A Systematic Literature Review of Recent Trends in Replication Technique

Albandari Alanazi, Almetwally Mostafa and Abeer Alnuaim

December 16, 2019

A Systematic Literature Review of Recent Trends in Replication Techniques

Albandari L. Alanazi
Department of Information Systems, King Saud University
Riyadh, Saudi Arabia
439203801@student.ksu.edu.sa

Almetwally M. Mostafa [1]
Department of Information Systems, King Saud University
Riyadh, Saudi Arabia
almetwaly@ksu.edu.sa

Abeer A. Alnuaim
College of Applied Studies and Community Services ,
King Saud University, Riyadh,
abalnuaim@ ksu.edu.sa

***Abstract-* Nowadays most software systems manage a huge amount of data. The clients depend heavily on these data and they expect the data to be available at all times. In order to use and manage these data in an efficient way to ensure availability, data replication technique is used. So far, three basic models for replication are exist with their variants. This paper reviews these three basic models of replication techniques and their variants regards to how the load is distributed among replicas, what is the total throughput for these set of replicas, and which type of consistency models is supported by them.**

Key words- Replication, Paxos, Primary Backup Replication (PBR), Chain Replication (CR).

I. INTRODUCTION

Replication is one way to provide the availability and tolerate the failure of the system. It is defined as the process of copying data over multiple servers by using replication techniques, in order to increase the data availability [1]. Replication is classified into two approaches. One approach of replication is called active replication, in which the client request is executed by all non-faulty replicas in the same order. On the other hand, in the passive replication approach; one of the replicas is considered as primary and the others are backup replicas. The client request is executed at the primary, and the state changes are propagated to all replicas. The system with replication needs to maintain data consistency among replicas. Therefore, a trade-off between consistency and performance must be considered. Some system requires strong consistency with replication, and this is achieved when all replicas must be identical to each other. On the other hand, weaker models for consistency, such as eventual consistency, allow the replicas to diverge [2].

The objective of this paper is to review three basic models of replication techniques used in distributed systems as well as their variants including the supported consistency model, their throughput, and the distribute load among replicas. [1]

This paper is organized as the following: First, in section 1, we discuss the basic classic Paxos with its variants. Then, the second model of replication is about primary backup and two improvement approaches is defined in section 2. After that, chain replication and a set of variants models are reviewed in section 3. Finally, a compression between three replication techniques is presented in the last section.

1. CLASSIC PAXOS

Paxos [1], [3], [4] is one of the oldest algorithms that was invited in the middle of 80's. It is mainly used for solving the distributed consensus in order to ensure all replicas are consistent. The Paxos algorithm guarantees that all non- faulty replicas choose a single value among the proposed values.

In this model, the processes are classified based on their roles: proposer, acceptor, and learner. These processes are communicated with each other by exchanging messages. And each one of them can play different roles They are defined as follows: *Proposers:* to suggest a value to be chosen; *Acceptors:* to agree which value to choose; *Learners:* to learn which value was chosen; *Coordinator/Leader:* at each round, one of the Proposers is chosen to be the "distinguished" one and acts as coordinator to allow a new round to start with a new ID and ensure that no conflicts happen.

Classic Paxos relies on one coordinator/leader to starting the round and manage the consistency. Therefore, there will be high loads and a bottleneck on a single server (coordinator) especially when the number of nodes is large. In addition, it suffers from poor utilization of resources, so the throughput is limited to a single server. Also, it requires that the

[1] A. M. Mostafa
Systems and Computers Department,
Faculty of Engineering, Alazhar University, Cairo, Egypt.

leader must hear from the majority for both phases and this might slow reaching the decision and increase the latency and decrease the performance. And in case of the failed of the majority, Paxos cannot reach to decision. *In the following subsections, we will review several variants of Paxos.*

1.1 Raft

The original Paxos algorithm was quite difficult to understand and consequently a lot of works has been done to explain it in an easier way. Different variants of Paxos algorithm have been proposed over the years to make the classic Paxos more efficient as mentioned before. On the other hand, Raft [5], [6]was invited by Diego Ongaro and John Ousterhout to be an alternative to Paxos. The main contribution of Raft is to improve the understandability by finding a straightforward consensus algorithm. It is considered as efficient as Paxos but with different structure.

