International Journal of Computing Science and Information Technology, **2018,**
ISSN: 2278-9669, April 2018 (http://ijcsit.org)

A Survey on Hashgraph The Future of Decentralized Technology

Monika M S , Bharat Kumar Malviya
MCA VI SEM Presidency College, Hebbal, kempapura, Bangalore, Karnataka
Email: Monikasomannams@gmail.com

Abstract

A new system, the Swirlds hashgraph consensus algorithm, is proposed for replicated state machines with guaranteed Byzantine fault tolerance. It achieves fairness, in the sense that it is difficult for an attacker to manipulate which of two transactions will be chosen to be first in the consensus order. It has complete asynchrony, no leaders, no round robin, no proof-of-work, eventual consensus with probability one, and high speed in the absence of faults. It is based on a gossip protocol, in which the participants don't just gossip about transactions. They gossip about gossip. They jointly build a hash graph reflecting all of the gossip events. This allows Byzantine agreement to be achieved through virtual voting. Alice does not send Bob a vote over the Internet. Instead, Bob calculates what vote Alice would have sent, based on his knowledge of what Alice knows. This yields fair Byzantine agreement on a total order for all transactions, with very little communication overhead beyond the transactions themselves.

1. Introduction

Distributed databases are often required to be replicated state machines with Byzantine fault tolerance. Some authors have used the term "Byzantine" in a weak sense, such as assuming that attackers will not collude, or that communication is weakly asynchronous. In this paper, "Byzantine" will be used in the strong sense of its original definition: up to just under 1/3 of the members can be attackers, they can collude, and they can delete or delay messages between honest members with no bounds on the message delays. The attackers can control the network to delay and delete any messages, though at any time, if an honest member repeatedly sends messages to another member, the attackers must eventually allow one through. It is assumed that secure digital signatures exist, so attackers cannot undetectably modify messages. It is assumed that secure hash functions exist, for which collisions will never be found. This paper proposes and describes the Swirlds hashgraph consensus algorithm, and proves Byzantine fault tolerance, under the strong definition.

2. CORE CONCEPTS

The hashgraph consensus algorithm is based on the following core concepts.
Transactions - any member can create a signed transaction at any time. All members get a copy of it, and the community reaches Byzantine agreement on the order of those transactions.
Fairness - it should be difficult for a small group of attackers to unfairly influence the order of transactions that is chosen as the consensus.
Gossip - information spreads by each member repeatedly choosing another member at random, and telling them all they know
Hashgraph - a data structure that records who gossiped to whom, and in what order.
Gossip about gossip - the hashgraph is spread through the gossip protocol. The information being gossiped is the history of the gossip itself, so it is "gossip about gossip". This uses very little bandwidth overhead beyond simply gossiping the transactions alone.
Virtual voting - every member has a copy of the hashgraph, so Alice can calculate what vote Bob would have sent her, if they had been running a traditional Byzantine agreement protocol that involved sending votes. So Bob doesn't need to actually her the vote. Every member can reach Byzantine agreement on any number of decisions, without a single vote ever being sent. The hashgraph alone is sufficient. So zero bandwidth is used, beyond simply gossiping the hashgraph.
Famous witnesses - The community could put a list of n transactions into order by running separate Byzantine agreement protocols on O(n log n) different yes/no questions of the form "did event x come before event y?" A much faster approach is to pick just a few events (vertices in the hashgraph), to be called witnesses, and define a witness to be famous if the hashgraph shows that most

members received it fairly soon after it was created. Then it's sufficient to run the Byzantine agreement protocol only for witnesses, deciding for each witness the single question "is this witness famous?" Once Byzantine agreement is reached on the exact set of famous witnesses, it is easy to derive from the hashgraph a fair total order for all events.
Strongly seeing - given any two vertices x and y in the hashgraph, it can be immediately calculated whether x can strongly see y, which is defined to be true if they are connected by multiple directed paths passing through enough members. This concept allows the key lemma to be proved: that if Alice and Bob are both able to calculate Carol's virtual vote on a given question, then Alice and Bob get the same answer. That lemma forms the foundation for the rest of the mathematical proof of Byzantine agreement with probability one.

3.Gossip about gossip: the hashgraph

Hashgraph consensus uses a gossip protocol. This means that a member such as Alice will choose another member at random, such as Bob, and then Alice will tell Bob all of the information she knows so far. Alice then repeats with a different random member. Bob repeatedly does the same, and all other members do the same. In this way, if a single member becomes aware of new information, it will spread exponentially fast through the community until every member is aware of it. The history of any gossip protocol can be illustrated by a directed graph like Figure 1. Each vertex in the Alice column represents a gossip event. For example, the top event in the Alice column represents Bob performing a gossip sync to Alice in which Bob sent her all of the information that he knew. That vertex has two downward edges, connecting to the immediately-preceding gossips for Alice and Bob. Time flows up the graph, so lower vertices represent earlier events in history. In a typical gossip protocol, a diagram such as this is merely used to discuss the protocol; there is no actual graph like that stored in memory anywhere.

