Quest Journals
Journal of Research in Applied Mathematics
Volume 9 ~ Issue 7 (2023) pp: 01-10
ISSN (Online): 2394-0743 ISSN (Print): 2394-0735
www.questjournals.org

Research Paper

Analysing the Performance of Raft Consensus Algorithm using M/M/C Queueing Model

Stuti Vora[1], Ravi Gor[2]
[1]*Research Scholar, Department of Applied Mathematical Science, Actuarial Science and Analytics, Gujarat University*
[2]*Department of Applied Mathematical Science, Actuarial Science and Analytics, Gujarat University*

Abstract: *Distributed systems are governed by consensus algorithms to ensure stability and availability of data across multiple nodes. Raft is a well-known consensus algorithm that provides fault tolerance in distributed systems. Examining the performance of Raft is crucial to understand its behaviour under diverse capabilities and system configurations. The aim of this paper is to analyse the performance of Raft using M/M/C queueing model. The arrival and service of client requests in the Raft consensus algorithm, considering the multi-channel nature of the follower nodes is modelled. The utilization, queue length, throughput, and latency metrics for the system are derived. It assesses the influence of various capabilities and system characteristics on performance. This analysis provides insights into the behaviour of Raft under different scenarios and can be used to guide system design and optimization efforts.*
Keywords: *Consensus, Raft Consensus, Queueing model, Throughput, Latency*

R*eceived 25 June, 2023;* ***R****evised 03 July, 2023;* ***A****ccepted 05 July, 2023 © The author(s) 2023.*
P*ublished with open access at* ***www.questjournals.org***

I. Introduction

Modern computing is increasingly using distributed systems because they provide advantages including fault tolerance, scalability, and high availability. Consensus algorithms are a key aspect of distributed systems as they permit several nodes to converge on a single value or choice, even in the failures or network splits. Raft is a widely used consensus algorithm that has gained popularity due to its simplicity and ease of understanding. To verify that Raft is appropriate for specific uses, it is essential to understand how it performs under different capabilities and system designs.

In this paper, the performance of Raft consensus algorithm using M/M/C queueing model is analysed. The arrival process and service process of client requests in Raft are modelled, considering the multi-channel nature of the follower nodes. Throughput and latency are important parameters for evaluating the performance of this distributed systems. They determine how many client requests can be processed within a given time frame and how fast they can be completed.

II. Literature Review

Ongaro and Ousterhout [5] (2014) proposed the Raft consensus algorithm as a more understandable and modular approach compared to other consensus algorithms. Their work discussed the details and design choices of the algorithm, highlighting its simplicity and ease of implementation.

Huang et. al. [1] (2018) analysed the distributed network split probability using a basic but accurate analytical approach. The network split probability was explicitly presented as a function of network size, packet loss rate, and election timeout length. To validate this work, a Raft simulator was used, and the simulation findings match the analytical conclusions. With the proposed model, one can predict the network split time and probability in theory and optimize the parameters in Raft consensus algorithm.

Memon et. al. [4] (2019) suggested a model to imitate a blockchain using queuing theory. The suggested model was created by using M/M/1 queue as a memory pool, and an M/M/c queue as a mining pool. This technique was a simple but effective way of revealing several essential metrics such as (a) The number of transactions per block (b) The mining time of each block (c) System Throughput/Transactions per second (d) Memory-pool count (e) Waiting time in memory-pool (f) Number of unconfirmed transactions in whole system (g) Total number of transactions and (h) Number generated of blocks.

Samy et. al. [6] (2021) aimed a modification to Istanbul Byzantine Fault Tolerance consensus algorithm to solve scalability issues of IBFT in private blockchain systems. They used Hyperledger Caliper, a blockchain benchmark tool, to evaluate the performance of open-source platforms like Quorum and Hyperledger Fabric. Fabric's utilization of Solo and Raft protocols as transactions ordering services in v1.4 produces a reliable throughput and low latency making Hyperledger Fabric a good candidate for enterprise blockchain-based solution. On the other hand, Quorum's implementation of Raft and IBFT as consensus protocols responsible for maintaining the chain of blocks makes it a more logical option to test modifications against.

Kim et. al. [3] (2021) proposed a mechanism for electing leaders via the Raft algorithm considering network stability. Developed method reduced the probability of network split using Raft. Also, prevented further performance degradation when a blockchain node network was unstable.

Tomić [9] (2021) reviewed functional advantages and disadvantages of observed protocol. Characteristics compared: security, trust among participants, throughput and scalability. It was concluded that no protocol showed complete dominance in all aspects of the comparison. Most protocols in practice show deviations from the theoretically stated optimal results all the observed protocols like PBFT, FBA, DBFT are intended for permissioned blockchains, not all have the same purpose. So, when choosing a consensus protocol for a blockchain applications, take into purpose priority.

