Fast Reconciliation with Time Ordered Patricia Chains

Abstract

Patricia trie based data structures for CRDT reconciliation incur expected cost logarithmic in the number of total messages. Instead, I propose a structure that orders messages by the local time at which they were sent. Assuming all nodes incur independent exponentially distributed clock drift, we can show that the distribution of local times for messages that have been sent but not yet received by some pair of nodes in a cluster of gossiping peers obeys a hypoexponential distribution, for which we can calculate an upper bound on the median time T. By using a separate Patricia trie for each interval of T seconds, we can bring the communication and time complexity of CRDT reconciliation down a constant. Formal derivations of these results are backed up by results from an empirical simulation on a network of 100 gossiping nodes observed over a period of twenty five minutes.

Introduction

State machine replication ensures that the same sequence of operations gets carried out in exactly the same order on every node in a cluster, and that reads from any node in the cluster at any time will all return the same state. Supporting this highly general model comes at a cost, however. These systems have trouble scaling to large numbers of nodes, generally require all clients to communicate with a single coordinator node on every write, and require complicated multi-phase commitment protocols.

State machine replication becomes a much easier problem if we assume a relaxed form of consistency known as *strong eventual consistency* (Shapiro et al. 2011). There are two requirements in this model:

- *Eventual update*: If an update is applied by a correct replica, then all correct replicas will eventually apply that update.
- *Convergence*: Any two correct replicas that have applied the same set of updates are in the same state (even if the updates were applied in a different order).

CRDTs

This form of consistency is easy to achieve, even in a fully asynchronous setting, when state machine operations form a join-semilattice (colloquially, a Conflict-free Replicated Data Type or CRDT). Specifically, all state-machine operations must be able to be combined by an associative, commutative and idempotent operation known as the *join* or *least upper bound* operation. For example, imagine if the state and all operations are sets. The join operation, in this case, is a union. Because set-union is commutative and idempotent, the order in which each replica joins the operations to produce the final state is arbitrary. Other classic examples of CRDTs are counters (in which the join of integer semilattice elements is taking the maximum), graphs (which are just sets of edges), timestamped operations (in which the join operation picks the element with the higher timestamp), and sorted lists (which are isomorphic to sets).

Recently, CRDTs have seen prominent use in peer-to-peer collaborative editing applications. A collaboratively edited text document can be represented as a sorted list of position-annotated characters, or alternately as a tree in which edges are strings of sequential characters. A collaboratively edited vector graphics document can be represented as a set of vector graphics primitives. A distributed key value is just a set of (key, value) pairs, where some of the values may be tombstones.

The Gossip Protocol

To achieve strong eventual consistency, most CRDT systems use a gossip protocol. Each node in a network of size N samples a waiting time from an exponential distribution with rate β/N, then contacts another node in the system at random. The two nodes take the join of each others' states, which propagates information throughout the network. In what follows, I will assume that the underlying CRDT is a set of 'messages', where each message has some hashable binary representation. This is not a restrictive assumption, as it corresponds to the so called "free" semilattice which is able to represent any other. Additionally, I will assume that each message is accompanied by metadata including its senders' unique identifier and local time upon message generation. Messages are assumed to be uniquely identifiable given their sender's identity and local generation time. Because every node has the opportunity to gossip with every other node, Gossip protocols are naturally Byzantine fault tolerant as long as each message is signed by its sender.

Patricia Structured Merkle Trees

For this system to be useful in practice, we need a fast way for each node to find the union of its message set with that of another node. The standard approach to this problem is to use a Merkle tree (Auvolat and Taıani 2019), (Ramabaja and Avdullahu 2020). A Merkle tree representing a set S partitions S into disjoint sets A and B, recursively constructs Merkle trees for these left and right children, and computes a hash for S combining the hashes of its left and right children. An example is shown in Figure 1. To check whether two sets represented by Merkle trees are equal, it is sufficient with high probability to simply compare the root hashes. To find the difference between two sets, we can descend recursively from the root along every edge with mismatched hashes.

Specifically, when a node Y wants to receive new messages from a node X, it follows the following algorithm:

- Ask X for the hashes of its root node and its children, along with the range of values contained in its left and right subtrees.
- If the range of values in Y's root node is entirely contained in one of X's subtrees, recurse on that subtree. All of the messages in the other subtree will be new to Y, and should be sent over as well.
- Otherwise, if the hashes of X's root node and Y's root node are the same, there are no new messages, and the algorithm halts.
- If the root hashes differ, then the hashes of at least one of their children must differ as well. Recurse on any child nodes with mismatched hashes.

