Creating a Novel Consensus Algorithm for Distributed Computing use Cases

R. Kannadasan, N.Prabakaran, A.Krishnamoorthy, K,Naresh, Saravana Balaji.B, A.S.Anakath,

Abstract*: *There are many consensus algorithms that exist in parallel computing that involve multiple computing units like virtual machines which make use of available resources and arrive at a single agreeable state for the combined system. This is done on the basis of voting which itself branches into several arrangements like voting, functions of central tendencies, weighted functions of central tendencies etc. Some applications that consensus algorithms try to cover are: deciding on transaction operations (read, write, commit); deciding on node leaders of a system; maintaining replicas in the state of a machine (also called a state machine) and creating consistency between them. Some common algorithms of this type are Proof of Work algorithm (PoW), the practical Byzantine fault tolerance algorithm (PBFT), the proof-of-stake algorithm (PoS) and the delegated proof-of-stake algorithm (DPoS), Paxos algorithm and the Raft consensus algorithm.

Keywords: Maintaining replicas, Parallel Computers, State Machines, Virtual Machines.

I. INTRODUCTION

Distributed Computing has been one of our best approaches to computing solutions for a given problem by making use of shared resources systems when it comes to memory, processing power and platforms in general. This has not only significantly reduced costs and overheads but also ensured increased fault tolerance when it comes to highly scalable systems.

To make this concept work, we make use of consensus algorithms to replicate the state of all involved machines such that after a certain time or number of steps of processing, each machine agrees to end its final state based on a consensus. These consensus algorithms are the basis of parallel and distributed computing. Consensus algorithms are designed such that there is a leader node in the network and by means of an election/consensus either a final state is agreed upon by the majority of the nodes in the network, or no state is agreed on in which case the system can proceed to send an error message or never reach consensus and deem our given problem as unsolvable in nature.

In this paper, our goal will be to quickly examine the flaws of existing consensus algorithms and develop a new and novel consensus algorithm that we can utilize to cover up for all the negative aspects of said algorithms.

II. REPLICATED STATES

As mentioned, the purpose of replicated consensus algorithms is to ensure that there is complete agreement between the final states of each node such that based on the voting, the whole connected system as a whole can issue out a final agreed value. This is done by keeping logs of the state of each node. Consider a distributed network as follows:

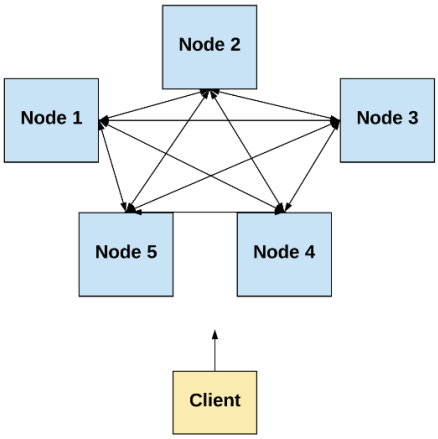

Fig. 1

Each node in the above network (Node1-Node5) has its own set of properties and functions based on their configurations. Based on our use case we are free to conduct any operation/computation that may require more than one node in usage. Our goal is that towards the end – the final output given back to the client is done by making use of a maximum number of resources in minimum time using minimum load on one node based on its tolerance (in time and resource). Each node has its own characteristic log file and based on operations carried out, these log files are modified and changed based on the sequence of computing and result generated. Simply stating- the job of replicated state machines is to manage consistency in these log files till the end and ensure that if the consistency is changing, then the log files have to be adjusted along with their corresponding operation.

Revised Manuscript Received on July 22, 2019.

* Correspondence Author

R. Kannadasan, Computer Science and Engineering, VIT University, Vellore, India. Email: kannadasan.r@vit.ac.in

N.Prabakaran, Computer Science and Engineering, VIT University, Vellore, India. Email: Prabakaran.n@vit.ac.in

A.Krishnamoorthy, Computer Science and Engineering, VIT University, Vellore, India. Email: krishnamoorthyarasu@vit.ac.in

K,Naresh, Computer Science and Engineering, VIT University, Vellore, India. Email: knaresh@vit.ac.in

Saravana Balaji.B, Collge of Engineering and Computer Science, Lebanese French University,Erbil-Kurdistan, Iraq. Email: saravanabalaji.b@lfu.edu.krd

A.S,Anakath, Professor and Director, Department of M.C.A, E.G.S. Pillay Engineering College(Autonomous), Nagapattinam , India. Email: anakatharasan@gmail.com

This is done by a series of rollbacks or additional operations depending on the requirements of our given node files.