Vora and Gor [10] (2022) studied M/G/1 queue with discrete time Markov chain (DTMC) in Blockchain system. Queueing theory was theoretically analysed for the block-generation and blockchain building processes. In this, the sum of the block-generation and blockchain-building times was considered as the transaction-confirmation time of a block. The Blockchain system was examined at arrival as well as departure point. Average transaction confirmation time, number of transactions in queue at arrival point and the number of transactions at departure points etc were observed with the help of Python.

Vora and Gor [11] (2022) provided Markovian queueing model of the form $M^b/M/1$ as the theoretical foundation for analysing a Proof of Authority (PoA)-based Blockchain system that focuses on the stochastic behaviour of the transactions, consensus process and throughput. Transactions were assumed to arrive in batches with a Poisson distribution, according to the model. The queueing model had been used to analyse the PoA consensus protocol. To demonstrate the reliability of the underlying simulation model, numerical experiments carried out in Python was presented. It had been noted that the block's optimal throughput rate is attained for small numbers of transactions.

III. Raft Consensus Algorithm [8],[1]

Raft is a consensus algorithm designed for distributed systems, which aims to ensure that all nodes in a cluster agree on the state of the system even in the face of failures or network partitions. The algorithm was first introduced in 2014 by Diego Ongaro and John Ousterhout. It is named after Reliable, Replicated, Redundant, and Fault-Tolerant [1].

Each node in the cluster of the Raft consensus process is either or a candidate. The leader is managing the cluster, handling client requests and making sure that all other nodes agree on the system's current status. Whereas the candidates are nodes that are presently seeking to become the leader, the followers are passive nodes that only react to requests.

Raft obtains consensus through a process known as terms. For each term, the cluster's nodes compete in an election to choose a new leader. If a candidate obtains votes from a majority cluster nodes, he or she is elected leader for the next term. The leader then transmits heartbeats to the followers to retain its leadership and ensure that all nodes agree on the system's status.

Once the leader is selected, the second part deals with the index. It determines how an index log or a transaction is committed are agreed by all the followers. The leader has the task to propose a new transaction. Now, if the leader wants to propose a new transaction, it acts as an entry log with the current term where everyone is. Then the current transaction index has new index i.e., current index +1. Therefore, this particular entry is done in the index log of the leader. Then the leader sends a message called append entries of new term and index (term and index) to all the followers.

These append entries contain the proposal coming from the leader. In a particular proposal, if the leader declares that now the process at next index, it means new round is started. It also means that the leader is proposing for the new transaction. So, once this particular transaction log is broadcast to all the followers, they collectively vote either for the transaction or against the transaction. If leader receives majority votes for the new transaction, it means that majority of the nodes are agreed with committing that particular log. The leader sends an accept message based on the majority voting to all the individual followers. Therefore, the followers update the committed new index.

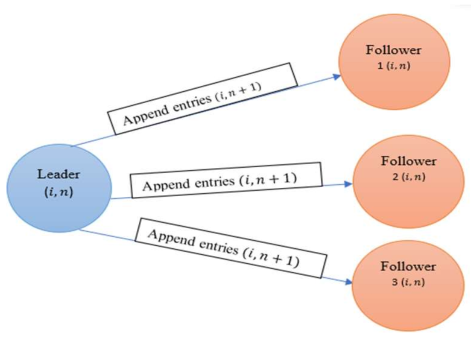

Leader
(i, n)
Append entries $(i, n+1)$
Append entries $(i, n+1)$
Append entries $(i, n+1)$
Follower
1 (i, n)
Follower
2 (i, n)
Follower
3 (i, n)

Figure 1

The simplicity of the Raft consensus method makes it simpler to comprehend and apply when compared to other consensus algorithms such as Paxos. Also, Raft delivers better safety assurances than existing consensus algorithms. It ensures that the system stays consistent even when numerous failures or network partitions occur. The well-known blockchain platforms, R3 Corda and Quorum, use Raft as a consensus protocol.

The performance measures of Raft Consensus:
The performance of the Raft consensus algorithm can be measured using different parameters, such as:

1. **Latency:** This is the time it holds for a client request to be managed and for the system to reach consensus on the response. Lower latency means faster response times for clients and a more responsive system overall.
2. **Throughput:** This is the number of client requests that can be processed by the system in a given period of time. Higher throughput means the system can handle more requests and support more users or applications.
3. **Scalability:** This is the ability of the system to handle an increasing number of nodes in the cluster without significantly impacting performance. A more scalable system can support larger clusters and more complex applications.
4. **Consistency:** This is the ability of the system to ensure that all nodes in the cluster agree on the state of the system and that this state remains consistent even in the face of failures or network partitions. A more consistent system provides stronger guarantees of correctness and reliability.