For a fully balanced Merkle tree containing n messages, this recursion will always terminate after $\log n$ steps. This means that the procedure above produces the set differences $X \setminus Y$ and $Y \setminus X$ in time and communication complexity $O((X \Delta Y) \log n)$ where Δ indicates the symmetric difference.

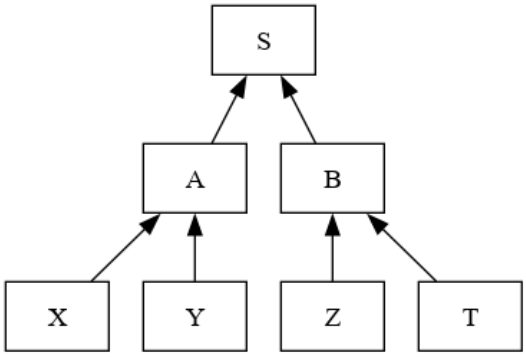

Figure 1: Generic Merkle Tree

State of the art Merkle trees partition S into A and B using the structure of a *Patricia trie*. A node in a Patricia trie represents a set of messages which have the same binary representation up to bit k. Its left child is a Patricia trie containing the subset of these messages with a one in bit position $k+1$, while its right child will be those with a zero. We can keep track of the bits shared by all the children of a Patricia trie node using two 64-bit words: a mask and a shared path. An example is shown in Figure 2. If messages' binary representations are distributed uniformly, the expected number of nodes in each branch is $|S|/2$, making the tree properly balanced.

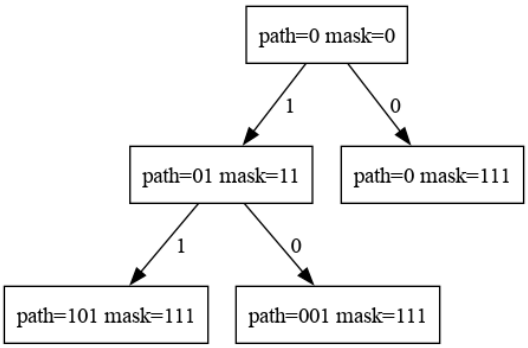

Figure 2: Patricia trie for 0, 001 and 101.

Improving on Patricia Trees

In this paper, I claim that seeking a well balanced Merkle tree is actually a mistake. If we arrange our Merkle tree as a time ordered linked list or *Merkle Chain* instead of a tree as in Figure 3, I claim that reconciliation is possible in *constant* rather than logarithmic expected time in the number of messages, leading to a protocol with constant rather than logarithmic communication complexity in the total number of messages. This improvement is possible because the exchange procedure described above neglects the fact that the probability a node lacks a given message decreases exponentially with time since it was sent.

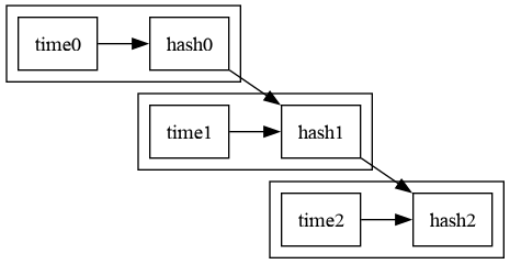

Figure 3: Merkle chain

Mean Field Analysis of the Gossip Protocol

To see the exponential decay in probability over time, we can turn to mean field analysis. In (Bakhshi et al. 2008), the authors proposed the use of mean field analysis

for analyzing the Gossip protocol, but failed to find an analytic form for the continuum limit of the underlying Markov chain. My analysis is instead based on (Draief and Massouli 2010), which applies mean field analysis to the classic Susceptible-Infected-Susceptible model of epidemics.

Let X be the number of nodes that know about a message in a network of N nodes. The gossip protocol described above, in which each node waits for a single exponentially distributed timer with rate β/N to expire before contacting a random peer, is equivalent to one in which each node starts a separate timer for each peer in the network with rate $\frac{\beta}{N(N-1)}$ and exchanges message with that peer once the timer goes off.