Our goal is to briefly look into the advantages and disadvantages of each of these algorithms and try to create a novel algorithm of our own that aims to do the following:

- Retain the benefits of the above algorithms
- Get rid of the disadvantage
- Maintain proper replicas of our finite state machine
- Maintain fault tolerance
- Use resources available in a distributed computing environment as efficiently as possible
- Make the best judgment of setting trade-off between accuracy, speed and usage of resources.

III. EXISTING WORK AND THEIR CONS

Over the years a lot of consensus algorithms have been developed for specific use cases and problem statements in the areas of blockchain, networking, finance, security, cloud computing and enterprise software. These algorithms are sufficient when it comes to their defined cases but fail in the following circumstances

1. Scalability factors
2. Change of use case
3. Change of operations
4. Change of nature of networks
5. Change of nodes and their respective load balancing capabilities
6. Change of problem types that can usually be solved by these algorithms
7. Increasingly complex computing architectures

These factors prevent any of the common algorithms to reach a state of consensus for general use cases. Before we propose our solution to this generalization issue, let us look at existing algorithms for consensus and their issues

IV. PROOF OF WORK ALGORITHM

It is used in decentralized ledger networks where all information related to all nodes are collected and maintained. Each block/node in the network is recorded and watched over by a special individual node called a miner. This technology is used to solve complex mathematical problems that usually require high resource usage in terms of memory, architecture and time. This encompasses problem statements like integer factorization, hashing operations and tour algorithms. This algorithm has amazing use cases when getting outputs based on unknown inputs but we will now look at certain flaws in the algorithm.

Flaw of Proof of work algorithm:

- High node requirement in the network
- Lengthy network
- Increased power consumption can sometimes outweigh the resource optimization costs
- The process increases the overall sensitivity of the system

V. PRACTICAL BYZANTINE FAULT TOLERANCE

This algorithm works on the basis of replicated state machines and solves the Byzantine general's problem. The Byzantine Fault is one of the more popular classical problems with distributed computing which features failure of components, and/or where there is misinformation of the failure of a component in a network. Because of this fault – a high-risk server can seem to be working as well as failed to a failure-detection system, thus presenting different perspectives of a state to different observers.

The PBFT algorithm starts by assuming that not all nodes in a distributed network are going to be active and functioning all the time. Because of this – some scale of failure is prepared for in advance. The nodes in this algorithm are arranged in a specific order that ensures that one of the nodes acts as a leader and the others are kept as backup nodes. This is done by a round of voting and the value with the highest number of votes gets elected as the state of the machine. This algorithm also features increased communication in the network and prevents any overlooking in the network as long as there is a majority of functioning nodes in the network.

Main drawbacks of this algorithm include:

- Communication Gap
- Sybil Attack

VI. PAXOS

Paxos algorithm is one of the most popular algorithms that is used to achieve consensus in a distributed network of systems over an asynchronous network. One or more nodes in a network propose a value and all the nodes in the same network have the liberty of proposing or agreeing with the proposed value. Paxos then assigns the node the leader that gets the maximum votes for the same. This is one of the most famous algorithms as it has been rigorously proven to be correct. In order to create replicated state machines in the form of log files for each node, we need to run Paxos algorithm repeatedly for all the available nodal zones so that each center can have a chance to initiate a proposal to submit as a value. Based on these elections and voting are based and carried out.

Paxos has 3 main entities:

- Proposers
- Acceptors
- Learners

Error Cases exhibited in Paxos (basic):

- When an acceptor fails
- When a redundant learner fails
- When a Proposer fails
- When multiple proposers conflict

There are some major issues with the Paxos algorithm. The first being that it is an incredibly complex algorithm to understand.

Even the recent definitions of the Paxos algorithm's functioning is based on single/basic Paxos functioning. When we start involving multiple Paxos on nodes in a network, we start observing an additional complexity in the implementation. Finally, this makes the decomposition of the problem statement complex and we can observe that there are in turn much simpler and obvious ways of slicing our operations to make use of resources in a shared ecosystem just as efficient.

The second being – Paxos is incredibly difficult to implement in a practical surrounding. Even the single-decree Paxos implementation that is usually theorized have been inconsistent with their data and environments and their results have not yet been published. The issue with such a system is that this algorithm is not suitable for types of statements where our log files are added and appended to each other if they're independent. It would be less costly to add them in a sequential manner directly without any voting/consensus instead of wasting computing resources to decide the order of logfile- operation execution.