Overall, the performance of the Raft consensus algorithm will depend on a variety of factors, such as the size and complexity of the system, the network conditions, the hardware and software configuration, and the specific use case or application requirements.

IV. Description of the Mathematical Model

The Raft consensus process is a distributed consensus algorithm used to maintain consistency across a cluster of computers. Thus, Raft decomposes the consensus problem relatively independent subproblems, which can be described as follow:
• Leader election: A new leader must be chosen when an existing leader fails.
• Log replication: The leader must accept log entries from clients and replicate them across the cluster, forcing the other logs to agree with its own.
In the context of a queuing model, *Log replication process* of Raft consensus process is considered as a multi-server queueing system. The M/M/C queueing model is taken for this consensus systems, where arrivals are modelled as a Poisson process and service times are exponentially distributed.
In the Raft consensus process, a server refers to a follower node in the distributed system that participates in the consensus protocol. Each server maintains a copy of the system's log and communicates with other servers to ensure that they agree on the current state of the log.
Client requests arrive at each follower node according to a Poisson process with rate λ, and it requires a service time of $1/\mu$.
Arrival process: *Once a leader has been elected, it begins servicing client requests. Each client request contains a command to be executed by the replicated state machines. The leader appends the command to its log as a new entry.*
The arrival process can be modelled as a Poisson process, with the rate determined by the frequency of leader messages sent to the followers. λ be the arrival rate of client request from the leader node to the follower nodes.

Service process: *The leader appends the command to its log as a new entry, then issues Append Entries RPCs in parallel to each of the other followers to replicate the entry. When the entry has been safely replicated (as described below), the leader applies the entry to its state machine and returns the result of that execution to the client.*
The service time can also be modelled as a Poisson process, exponential distribution; with the rate determined by the time it takes for the followers to replicate the leader's log entries. μ be the service rate of each follower(channel), representing the rate at which log entries are replicated from the follower to the leader.

Terminology
Table 1 shows the notations of terminology used in this work.

Table 1: Terminology

λ	Arrival rate of a transaction
μ	Service rate of transaction
ρ	Utility factor of blockchain system
$N(t)$	Total Number of transactions in the system at time t
$N_q(t)$	Number of transactions in queue at time t
T_q	Time a transaction spends to wait in a queue
T	Total time a transaction spends in the system
$E[N]$	Expected number of transactions
$E[N_q]$	Expected number of transactions in queue
$E[T_q]$	Expected waiting time in queue
$E[T]$	Expected waiting time in the system

V. Analysis

Here, the Raft consensus algorithm is analysed using the $M|M|C$ queueing model. Assuming client requests arrive at an average rate of λ following the Poisson fashion.

$$\lambda_n = \lambda$$

There are c parallel identical followers, each follower according to an exponential distribution with an average rate μ. For the n client requests, the service rate of the server can be obtained by following situations:

1. $n < c$, all the client requests may be served simultaneously. There will be no queue.$(c-n)$ number of servers remain idle. In this case $\mu_n = n\mu, n = 0,1,2,\dots,c$.
2. $n \geq c$, all the servers are busy and the maximum number of client requests waiting in queue will be $(n - c)$. Then, $\mu_n = c\mu$, for $n = 0,1,2,\dots$

$$\mu_n = \begin{cases} n\mu & n < c \\ c\mu & c \leq n \end{cases} \qquad \dots(1)$$

The steady stat difference equations can be written as

$$P_0(t+\Delta t) = P_0(t)(1-\lambda\Delta t) + P_1(t)(\mu\Delta t)$$

$$P_n(t+\Delta t) = P_n(t)[1-(\lambda+n\mu)\Delta t] + P_{n-1}(t)(\lambda\Delta t) + P_{n+1}(t)(n+1)(\mu\Delta t), \text{ for } n < c$$

$$P_n(t+\Delta t) = P_n(t)[1-(\lambda+c\mu)\Delta t] + P_{n-1}(t)(\lambda\Delta t) + P_{n+1}(t)(n+1)(\mu\Delta t), \text{ for } c \leq n$$