The consensus is implemented as the following first a leader must be elected by leader election mechanism. After that, the elected leader has to manage the replicated log, and all clients request (command) are processed by that leader. The received command will be appended to the leader's log as a new entry. The entry will be sent in parallel to all other servers in order to replicate it to their logs. Once the process of replicated the entry is done by majority of the servers, the command is committed, and the leader notifies the client about the result.

Raft can be compared to the basic Paxos. First, the performance of Raft is better than Paxos. Raft need only one round trip in order to disseminate the new entry to other servers. While Paxos needs two round trips. So, this results in less latency than in Paxos. Second, all of them use a leader election protocol. In Raft the protocol is separated from the consensus algorithm. However, the leader election in Paxos is considered as an important part of the consensus algorithm. As a result, Paxos requires more mechanism than Raft.

1.2 Mencius

Mencius[7] is an algorithm that is derived from Paxos. It is proposed to solve leader bottleneck and, to achieve high throughput when the clients load is high and low latency when clients load is low.

In Paxos, only the leader in each round can propose values. While in the Mencius, all servers can take turns for proposing values, and this is done through pre-partitioning consensus instance among replicas. So, it has better utilization of resources and load balance. As a result, throughput will be increased. In addition, it will reduce the latency, because a local server can be used as the leader (coordinator) for the client request.

The drawback of this approach is that in order to committee the command, the coordinator of this command must gather information from all other replicas in the system. So, in case of the failure of response from any replica, then coordinator cannot make any progress so this will result a poor performance.

1.3 Egalitarian Paxos

EPaxos[8], [9] is an improvement of Paxos that has no central leader process. In this approach, a client can send the request at any replica. This enables that all replicas can act as proposers at the same time, so better utilization of resources is provided. When the replica receives the request s, it will act as the leadership of that command and it is responsible to collect dependencies from other nodes which are conflicting command with s . And these conflicting commands is ordered using a graph-based mechanism to represent the dependencies. So, the request can be committed after communicating with subset of replicas. Hence, EPaxos requires fewer messages to process than Mencius, so its latency is lower, and its throughput is commonly higher.

While this approach has advantages over Mencius, but it needs an expensive computation in case with a graph that has complicated dependency patterns . Consequently, both of two previous models improves the performance and enable load balancing but they need to exchange dependency among nodes, and this is considered a costly process since it requires a higher bandwidth. In addition, reaching the decision will be slow if the size of the system is increased because the fact that multiple replicas may produce conflict transactions.

1.4 M^2 Paxos

M^2 Paxos [10], [11] is an algorithm presented to overcome the drawback of keeping track of dependencies (conflicting command) as in EPaxos. In M^2 Paxos, the commands can be decided without the need of exchanging dependencies or a designed leader. Instead, it requires that the node must have the ownership of all objects for command c.

Three cases within this protocol: If the proposer node (p) has the ownership of c, then (p) can execute the command. If the proposer node (p) of the command does not have the ownership of c, but other node in the system does have then M^2 Paxos will forward c to that node. Otherwise, the node that proposed a command must first acquire the ownership of all objects in c.

Leader election protocol is used with M^2 Paxos for changing the ownership of objects if the owner is failed. As a result, this effects on the availability of the system i.e. if the node fails its owned objects will become unavailable until electing a new node to takes the objects ownership.

1.5 A generalized consensus

Finally, in 2019, a generalized consensus algorithm[12] is proposed as an improvement of Paxos. The aim of this solution is to improve the performance of Paxos by handling its limitation. This improvement is done by changing some requirements of classic Paxos regarding to quorums intersection, quorums agreement(value) and epochs (proposal ID). As a result, the algorithm provides a flexibility of choosing the trade-off depends on the system needs.

In the proposed approach, each node acts as a proposer has a stable table in which contains the set of registers and the quorums is defined. While in Paxos, the majority is required for all registers set to decide the value. So, less participant is involved for deciding the value in the generalized consensus. As a result, it guarantees less latency than Paxos.

In both algorithms, the epochs associated with each value must be unique. This is achieved in class Paxos by a prior allocation of epochs among the proposers or by voting on a unique epoch. On the other hand, the proposed algorithm overcome these requirements by introducing three mechanism for selecting epochs. So, it provides more flexibility mechanism for assigning epoch value.

