Bidirectional Chain Replication for Higher Throughput Provision

Almetwally M. Mostafa[1,2], Ahmed E. Youssef[1,3] and Yazeed Ali Aljarbua[1]
[1] College of Computer and Information Sciences, King Saud University
Riyadh, Saudi Arabia
[e-mail: almetwaly@ksu.edu.sa, ahyoussef@ksu.edu.sa, yaljarboa@ksu.edu.sa]
[2] Faculty of Engineering, Alazhar University
Cairo, Egypt
[3] Faculty of Engineering at Helwan, Helwan University
Cairo, Egypt
*Corresponding author : Ahmed E. Youssef

Received February 16, 2018; revised May 15, 2018; accepted June 10, 2018; published February 28, 2019

Abstract

Provision of higher throughput without sacrificing consistency guarantees in replication systems is a critical problem. In this paper, we propose a novel approach called Bidirectional Chain Replication (BCR) to improve throughput in traditional Chain Replication (CR) through better utilization of computing and communication resources of the chain. Unlike CR where the whole replicated data store is treated as a single unit, in BCR the replicated shared data at each server in the chain is split into two disjoint Logical Partitions (LP_1, LP_2). This forms two chains running concurrently on the same hardware in two opposite directions; the first chain (CR_1) exclusively manipulates data objects in LP_1, while the second chain (CR_2) exclusively manipulates data objects in LP_2, therefore, conflict is avoided and concurrency is guaranteed. The simultaneous employment of these two chains results in better utilization of hardware in the sense that the two chains can evenly share the workload, hence, throughput can be improved without sacrificing consistency. Experimental results showed an improvement of approximately 85% in throughput of BCR over CR.

Keywords: Chain Replication, Consistency, Throughput, Data Partitioning, Distributed Systems.

This work is supported by the Research Center of College of Computer and Information Sciences (CRC) at King Saud University. The authors are grateful for this support.

1. Introduction

The fact that databases are increasingly deployed by various distributed systems over the recent years, has caused a dramatic growth in the importance of data replication. Replication is the process of copying data in one server to other (n-1) servers (i.e., replicas) to provide high availability and fault tolerance. In replication systems, as many as (n-1) servers can fail without compromising data availability. In addition, client request processing proceeds in case of server failure. However, maintaining data consistency represents a critical challenge in these systems. Consistency guarantees assert that operations to query and update individual objects are executed in some sequential order and the effects of update operations are necessarily reflected in results returned by subsequent query operations [31 and 32]. In order to ensure that all data ends up on all replicas, every write (update) request to the data needs to be processed by every replica in its local data store, otherwise the replicas would no longer contain the same data.

Strong consistency guarantees are often thought to be in conflict with high throughput and availability [23, 32 and 34]. Hence, many replication systems sacrifice throughput or availability to support strong consistency guarantees. For example, Paxos [15-18 and 26] and Primary Backup Replication (PBR) [11] employ a single server (i.e., leader) to ensure that consistency and serialization are applied all the time. Accordingly, during efficient, failure free operations, all clients communicate with the single leader at all times. This limitation has an important consequence since it reduces throughput by placing a disproportionality high load on the leader, which must process more messages than the other replicas [28 and 30]. Moreover, in case of leader failure, throughput drops to zero until a new leader is elected. Hence, the throughput of the replication cell is limited by the performance of the leader. In other words, the employment of single leader technique to maintain consistency causes a problem of inefficient utilization of the available servers in the replication cell and imbalanced load distribution among these servers. This problem, in turns, results in performance degradation.

Chain replication (CR) [32-36] was first proposed by Renesse et. al. [32] to improve throughput in replication systems while maintaining consistency. They showed that in a large-scale storage system, maintaining strong consistency is not in conflict with achieving high throughput and availability. The chain consists of four linearly ordered servers, the first server is called “the head” and the last server is called “the tail”. The main idea is to classify client requests into two different classes, the write-requests and the read-requests. The head is assigned to exclusively process the write (update) requests over the entire replicated data, while the tail exclusively performs the read (query) requests over the whole replicated data store. In CR, workload (update and query processes) is divided between the head and the tail, resulting in better resource utilization and an improved throughput. To maintain consistency, CR employs a single writer (head) and a single reader (tail), consistency is guaranteed since read requests are processed only by the tail. Although CR simultaneously supports strong consistency and high throughput, it suffers from some limitations. Firstly, communication and computing resources of the chain are not fully utilized since the data are transferred and processed in only one direction (i.e., from the head to the tail). Secondly, the replicated data are still treated as a single unit with coarse-grained.

