International Journal For Advanced Research In Science & Technology
A peer reviewed international journal www.ijarst.in
ISSN: 2457-0362

IMPROVING QUALITY OF SERVICE IN WIRELESS NETWORKS USING CONSENSUS ALGORITHM

Dr R Baby Munirathinam , M.C.A , M. PHIL, PH.D.,
babyrathinam@gmail.com
Associate Professor / Department of Computer Science
MALLA REDDY ENGINEERING COLLLEGE FOR WOMEN
Secunderabad, Telangana, INDIA -500 100

Abstract:

The consensus algorithm has become more common in contemporary distributed systems as a result of its improved efficiency in resolving server unreliability. It ensures that several servers may work together to build a system and that the system continues to function even if one service point malfunctions. The widely used distributed consensus algorithm Raft constantly sacrifices performance in order to achieve its fundamental goal of comprehensibility. In this study, we primarily concentrate on the performance issue with the traditional Raft consensus algorithm, particularly under conditions of high concurrency. To improve efficiency through disc flushing and batch asynchronous log replication, we add a pre-proposal stage to the approach. The trial showed the improved Raft's method, and this study focuses mostly on the performance issues with the traditional Raft consensus process, particularly under conditions of high concurrency. To improve efficiency through disc flushing and batch asynchronous log replication, we add a pre-proposal stage to the approach. The experiment showed that the improved Raft may increase the system's throughput by 2-3.6 times and its efficiency for handling parallel requests by at least 20%.

Keywords:

Raft, RAmCloud, Consensus Algorithm, Distributed system, Asynchronous .

1.Introduction

A group of machines can function as a cohesive unit that can endure the failure of some of its members thanks to consensus methods. They are essential in creating dependable large scale software systems as a result.The consensus algorithm maintains a replicated log with client supplied a state machine command. The state machines employ identical command sequence from the log to process, producing similar outputs which is used to solve. A separate replicated state machine is often used to handle large-scale systems with a single cluster leader such as GFS[8] , HDFS[18] and RAMCLOUD[23] typically manage leader election and store configuration data that has to survive all the leader crashes in a separate replicated state machine and several issues with a Fault tolerance in distributed systems. Replicated state machines have examples like Chubby[2] ZooKeeper[11]. Raft shares many similarities with other Consensus Algorithms such OKI and Liskov's Replications[29,22] but it also a few new features:

- **Solid Pioneer** : Pontoon utilizes a more ground type of initiatives than other agreement calculations. Log entries , for instance , only flow to other servers from the leader. Raft is made easier to comprehend and manage the replicated log and made simpler as a result.
- **Election of leaders**:

Leaders are chosen in Raft through the use of the random timers. This adds just a modest quantity of system to the pulses previously expected for any agreement calculation, while settling the clashed quickly and changes in the membership.

A Novel joint consensus approach is used in Raft's mechanism for changing the number of servers in the cluster when the majorities of the two different configurations overlap during when the configuration changes, The cluster continue to function normally because of this replicated state machines as seen in Figure 1. Each server keeps a record with a list of commands that is state machine sequentially executes. Each state machine processes the same set of commands since each log contains the same commands in the same order and output sequence because they are deterministic. This algorithm's sole responsibility is to maintain the consistency in the replicated log. The main characteristics are found in this algorithm for practical systems.

- .The failure of any two server can therefore to be tolerated in a typical cluster of five servers, It is assumed that servers would fail, however they might later recover from a filed state using stable storage and re-join the clusters. And they are independent of timing to maintain consistency.

2. Description of the protocol

The Raft protocol is designed to be without difficulty understandable thinking about that the maximum popular manner to achieve consensus on the allotted structures was the Paxos, set of rules, which became very tough to apprehend and enforce. Each person with the basic knowledge and common place experience can apprehend the essential elements of the protocol and the research paper published by the way of Diego Ongaro and John Ouster out. It is a miles comparatively easy to put into effect than other options, on the whole the Paxos, due to a more cantered usage of case phase and assumptions about the disbursed system. Many open supply implementations of the Raft to be had at the C++ and Java. The Raft protocol has been decomposed into smaller sub issues which can be tackled fantastically independently for higher information, implementation and debugging, optimizing performance for a greater particularly of the use case.

- ➢ The disbursed machine following the Raft consensus protocol will remain operational even if minority of the server fail. For instance, if we have got a five server node cluster , if 2 nodes fails, this device can still function.
- ➢ The leader election mechanism in the Raft is so designed that one node will constantly gain most of the people votes within the most of the two terms.
- ➢ The Raft employs RPC (Remote Procedure Call) to the votes and sync up the clusters., so the load of the calls does not fall at the leader node in the cluster now. Raft changed into design later, so it employes modern day concepts which have been no longer but understood at the time of the formulation of the paxos and comparable protocols.

