Journal of Engineering Science and Technology
Vol. 11, No. 2 (2016) 193 - 211
© School of Engineering, Taylor's University

ANALYSIS OF A DATABASE REPLICATION ALGORITHM UNDER LOAD SHARING IN NETWORKS

SANJAY KUMAR YADAV[1,*], GURMIT SINGH[1], DIVAKAR SINGH YADAV[2]

[1]Department of Computer Science & IT, Sam Higginbottom Institute of Agriculture, Technology & Sciences, Allahabad, India
[2]Department of Computer Science & Engineering, Institute of Engineering and Technology, Lucknow, India
*Corresponding Author: yadav_sk@rediffmail.com

Abstract

Recently, (PDDRA) a Pre-fetching based dynamic data replication algorithm has been published. In our previous work, modifications to the algorithm have been suggested to minimize the delay in data replication. In this paper, a simulation framework is presented and results are obtained to estimate the throughput and average delay. The overall network is divided into two parts as local and global networks. The data requests are generated only at the local nodes. However, the service can be obtained form both local and global servers. In our previous work it has been found that the throughput and average delay heavily depends on buffer capacity of sever node and if server load is below 80% then, nearly 100% throughput is possible with very small average delay. In this paper, we have shown that shown the delay can be further minimized by sharing the load among servers, still throughput remains nearly 100 percent.

Keywords: Database replication, Throughput, Average delay.

1. Introduction

The growing need towards decentralization in any enterprise has created a strong necessity for database replication. This is because businesses today are more geographically dispersed and it is expected to provide location transparency to the employees of an organization [1]. Data replication and synchronization have been topics of research for quite some time in the area of distributed databases. Nowadays database replication techniques see applications in many fields like mobile computing, wireless sensor networks, mobile ad-hoc networks, etc. [2]. Thus, research has to be applied to a new area of applications.

Nomenclatures

a	Fraction of load transferred to the global network
B	Buffer size
B_R	Bit rate
D	Total delay
D^L	Delay at local server
D^G	Delay at global server
D_F	Latency due to frame size
$D_{N/W}$	Round trip delay
F_s	Frame size
K	Number of request arrive at a server
l	Number of request served at local network
N	Number of servers
P	Probability of local service
$P[K]$	Probability that K requests arrive at the particular server
P_r	Probability of request generation at local network
r	Total number of requests generated at local network
t_q	Access time of master node
t_s	Access time of local node
S	Server
T	Throughput

Greek Symbols

λ	Transaction arrival rate
λ^G	Mean value of request served globally
λ^L	Mean value of request served locally

Abbreviations

M-PDDRA	Modified-PDDRA
PDDRA	Pre-fetching Based Dynamic Data Replication Algorithm
VO	Virtual Organisation

In distributed systems, identical data items may be managed at physically distributed places or machines. Data items that are located within a database, a file system or in memory as a variable within a program can be modified by processes If only one process at a time is trying to modify a data item, there is no need for synchronization. If more than one process at a time try to modify a data item, it is necessary to ensure that no two processes access the data item at the same time. These data items can be manipulated independently from each other, resulting in different versions. These logically connected data items need to be merged back into a consistent state. This process is called data synchronization [3].

In Internet applications, a large number of users that are geographically dispersed all over the world may routinely query and update the same database. In the environment which generally changes with time, the location of the data can have a significant impact on an application's response time and availability [4]. In a centralized approach, only one copy of a database is managed. This approach is simple since contradicting views among replicas are not possible. However, the centralized replication approach suffers from two major drawbacks:

- High server load or high communication latency for remote clients.
- Sometime server may not be available due to the down time or lack of connectivity. Clients in portions of the network that are temporarily disconnected from a server cannot be serviced.

A server load and server downtime problems can be addressed by replicating the database servers to form a cluster of peer servers that coordinate updates. Wide area database replication coupled with a mechanism to direct clients to the best available server (network-wise and load-wise) [5] can greatly enhance both the response time and availability.

Wahid et al., [11] describe distribution problem of replicated databases and its optimization in a computer network. As replication enables data availability in case of any site failure and also provides local access of data. This paper presents a bio-inspired replication management approach which is based on swarm intelligence. Chen et al., [12] discuss structure for grid databases replication. As database replication enables data availability, fault tolerance and minimal access time in grids. Most systems that use grid replication now-a-days deals with only read files. Some of the relational database products offer transaction based replication but these tools cannot cope with the grid issues. The approach discussed in this paper provides metadata registry using grid mechanism and also makes replicas of data resources.

In [14], Goel and Buyya state that replication is one of the known phenomena in which copies of data are stored at different locations. In a distributed environment through replication technique data is accessed efficiently. Replication provides data consistency and availability each time without bothering failure of any site because of its data replicas. If any request of data access is originated, it finds its closely located replica which increases performance of system. Sears et al., [16] propose a database replication engines which provides high throughput regardless of database size and query content. Serrano et al., [19] discuss the performance gain which is achieved by partial replication configurations is analyzed analytically, like the configuration of all sites which does not store all data. A partial replication protocol is also derived which provides 1-copy snapshot isolation, which is correctness criteria.

When taking a look at the types of communication models that are being used, it is like a client server model. In the client-server model, the request will be generated by a client, from another computing system, to a system which plays role of a server, which is located at the fixed network. Once data replication is done at the network layer, there are many issues that come into the picture.