In this model, X evolves according to a continuous time Markov chain. Each of the X nodes with the message will tell one of their neighbors as soon as one of their $X-1$ exponential clocks expires. Each clock expires with rate $\frac{\beta}{N(N-1)}$, making the rate at which the first timer goes off (and therefore the jump rate between states X and $X+1$) equal to $X(N-X)\frac{\beta}{N(N-1)}$. This rate can equivalently be expressed as $\frac{N}{N-1}x(1-x)\beta$, where $x=\frac{X}{N}$ is the fraction of nodes that know about the new message. This means we can express $X(t)$ in terms of a rate-1 Poisson process $\Gamma(t)$ as

$$X(t) = X(0) + \frac{N}{N-1}\Gamma\left(\int_0^t x(1-x)\beta\, dt\right)$$

Similarly,

$$x(t) = x(0) + \frac{1}{N-1}\Gamma\left(\int_0^t x(1-x)\beta\, dt\right)$$

By Kurtz's theorem (Kurtz 1971), this means that in the limit as $N\to\infty$, x almost surely follows the deterministic ODE $\frac{dx}{dt} = \beta x(1-x)$. Intuitively, this means that at any given moment in time, the fraction x of nodes with the message are contacting the fraction $(1-x)$ of nodes without it at rate β. The ODE is solved as

$$x(t) = \frac{e^{\beta t}}{e^{\beta t}+c}$$

where $1/(1+c)$ is the fraction of nodes that know about the message at time 0.

Let t be the number of seconds that have passed since a given message was sent. Assume that message sending started n seconds ago, so t is bounded by n. Let Z be the event that this message is not yet known to a given node. We know that $p(Z|t) = \frac{c}{e^{\beta t}+c}$, so we can compute $p(t|Z)$ using Bayes' rule:

$$p(t|Z) = \frac{p(Z|t)p(t)}{\int_0^n p(Z|t)p(t)dt}$$

This simplifies to

$$p(t|Z) = \frac{\beta c}{(c+e^{\beta t})\left(-\log\left(c+e^{\beta n}\right)+\log(c+1)+\beta n\right)}$$

When we take the limit as the time horizon goes to infinity, we get

$$\lim_{n\to\infty} p(t|Z) = \frac{\beta c}{\log(c+1)\,(c+e^{\beta t})}$$

I will assume that when a message is generated, it is immediately forwarded to 1% of the the total nodes before gossip begins, making c=99. In this case, a plot of $p(t|Z)$ is shown in Figure 4 using $\beta = 2$. The probability of needing to exchange a message sent more than 6 seconds ago is negligible.

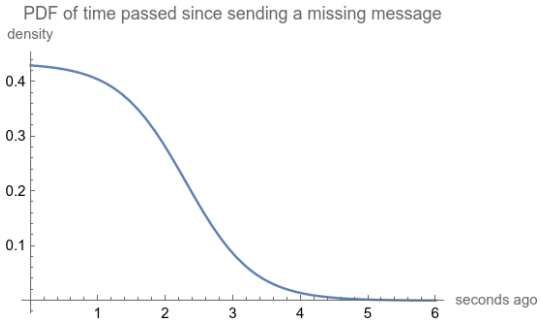

Figure 4: Distribution of time since a message was sent given that it was not yet received on a given node

Patricia Chains

Using this result gives a simple way to get reconciliation times far better than what is possible with Patricia tries. Consider when nodes X and Y attempt to reconsile their message sets. Arrange all of X's messages in a list l ordered by the timestamp at which they were generated. This gives a Merkle *Chain* rather than a Merkle tree. If we assume that new messages get generated with constant rate k, this means that any message in l that Y is missing will lie within the first $6k$ entries with high probability.

In practice, nodes do not know the true timestamp at which each message is generated; they each have noisy estimates of the true time, as their internal clocks will drift in between NTP synchronizations. We can model this phenomenon by assuming that the timestamp attached to each message is the true timestamp plus an exponentially distributed drift term with scale ν. For times greater than 1 second, the pdf in Figure 4 is well approximated by that of an Exponential distribution. The sum of two independent exponentially distributed random variables with scales ν and β^{-1} has a hypoexponential distribution with median upper bounded by its mean of $T = \nu + \beta^{-1}$.

