 2*av* messages, it responds to the leader's ballot-*b* 1*a* message with a 1*b* message containing the number of the highest-numbered ballot in which it has voted and the value it voted for in that ballot, or saying that it has cast no votes.
    - 1*c* - Using the 1*b* messages from a weighted byzquorum > 1 - ρ of acceptors, the leader chooses a set of values that are safe at *b* and sends a 1*c* message for each of those values. The acceptors compare these values with their own safe values and accept if it matches.
- Phase 2
    - 2a - For a safe value *v* already sent in a 1*c* message, the ballot-*b* leader sends a 2*a* message containing *v* to the acceptors.
    - 2*av* - An acceptor broadcasts a 2*a* request to all other acceptors in a 2*av* message
    - 2b - Upon receipt of a weighted byzquorum > 1 - ρ of 2*av* messages, an acceptor votes for *v* in ballot *b* by sending a 2*b* message to the acceptor.

Weighted Byzantine Paxos - Example

- Assume you have 4 servers, {P1, P2, P3, P4} with weights {0.25, 0.25, 0.25, 0.25} respectively in order to simulate unweighted Byzantine Paxos
    - In this case, the loss of 1 server makes consensus impossible due to quorum requirement of N > 4f
- Now, assume they have weights of {0.3, 0.3, 0.2, 0.2} respectively
    - We can now lose P3 or P4 with no affect on our ability to achieve consensus
    - However, loss of P1 or P2 or any combination of processes makes consensus impossible
- Now, assume we have have weights {0.4, 0.4, 0.1, 0.1}
    - P3 and P4 can be lost separately or together with no affect on our ability to achieve consensus
    - Loss of P1 and P2 still make consensus impossible

Weighted Paxos - Example

- Assume you have 4 servers, {P1, P2, P3, P4} with weights {0.25, 0.25, 0.25, 0.25} respectively in order to simulate plain Paxos
  - In this case, the loss of 2 servers makes consensus impossible
- Now assume they have weights of {0.3, 0.3, 0.2, 0.2} respectively
  - We can now lose P3 and P4 with no affect on our ability to achieve consensus
  - However, loss of any combination of P1 or P2 and any other process makes consensus impossible

Weighted Byzantine Paxos - Uses

- An example use could be a dynamic system in which new processes are added regularly
- No information on whether new processes can be trusted
  - Assign low initial weight
- Long running processes with a non malicious history are more trusted
- Update weights as processes run longer with a trustworthy history
- Increases tolerance for malicious newcomers vs unweighted Byzantine Paxos

Future Work

- Implement the $N > 3f$ version of byzantine Paxos using cryptographic signatures, and apply weights to this version
- Implement learner portion of the algorithm
- Continue validation of Byzantine Paxos implementation
- Implement a solution for updating weights of more trustworthy processes and eliminate untrustworthy processes from the system
- Add support for reconfiguration

Demo