The objective of this research is to improve throughput in CR without sacrificing consistency through better utilization of computing and communication resources. To achieve this goal, we propose a novel approach called Bidirectional Chain Replication (BCR). Like CR, BCR consists of four linearly ordered servers connected to form a chain, thus, CR and BCR have the same hardware. In contrast to CR where the replicated data store on each server are treated as a single unit, in BCR the replicated data on each server in the chain are split into two disjoint Logical Partition (LP_1, LP_2). BCR forms two chains; the first chain (CR_1) exclusively manipulates data objects in LP_1 and the second chain (CR_2) exclusively manipulates data objects in LP_2, therefore, conflict is avoided and the two chains can work concurrently on the same hardware in two opposite directions. The simultaneous deployment of these two chains results in better utilization of hardware in the sense that the two chains can evenly share the workload, therefore, throughput can be improved.

Experimental performance evaluation of BCR showed an improvement of approximately 85% in throughput over traditional CR. Practically, we can merge every two physical partitions used by CR into a single physical partition in BCR which reduces the number of the needed physical partitions using the same set of servers in a datacenter. The main contribution of this work is to provide a method for attaining higher throughput in replication systems through better utilization of computing resources without sacrificing consistency.

The rest of this paper is organized as follows: in Section 2, we briefly review essential concepts related to chain replication. In Section 3, we review related work. In Section 4, we describe our proposed BCR approach in detail. In Section 5, emprical evaluation for BCR is described and the results are analyzed and interpreted. Finally, in Section 6, we give our conclusions and suggestions for future work.

2. Chain Replication

Renesse et. al. [32] proposed Chain Replication, CR, an approach that simultaneously supports high throughput, availability, and strong consistency in large-scale storage service. In this approach, the servers replicating data are linearly ordered to form a chain, as shown in **Fig. 1**, the first server in the chain is called “the head” and the last server is called “the tail”. Request execution is implemented by the servers roughly as follows: each update-request is directed to the head and executed at its local store, then the state changes are forwarded along a reliable FIFO link to the next server of the chain which updates its local data store and forwards the changes to the next server and so on until the changes reach the tail. At this point, the tail sends a reply (write-notification) to the client and the write finishes. An update acknowledgement is generated at the tail and is sent along the chain until it reaches the head. Query requests are directed to the tail and processed there automatically. The reply for every query request is generated by the tail and sent to the client. Since all the values stored in the tail are guaranteed to have been propagated to all replicas, reads are always consistent [36].

Unlike Paxos and Primary Backup Replication (PBR) where all decisions for update and query requests and their replies are made by a single server (i.e., the leader), CR deploys a better resource utilization procedure to improve throughput. The workload is distributed among the head and the tail by splitting client requests into updates (writes) which are processed by the head and queries (reads) which are processed by the tail, the two servers (head and tail) work concurrently. However, communication and computing resources are not fully utilized since the data are transferred and processed in only one way (i.e., from the head to the tail).