Finally, the Paxos architecture utilizes the symmetric peer-to-peer approach, according to which making unary decisions becomes extremely convenient and realistic, but in a practical situation-set; for more complex or serial decisions it is more efficient to elect a leader and then initiate voting. These are some of the reasons why even the classic Paxos algorithm fails in certain use cases and succeeds in others.

VII. RAFT CONSENSUS

The biggest issue with the Paxos algorithm was its increasingly complex architecture and algorithmic structuring. To break down the algorithm into simpler phases and increase industrial scalability needs we found the need to shift to a new algorithm that deals with the drawbacks of Paxos and also has some additional features of its own. Raft algorithm enables us to distribute a state machine (log file-replicas) over a network of nodes and provide a way to obtain general consensus such that concurrent set of transactions and logic routes are followed that ultimately reach an agreeable state. It is important to note that Raft consensus algorithm isn't Byzantine Fault-tolerant and hence elects a leader node and carries out its computation in a distributed system of networks.

Raft consensus differs from Paxos in the sense such that it follows a 3-subcomponent method of implementation

1. Leader election
2. Log replication
3. Safety

This algorithm firstly elects a leader in the node of the network. Depending on the votes obtained a leader is elected and is given full power over accepting and implementing the log replication of the network based on the client's requirements. This ensures that the leader need not waste time or energy in consulting other nodes to make modifications in the log and execution flow as long as it is connected to the network. The moment a network failure occurs that causes the leader node to get disconnected from the network a new election is held to contest a new leader for the same as a part of this algorithm's implementation.

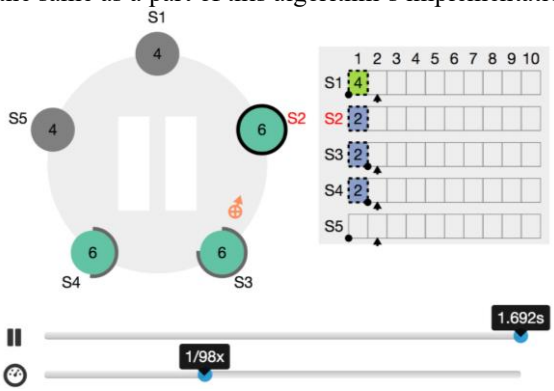

Fig. 2

While Raft Protocols solve some issues faced by Paxos, it is incredibly important to know its limitations as well.

Raft protocols follow the single leader approach. While this can help a lot of redundancy test cases pass with a reduction in processing time, the protocol can fail in a high-pace request system scenario in a network.

Raft protocols do not prove to be generally good algorithms for all consensus problems but only for a very specific and specialized of problem statements.

Raft protocols sideline real-life scenarios which face the highest occurrences of Byzantine failures. This reduces the application of the algorithm further because of anomalies.

VIII. PROPOSED WORKING CONCEPT

We have explored and studied existing algorithms for finding consensus in a distributed network. Each algorithm has its own set of use cases and works well in highly specialized environments and problem statements. However, as we have investigated the flaws of all the above-mentioned algorithms, it is also evident that none of these algorithms is useful in a general scenario of Parallel and Distributed Computing - the biggest reason being the existence of Byzantine Failure.

This section will deal with proposing a new concept to eliminate rate of failure of distribution of resources in a shared system-environment by considering data points from numerous simulations of the above algorithms and finding failure and success test cases, and ensemble the results such that a neural network is able to identify which set of combinations of algorithms can achieve a better set of generalization.

We can add on to this approach by using Generative Adversarial Networks (GANs) by keeping the generator as a set of Long Short-Term Memory networks (LSTMs) and the discriminator as a set of another LSTMs that are pre-trained from existing data points. The generator keeps generating series of combinations (randomly in the beginning) and keeps validating with the discriminator and based on the correct result, the complete GAN is retrained dynamically.

Let us look at the diagrammatic approach for the same.