Challenges in database replication:

- Response time is an important parameter. It is defined as the time taken for the client to access the data from servers. If a server is far away from the network; response time will be more to service a client.
- Another issue in data replication is to synchronize the replicas of data at all nodes.
- A fundamental challenge in database replication is maintaining a low cost of updates while assuring global system consistency. The problem is magnified

for wide area replication due to the high latency and the increased likelihood of network partitions in wide area settings [5].

Therefore, in database replication, the location of nodes and their availability is important. In our previous work, PDDRA (Pre-fetching Based Dynamic Data Replication Algorithm [6]) has been modified to reduce the network based latency and a mathematical model is presented to evaluate the throughput and average delay [7]. In this paper, main points of the algorithm are highlighted, simulation environment is created and results are presented to evaluate the throughput and average delay.

The rest of the paper is organized as follows: In section 2, the points of M-PDDRA algorithm is highlighted. The simulation framework and results are discussed in section 3. Network propagation delay is discussed in section 4. The major conclusions of the paper are discussed in section 5.

2. M-PDDRA Algorithm

In past, A PDDRA (Pre-fetching Based Dynamic Data Replication Algorithm) is presented. The main idea is to pre-fetch some data using the heuristic algorithm before actual replication start to reduce latency. In our previous work [9], modifications in PDDRA (M-PDDRA) are suggested to further reduce the latency. For more detail please refer to Refs. [7, 9]. In summary, the main points of the algorithm are:

- In the modified scheme the internet cloud will be considered as master node as it can be assumed that the data is available in the internet for the replication (Fig. 1).
- If any replication request is generated by a node then via edge node first it will be searched in same local network, then it will search in internet, if data is locally available at any node then it will be replicated and there will not be any need to connect through the master node.
- There is a possibility that the data may not be available at any local node or waiting time is too large, as simultaneous request is sent to both to a local node and master node, if access of master node is in queue for let's say time t_q then local search will only be done for time $t_s < t_q$. These simultaneous requests to both local and global network will reduce latency in comparison to first request is sent to local network then thereafter to global network.

3. Simulation Framework and Results

The replicated data is either available locally or it is available globally i.e., at the internet. Therefore, when requests are generated, some of the generated requests will be full-filled locally and leftover requests will be fetched from internet (master node). In this section simulation framework is developed to estimate the average response time of all the transactions.

In nutshell, there are four processes in database replication:

- Request generation at a local node

- Request serving at a local node
- Network propagation
- Servicing at a remote site in the global network

The important network parameters are:

Network Throughput: It refers to the volume of data that can flow through a network, or in other words, the fraction of the generated request which can be served.

Network Load: In networking, load refers to the amount of data (traffic) being carried by the network.

Network Delay: It is an important design and performance characteristic of data network. The delay of a network specifies how long it takes for a request to travel across the network from one node or endpoint to another. Delay may differ slightly, depending on the location of the specific pair of communicating nodes.

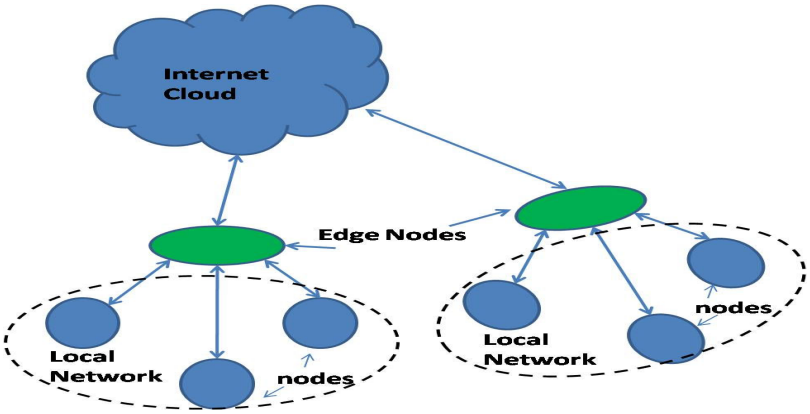

Fig. 1. Schematic of the database replication in network.

Let in the local network, r requests are generated with probability p_r (it is assumed that each request are generated with equal probability), out of which l requests are served locally with probability p, and then (r - l) requests will be transmitted to the outer world (global network). Hence, transaction arrival rate is represented by λ

$$\lambda_i = r p_r \tag{1}$$

Then the mean value of the requests served locally is

$$\lambda_{av}^{L} = \lambda_i p \tag{2}$$

and the requests served globally are

$$\lambda_{av}^{G} = \lambda_i (1-p) \tag{3}$$

Total requests served are

$$\lambda_i = \lambda_{av}^L + \lambda_{av}^G \tag{4}$$

$$\lambda_i = rp_r p + r(1-p)p_r \tag{5}$$

The throughput can be calculated as

$$T = \frac{(1-a)}{100} T_{av}^L + \frac{a}{100} T_{av}^G \tag{6}$$

where, '***a***' is the faction of load transferred to the global network.

The total delay can be evaluated as

$$D = D_{av}^L + D_{av}^G \tag{7}$$

The simulation is done in MATLAB. The simulation pattern is based on random number generation and well known as Monte Carlo simulation. In the simulation first random traffic model is considered and in the later part Poison traffic model is considered.

3.1. Random traffic model

In the simulation we have assumed:

- Request can be generated at any of the input with probability p_r .
- With probability p the generated request served locally.
- Each request is equally likely to go to any of the server (locally or globally) with probability $\frac{1}{N}$, where *N* is the number of servers available.