Any node in the cluster can comes to be chief, so it has fairness . Raft nodes are always be the candidates, followers ore leaders. Every node begins its life as admirer. Nodes can agree to take log entries from the leader and cast votes at this stage. Node stage-promoters to the candidate state over a period of time without receiving any entries in the candidate state and ask their follow nodes for the votes. A candidate is elevated to be the leader if quorum of votes is case infavor of them . A new log entry must be accepted by the leader , who must then repeat it to every other follower. All queries must also be executed on the leader if stale readings are unacceptable. Describe the protocol that fact that a replicated log is unbounded raises an obvious question. Raft offers a method for taking a snapshot of the current state and compressing the log. The FSM abstraction requires that log for restoring the state of the FSM, all queries must also be performed by the leader. A leader then attempts to duplicate the entry to quorum followers by writing it to reliable storage. Once the log entry is regarded as committed means, then a finite state machine can use it. The interface is used to implement the finite state machine, which application must result in the same state as that of the old logs. A cluster can accept fresh log entries once it has a leader and FSM client, Raft will then be able to delete all the logs that were used to get the FSM state which was captured at the particular time. This is done automatically, without user input and limits the amount of disk space that can be used while also cutting down the amount of time that logs muse be replayed.

3. Consensus:

In order to comprehend Raft, it is to be consider the challenge of reaching consensus, when the Raft aims to address multiple servers are coming to consensus on the same information is necessary to create fault tolerant distributed systems. Let's explain it by few illustrations. It is better to understand the process, let us first outline the procedure followed when a client communicated with a server process. The server received a message from the client and responds with a reply.

The following characteristics are essential for a consensus procedure tolerating the failures:

- Validity : A value must have been provided by a another valid process if a process decides (either to read or writes) it.
- Consensus: Every valid procedure must occur on the same value.

Termination: Every correct procedure must come to an end after a number of steps.

Integrity: Any process has the specified value if all the correct processed reach the same. Now let us assume that just one client (for the sake of clarity) there are two different types of systems that could exist. Client communication takes place in a system with a single server and no backup in a system with a single server and no backup. In such a system, reaching consensus is not an issue.

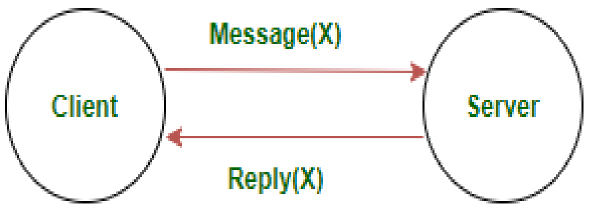

Figure 1: Visual Single Server Raft.

Multi Server System: The client communicates with a system of several servers. Then such system comes in two varieties:

When a client requests a response from one of the many servers, any of the many servers are expected to synchronizes with that server. A cluster can accept fresh log entries once it has a leader.

Asymmetric: Once the leader server can react to the client. Any of the numerous server may respond to the client and all the other servers are expected to sync, up with the one that did

Symmetric: Any of the numerous servers may respond to the client, and all servers are expected to synchronize with the one that did. The leader server synchronizes with the other servers. For the time being. system like this can be referred to replicated state machine all of the servers preserve similar data(shared data) throughout the time. The words used to a specific servers in the distributed system will now to be defined. Only the server chosen as the leader that can communicate with the client. Each of other server synchronizes with the leader , there can be only one leader at a time of process. (perhaps 0, as it will be ex plain later.)

Follower:
After a regular time intervals, the follower server sync their copies of the data with the leaders. One of the followers can run leadership in the event that the leader server goes offline (for any reason).

Candidate: When competing in an election to select the leader server, the servers may solicit votes from other servers. As a result , when they ask for votes, they are referred to as candidates.