4. Consensus algorithm

It is not enough to ensure that every member knows every event. It is also necessary to agree on a linear ordering of the events, and thus of the transactions recorded inside the events. Most Byzantine fault tolerance protocols without a leader depend on members sending each other votes. So for n members to agree on a single YES/NO question might require O(n 2) voting messages to be sent over the network, as every member tells every other member their vote. Some of these protocols require receipts on votes sent to everyone, making them O(n 3). And they may require multiple rounds of voting, which further increases the number of voting messages sent.

5. Proof of Byzantine fault tolerance

This section provides a number of useful definitions, followed by several proofs, building up from the Strongly Seeing Lemma (lemma 5.12) to the Byzantine Fault Tolerance Theorem (theorem 5.19). In the proofs it is assumed that there are n members ($n > 1$), more than 2n/3 of which are honest, and less than n/3 of which are not honest. It is also assumed that the digital signatures and cryptographic hashes are secure, so signatures cannot be forged, signed messages cannot be changed without detection, and hash collisions can never be found. The syncing gossip protocol is assumed to ensure that when Alice sends Bob all the events she knows, Bob accepts only those that have a valid signature and contain valid hashes corresponding to events that he has. The system is totally asynchronous. It is assumed that for any honest members Alice and Bob, Alice will eventually try to sync with Bob, and if Alice repeatedly tries to send Bob a message, she will eventually succeed. No other assumptions are made about network reliability or network speed or timeout periods. Specifically, the attacker is allowed to completely control the network, deleting and delaying messages arbitrarily, subject to the constraint that a message between honest members that is sent repeatedly must eventually have a copy of it get through.

6. Fairness

Most existing systems for distributed consensus can fail to be "fair" in their consensus ordering of transactions. To see this, first consider a stock market that is run by a single server. Alice and Bob each submit a bid to that server, with Alice submitting it just before Bob. If the server is fair, then it will count Alice's transaction as occurring before Bob's. For some applications, the exact order does not matter, but for a stock market it can be critically important that this decision be made fairly. Now consider a distributed peer-to-peer system, where there is no single server, but there is a community that will reach consensus on whose transaction was first. It may still be critically important that the consensus decision is fair. But what should be the definition of "fair"?

7. Generalizations and enhancements

7.1 proof-of-stake. So far, it has been assumed that every member is equal. The above algorithms refer to things depending on "more than 2n/3 of the members" and "at least half of the famous witness events". They also use the idea of a "median" of a set of numbers. The proof shows Byzantine convergence when more than 2n/3 of the members are honest. It is easy to modify the algorithm to allow members to be unequal. Each member can be assumed to have some positive integer associated with them, known as their "stake". Then, the votes would be replaced with weighted voting, and the medians with weighted medians, where votes are weighted proportional to the voter's stake. In all of the above definitions, algorithms, and proofs, define "more than 2n/3 members" to mean "a set of members whose total stake is more than 2n/3, where n is the total stake of all members". The "median of the timestamps of events in S" would become "the weighted median of the timestamps in S, weighted by the stake of the creator of each event in S". The weighted median can be thought of as taking each event y in S, and putting multiple copies of the timestamp of y into a bag, where the number of copies equals the stake of the member who created y. Then take the median of the timestamps in the bag. The Byzantine proof applied as long as the attackers constituted less than 1/3 of the population. With these new definitions, it will now apply when the attackers together have a stake that is less than 1/3 of the total stake of all members. This new proof-of-stake system is more general than the unweighted system. It can still be used to implement the unweighted system, by simply giving every member a stake of 1. But it can also be used to provide better behavior. For example, the stake might be proportional to the degree to which a member is trusted. Perhaps members who have been investigated in some way should be trusted more than others. Or it could be used to give greater weight to members who have a greater interest in the system as a whole working properly. A cryptocurrency might use each member's number of coins as their stake, on the grounds that those with more coins have a greater interest in ensuring the system runs smoothly. Or a community could be started by a group of members with mutual trust, each of which is given an equal stake. Then, each existing member could be allowed to invite arbitrarily many new members to join, subject to the constraint that the inviter must split their stake with the invitee. This would discourage a Sybil attack, where one member invites a huge number of sock puppet accounts, in order to control the voting.