Dividing these equations by Δt and taking limit as $\Delta t \to 0$, the difference equations are

$$P'_0(t) = -\lambda P_0(t) + \mu P_1(t)$$

$$P'_n(t) = P_n(t)[-(\lambda+n\mu)] + P_{n-1}(t)(\lambda) + P_{n+1}(t)(n+1)(\mu), \text{ for } n < c$$

$$P'_n(t+\Delta t) = P_n(t)[-(\lambda+c\mu)] + P_{n-1}(t)(\lambda) + P_{n+1}(t)(n+1)(\mu), \text{ for } c \leq n$$

In steady state condition, when $t \to \infty$, $P_n(t) \to P_n$, $P'_n(t) \to 0$, for all n. So, the following equations become,

$$0 = -\lambda P_0(t) + \mu P_1(t), \text{ for } n = 0 \qquad \dots(2)$$

$$0 = -(\lambda+n\mu)P_n(t) + (\lambda)P_{n-1}(t) + (n+1)(\mu)P_{n+1}(t), \text{ for } n < c \qquad \dots(3)$$

$$0 = -(\lambda+c\mu)P_n(t) + (\lambda)P_{n-1}(t) + (c)(\mu)P_{n+1}(t), \text{ for } c \leq n \qquad \dots(4)$$

From equation (2),

$$P_1 = \frac{\lambda}{\mu}P_0$$

Similarly, for$n = 1$,

$$P_2 = \frac{\lambda}{2\mu}P_1 = \left(\frac{1}{2!}\right)\left(\frac{\lambda}{\mu}\right)^2 P_0$$

Similarly, for $n = 2$,

$$P_3 = \left(\frac{1}{3!}\right)\left(\frac{\lambda}{\mu}\right)^3 P_0$$

$$\vdots$$

$$P_n = \frac{\lambda}{n\mu} P_{n-1} = \left(\frac{1}{n!}\right)\left(\frac{\lambda}{\mu}\right)^n P_0, \qquad For\ 1 \leq n \leq c$$

From the equation (4),

$$c\mu\, P_{c+1} = (\lambda + c\mu)\, P_c - \lambda P_{c-1}$$

$$P_{c+1} = \frac{(\lambda + c\mu)}{c\mu} P_c - \frac{\lambda}{c\mu} P_{c-1}$$

$$= \frac{\lambda}{c\mu} P_c + P_c - \frac{\lambda}{c\mu} P_{c-1}$$

$$= \frac{\lambda}{c\mu} P_c + \frac{\lambda}{c\mu} P_{c-1} - \frac{\lambda}{c\mu} P_{c-1} = \frac{\lambda}{c\mu} P_c$$

$$P_{c+1} = \frac{1}{c}\left(\frac{1}{c!}\right)\left(\frac{\lambda}{\mu}\right)^{c+1} P_0$$

Similarly,

$$P_{c+2} = \frac{\lambda}{c\mu} P_c = \frac{1}{c^2}\left(\frac{1}{c!}\right)\left(\frac{\lambda}{\mu}\right)^{c+2} P_0$$

In general,

$$P_n = P_{c+(n-c)} = \frac{1}{c^{n-c}}\left(\frac{1}{c!}\right)\left(\frac{\lambda}{\mu}\right)^n P_0, \qquad for\ c \leq n \leq N$$

$$P_n = \begin{cases} \left(\frac{1}{n!}\right)\left(\frac{\lambda}{\mu}\right)^n P_0 & n < c \\ \frac{1}{c^{n-c}}\left(\frac{1}{c!}\right)\left(\frac{\lambda}{\mu}\right)^n P_0, & c \leq n \end{cases} \qquad \dots (5)$$

Moreover,

$$\sum_{n=0}^{\infty} P_n = 1 \Rightarrow \sum_{n=0}^{c-1} P_n + \sum_{n=c}^{\infty} P_n = 1$$

That is,

$$\sum_{n=0}^{c-1}\left[\left(\frac{1}{n!}\right)\left(\frac{\lambda}{\mu}\right)^n P_0\right] + \sum_{n=c}^{\infty}\left[\frac{1}{c^{n-c}}\left(\frac{1}{c!}\right)\left(\frac{\lambda}{\mu}\right)^n P_0\right] = 1$$

$$P_0\left[\sum_{n=0}^{c-1}\left(\frac{c^n}{n!}\right)\left(\frac{\lambda}{c\mu}\right)^n + \frac{1}{c!}\sum_{n=c}^{\infty}\frac{c^n}{c^{n-c}}\left(\frac{\lambda}{c\mu}\right)^n\right] = 1$$

$$P_0 = \left[\sum_{n=0}^{c-1}\left(\frac{c^n}{n!}\right)\left(\frac{\lambda}{c\mu}\right)^n + \frac{1}{c!}\sum_{n=c}^{\infty}\frac{c^n}{c^{n-c}}\left(\frac{\lambda}{c\mu}\right)^n\right]^{-1}$$