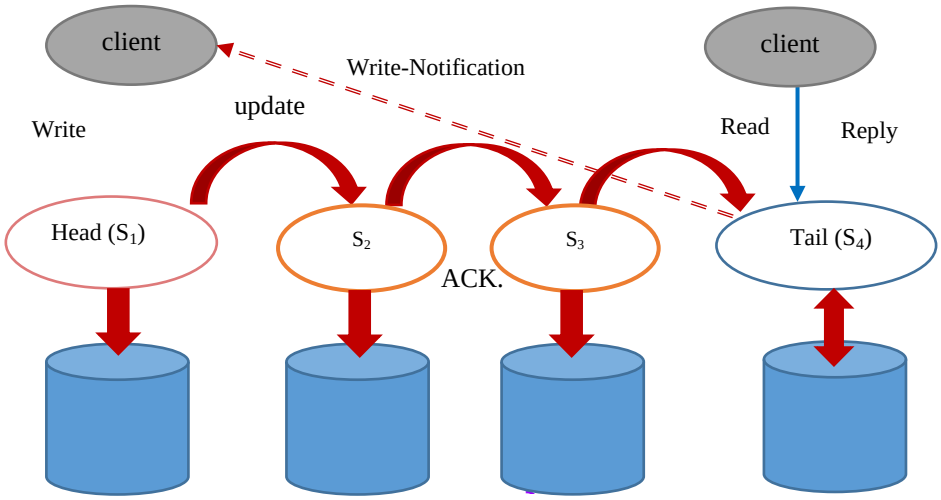

Fig. 1. Chain Replication

3. Related work

A wide variety of replication techniques has been introduced in literature [1-4]. Generally speaking, replication techniques can be classified into active replication and passive replication. In active replication [5-7], all replicas execute all client requests in the same order, assuming that the process hosted by the replicas are deterministic. Deterministic means that, given the same initial state and a request sequence, all processes will produce the same response sequence and end up in the same final state. The Chubby lock service for loosely coupled distributed systems [20], the Spanner, as Google's globally distributed database [21], and PaxStore [19] are examples of services that utilize active replication. The disadvantage of active replication is that in practice most of the real world servers are nondeterministic. In passive replication [8-14], there is one replica, the primary, which executes the client requests and propagates the new states to all other replicas (backup servers). Then, the backups apply updates in the same order sent by the primary. If the primary server fails, one of the backup servers takes its place. Passive replication may be used even for nondeterministic processes. One of the service that sometimes utilizes passive replication is Zookeeper [22]. However, the disadvantage of passive replication compared to active replication is that in case of failure the response is delayed.

Paxos [15-18 and 26] is a widely used active replication technique. It is a consensus protocol that results in an agreement on an order of inputs among a group of replicas, even when the replicas in the group crash and restart or when a minority of them permanently fail. Paxos classifies the processes by their roles in the protocol: Client, Proposer, Acceptor, Learner, and Leader. A single processor may play one or more roles at the same time without affecting the correctness of the protocol. The Client issues a request to the proposer, and waits for a response. The proposer promotes the client request, attempting to convince the acceptors to agree on it. Learners act as the replication factor for the protocol. Once a client request has been agreed on by the majority of the acceptors, the learner may execute the request and send a response to the client. The leader is a distinguished proposer that acts as a coordinator to move the protocol forward when conflicts occur.

The Primary-Backup Replication (PBR) protocol [11] is massively used in passive replication. In this approach, one server, the primary, perform the following jobs [32]: 1) imposes a sequencing on client requests to ensure that strong consistency holds, 2) executes locally the client requests, 3) distributes to other servers (backups) the client requests resulting updates, 4) awaits for acknowledgement from all non-faulty backups, and 5) send a reply to the client after receiving those acknowledgements. If the primary fails, one of the backups is elected to that role.

Both Paxos and PBR employ a single server (leader/primary) to manage consistency and serialization. Accordingly, during efficient, failure free operations, all clients communicate with the single leader at all times. This limitation has an important consequence since it impairs throughput and scalability by placing a disproportionality high load on the leader, which must process more messages than the other replicas [28 and 30]. Moreover, in case of leader failure, throughput drops to zero until a new leader is elected. Several approaches have been proposed to resolve the issue of a single leader bottleneck. Their main objective is to improve replication cell performance by distributing consistency management responsibilities among servers. These approaches include different variants of Paxos such as Multi-Paxos [16 and 17], Fast Paxos [27], Mencius [28], generalized Paxos [29], EPaxos [30], Object Ownership Distribution (OOD) [13], and chain replication [32-36].