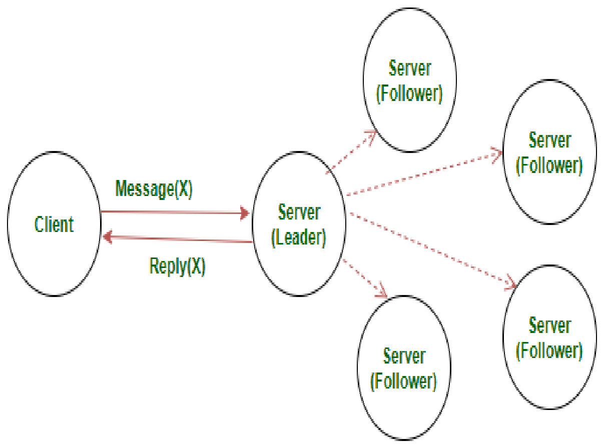

Figure 2: Multiple Server Labeled Raft Visual

3. Proposed Model:

Every client operation that the servers' state machine is This model refers to a proposal as one that is process-capable. A thorough and efficient proposal process normally consists of an event request Invocation (hence referred to as Inv) and an event response (hence referred to as Res). In the end, the state machine submits the non-read type Write for each operation that contains a Write or Read operation. Each task proposal will be converted into a log as shown in Figure 4 as long as it is there and can be completed by the state machine. Proposals can be handled once a response is returned to the consistency module of the leader node, and for the Inv and Res events, read operations should return a new value after each read operation. The Raft system can then handle simultaneous client requests while maintaining linear consistency. This entails ensuring a complete order relationship and concurrent Read/Write requests with the real-time order.
The client A's procedure is depicted in Figure 3(a), from commencement to response. According to Raft, a system proposal that satisfies linear consistency must accomplish the following process.

A parallel client request with linear consistency in Raft is shown in Figure 3(b). The client A To E starts a parallel Read/Write request for the same piece of data V at a specific time, and Raft receives the proposal in Real-Time Order. The request complies with the following total order relationship, as depicted in the Figure. After first adding the log to the log collection and sending Append Entries to the remaining followers nodes, the leader distributes the log items using RPC technique. After receiving the requests, the follower node will also copy the log items to its log collection, regardless of events like network partition and outages, and respond to the leader node with an ACK to indicate a successful append. When the leader receives more than one Ack (acknowledgement) message from the followers, the state machine will submit the log and the ACK will be relayed to other followers nodes to submit, concluding a cluster log submission. The following proposals can only be handled after a proposal returns a Response(res), despite the fact that proposals may be submitted concurrently. In the highly concurrent scenario, the log items to be processed can be understood as an infinitely growing task queue. The leaders wait for the response from half of the nodes before continuing to send AppendEntries RPC messages to the followers. The Leader essentially establishes a TCP relationship with the follower and launches many TCP packets based on the TCP protocol's sliding window technique when several successive AppendEntries RPCs are launched. Instead of pausing to confirm each group, the sender can send multiple packets in a succession using the sliding window approach before a stop-and-wait confirmation. The maximum amount of data packets that may be sent is determined by the window size, and wait times are longer as the window fills up. A long fat network (LFN) will grow when several TCP data packets are delayed in arriving, which causes the data packets to time out and retransmit. Retransmissions with no purpose add significantly to network overhead. The response can be appropriately received by delivering many data packets constantly and not being retransmitted, provided the window is large enough. If additional network overheads are not taken into consideration, the throughput of the network is equal to the amount of data transmitted each second. Based on this concept, the suggested system converts the synchronous wait of a continuous append entries to an asynchronous wait, preventing the stopping of additional ACKs and enhancing network speed. The solution to this problem is that a network is considered smooth when replies to the leader's constant heartbeat confirmation are prompt since the logs are within a tolerable range prior to out-of-order sequences occurring. The network throughput is equal to the quantity of data transmitted per second if additional network overheads are not taken into account.

This asynchronous processing of logs is done through the batch. We introduce a pre-proposal phase between the client-initiated proposal and the leader-analyzing the proposal in order to pre-process concurrent proposals. The proposal is loaded utilizing a highly parallel synchronization queue in FIFO (First in First out) order during this period. Once it starts processing a proposal, the Leader will sequentially remove each proposal from the synchronization queue until it reaches the first read-only request in the queue. Following the Leader's initial processing of the proposal, each proposal in the queue will be removed one at a time. The modified algorithm's premises and goal are a replica state machine.

International Journal For Advanced Research In Science & Technology
A peer reviewed international journal
www.ijarst.in
ISSN: 2457-0362
IJARST

According to the original Raft algorithm's principles, the security mechanism should essentially provide the following guarantees:

The term number (Term) in the cluster continues to increase monotonically. Synchronization must be kept going until the first read-only request is received.
There won't be any byzantine errors because the network communication between clusters is reliable to pack loss, delay, network jitter, etc.
There will only ever be one leader chosen for the cluster, and that leader will always have the same term number. Requests from clients that are received by other nodes are forwarded to the leader.

Additionally, if a failure unsafe situation occurs, general reliable communication like TCP have retransmission mechanism, with be lost packets will retransmit instantly, so it is possible to recover in a short time. Although node unavailability and network locations will arise, it may be presumed that communication made between the leader and the other followers is secure and that these issues are undercontrol.