By taking$r = \frac{\lambda}{\mu}$ $, and\ \rho = \frac{\lambda}{c\mu}$,

$$P_0 = \left[\sum_{n=0}^{c-1}\left(\frac{r^n}{n!}\right) + \sum_{n=c}^{\infty}\frac{r^n}{c!\, c^{n-c}}\right]^{-1}$$

Considering infinite series of the above equation

$$\sum_{n=c}^{\infty}\frac{r^n}{c!\, c^{n-c}} = \frac{r^c}{c!}\sum_{n=c}^{\infty}(\rho)^{n-c}$$

$$= \frac{r^c}{c!}\sum_{m=0}^{N}\left(\frac{r}{c}\right)^m$$

$$= \frac{r^c}{c!}\frac{1}{1-\frac{r}{c}} \qquad (\frac{r}{c} = \rho < 1)$$

Hence,

$$P_0 = \left[\sum_{n=0}^{c-1}\left(\frac{r^n}{n!}\right) + \frac{r^c}{c!\, 1-\rho}\right]^{-1}\left(\frac{r}{c} = \rho < 1\right) \qquad \dots (6)$$

And from this performance Measures are:

1. Expected number of client requests in queue$E[N_q]$:

$$E[N_q] = \sum_{n=c+1}^{\infty} (n-c)P_n$$
$$= \sum_{n=c+1}^{\infty} (n-c)\frac{1}{c^{n-c}}\left(\frac{1}{c!}\right)\left(\frac{\lambda}{\mu}\right)^n P_0$$
$$= \frac{P_0 r^c}{c!}\sum_{n=c+1}^{\infty} (n-c)\frac{r^{n-c}}{c^{n-c}}$$

$$= \frac{P_0 r^c}{c!}\sum_{n=c+1}^{\infty} (n-c)\,\rho^{n-c}$$
$$= \frac{P_0 r^c}{c!}\sum_{m=1}^{\infty} m\,\rho^{m}$$
$$= \frac{\rho P_0 r^c}{c!}\sum_{m=1}^{\infty} m\,\rho^{m-1}$$
$$= \frac{\rho P_0 r^c}{c!}\frac{d}{d\rho}\sum_{m=1}^{\infty} \rho^{m}$$
$$= \frac{\rho P_0 r^c}{c!}\frac{d}{d\rho}\left(\frac{1}{1-\rho}-1\right)$$
$$E[N_q] = \frac{\rho r^c}{c!\,(1-\rho)^2}P_0 \qquad \dots(7)$$

And by using Little's formula other performance measures are

2. Expected waiting time of client requests in queue $E[T_q]$:

$$E[T_q] = \frac{E[N_q]}{\lambda}$$
$$= \frac{r^c}{c!\,c\mu(1-\rho)^2}P_0 \qquad \dots(8)$$

3. Expected number of client requests in system $E[N]$:

$$E[N] = E[N_q] + \frac{\lambda}{\mu}$$
$$= r + \frac{\rho r^c}{c!\,(1-\rho)^2}P_0 \qquad \dots(9)$$

4. Expected time of client requests in system $E[T]$:

$$E[T] = E[T_q] + \frac{1}{\mu}$$
$$= \frac{r^c}{c!\,c\mu(1-\rho)^2}P_0 + \frac{1}{\mu} \qquad \dots(10)$$

Eq. (10) use to evaluate the Latency or Response time of the system.

As $W_q(0)$ gives the probability that client requests have zero delay in queue before getting the service. So $1 - W_q(0)$ is the probability that client requestshave nonzero delay before getting the service. $1 - W_q(0)$ does not directly indicate the length of time in queue but the probability that a transaction not able to immediately access a server.

To find $W_q(0)$,

$$W_q(0) = \Pr\{T_q = 0\}$$
$$= \sum_{n=0}^{c-1} P_n = \sum_{n=0}^{c-1}\left(\frac{r^n}{n!}\right)$$

To find $\sum_{n=0}^{c-1}\left(\frac{r^n}{n!}\right)$, from equation (6)

$$\sum_{n=0}^{c-1}\left(\frac{r^n}{n!}\right) = \frac{1}{P_0} - \frac{r^c}{c!\,(1-\rho)}$$

$W_q(0)$ becomes,

$$W_q(0) = P_0\left(\frac{1}{P_0} - \frac{r^c}{c!\,(1-\rho)}\right)$$
$$= 1 - \frac{r^c P_0}{c!\,(1-\rho)} \qquad \ldots(11)$$

And the probability that an arriving client request has a nonzero wait in queue is,

$$C(c,r) \equiv 1 - W_q(0)$$

$$C(c,r) = \frac{r^c}{c!(1-\rho)} \Bigg/ \frac{r^c}{c!(1-\rho)} + \sum_{n=0}^{c-1}\left(\frac{r^n}{n!}\right) \qquad \ldots(12)$$

The above equation is Erlang - C formula which gives the probability that an arriving client requestsis delayed in the queue. Message delay can be analysed using this formula.