The probability that K requests arrive at the particular server is given by

$$P[K] = {}^{N}C_{K} \left(\frac{p}{N}\right)^{K} \left(1 - \frac{p}{N}\right)^{N-K} \tag{8}$$

In the simulation synchronous network is considered hence time is divided into slots. Therefore following assumptions are made:

- The requests can be generated at the slot boundary only.
- Updates can be made at the slot boundary only.
- Once replication is started, then further updation is not allowed till replication complete.

In the previous work, we have obtained the simulation results under various buffering conditions and it has been fount that, as the buffer space increases the average throughput increases, with reasonable amount of increase in the average delay [8]. In [8], we have also shown that the sharing of the load on the local and global networks further improves the system.

In this paper, we have analyzed that if load is distributed among the servers, then how delay will reduce as throughput will surely be increased. Here, we have simulated three cases, for the load distribution as shown in Figs. 2 to 11.

In Fig. 2, the value of 'a' is considered to be 100% and 20%. It is examined from the figure if the entire load is given to a particular server then at the higher load throughput decreases, and if load is shred as 20% of the load is on the server, the throughput always remains one.

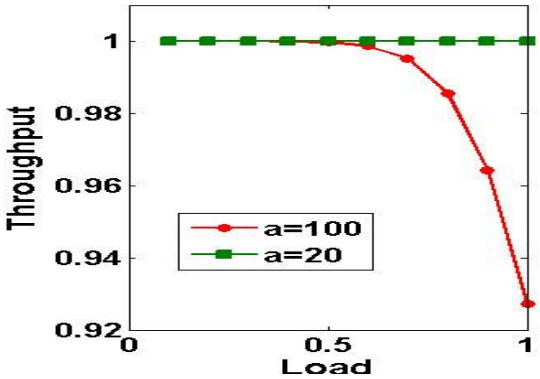

Fig. 2. Throughput vs. load with fixed buffering capacity (*B*) of 4 while considering request generating nodes and servers are 4 and assuming *a*=20% and 100%.

Considering Fig. 3, it is also seen that when the entire load is given to a particular server than the average delay increases and attain the value of 4 at the load one. However when the load is shared, the average delay at the load of 1 is only 0.8 slots.

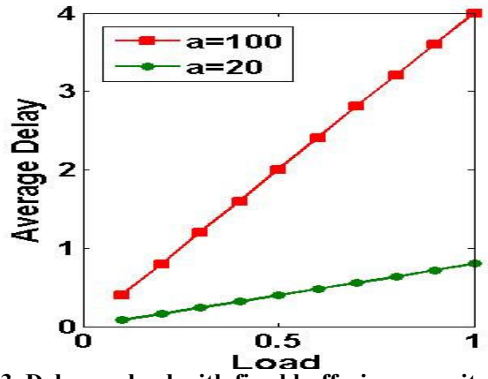

Fig. 3. Delay vs. load with fixed buffering capacity (*B*) of 4 while considering request generating nodes and servers are 4 and *a*=20% and 100%.

Figure 4 is very much similar to Fig. 2, here the value of 'a' is chosen to be 100% and 30%. It is seen from the figure if the entire load is given to a particular server than at the higher load throughput decreases, and if load is shared as 30%

of the load is on the server, the throughput always remains one. Hence, 30% of the load does not affect the throughput performance. Considering Fig. 5, it is also seen that when the entire load is given to a particular server than the average delay increases and attains the value of 4 slots at the load of 1. However, when the load is shared, the average delay at the load of 1 is only 1.2 slots.

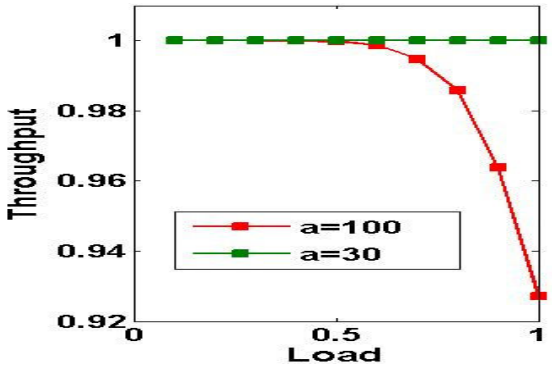

Fig. 4. Throughput vs. load with fixed buffering capacity (*B*) of 4 while considering request generating nodes and servers (*N*) are 4 and assuming *a*=30% and 100%.

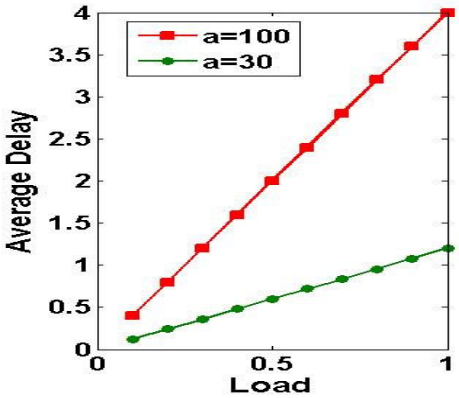

Fig. 5. Delay vs. load with fixed buffering capacity (*B*) of 4 while considering request generating nodes and servers (*N*) are 4 and assuming *a*=30% and 100%.

In Fig. 6, the value of 'a' is considered to be 100% and 50%. It is clear from the figure, even for 'a' is 50%, the throughput performance remains the same as for a=20 and a=30%. Considering Fig. 7, the average delay at the load of 0.5 is one slot, and at the load of one average delay is of only 2 slots.