We can use this fact to improve the constant factor for the $O(1)$ reconciliation time in Merkle chains. Bin all messages into T second chunks of time. Make a Patricia

structured Merkle tree for each bin and collect the bins into a Merkle chain, as shown in Figure 5. I call the result a *Patricia Chain*. A missing message will land in the most recent T-second block with probability at least $\frac{1}{2}$, in which case it can be found in the relevant Patricia trie in expected time $\log(kT)$. This means the expected lookup time to find a missing message in a set of n messages obeys the recurrence $f(n) \leq 1 + \frac{1}{2}f(n - kT) + \frac{1}{2}\log(kT)$. Solving the recurrence tells us that expected lookup time is at most $2 + \log(kT)$.

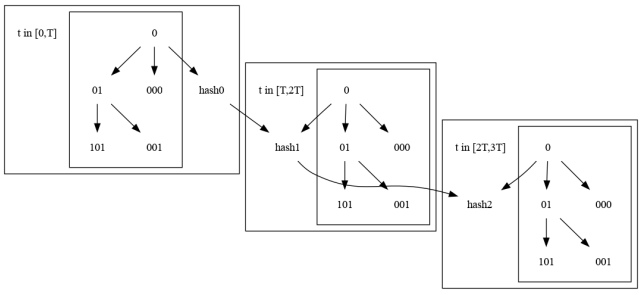

Figure 5: A patricia chain for messages with timestamps in [0,3T]

Related Work

Causal DAGs

Instead of ordering messages by a local timestamp, (Kleppmann and Howard 2020) use a causal ordering determined by the sender of each message. Each node keeps track of a DAG of messages. When a node generates a new message, it collects the hashes of root nodes in its DAG and attaches this set of hashes as metadata. This is similar to the timestamp we attached when using Merkle chains: the set of hashes acts as a *causal timestamp*. Message a is considered a parent of message b in the DAG if a has the hash of b included in its metadata. An example of the state stored at a node using this strategy is shown in Figure 6. Gathering messages from node X to send to node Y using this representation proceeds similarly to reconciliation in a Merkle chain. X sends Y the hashes of the roots of its DAG. Y responds with the hashes it did not already know about. X tries again, sending the children of these hashes. The process repeats until Y finds all its missing messages.

As with Merkle chains, this algorithm only requires searching through recently sent messages to reconcile different sets. For this algorithm to perform well, however, the number of root nodes in each node's DAG must never get too large, as each round of reconciliation requires sending this set. Full analysis of when these conditions are met is beyond the scope of this paper, but we can see intuitively that this situation can easily develop if the gossip rate is too low relative to the aggregate generation rate. Multiple heads occur any time messages are generated in parallel and are not causally ordered. In practice, we will see that the performance of this algorithm is similar to that of Merkle chains. The number of root nodes stays roughly constant over time, but this constant is much higher than what we can obtain with Patricia chains.

The authors augment this base algorithm with an optimization which, for each node, keeps track of the the most recently seen set of root messages for every other node. On reconciliation between nodes X and Y, Y constructs a Bloom filter with all the messages it has received that are ancestors of its cached roots for node X. This allows X to quickly determine a large subset of the messages it needs to forward to Y by checking if any of its new messages are not in the Bloom filter. Unfortunately, as this requires each node to keep track of $O(m)$ root message sets, the optimization is impractical for large clusters.

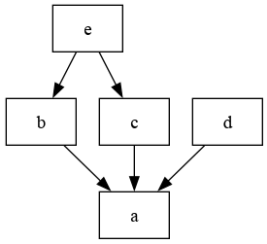

Figure 6: Causal Merkle DAG. Messages b, c and d were sent concurrently by nodes that knew about a. Message e was sent by a node that did not yet know about d.

Read Repair

Instead of using a gossip protocol to bring nodes towards consistency, we could use read repair. The idea requires a full lattice (with a commutative 'meet' operation) rather than simply a semilattice (with only a 'join') . For example, if the replicated state is a set of strings, the join operation corresponds to set union and the meet to set intersection. Clients ask multiple random servers about whether they have the given string in their states, asking for a 'meet' operation with a provided singleton set. The client will perform a join of all the responses, and then forward the result back to the nodes it asked. These nodes will join the response into their own states. With enough clients issuing requests for different lattice elements, strong eventual consistency is still possible, although convergence can take much longer. Note that this consideration of read repair as a general operation on lattices is novel, even though the underlying idea is widely established.