Throughput: Throughput is to measure a system's ability in handling issues, requests, and transactions per unit time. It is also an important indicator to measure the system's concurrency. Here, the throughput of the system can be calculated as the average number of verified requests per unit time. In this case, a verified request refers to a log entry that has been successfully replicated across the follower nodes and applied to the leader's state machine. The formula is as follows:

$$TPS_{\Delta T} = \frac{Total\ Number\ of\ Verified\ client\ requests}{Taken\ time} \qquad \ldots(13)$$

Latency: This is the time it holds for a client request to be managed and for the system to reach consensus on the response. It is defined as Expected time of client requests in system $E[T]$ which is same as equation (10):

$$E[T] = E[T_q] + \frac{1}{\mu}$$
$$= \frac{r^c}{c!\,c\mu(1-\rho)^2} P_0 + \frac{1}{\mu}$$

As Raft consensus is for permissioned blockchain the cluster of nodes are limited. This model ignores complexities such as different capacities of nodes as a system, network conditions and system-specific implementation details.

VI. Numerical Experiments

This analysis gives understanding on how Raft performs in various contexts and can be applied to optimize system design processes. Raft's performance can be increased by changing system characteristic such as the number of follower nodes, their service capacity, arrival of client requests etc. The results will be analysed between the trade-offs of throughput, latency, and probability of delay. It can aid system designers in making feasible choices regarding resource allocation and system setup.

The numerical experiments are to demonstrate various primary parameters for the relevant raft consensus mechanism of blockchain technology model in the context of a queueing system. Performance measures such as Expected time of client requests in system can be referred as latency, expected number of client requests in the system and queue, throughput, probability of message delay, scalability etc. of Raft consensus algorithm are analysed. Numerical experiments of theoretical results are carried out in python.

For different values of μ, parameters such as number of followers i.e., channels, expected time of client request in system i.e., latency, probability of message delay etc are measured for $\lambda \in [150, 600]$

Figure 2(a)and 2(b) shows graph of expected time of client request which is refer to as latency and probability of massage delay for $\mu = 650$. As arrival rate increases expected time of client request increases for $c = 3$. For $c = 5\ and\ 11$, it refers that higher number of followers; parameter latency and message delay does not depend on arrival rate. Figure 2(c) follows the same pattern for the latency and throughput.

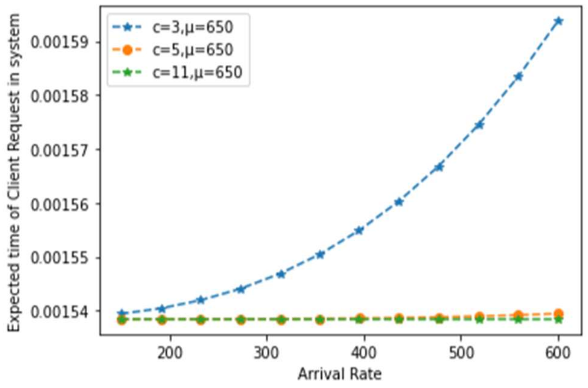


Figure 2(a): Graph of Expected time of Client Request in system for μ = 650

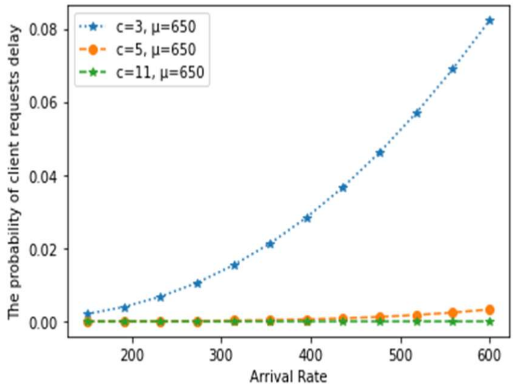


Figure 2(b): Graph of the probability of client request delay μ = 650

Figure 3(a) and 3(b) shows graph of expected time of client request which is refer to as latency and probability of massage delay for $c = 3$ and different μ. As arrival rate increases expected time of client request increases for $c = 3$ and different μ. Figure 3(c) follows the same pattern for the latency and throughput.
Figure 4(a) and 4(b) shows graph of expected time of client request which is referred as latency and probability of massage delay for combinations of c and μ. As arrival rate increases expected time of client request increases for combinations of c and μ. Figure 4(c) follows the same pattern for the latency and throughput.