In Fig. 8, the value of 'a' is considered to be 100% and 25%. It is examined from the figure, even for 'a' is 25%, the throughput performance remains the same as for a=20 and a=30%. Considering the Fig. 9, the average delay at the load of 0.5 is 0.4 slots and at the load of 1 average delay is of only 1.0 slot.

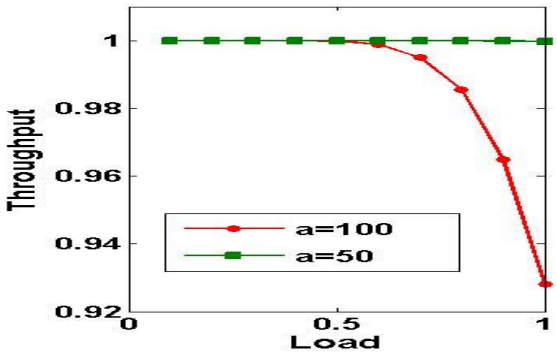

Fig. 6. Throughput vs. load with fixed buffering capacity (*B*) of 4 while considering request generating nodes and servers (*N*) are 4 and Assuming *a*=50% and 100%.

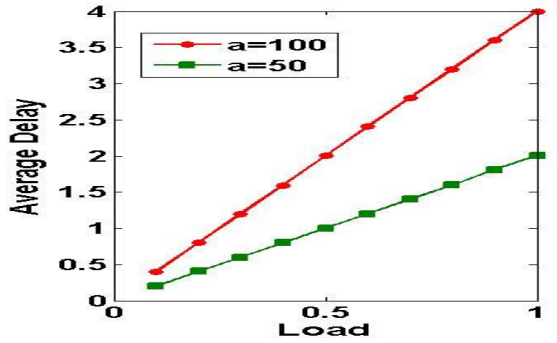

Fig. 7. Delay vs. load with fixed buffering capacity (*B*) of 4 while considering request generating nodes and servers (*N*) are 4 and assuming *a*=50% and 100%.

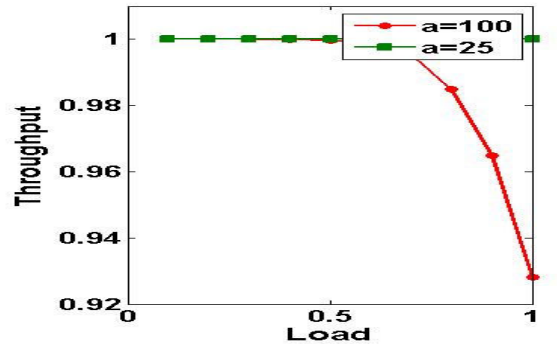

Fig. 8. Throughput vs. load with fixed buffering capacity (*B*) of 4 while considering request generating nodes and servers (*N*) are 4 and Assuming *a*=25% and 100%.

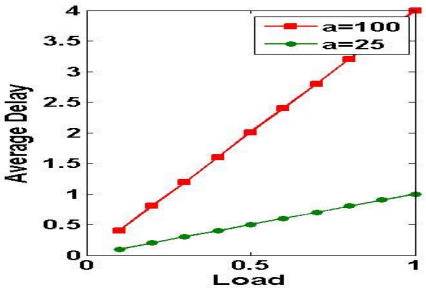

Fig. 9. Average delay vs. load with fixed buffering capacity (*B*) of 4 while considering request generating nodes and servers (*N*) are 4 and assuming *a*=25% and 100%.

In Fig. 10, the various values of value of **'**a**'** are considered. It is seen from the figure if all the load is given to a particular server than at the higher load, throughput decreases and if load sharing is increased the throughput start improving. If the value of 'a' is 100% the throughput is 93% and when 'a' is start decreases the throughput improves and finally attains a value 100%. Considering Fig. 11, it is also seen that when the load increases, the average delay and as the value of 'a' increases the average delay also increases.

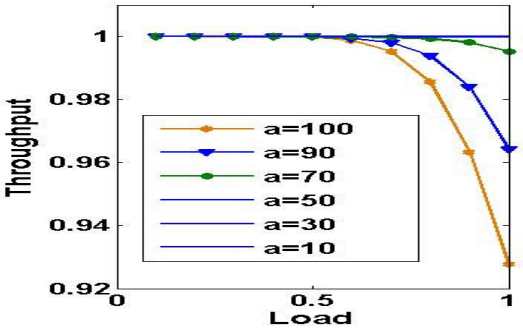

Fig. 10. Throughput vs. load with fixed buffering capacity (*B*) of 4 while considering request generating nodes and servers (*N*) are 4 and assuming various values of 'a'.

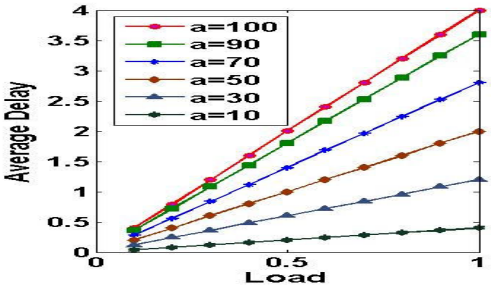

Fig. 11. Delay vs. load with fixed buffering capacity (*B*) of 4 while considering request generating nodes and servers (*N*) are 4 and assuming various values of '*a*'.

From Figs. 2 to 11, the main noticeable points are

- If load is distributed that the throughput is very high and nearly one.
- The distribution of load among various servers reduces the waiting time latency to a significant low level.