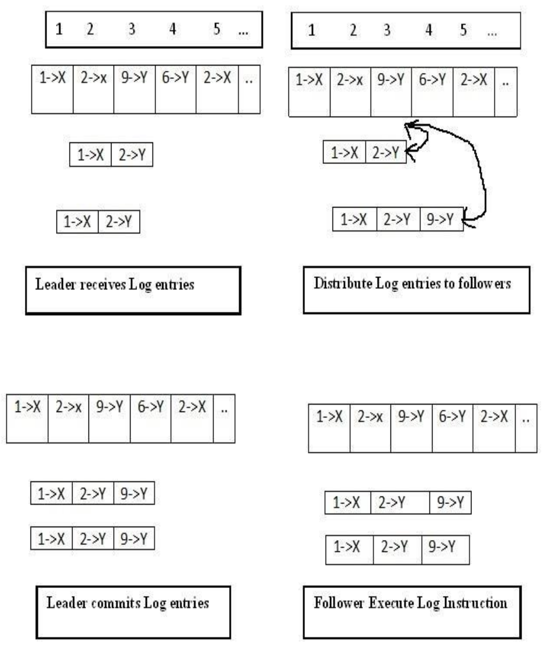
1 2 3 4 5 ...
1->X 2->x 9->Y 6->Y 2->X ..
1->X 2->Y
1->X 2->Y
Leader receives Log entries
1 2 3 4 5 ...
1->X 2->x 9->Y 6->Y 2->X ..
1->X 2->Y
1->X 2->Y 9->Y
Distribute Log entries to followers
1->X 2->x 9->Y 6->Y 2->X ..
1->X 2->Y 9->Y
1->X 2->Y 9->Y
Leader commits Log entries
1->X 2->x 9->Y 6->Y 2->X ..
1->X 2->Y 9->Y
1->X 2->Y 9->Y
Follower Execute Log Instruction

Figure (a) The process of Proposal Figure (b) The parallel Process of Proposal

A proposal is any client operation that the server's state machine is capable of handling. A complete proposal typically consists of an EventRequest (also known as an Invocation afterwards) and an EventResponse (also known as Res). The state machine finally submits a request with the types Writer or Read and the non-Ready only type write.
The process of a proposal from client A is depicted in Figure 3(a) from the viewpoint of Raft, a system that satisfies the requirements for linear consistency to accomplish the following points.

4. Experiments and analysis

The experimental surroundings is as follows:
The server host had 32 GB of memory , the CPU is Intel Xeon (cascade Lake) Platinum 8269 Y2 Five GHz with 8 cores. The proposed set of rules is administered in the digital box of he server, three nodes are simulated, with each node specifies four GIB memory and a pair of CPU cores, the working device is Centos, and this system code is programmed in Java. So that we can evaluate the efficiency of the progressed with Raft , set of rules , assessment experiment procedure.
Multithreading changed into and used to ship the concurrent requests. In general 17 sets of experiment have been executed for the evaluation, with distinct request concurrency tiers- from one thousand log entries to up to 13000 log entries. In total 13000 log entries . The very last effects are proved in Figure 4 Figure 5 and Table 1 .

The programme will unavoidably attain the processing bottlenecks because the concurrency degree rises, or the point at which the processing velocity is far slower than the task increments.

The bottleneck is placed around the log concurrency of 12000, as seen in picture 5. The processing electricity for both the algorithm will degrade dramatically if the wide variety of requests exceeds this.Earlier than the bottleneck, it is miles obvious that the recommended method can asure 20% or more development over the present algorithm, because the addition of the batch method allows red for the concurrent tasks queue , the consoled algorithm over the process time can adjust to stable even after the bottleneck .

Contrarily, when obligations and the log backlog grow, the vintage algorithm overall performance policies for safety within the protocol performance policies for safety only within the protocol for Rafts .

Through distinctive feature of its is design the Raft protocol ensures the following protection towards consensus malfunctions. The very last outcomes are shown in Figure 4 , figure5 and table 1 . Table 1 fact the improvement fee of the stepped forward algorithm in the device the throughput and log processing time. It is able to be be seen that the proposed algorithm can at least double the device throughput and the processing time of the customer request also tobe improved by way of more than 20%.