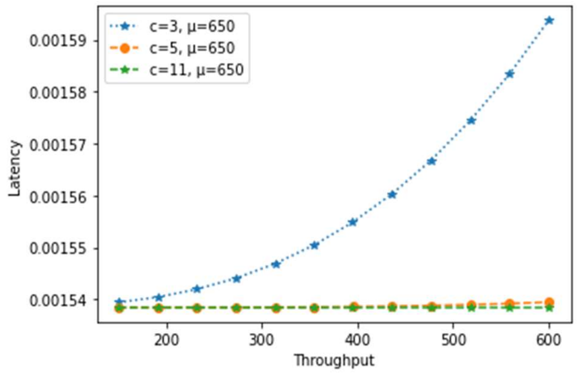


Figure 2(c): Graphs of Throughput vs Latency for μ = 650

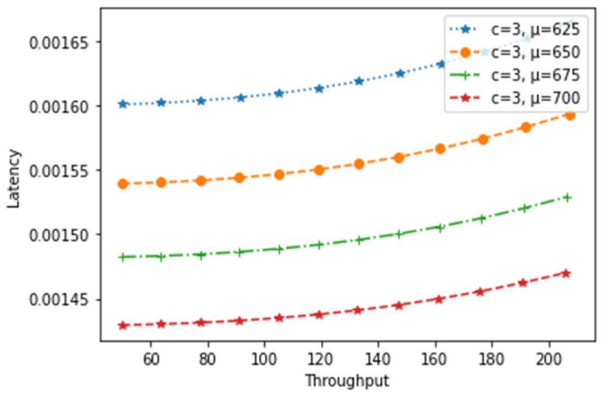
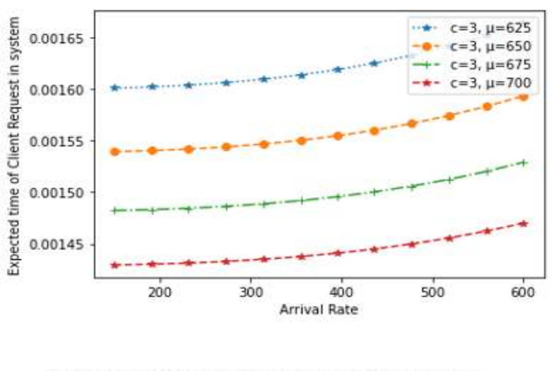


Figure 3(c): Graphs of Throughput vs Latency for c = 3



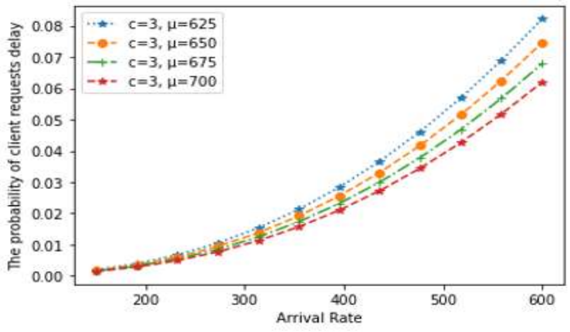


Figure 3(a): Graph of Expected time of Client Request in system for $c = 3$

Figure 3(b): Graph of the probability of client request delay $c = 3$

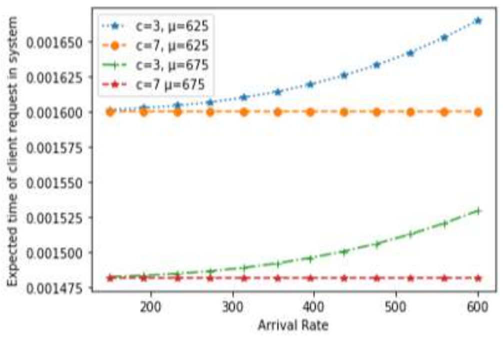


Figure 4(a)Graph of Expected time of Client Request in system for combinations ofc and μ

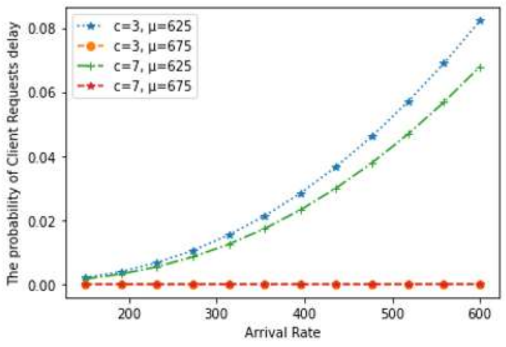


Figure 4(b): Graph of the probability of client request delay for different combinations ofc and μ