Considering Figs. 2 to 7, where we have considered that the load is shared between the three servers with the values of **'a'** as 20%, 30% and 50%. The throughput is given by

$$T = \frac{20}{100} T_{av}^{S_1} + \frac{30}{100} T_{av}^{S_2} + \frac{50}{100} T_{av}^{S_3}$$

$$T = 0.2 \times 1 + 0.3 \times 1 + 0.5 \times 1 = 1.0$$

and the average delay at the load of 1 is given by

$$D = 0.8 + 1.2 + 2.0 = 4.0 \text{ slots}$$

But the above delay will happen only when one request is sent after another, but as we suggested, once requests are generated they are shared among the servers. Hence, simultaneous request to various servers will reduce the delay and net delay will be given by

$$D = \max(0.8, 1.2, 2.0) = 2.0 \text{ slots.}$$

In the same view considering the Figs. 8 and 9, it is very clear that if the load is shared between four servers 25% to each, then the throughput will be 100% and average delay will be 1.0 slot. Similarly for the other values of **'a'** the throughput and average delay can be obtained by considering the Figs. 10 and 11.

Hence, it can be summarized that if the load is shared between the servers, then the throughput increases and average delay decreases. But this distribution of load will lead to the deployment of more number of servers or same data has to be placed on the large number of servers. The optimal value of the server that can maintain high throughput and reasonable delay depends on various parameters like; type of networks, quality of service, server capacity and request type etc. that analysis can be part of further study.

3.2. Results based on Poisson traffic arrival

Till in chapter we have assumed that the generation of requests and service is random in nature, and a particular server *K* request arrive is given by following equation

$$P[K] = {}^{N}C_{K} \left(\frac{p}{N}\right)^{K} \left(1 - \frac{p}{N}\right)^{N-K}$$

Now, if the generation of requests are relatively small and requests generating nodes are large in numbers (generally it is true in any networks) then under the condition the above equation will be modified as

$$P[K] = \frac{e^{-m} m^{K}}{K!}, \text{ where } m = NP_r$$

In Fig. 12, the random traffic data is compared with the Poisson data. Here, number of request generating nodes and servers considered as four. It can be easily examined that in case of random traffic, as the load increases, the

throughput decreases. However, in case of Poisson traffic, due to the less number of arrivals the throughput remains at one.

In Fig. 13, the random traffic data is again compared with the Poisson data for average delay. Here, number of request generating nodes and servers considered as four. It can be easily seen that in case of random traffic, as the load increases, the delay increases. However, in case of Poisson traffic, due to the less number of arrivals the average delay remain nearly zero at lower load at attain a value of 0.2 slot at load of 1.

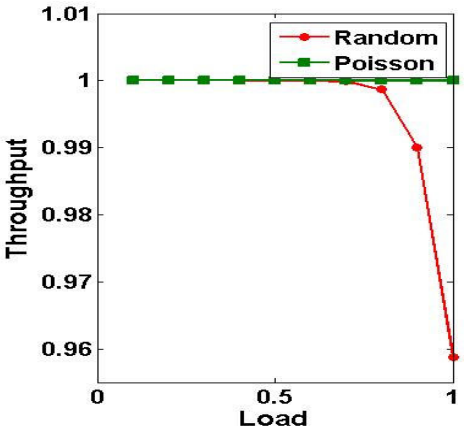

Fig. 12. Throughput vs. load with fixed buffering capacity (*B*) of 8 while considering request generating nodes and servers (*N*) are 4 and considering random and Poisson traffic.

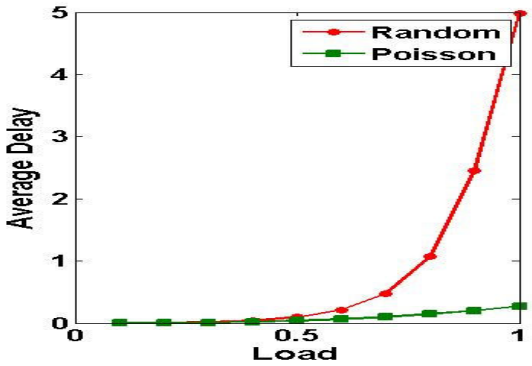

Fig. 13. Average delay vs. load with fixed buffering capacity (*B*) of 8 while considering request generating nodes and servers (*N*) are 4 and considering random and Poisson traffic.

In Fig. 14, throughput vs. load is plotted. Here, number of request generating nodes and servers are varying from 2 to 8 and the buffering capacity is fixed of 4 requests. It can be examined from the figure, as the request generating nodes increases the throughput decreases. However, in the each case, due to the less number of arrivals the throughput is nearly one.

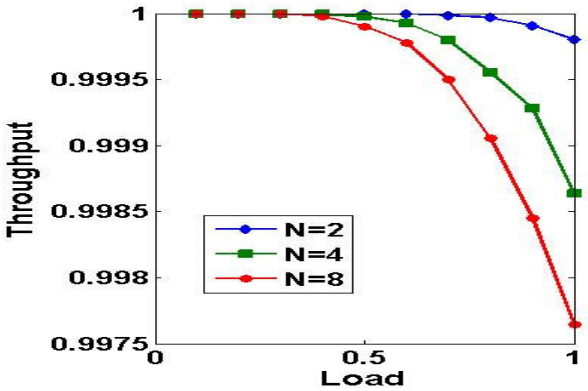

Fig. 14. Throughput vs. load with buffering capacity (*B*) of 4 while varying the request generating nodes and servers (*N*) and considering Poisson traffic.