In summary, as shown in the table (1), Paxos rely on a single leader at all time to ensure strong consistency. This introduce an important consequence which is limited throughput and scalability to only one server, and as a result high load on that server. Other variants of Paxos algorithms that eliminate the bottleneck created by the unique leader by allowing multiple replicas to act as leaders at the same time such as Mencius, Egalitarian Paxos. As a result, this improves resource utilization since the load is distributed among all replicas. M^2 Paxos has the same advantages of Mencius, Egalitarian Paxos, and additionally a better performance is guaranteed since no need for collecting and exchanging conflict commands. Finally, Raft is considered as understandable consensus algorithm and alternative to Paxos.

Table 1 Important Information about Paxos and its variants.

	Classic Paxos	**Mencius**	**EPaxos**	**M^2 paxos**
Mini. comm. delay for learning a value	4	2	2	2
Leader approach	Single leader	Multi-leader	Multi-leader	Exclusive ownership

2. PRIMARY BACKUP REPLICATION

Primary backup[13], [14] was first proposed in 70's as a replication model used in most practical systems that require high availability and consistency. One server is considered to be the primary server and all the other backups. In this model, the primary server is exclusively responsible for processing all requests from the client. When the primary receives a request from the client, first it has to check the request type. If the request is a query (read), the primary server processes the request directly without communicating with the backups. However, if the received request is update (write), the primary needs first to communicate with every backup node. This is done by disseminating the update request to all replicas (backups) in parallel. Then, the primary has to await acknowledgments from all backups. After receiving acknowledgments, a reply (write notification) is sent to the client.

Primary backup replication uses a master coordination for failure detection, and a leader election algorithm for recovery. So, in case of the primary failure, one of the backups is elected to be the primary. The primary backup model can tolerate f failure with f+1 replica.

Since the primary server exclusively processed all types of requests, strong consistency among distributed objects can be provided. On the other hand, this leads to imbalanced load distribution and limits the system scalability and throughput to only a single server (primary). In addition, it may cause a bottleneck of the primary server. Also, in case of a failure of the primary, the availability of the system and data accessibility are lost until electing a new primary. On the other hand, under weak consistency model, any replica can handle the query request. As a result, read throughput is increased, and not limited to only the primary server. So, read throughput in this case is equivalent to the total number of servers in the system. *The following subsections review some variants of PBR.*

2.1 Object Ownership Distribution

Object ownership distribution (OOD)[15] is an enhancement on primary backup protocol in order to overcome some of its limitation. It is considered as a static logical partition in which the ownership of the objects is distributed between the replicas. Unlike primary backup model, in which the primary node is exclusively responsible for maintaining the objects consistency among replicas. While in OOD all replicas are participated to maintain the consistency. As a result, this leads to better utilization of resources as well as load balancing.

In OOD, each replica owns subset of objects at which they are created by the client. The ownership table is used in this model in order to keep the owner information such as the object ID associated with the owner ID. This table is replicated at each replica and updated if there is a new creation of object, so every replica knows about its owned objects as well as the owners of other objects in the system. Any updates associated with item should be authorized by the owner's replica and it has to control all updated request of its owned objects. In this model, as a result, the owner ensures the consistency of the set of objects it owns.

OOD can be compared with PBR, in OOD model a better resource utilization is provided because the fact that OOD uses active replication instead of passive replication as in PBR and this leads to higher throughput than PBR. Throughput is not limited to one server (primary) as in the PBA. In addition to that, availability is increased because there is no leader failure like in primary backup approach So, only subset of data that owned by failed replica will become unavailable.

2.2 Dynamic Health-based Objects Ownerships Distribution Protocol (DHOOD)

In the previous model (OOD), the partitioning of the objects among replicas is static in which the objects distribution depends on the object creation. That's mean, this approach doesn't consider the differentiation between nodes regarding their capabilities such as some of resources are poor while other resources are powerful. This might lead to waste of resources either underutilization or overutilization.