Chain replication (CR) is intended for supporting large-scale storage services that exhibit high throughput and availability without sacrificing strong consistency guarantees. In chain replication, the primary's role in sequencing requests is partitioned between two servers. The head sequences update requests; the tail extends that sequence by handling query requests. This sharing of responsibility enables lower-latency and lower-overhead processing for query requests, because only the tail is involved in processing a query and that processing is never delayed by activity elsewhere in the chain. This is contrast to the primary backup approach, where the primary must await acknowledgements from backups for prior updates before responding to a query. In both approaches, update requests must be distributed to all servers replicating an object otherwise the replicas will deviate. Chain replication does this broadcasting serially, resulting in higher latency than the primary/backup approach where updates are disseminated to backups in parallel. With parallel dissemination, the time needed to generate a reply is proportional to the maximum latency of any non-faulty backup; with serial dissemination, it is proportional to the sum of those latencies [32]. Chain replication exhibits a higher latency than multicast-based replication solutions but, on the other hand, it is extremely resource efficient and, therefore, it has been adopted in several practical systems. FAWN-KV [39] and Hyperdex [40] are two data stores that offer strong consistency using chain-replication as the main replication technique. BCR extends CR to achieve better utilization of computing and communication resources and to gain higher throughput.

The work in [34] proposed a novel datastore design, named ChainReaction. The proposed solution relies on a novel variant of chain-replication that offers the causal+ consistency criteria [37and 38] and is able to leverage the existence of multiple replicas to distribute the load of read requests. As a result, ChainReaction avoids the bottlenecks of linearizability while providing competitive performance when compared with systems merely offering eventual consistency. ChainReaction can be deployed either on a single datacenter or on Geo-replicated scenarios, over multiple datacenters. Additionally, ChainReaction provides a transactional construct that allows a client to read the value of multiple objects in a causal+ consistent way.

The work in [35] presents the design, implementation, and evaluation of CRAQ (Chain Replication with Apportioned Queries), an object storage system that, while maintaining the

strong consistency properties of chain replication, provides lower latency and higher throughput for read operations by supporting apportioned queries: that is, dividing read operations over all nodes in a chain, as opposed to requiring that they all be handled by a single primary node. CRAQ enables any chain node to handle read operations while preserving strong consistency, thus supporting load balancing across all nodes storing an object. Furthermore, when workloads are read mostly, an assumption used in other systems such as the Google File System [24] and Memcached [25], the performance of CRAQ rivals systems offering only eventual consistency. In addition to strong consistency, CRAQ's design naturally supports eventual-consistency among read operations for lower-latency reads during write contention and degradation to read-only behavior during transient partitions. CRAQ allows applications to specify the maximum staleness acceptable for read operations. CRAQ techniques can be directly supported in BCR.

4. The Proposed BCR

4.1 BCR Intuitions and Contributions

The main objective of BCR is to improve throughput in CR without sacrificing consistency through better utilization of computing and communication resources of the chain. As shown in **Fig. 2**, BCR is composed of four uniquely identified servers (S_1, S_2, S_3, and S_4) which are linearly connected to form a chain. To maintain availability, data is replicated on each server, the replicated data is divided into two logical partitions (LP_1, LP_2). In order to maintain data availability in BCR, data is replicated on each server, the replicated data is divided into two logical partitions (LP_1, LP_2). Thus, data redundancy is achieved by this replication. If LP_1 or LP_2 is corrupted on one server, data can be recovered from another server. It is worthy to mention here, that fault tolerance techniques used in traditional CR are valid in BCR, hence, it is out of scope of this work. On server level, the first server in BCR, S_1, (formerly the head in CR) has the exclusive right to concurrently write to the LP_1 and read from the LP_2. Conversely, the last server in BCR, S_4, (formerly the tail in CR) has the exclusive right to concurrently write to the LP_2 and read from LP_1. Thus, S_1 employs two processes: H_1 to exclusively write to LP_1 and T_2 to exclusively read from LP_2. Conversely, S_4, employs two processes: H_2 to exclusively write to LP_2 and T_1 to exclusively read from LP_1. The inverse assignment of the read and write operations on S1 and S_4 with respect to the logical partitions is motivated by attaining concurrent operation of S1 and S_4 without conflict. Server, S_2, writes on LP_1 when triggered by H_1 and writes to LP_2 when triggered by S_3. On the contrary, server, S_3, writes on LP_1 when triggered by S_2 and writes to LP_2 when triggered by H_2. On chain level, BCR forms two chains running concurrently in two opposite directions (bidirectional chain). The first chain (CR_1) runs from left to right (from S_1 to S_4) and manipulates data objects belonging to LP_1. The second chain (CR_2) runs from right to left (from S_4 to S_1) and manipulates data objects belonging to LP_2. The processes H_1 and T_1 forms respectively the head and the tail of CR_1, while the process H_2 and T_2 forms respectively the head and the tail of CR_2.