In Fig. 15, the average delay vs. load is plotted. Here, number of request generating nodes and servers are varying and the buffering capacity is kept of 4 requests. It can be seen that the average delay increases as the request generating nodes increases and is 0.05 slot for N=2 and is of ~0.7 slot for N=8.

In Fig. 16 the throughput vs. load is plotted. Here, number of request generating nodes and servers are 4 and the buffering capacity is varied from 2 to 8 requests. It can be seen from the figure, as the buffering capacity increases the throughput increases. However, still throughput attains a very high value in each case.

In Fig.17, the average delay vs. load is plotted. Here, number of request generating nodes and servers considered as four and the buffering capacity is varied from 2 to 8 requests. It can be visualized that the average delay increases as the buffering capacity of server increases and is 0.54 slot for B=2 and is of ~0.78 slot for B=8.

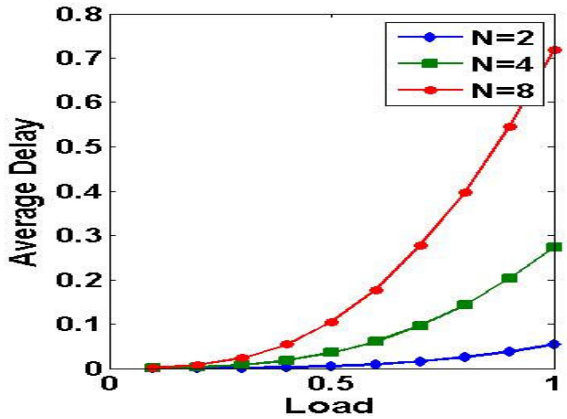

Fig. 15. Average delay vs. load with buffering capacity (*B*) of 4 while varying the request generating nodes and servers (*N*) and considering poisson traffic.

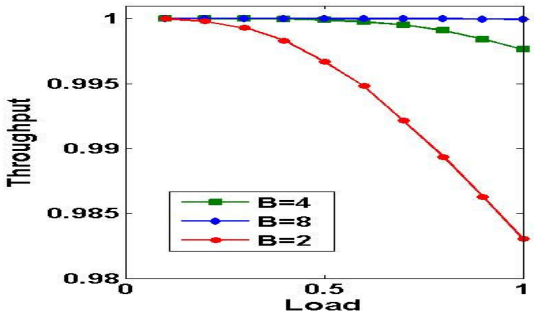

Fig. 16. Throughput vs. load with varying buffering capacity (*B*) while the request generating nodes and servers (*N*) as 4 and considering poisson traffic.

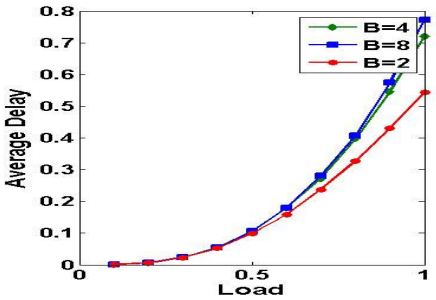

Fig. 17. Average delay vs. load with buffering capacity (*B*) of 4 while varying the request generating nodes and servers (*N*) and considering Poisson traffic.

The results obtained under the Poisson traffic clearly show that, the throughput is nearly one with very less average delay. Hence, data distribution among the servers is not necessary. The Poisson traffic analysis is valid till date, but the more data centric applications are coming up, in near future the random traffic analysis needs to be considered for real time data replication processes.

In the above analysis, the server delay is taken into considerations. However, in WAN data may have to traverse to a geographically far node in such a case the network delay becomes an important parameter. The network delay estimation is presented in the next section.

3.3. Performance comparision of the proposed M-PDDRA and the PDDRA schemes

The PDDRA scheme mailny relies on the pre-fetching of the probable data to reduce the overall latency. In PDDRA scheme, required data is fetched form the master node for replication i.e., the internet. There is no provision to fetch data from the other nodes in the local network where data may be available. As in PDDRA scheme all the generated requests are sent to the global network, i.e. 100% of the load is given to the global network. It fetches the required data along with the probable data that may be needed in future from global network in order

to reduce the overall latency. As the bandwidth between RS and global network is lesser than that between VOs, it takes too much time to replicate the data as the existing PDDRA algorithm has to perform pre-fetching of probable data along with the required data. Howerver, in the proposed scheme a simultaneous request is sent to both local and global networks (master node) by taking into consideration that sometimes it may possible that data may be available locally at some other node.

In the following figures throughput and average delay of the proposed and PDDRA schemes have been given to analyze the comparative performance of both the schemes. Consider Fig. 18., here, it is considered that the number of request generating nodes are four and servers are also four and each server has a buffering capacity of a 8 requests. This figure shows the simulation results of throughput with varying loads.

According to the PDDRA scheme, all the generated requests are sent to the global server; hence, '*a*' which denotes the fraction of traffic being sent to the global server will be 100% i.e. 'a'=100%. It can be seen that till load 0.7 both the schemes have same throughput, but as the load increases beyond the 0.7 mark, the throughput starts decreasing and attains the value of 0.96 at the load of 1.

In comparison to this, the proposed scheme also looks for the availability of the data at the local servers. If data is not available at local network then the proposed scheme has the same performance as the PDDRA scheme. However, if data is available at a local server, then the proposed scheme performs better than the PADRA scheme as shown in Fig. 18.