DHOOD [16] is an improvement on OOD in which the redistribution of objects' ownerships is done according to replica's health. The proposed model has the same architecture and additionally in has one module for evaluating the health of replica (HE) and another module for redistributing the object's ownership (OM). The replica's health is assessed based on its resources including CPU, memory, and network bandwidth. In this protocol, the redistribution process is conducted in three steps. First, health evaluation is done by HE module. In this step, a health tabled is maintained to evaluate the health level of resources in the system. So, replica's health is evaluated regularly. And any updates associated with that table will be notified to all other replicas. Then, object ownership transformation; in this step OM module is responsible to defines the number of objects that should be transferred to other replicas based on the replica's health. Finally, the ownerships will be distributed among other replicas in the system according to their health level (from lower level to higher level transformation).

In this section so far , we conclude that the primary backup replication is one of the passive replication techniques that suffers from poor utilization of resources. As a result, the performance is considered low due to the restriction of throughput to a single server (primary). Moreover, in case of leader failure, throughput drops to zero until a new leader is elected. While OOD and DHOOD which are variants of PBA there is no single point of failure since the load is distributed among replicas. Furthermore, they have better utilization of resources, and higher throughput than PBA.

3. CHAIN REPLICATION

CR has been proposed by Van Renesse and Schneider [17], [18] in 2004 as a variation on primary backup replication to be used mainly with storage system. . All replica nodes are serially ordered to form a chain. In chain replication, the role of request execution is shared by two replicas.

CR works as follow: The first node in the chain is called the Head (H), which is responsible for handling write operations, and prorogating the updates to the next replica of the chain until the request reaches the tail. while the last node is called the Tail (T) receives query operations and reply generation. So, all updates and query requests are processed at the tail. As a result, strong consistency is guaranteed.

The most important advantage of CR is better utilization of computing resources comparing with other models. In chain replication, if the chain has n servers , we can get (n) physical chains. The following table (2) shows the possible combinations of physical chains with different roles of servers (head of the chain, tail of the chain, and middle) . Also, it has higher read throughput because total throughput in CR is equivalent to two servers which are the head and the tail. Also, it provides lowest latency than other models for read operations

Table 2 list of physical chains with n servers in CR.

Chain #	**S1**	**S2**	**S3**	**S4**
Chain 1	Head	middle	middle	Tail
Chain 2	middle	Head	Tail	middle
Chain 3	middle	Tail	Head	middle
Chain 4	Tail	middle	middle	Head

Chain replication handles server failures well due to its serial chain structure. It needs a single service for fault tolerance coordination, which is called the master and is responsible for detecting failure, removing the failed server, adjusting the successor and predecessor for each sever and informing the client if there is a new head or tail.

In chain replication, all updates are disseminated serially to other replicas before reply generation. This results a high latency for update request, and it needs n+1 latency when there are n servers to reply write notification to the client.

The problem with chain replication is load balancing under strong consistency model, especially for query-intensive applications. Since all query requests and reply notifications are processed by the tail, read throughput will be limited to a single node. This might cause a bottleneck for the tail. While under weak consistency model, a query request can be handled at any replica, but this leads to read a stale value. Moreover, communication and computing resources of the chain are not fully utilized because the data are transferred and processed in only one direction (from the head to the tail). *In the following subsections CRAQ and BCR which are variants of CR will be reviewed.*

3.1 Chain Replication with Apportioned Queries (CRAQ)

CRAQ [19] is an improvement on chain replication in which the queries are apportioned, in order to provide lower latency and higher throughput for query requests. Apportioned queries are done through splitting read request between replicas in the chain and any replica can carry out query request with guaranteeing strong consistency. The architecture of CRAQ is the same as chain replication, except that it allows the handling of read requests by all replicas not only the tail like in chain replication. As a result, it eliminates the possibility of making a hotspot on the tail. The fault-tolerance and recovery mechanisms that are applied in CRAQ are the same as the basic chain replication model.

This approach can be compared with the basic model of chain replication. Since the read requests can be processed by all replicas in the chain for CRAQ, throughput increases for query requests and hotspots are reduced on the tail in the case of weak consistency model (eventual consistency) approach. Also, this approach better utilizes resources since the intermediate nodes can process query requests, while in chain replication these nodes are only responsible for propagating the updates.