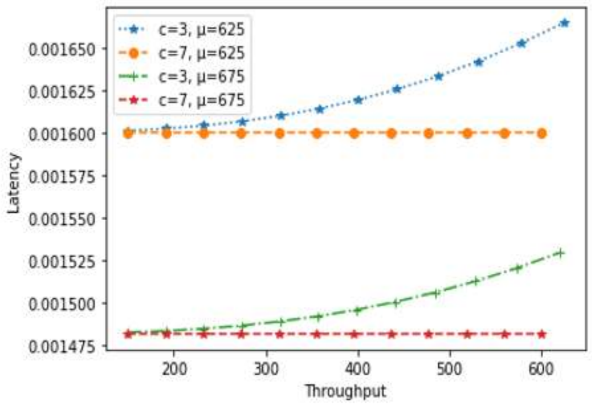


Figure 4(c): Graph of throughput and latency for combinations of c and μ

It is important to note that the work done for the system is more general for M/M/C queue. A system in terms of its input i.e., client request arrived at system, probability of message delay and expected time of client request in system i.e., latency are observed for different number of followers node. Client requests depart after taking the service., are observed. We do not need to specify how the client request are arrived within the system, configured of follower node.

VII. Conclusion

In this paper, the performance of Log replication process in the Raft consensus algorithm using an M/M/C queueing model is analysed. It models the arrival and service processes of client requests while considering the multi-channel nature of the follower nodes. To examine the performance of distributed systems, key measures include throughput, latency, and message delay. These measures determine how many client requests can be processed within a given time and how quickly they can be completed. This study can be used to optimize system design procedures by providing an understanding of how Raft works in various circumstances. Latency and probability increase with increasing arrival rates for lower follower networks. However, by increasing the number of follower nodes, latency, and probability decrease. Raft's performance can be enhanced by changing system characteristics such as the number of follower nodes. Numerical experiments performed in Python are provided to demonstrate the validity of the baseline simulation model in the context of a queuing system. These experiments can assist system designers in making practical decisions about resource allocation and system configuration.

References

[1]. Huang, D., Ma, X., & Zhang, S. (2019). "Performance analysis of the raft consensus algorithm for private blockchains". IEEE Transactions on Systems, Man, and Cybernetics: Systems, 50(1), 172-181.

[2]. Kang, H., Chang, X., Mišić, J., Mišić, V. B., Yao, Y., & Chen, Z. (2020). "Stochastic modelling approaches for analyzing blockchain: A survey". arXiv preprint arXiv:2009.05945.

[3]. Kim, D., Doh, I., & Chae, K. (2021, January). "Improved raft algorithm exploiting federated learning for private blockchain performance enhancement". In 2021 International Conference on Information Networking (ICOIN) (pp. 828-832). IEEE.

[4]. Memon, R. A., Li, J. P., & Ahmed, J. (2019). "Simulation model for blockchain systems using queuing theory". Electronics, 8(2), 234.

[5]. Ongaro, D., & Ousterhout, J. (2014). In search of an understandable consensus algorithm. In Proceedings of the 2014 USENIX conference on USENIX annual technical conference (pp. 305-319). USENIX Association.

[6]. Samy, H., Tammam, A., Fahmy, A., & Hasan, B. (2021). Enhancing the performance of the blockchain consensus algorithm using multithreading technology. Ain Shams Engineering Journal, 12(3), 2709-2716.

[7]. Shortle, J. F., Thompson, J. M., Gross, D., & Harris, C. M. (2018). Fundamentals of queueing theory (Vol. 399). John Wiley & Sons.

[8]. Subramanian, C., George, A., Abhilash K., A., & Karthikeyan M., (2021). Blockchain Technology, Universities Press (India) Private Limited.

[9]. Tomić, N. Z. (2021). "A review of consensus protocols in permissioned blockchains". Journal of Computer Science Research, 3(2), 32-39.

[10]. Vora S.& Gor R. (2022), "Study of Average Transaction Confirmation time in a Blockchain using Queueing Model", IOSR Journal of Computer Engineering (IOSR-JCE),24(3), 2022, pp.60-68.

[11]. Vora S. & Gor R. "Queueing Model for PoA based Blockchain system." IOSR Journal of Mathematics (IOSR-JM), 18(6), (2022): pp. 31-38.