Now considering the value of '*a*' as 20%, meaning 80% of the data can be fetched from the local servers. In such a case, the throughput of the proposed scheme is one irrespective of the load. Similarly, if the value of '*a*' is considered to be 80%, meaning 20% of the data can be fetched from the local servers, and results are shown in the figure. Here, the throughput remains one till load 0.9 and it decreases slightly at the load of 1.0. In a nut shell both the schemes provide very high throughput even at the higher loads.

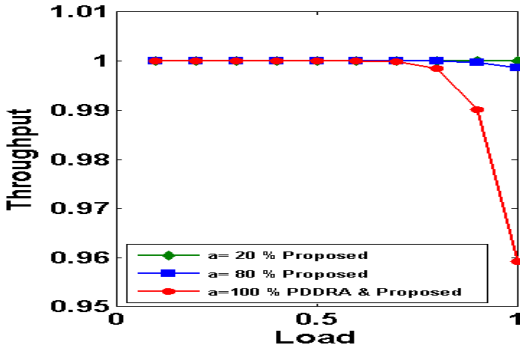

Fig. 18. Throughput vs. load for *N*=4, *S*=4, *B*=8 with '*a*'=20%, 'a'=80%, '*a*'=100%.

Another important performance criterion is the average delay. In Fig. 19, the average delay is plotted vs. load on the servers for both the schemes. Here, it is very clear that the average delay in PDDRA scheme is much larger in comparison

to the proposed scheme. As the load crosses the 0.7 mark, the delay rises exponentially to attain a value of 5 slots at the load of 1.0.

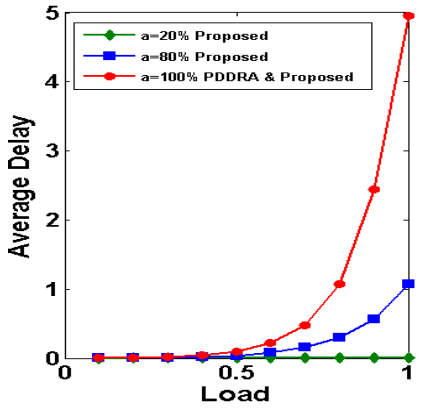

Fig. 19. Average delay vs. load for *N*=4, *S*=4, *B*=8 with '*a*'=20%, '*a*'=80%, '*a*'=100%.

However, with the proposed scheme while considering 80% of the traffic being transferred to the global servers the average delay is zero till 0.5 load and attains a value of one slot at the load of 1.0, which is very small in comparison to the PDDRA scheme. Similarly considering that 20% of the traffic being transferred to the global servers, in such a case the average delay remains zero for all the loads.

From Figs. 18 and 19, it is concluded that the throughput performance of both the proposed and the PDDRA schemes is nearly same, but on the higher loads the performance of the proposed scheme is slightly better than the PDDRA scheme. However, in case of average delay the proposed scheme performs much better than the PDDRA scheme as the PDDRA algorithm requires much time for mining the file access sequences and patterns for pre-fetching.

4. Network delay

When data request traverse through the network, the network delay may be significant, and it becomes an important delay parameter. If we incorporate the network delay ($D_{N/W}$), then the total delay (D) will be formulated as

$$D = D_{av}^{L} + D_{av}^{G} + D_{N/W}$$

Here, $D_{N/W}$, is the round trip delay.

Now, as the more data centric applications are coming up, the whole computer network system is slowly transferring in fiber optic network.

Consider the fiber optic network the latency in the network would be

$$D_{N/W} = \frac{L}{v} = \frac{Ln}{c} = \frac{100 \times 1000 \times 2}{3 \times 10^{8}}$$

$$D_{N/W} = 0.67 \times 10^{-3} \approx 1ms$$

Let the global node is 100 km away for the request generating node, then there will be a network delay of 1.0 ms and thus the round trip delay would be 2.0 ms. Similarly, if a local node is only 100 m away from the request generating node then the round trip network delay would be 2 μs. It also introduces a latency that is proportional to the size of the frame being transmitted and inversely proportional to the bit rate as follows:

$$D_F = \frac{F_S}{B_R}$$

In the above equation, F_S is the frame size and B_R is the bit rate. For a frame of 64 bytes and data rate of 1000 Mbps the delay is $0.5\,\mu s$. As average queuing delay is in terms of frame size, for example a delay of 4 slots will be equal to $0.5 \times 4 = 2.0\,\mu s$.

In case of global network, the main contribution in the delay is due to the propagation delay. Considering the case, when all the generated request are transferred to global network (refer Fig. 3) is

$$D = D_{av}^{G} + D_{N/W} = 0.5 \times 4\,\mu s + 2.0\,ms \approx 2.0\,ms$$

Again considering the case, when all the generated request are served at local network (refer Fig. 3) is

$$D = D_{av}^{L} + D_{N/W} = 0.5 \times 4\,\mu s + 2.0\,\mu s \approx 2.2\,\mu s$$

Overall, it can be observed that on the overall delay the network delay plays very significant role in case of global network. However, in case of local network it plays a minor role in the overall delay.