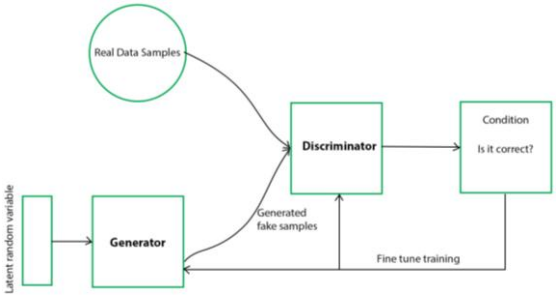

Fig. 3

The above is a simple GAN implementation that consists of our Discriminator and Generator. Let us look at the architectures of our Discriminator and Generator seperately

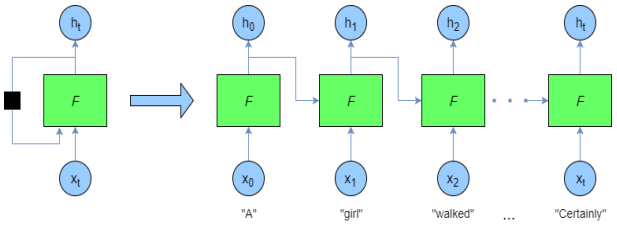

Fig. 4. Generator

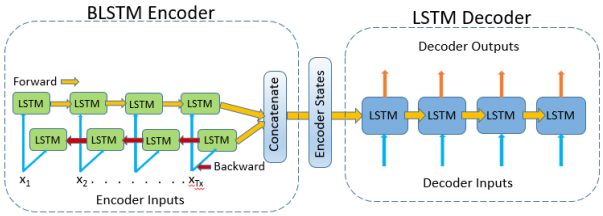

Fig. 5. Discriminator

Before we dive deep into the algorithm process flow, let us familiarize ourselves with some terms closely associated with our neural network.

Neural Network: A neural network is defined as a weighted directional graph which is used to calculate the cost of traversal of the path between any number of nodes defined within the path. We usually use different architectures of Neural Networks to solve complex problems that cannot be solved by modern-day computing. This is done leveraging numerous amounts of data of similar kinds such that our network (weighted graph) is fed with the same repeatedly and its' output is measured and compared with respect to the correct output and the weights of the connections are constantly updated to best match the final computation as the result desired.

Long Short-Term Memory networks (LSTMs): LSTMs is a kind of Artificial Neural Network that is used to process sequence of data such that they can very accurately work on predicting and computing results on time-series data. This makes them very powerful and unlike normal feed-forward neural architectures, LSTMs have backwards connections as well. This ensures that memory of the past learning iteration is retained (short-term memory) and this helps us detect patterns in a certain set sequence very easily.

General Adversarial Networks (GANs): GAN is an architecture which consists of two sets of neural networks that constantly compete with each other on deriving the correct output. This makes use of a popular concept from principles of Game theory - which is called a zero-sum game. This is done by keeping one set of a network as a discriminator and the other as a generator. The generator is responsible for using data distribution to find certain patterns (which is random initially and represents plain-high dimensional signal noise) and the discriminator validates the obtained data distribution's pattern with the true data distribution metrics.

Convolutional neural networks (CNNs): This kind of network deals with finding patterns in a distributed data set. This is done by adding a layer of convolution (filter) after every 'n' number of artificial neural layers of an ANN. This ensures that the process of gradient descent is applied to the convolutional layer as well.

Now that we have our literature review of existing concepts that we require for our algorithm, let us look at the main steps of the entire process flow.

Step 1: Prepare a dataset for our General Adversarial Network to train upon.

This is done by simulating the algorithms that we have discussed above on 'm' a number of distributed computing use cases, like - PoW, PoS, PBFT, Paxos and Raft. We can consider the following points for our use case distribution:

- Time elapsed (Total)
- State of Success/Failure (S/F)
- Resource optimization vs. Resource availability (Resource used/Total resource available)
- Even in resource optimization – Maintain a separate count of different categories of resources used.

Once our basic metrics are isolated, we proceed to pass the above algorithms - PoW, PoS, Paxos and Raft consensus algorithm one by one to make use of our distributed network of resources (processors, memory units, displays, etc). We then measure the above metrics and record the same for 'n' iterations of each algorithm. This can be stored as a separate CSV file on which our model will train on the same.

Step 2: Create your own General Adversarial Network (GAN) to train on the same dataset.
This step involved setting up our machine learning model by initializing a GAN and setting up its discriminatory and generator by declaring their innate properties. Our Generator will be an untrained LSTM network and our discriminator will be an LSTM network trained on our dataset that is created in step 1. This process trains our generator to eventually find patterns in specific distributions of data and correctly identify faulty outcomes and change the path of traversal in our distributed systems architecture.

Step 3: Dynamically validate your GAN on newer inflows of data by setting up required pipelines.
We can make the prediction metrics of our GAN much better by setting up pipelines to dynamically collect data every time any distributed system utilizes a classical algorithm for consensus purposes and retrain our network on the particular set of data points acquired in the process. We can additionally, set up a greater number of GANs to train on our data points but with different hyper-parameter configurations and then ensemble the outputs to reduce our validation loss even further.