Node-Wide Dot Based Clocks

Each message can be uniquely associated with the node that introduced it paired with a per-node, monotonically increasing identifier. This pair is referred to as a "dot based clock" in the literature. A set of messages,

therefore, can be described as a map from originating nodes to a set of integer intervals. This idea comes from (Gonçalves et al. 2017). Although the original paper did not include a probabilistic analysis, we can see from the same argument as above that there is a length of time T for which all messages sent more than T seconds ago have already been received with high probability. This means the number of intervals stored for each originating node will remain constant even as the total number of messages grow. Therefore, with high probability, the computational cost of computing a set difference using dot based clocks scales only with the number of nodes, not the number of messages. This linear dependence on the number of nodes, however, can present a problem in systems of large enough scale.

Polynomial Based Reconciliation

An alternative scheme (Minsky et al. 2003) interprets the hash of each message as an element of the finite field $\mathbb{F}_{2^{64}}$. Each node computes polynomials which have their message sets as roots. If X has messages 2, 5, and 67, for example, it would create the monic polynomial $p_X(z) = (z-2)(z-5)(z-67)$. To find the roots present in p_X that are not in Y's polynomial p_Y, it suffices to find the roots of p_X/p_Y. If the symmetric difference in their message sets is small, the polynomial p_X/p_Y will have low degree d, which means it can be uniquely recovered from a small number $m \geq d$ of evaluation points using rational function interpolation. Because of the efficiency of the Gossip protocol, d will be small with high probability.

To turn this representation into a protocol, Y can send X the values of p_Y at these m evaluation points. X can compute p_X at these points and divide to find p_X/p_Y. Once X finds the roots of this polynomial, it can send Y the associated messages. Likewise, Y can compute the roots of p_Y/p_X to send X its missing messages.

With high probability, we can tell that the chosen m will appropriately upper bound d by computing p_X and p_Y at an extra k validation points as well. If the recovered polynomial agrees with p_X/p_Y at these points, the authors show that $m \geq d$ with probability over $1 - 10^{-11}$. While this technique has nearly optimal communication complexity, the computation time needed to factor the resulting polynomial and resolve which messages are missing is much higher than with Merkle trees, which explains why this approach is not used more widely in practice.

Evaluation

To confirm the theoretical results derived above, I compared the performance of Patricia structured Merkle trees, list structured Merkle tries, Patricia Chains and causal DAGs described above by simulating a large network of gossiping peers using a pool of coroutines on a single machine. For each strategy, I keep track of the tree-traversal depth needed during each least-upper bound computation. I average all nodes' cumulative tree depth explored up to each time t. This number is proportional to both the average computation time necessary on each node, as well as the network communication cost necessary for all set reconciliations.

I conducted simulations varying the scale ν of the time skew distribution, the scale β^{-1} of the gossip process and the scale α of the message generation process from their default values of $\nu = 1, \beta^{-1} = \frac{1}{2}, \alpha = 1$. All simulations were run for 25 virtual minutes. As expected, Patricia structured Merkle tree performance deteriorated with time as the number of messages increased, while reconciliation cost for chain structured Merkle trees remained constant.

As shown in Figure 7, when the noise in message generation time estimates was high, Patricia tries fared substantialy better than chains while the number of messages was small. Patricia Chains (or *PatChains*) had the same constant scaling as Merkle Chains, but with a much smaller constant factor, beating the other two data structures by a large margin regardless of the total number of messages. Causal DAGs were unaffected by local time skew, tracking the overall performance of Merkle chains.

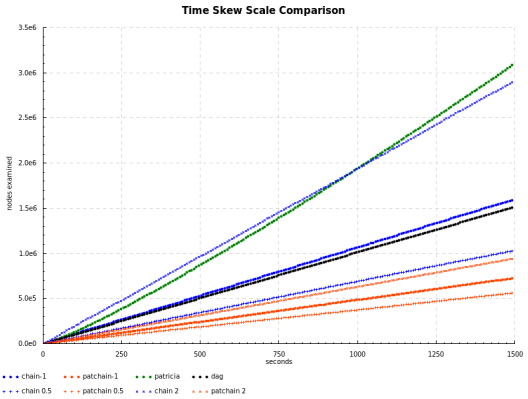

Figure 7: Algorithm Comparison for Timestamp Noise

With increased message generation rates, all the studied data structures incurred increased reconciliation times, as shown in Figure 8. This is unavoidable, given that an increased generation rate produces more messages to reconcile. Causal DAGs performed especially well with high generation rates, as each generated messages collapses the number of root nodes, improving the performance of future reconciliations.