5. Conclusions

In this paper, detailed simulation of the database replication based algorithm is presented. The main idea behind the algorithm is to reduce the network latency in WAN. Simulation results are presented to obtain the mean waiting time and throughput for a database replication algorithm. For the in-depth analysis of the algorithm various cases are considered and it has been found that the storage capacity has deep impact on the throughput and average delay. If server load is below 80% then, nearly 100% throughput is possible with very small average delay (in slots), even at the higher load the throughput is very acceptable. It is also found that if the numbers of servers that can serve the request are larger, than throughput is very high and average delay is very less. The overall, throughput and average delay also depends heavily on the load and if load is comparatively less (0.8) then the throughput is very high and average delay is nearly zero. In case of the Poisson traffic arrivals the throughput and average delay are under acceptable limits. However, if data has to be replicated form a distant node then the propagation delay overshadowed other delays. Finally, it is concluded that the throughput performance of both the proposed and the PDDRA schemes is nearly same, but on the higher loads the performance of the proposed scheme is slightly better than the PDDRA scheme. However, in case of average delay the proposed scheme performs much better than the PDDRA

scheme as the PDDRA algorithm requires much time for mining the file access sequences and patterns for pre-fetching.

References

1. Kemme, B.; and Alonso, G. (2010). Database replication: a tale of research across communities. *Proceedings of the International Conference on VLDB Endowment*. Switzerland, 3(1), 5-12.
2. Ratnasamy, S.; Karp, B.; Yin, L.; Yu, F.; Estrin, D.; Govindan, R.; and Shenker, S.G. (2002). A geographic hash table for data-centric storage. *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications.* Atlanta, GA, USA, 78–87.
3. Wiesmann; Pedone; Schiper; Kemme; and Alonso (2000). Understanding replication in databases and distributed systems. *Proceedings of 20th International Conference on Distributed Computing Systems.* Zurich, 1-23.
4. Yair, A.; Claudiu, D.; Michal, M.A.; Jonathan, S.; and Ciprian, T. (2002). Practical wide-area database replication. Retrieved December 10, 2011, from http://www.cnds.jhu.edu/publications.
5. Amir, Y. (1995). Replication using group communication over a partitioned network. Retrieved October 5, 2010, from www.cs.jhu.edu/~yairamir.
6. Saadat, N.; and Rahmani, A.M. (2012). PDDRA: A new pre-fetching based dynamic data replication algorithm in data grids. *Springer: Future Generation Computer Systems,*28, 666-681.
7. Yadav, S.K.; Singh, G.; and Yadav, D.S. (2013). Mathematical framework for a novel database replication algorithm. *International journal of Modern Education & Computer Science,* 5(9), 1-10.
8. Yadav, S.K.; Singh, G.; and Yadav, D.S. (2013). Analysis of database replication algorithm in local and global networks. *International Journal of Computer Applications,* 84(6), 48-54.
9. Yadav, S.K.; Singh, G.; and Yadav, D.S. (2013). Throughput and delay analysis of database replication algorithm. *International journal of Modern Education & Computer Science,* 5(12), 47-53.
10. Khan, M.; and Khan, M.N.A. (2013). Exploring query optimization techniques in relational databases. *International Journal of Database Theory & Application*, 6(3), 11-20.
11. Abdul-Wahid, S.; Andonie, R.; Lemley, J.; Schwing, J.; and Widger, J. (2007). Adaptive distributed database replication through colonies of pogo ants. *Proceedings of IEEE International Symposium on Parallel and Distributed Processing Symposium.*USA*,* 1-8.
12. Chen, Y.; Berry, D.; and Dantressangle, P. (2007) Transaction based grid database replication. *Proceedings of UK e-Science. Edinburgh,* U.K. 166-173.
13. Correia, A.; Pereira, J.; Rodrigues, L.; Carvalho, N.; Vilaça, R.; Oliveira, R.; and Guedes, S. (2007). GORDA: An open architecture for database replication. *Proceedings of Sixth International Symposium on Network Computing and Applications.* Boston, USA, 287-290.

14. Goel, S.; and Buyya, R. (2006). Data replication strategies in wide area distributed systems. Retrieved September 12, 2011, from http://www.buyya.com
15. Daudjee, K..; and Salem; K. (2006). Lazy database replication with snapshot isolation. *Proceedings of the 32nd International Conference on Very large data bases Endowment*. Seoul Korea, 715-726.
16. Sears, R.; Callaghan, M.; and Brewer, E. (2008). Rose: compressed, log-structured replication. *Proceedings of the International Conference on VLDB Endowment*. Auckland, New Zealand, 1(1), 526-537.
17. Agrawal, P.; Silberstein, A.; Cooper, B.F.; Srivastava, U.; and Ramakrishnan, R. (2009). Asynchronous view maintenance for VLSD databases. *Proceedings of the ACM SIGMOD International Conference on Management of data.* New York, USA, 179-192
18. Thomson, A.; Diamond, T.; Weng, S.C.; Ren, K..; Shao P.; and Abadi, D.J. (2012). Calvin: fast distributed transactions for partitioned database systems. *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Scottsdale, Arizona, USA, 1-12.
19. Serrano, D.; Patiño-Martínez, M.; Jiménez-Peris, R.; and Kemme, B. (2007). Boosting database replication scalability through partial replication and 1-copy-snapshot-isolation. *Proceedings of* IEEE *13th Pacific Rim International Symposium on Dependable Computing*. Melbourne, 290-297.
20. Armendáriz-Inigo, J.E.; Mauch-Goya, A.; de Mendívil, J.R.; and Muñoz-Escoí, F.D. (2008). SIPRe: a partial database replication protocol with SI replicas. *Proceedings of the ACM Symposium on Applied Computing*. Fortauza, Ceara, Brazil, 2181-2185.