7.2. Signed state. Another enhancement to the system is to have signed states. Once consensus has been reached on whether each witness in round r is famous or not, a total order can be calculated for every event in history with a received round of r or less. It is also guaranteed that all other events (including any that are still unknown) will have a received round greater than r. In other words, at this point, history is frozen and immutable for all events up to round r. A member can therefore take all the transactions from those events, and feed them into a database in the consensus order, and calculate the state that is reached after processing those transactions. Every member will calculate the same consensus order, so every member will calculate the same state. This is a consensus state. Each member can take the hash of this state and digitally sign it, and put the signature into a new transaction. Soon after, every member will have received by gossip many signatures for the consensus state. Once signatures are collected from at least 1/3 of the population, that consensus state, along with the set of signatures, constitutes a signed state that is an official consensus state for the system at the start of round r. It can be given to people outside the community, and they can check the signatures, and therefore trust the state. At this point, a member can feel free to delete all the transactions that were used to create the state, and delete all the events that contained those transactions. Only the state itself needs to be kept. It might be possible to do this every few minutes, so there will never be a huge number of transactions and events stored. Only the consensus state itself. Of course, a member is free to preserve the old events, transactions, and states, perhaps for archive or audit purposes. But the system is still immutable and secure, even if everyone discards that old information. Given the assumption that less than 1/3 of the population is dishonest, the signed state is guaranteed to have at least one honest signature, and so can be trusted to represent the community consensus, as found by the consensus algorithm. If the set of members (or their stake) can change over time, then that stake record (and its history) will also be part of the state. The threshold of 1/3 could be replaced with something else, such as more than 2/3, and the system would still work.

7.3. Efficient gossip. The gossip protocol makes very efficient use of bandwidth. Suppose there are enough transactions being created that every event contains at least one transaction. In any replicated state machine, using a point-to-point network such as the internet, it will be necessary for each member to receive each signed transaction once, and to also send each signed transaction on average once. For the hashgraph gossip, the same is true, except that the signature is for the event containing the transaction, rather than for the transaction itself. The only additional overhead is the two hashes and the timestamp, plus the array of counts at the start of the sync. However, the hashes themselves don't have to be sent over the internet. It is sufficient to merely send the identity of the creator of the event, and the sequence number of its other-parent.

7.4. Fast elections. That second part of the algorithm is a Byzantine agreement step for deciding fame. It has an interesting property. When a group of members are all online and all participating regularly, the Byzantine agreement will be applied to a set of elections where almost all the voters start with identical votes. That is because a round r + 1 witness will strongly see many of the round r witnesses, so a round might be expected to last about two "gossip periods", where a gossip period is the time it takes for a message to propagate through the entire community. This should be the time to do log2(n) syncs, when there are n members online. For a round r + 1 witness x to vote YES on the fame of a round r witness y, it isn't necessary for x to strongly see y. It can merely see y. It would be expected that y would propagate to all the online members in a single gossip period. So there is an overwhelmingly high probability it will propagate to them within two gossip periods. So in practice, when everyone is online and participating, the fame of witnesses is almost always decided immediately, without the need for many rounds of voting.

7.5. Efficient calculations. The first part step of the algorithm is to assign a round of either r or r + 1 to an event, based on whether it can strongly see enough round r events. So it is necessary to calculate whether a round r witness event x can be strongly seen by an arbitrary event y. The following is one way to calculate that answer. Give each event a sequence number that is one greater than the sequence number of its self-parent. Store an array for y and an array for x. The y array remembers the sequence number of the last event by each member that is an ancestor of y. The array for x remembers the sequence number of the earliest event by each member that is a descendant of x. Compare the two arrays, and find how many elements in the y array are greater than or equal to the corresponding element of the x array. If there are more than 2n/3 such matches, then y strongly sees x. The comparison of the x and y arrays can be sped up by multithreading (to use more cores), packing multiple elements into one integer (to use the ALU more efficiently), using assembly language (to access the CPU vector instructions) or using the GPU (for more vector parallelism).

8. Conclusions

A new system has been presented, based on the Swirlds hashgraph data structure, and the Swirlds hashgraph consensus algorithm. It is fair, fast, Byzantine fault tolerant, and extremely bandwidth efficient due to virtual voting. The algorithm is given in pseudocode in the figures, using an imperative language, but it is also very natural to describe it in a functional form. The appendix gives the algorithm in a functional form, which is concise, and may be of interest.

9. References

[1] www.Hashgraph.com
[2] www.swirlds.com
[3] www.ipfs.io
[4] www.thewindowsclub.com
[5] www.hiddenforcespod.com
[6] www.cryptoslate.com