On the other hands, some limitations are present in this approach. First, in the case of strong consistency, when the request query is forwarded to a replica with dirty version, this replica needs to communicate with the tail in order to get a clean version, so this is not aligned with the main contribution of this approach, which is the elimination of the hotspot on the tail. So, this will increase the latency for query requests. Second, reply notification for update requests are done by the tail in chain replication, while the head replica in CRAQ is responsible for handling this reply. This means that after the tail receives the updates, the acknowledgment will be propagated back until it is received by the head, and then the head will respond to the client. Consequently, the updates latency is double in CRAQ what it is in CR.

3.2 Bidirectional Chain Replication (BCR)

BCR [20] is an improvement on chain replication it tries to gain higher throughput through better utilization of network resources including computing and communication while maintaining consistency. The proposed model, however, doesn’t address fault-tolerance and recovery mechanism.

BCR has the same architecture as CR, but chains run concurrently in opposite directions. The following table (3) shows the possible combinations of physical chains with different roles of servers (head of the chain, tail of the chain, and middle) . So, with n servers, we can get 2N physical chains as each server can play 2N roles; L: denoted that a chain run form left to right, while R: denoted that a chain run from right to left.

Table 3 list of physical chains with n servers in BCR.

Chain #	S1	S2	S3	S4
Chain 1	Head	Middle L	Middle L	Tail
Chain 2	Middle L	Head	Tail	Middle L
Chain 3	Middle L	Tail	Head	Middle L
Chain 4	Tail	Middle L	Middle L	Head
Chain 5	Middle R	Middle R	Tail	Head
Chain 6	Middle R	Middle R	Head	Tail
Chain 7	Tail	Head	Middle R	Middle R
Chain 8	Head	Tail	Middle R	Middle R

So, in BCR there are two logical partitions for splitting the replicated data at each server, and two heads and two tails can process update requests and query request, respectively, at the same time. Thus, this leads to higher throughput since four requests can be achieved concurrently without affecting consistency.

One limitation of BCR is about the distribution of requests among two partitions, because some data objects may belong to either of the partitions which are frequently requested.

In summary, chain replication with its variants including CRAQ, BCR have better utilization of resources comparing with PBR, since the load is not limited to one server(primary) as in PBR. In addition, throughput for both update and query request are higher than PBA. So , they provide better performance.

II. DISCUSSION& CONCOLUSION

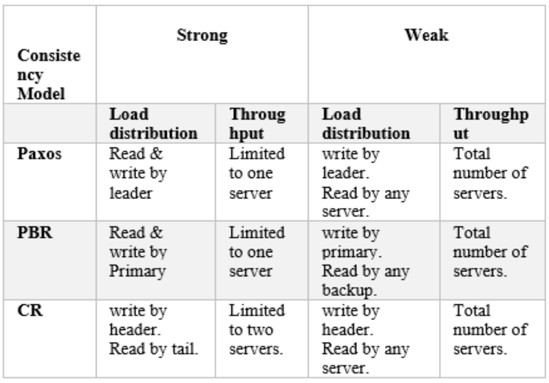
In this paper, a systematic review about data replication techniques in distributed systems has been presented.

A common challenge in the three-replication technique is that they suffer from poor utilization of computing resources. In addition, they have low performance due to the restriction of throughput to a single server as in Paxos and PBR or two servers as in CR under strong consistency model. Also, in case of leader/coordinator failure, throughput drops to zero until a new leader is elected such as Paxos and PBR. Moreover, in case of the failure response from one server in the system , this will negatively effect on the performance. In addition, the process of extension or configuration of adding a new server while the system is running is considered as challenges and difficult task.

In general, we can compare these models with the client/server model. Client server model suffers from unavailability in case of failure, but it provides better performance comparing with replication models since a response is completed with one round trip. Replication models under strong consistency behaves differently in order to response , Paxos requires three round trips while PBR needs only two round trips. In chain replication under strong consistency when there are (n) servers, the clients have to wait (n+1) latency to get the reply notification. Nevertheless, under weak consistency, Paxos, PBR, and CR needs only one round trip to provide the response which is the same performance as in the client server model.

The following table (4) summarizes consistency model ,load distribution and throughput for each basic replication model.

Table 4 comparison between three basic models for replication.