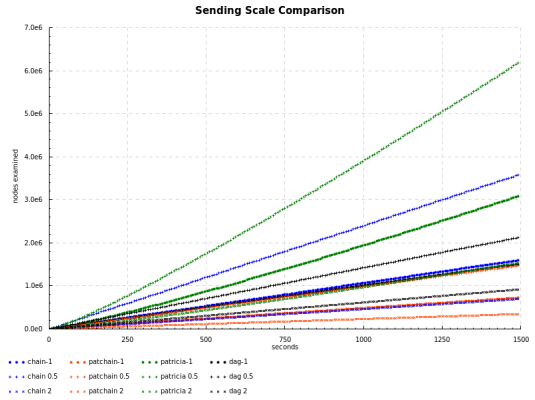

Figure 8: Algorithm Performance Across Message Rates

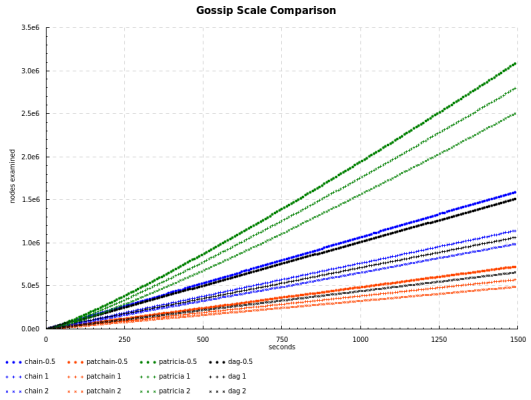

Figure 9: Algorithm Performance Across Gossip Rates

As nodes gossiped among themselves more frequently, the benefits of using a chain representation became more pronounced. We see in Figure 9 that a fast gossip rate can make up for time skew. This is anticipated, as the expected number of messages in each patchain block is $k(\nu+\beta^{-1})$. A more frequent gossip rate also makes causal DAGs perform better, as fewer pairs of messages end up without a causal order. Patricia tries, in contrast, are unaffected.

Conclusion and Further Work

Because of the efficiency of the gossip protocol, data structures for set reconciliation should prioritize access to recently sent messages. Even when this exact sending time is unknown, the noisy estimate provided by a nodes' local time is sufficient to allow for optimal communication complexity up to constant factors.

That said, my analysis has assumed that all nodes remain on-line throughout the gossip protocol, quickly receiving updates about new messages. While this modeling assumption may not be far from the truth in relatively centralized settings like data centers, it does not accurately describe the asynchronous peer to peer environment for collaborative document editing in which many CRDTs are used in practice. Allowing for new nodes joining and leaving the network would considerably complicate mean field analysis, and it is not clear that my results would continue to apply under such a regime. Practically, one could imagine using a hybrid reconciliation protocol in which, for the first reconciliation a node goes through after coming back on-line, a traditional Patricia-trie based strategy is used, continuing thereafter to use Patricia chains. I leave such extensions to further research.

References

[1] Alex Auvolat and Francois Taıani. "Merkle Search Trees: Efficient State-Based CRDTs in Open Networks". In: *IEEE Reliable Distributed Systems* (2019).

[2] Rena Bakhshi et al. "MeanField analysis for the evaluation of gossip protocols". In: *International Conference on the Quantitative Evaluation of Systems (QEST)* (2008).

[3] Moez Draief and Laurent Massouli. *Epidemics and rumours in complex networks*. Cambridge University Press, 2010.

[4] Ricardo Jorge Tomé Gonçalves et al. "DottedDB: Anti-Entropy without Merkle Trees, Deletes without Tombstones". In: *IEEE Reliable Distributed Systems* (2017).

[5] Martin Kleppmann and Heidi Howard. "Byzantine Eventual Consistency and the Fundamental Limits of Peer-to-Peer Databases". In: *ArXiv* (Dec. 1, 2020).

[6] T. G. Kurtz. "Limit Theorems for Sequences of Jump Markov Processes Approximating Ordinary Differential Processes". In: *Journal of Applied Probability* 8.2 (1971), pp. 344–356.

[7] Y Minsky et al. “Set reconciliation with nearly optimal communication complexity”. In: *IEEE Transactions on Information Theory* 49.9 (2003).
[8] Lum Ramabaja and Arber Avdullahu. “The Bloom Tree”. In: *ArXiv* (Feb. 19, 2020).
[9] Marc Shapiro et al. *Conflict-free Replicated Data Types*. 7687. INRIA, 2011.