Number of log entries(size/MS)		Improvement Rate	
Data Volume (size in g roups)	Throu ghput in MS	**Proposal Process**	**Throughp ut**
1	1000	0. 537	1.62
2	2000	0. 472	1.02
3	4000	0. 442	1.238
4	5000	0 .483	0.592
5	10000	0.353	1.580
6	12000	0. 220	1.354
7.	13000	0.269	1.353

Table 1 -Performance Improvement Rate

Figure 4:

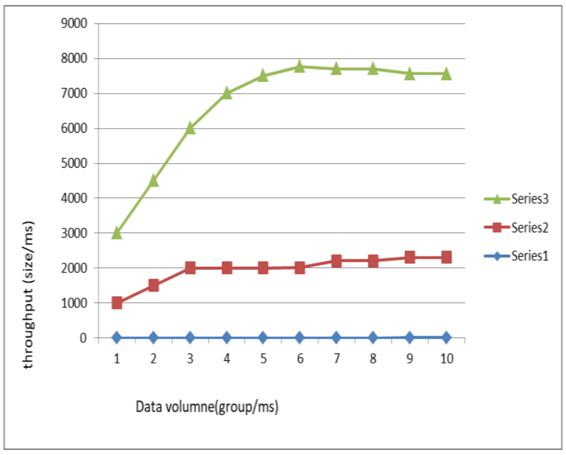

Figure 5:

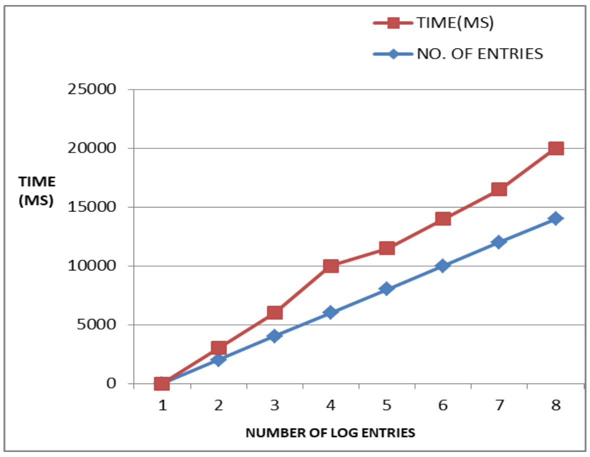

Figure 4 and Figure 5 : Performance of throughput of throughput of with different size of volume

5. Conclusion

Any client operation that the server's state machine can handle is referred to as a proposal. An EventRequest (also known as an Invocation later) and an EventResponse (also known as Res) often make up a complete proposal. Finally, the state machine sends a request with the types Writer, Read, and non-Ready only type write.
From the perspective of Raft, a system that satisfies the requirements for linear consistency to achieve the following goals, the process of a proposal from client A is shown in Figure 3(a).

References

[1] Kleppmann, M.: A Critique of the CAP Theorem.
arXiv:1509.05393 (2015) Brewer, E.: Spanner, TrueTime and the CAP Theorem (2017).

[2] Huang, D., Liu, Q., Cui, Q., et al.: TiDB: a Raft-based HTAP database. Proc. VLDB Endowment 13(12), 3072–3084 (2020)

[3]Taft, R., Sharif, I., Matei, A., et al.: Cockroachdb: the resilient geodistributed SQL database. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, pp. 1493–1509 (2020)

[4] Huang, D., Ma, X., Zhang, S.: Performance analysis of the raft consensus algorithm for private blockchains. IEEE Trans. Syst. Man Cyber net. Syst. 50(1), 172–181 (2020)

[5] Mingxiao, D., Xiaofeng, M., Zhe, Z., Xiangwei, W., Qijun, C.: A review on consensus algorithm of block chain. In: IEEE International Conference on Systems, Man, and Cybernetics (SMC), (Banff, AB, Canada), pp. 2567–2572 (2017)

[6] Wang, G., et al.: Building a replicated logging system with Apache Kafka. Proc. VLDB Endow. 8(12), 1654–1655 (2015)

[7] Van Renesse, R., Altinbuken, D.: Paxos made moderately complex. ACM Computer. Survey. 47(3), 36 (2015)

[8] Ongaro, D., Ousterhout, J.: In search of an understandable consensus algorithm. In: 2014 USENIX Annual Technical Conference, pp. 305–319 (2014) .

[9] Frömmgen, A., Haas, S., Pfannemüller, M., et al.: Switching ZooKeeper's consensus protocol at runtime. In: 2017 IEEE International Conference on Autonomic Computing (ICAC), pp. 81–82 (2017) https://github.com/tikv/tikv

[10] Ailijiang, A., Charapko, A., Demirbas, M.: Consensus in the cloud: Paxos systems demystified. In: 25th International Conference on Computer Communication and Networks (ICCCN), pp. 1–10 (2016)

[11] Lamport, L.: Time, clocks, and the ordering of events in a distributed system. In: Concurrency: The Works of Leslie Lamport, New York, USA, pp. 179–196 (2019)