Consistency Model	Strong		Weak	
	Load distribution	**Throughput**	**Load distribution**	**Throughput**
Paxos	Read & write by leader	Limited to one server	write by leader. Read by any server.	Total number of servers.
PBR	Read & write by Primary	Limited to one server	write by primary. Read by any backup.	Total number of servers.
CR	write by header. Read by tail.	Limited to two servers.	write by header. Read by any server.	Total number of servers.

III. REFRENCES

[1] L. Lamport, "Paxos Made Simple," in *ACMSIGACT News*, 2001, vol. 32, pp. 18 - 25.

[2] Jay Kreps .., "Distributed Systems for fun and profit." [Online]. Available: http://book.mixu.net/distsys/index.html. [Accessed: 15-Oct-2019].

[3] R. van Renesse, "Paxos Made Moderately Complex," *ACM Computing Surveys (CSUR)*, vol. 47, no. 3, pp. 1–16, 2015.

[4] W. J. Bolosky, D. Bradshaw, R. B. Haagens, N. P. Kusters, P. Li Microsoft, and M. Research, "Paxos Replicated State Machines as the Basis of a High-Performance Data Store," 2011.

[5] D. Ongaro and J. Ousterhout, "In Search of an Understandable Consensus Algorithm (Extended Version)," in *USENIX*

ATC'14 Proceedings of the 2014 USENIX conference on USENIX, 2014, pp. 305–320.

[6] H. Howard, "ARC: Analysis of Raft Consensus," 2014.

[7] Y. Mao, F. P. Junqueira, and K. Marzullo, "Mencius: Building Efficient Replicated State Machines for WANs," *8th USENIX Symposium on Operating Systems Design and Implementation*, pp. 369–384, 2008.

[8] I. Moraru, D. G. Andersen, and M. Kaminsky, "There is more consensus in Egalitarian parliaments," in *SOSP 2013 - Proceedings of the 24th ACM Symposium on Operating Systems Principles*, 2013, pp. 358–372.

[9] P. Bailis, A. Davidson, A. Fekete, A. Ghodsi, and J. M. Hellerstein, "Highly Available Transactions: Virtues and Limitations," *Proceedings of the VLDB Endowment*, vol. 7, no. 3, pp. 181–192, 2013.

[10] R. Palmieri, "Leaderless consensus: The state of the art," in *Proceedings - 2016 IEEE 30th International Parallel and Distributed Processing Symposium, IPDPS 2016*, 2016, pp. 1307–1310.

[11] V. Tech and R. Palmieri, "Making Fast Consensus Generally Faster Sebastiano Peluso Alexandru Turcu Giuliano Losa Binoy Ravindran," in *Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2016, pp. 1–31.

[12] H. Howard and R. Mortier, "A Generalised Solution to Distributed Consensus," Feb. 2019.

[13] N. Budhiraja, K. Marzullo, F. B. Schneider, and S. Toueg, "Chapter 8: The Primary Backup Approach," New York , USA.

[14] L. Lamport, D. Malkhi, and L. Zhou, "Vertical Paxos and Primary-Backup Replication," 2009.

[15] A. M. Mostafa and A. E. Youssef, "Improving Resource Utilization, Scalability, and Availability in Replication Systems Using Object Ownership Distribution," *Arabian Journal for Science and Engineering*, vol. 39, no. 12, pp. 8731–8741, Nov. 2014.

[16] S. Albassam and A. M. Mostafa, "Dynamic Health-based Objects Ownerships Distribution Protocol (DHOOD)," in *Sixth International Conference on Digital Information Processing and Communications (ICDIPC)*, 2016, pp. 13–18.

[17] R. van Renesse and F. B. Schneider, "Chain Replication for Supporting High Throughput and Availability," in *USENIX Symposium On Operating Systems Design and Implementation (OSDI04)*, 2004, vol. 4, pp. 91–104.

[18] S. L. Fritchie, "Chain Replication In Theory and in Practice Working Title, rough draft," 2010.

[19] J. Terrace and M. J. Freedman, "Object Storage on CRAQ High-throughput chain replication for read-mostly workloads," 2009.

[20] A. M. Mostafa, A. E. Youssef, and Y. A. Aljarbua, "Bidirectional chain replication for higher throughput provision," *KSII Transactions on Internet and Information Systems*, vol. 13, no. 2, pp. 668–685, Feb. 2018.