IX. ADVANTAGES OF THE PROPOSED CONCEPT

- Removes uncertainty of Byzantine Faults in distributed systems because of the ability to map mathematically complex functions to path traversals in the system such that all possible scenarios of byzantine fault are eliminated. This also can be modified such that the algorithm suggests additions/deletions in the network of resources to overcome the halt.
- Ability to find the best fitting path of traversal in the system for all problem statements- Can achieve generalization in consensus building through the concept of training on as much variety of data as possible. Leverages the availability of memory to a great extent.
- Promises optimal resource usage of our distributed system.
- Solves the issue of single leader election in situations of no majority - leverages on all possible nodes in the network during GAN-training and is able to predict the best choice for leader election for the same, considering system failure situations.

X. DISADVANTAGES OF PROPOSED CONCEPT

- This method has high memory requirements for our GAN storage.
- Our method will require a lot of time to train our proposed architecture of GAN and depending upon the number and variety of our data-points this time will increase. However, training is a very periodic process that is completely dependent on our usage and requirements of the consensus algorithm.

XI. CONCLUSION

In this paper, we have witnessed and discussed different methods of evaluating consensus from a distributed system by keeping in mind that the state of all nodes should replicate to same values at the end of one processing iteration. We have also identified the many flaws that exist in the same methods - the biggest being the issue of none of the algorithms having a generalized use case but only very specific and user specialized use cases.
Hence our main motive has been to solve this problem by proposing a novel solution using Machine Learning and General Adversarial networks. We have introspected on the process flow, advantages and disadvantages of our approach. In our results, we have concluded that from a theoretical standpoint our method has two main cons but they're short term and the advantages greatly outweigh the former due to the fact that with proper data collection and resource allocation we can actually solve the problem of generalization very real when it comes to determining consensus in a distributed network.

REFERENCES

1. Bruner, I. S. and Tagiuri, R. The perception of people. In Handbook of Social Psychology, Vol. 2, G. Lindzey, Ed., Addison-Wesley, Reading, MA, 634–654.1954
2. Bledsoe, W. W. The model method in facial recognition. Tech. rep. PRI:15, Panoramic research Inc., Palo Alto, CA.1964
3. Ekman, P. Ed., Charles Darwin's The Expression of the Emotions in Man and Animals, Third Edition, with Introduction, Afterwords and Commentaries by Paul Ekman. Harper- Collins/Oxford University Press, New York, NY/London, U.K.1998
4. Kelly, M. D. Visual identification of people by computer. Tech. rep. AI-130, Stanford AI Project, Stanford, CA. 1970
5. Kanade, T. Computer recognition of human faces. Birkhauser, Basel, Switzerland, and Stuttgart, Germany 1973
6. Y. Cheng, C.L. Wang, Z.Y. Li, Y.K. Hou and C.X. Zhao,‖ Multiscale principal contour direction for varying lighting face recognition‖,Proceedings of IEEE 2010
7. F. Al-Osaimi·M. Bennamoun · A. Mian,‖ An Expression Deformation Approach to Non-rigid 3D Face Recognition‖, Springer Science+Business Media, LLC 2008

AUTHORS PROFILE

R. Kannadasan is an Assistant Professor (Senior) at School of Computer science and Engg., (SCOPE), VIT University, Vellore, India. His research activities are carried out in bio informatics, language translators and DNA computing.

N.Prabakaran, is an Assistant Professor(Sr), School of Computer science and Engg., (SCOPE), VIT University, Vellore, India. His research activities are carried out in Networks and Distributed computing.

A.Krishnamoorthy is an Assistant Professor (Senior) at School of Computer science and Engg., (SCOPE), VIT University, Vellore, India. His research activities are carried out in VLSI, Embedded system and theory.

K.Naresh, is an Assistant Professor(Sr), School of Computer science and Engg., (SCOPE), VIT University, Vellore, India. His research activities are carried out in Networks and Distributed computing.

Saravana Balaji.B is an Assistant Professor , Collge of Engineering and Computer Science, Lebanese French University,Erbil-Kurdistan, Iraq.. His research activities are carried out in Embedded system ,Cloud computing and Simulators.

A.S,Anakath, Professor and Director, Department of M.C.A, E.G.S. Pillay Engineering College(Autonomous), Nagapattinam , India.. His research activities are carried out in Operating System, Cloud Computing and Network and Theoretical computing..