From the client point of view, BCR has two servers (S_1, S_4), each one of them can execute both write and read requests, but on two different data slices. This is contrary to traditional CR where S_1 can only write and S_4 can only read. In BCR, a client request manipulating a data object needs to be directed to the partition to which this object belongs (either LP_1 or LP_2). This is illustrated in **Table 1** shown below. From this table, we notice that BCR is able to perform four requests concurrently without affecting consistency. CR_1 is able to write (through H_1) to LP_1 and read (through T_1) from LP_1. Conversely, CR_2 is able to write (through

H_2) to LP_2 and read (through T_2) from LP_2. This represents twice the number of requests that traditional CR can perform concurrently since it can only write through the head and read through the tail. Servers S_1 and S_4 shares write and read workload since S_1 writes to LP_1 and reads from LP_2 while S_2 writes to LP_2 and reads from LP_1. The intermediate servers (S_2, S_3) are able to write to both LP_1 and LP_2. Hence, BCR distributes the workload (write and read requests) on both chains without conflict between the two chains since data on each server is partitioned into two disjoint sets. The concurrent deployment of the two chains provides better utilization of the chain hardware which results in an improved throughput compared to traditional CR.

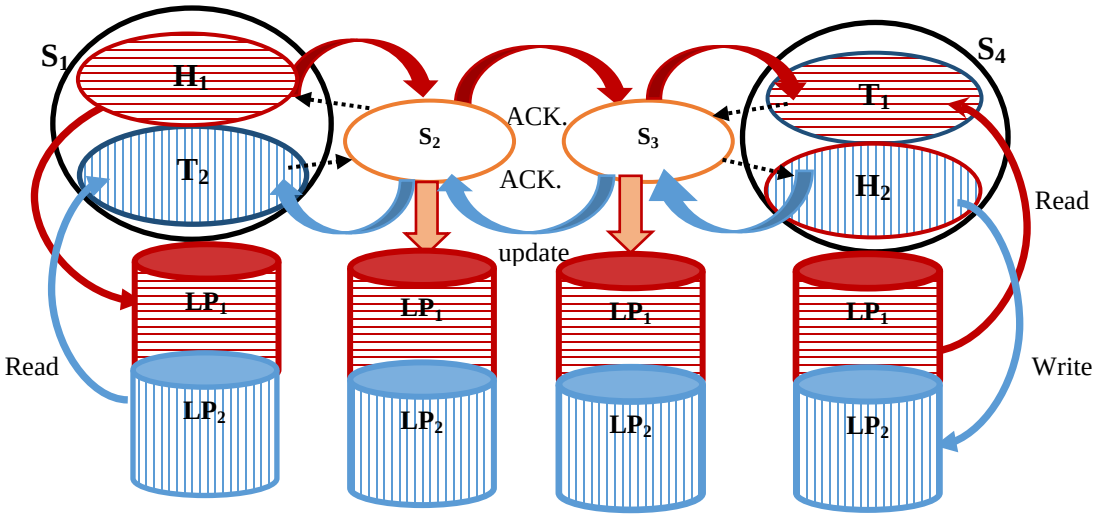

Fig. 2. BCR architecture

Table 1. BCR form the client view

client request	corresponding partition	sever	process	chain
Write to	LP_1	S_1	H_1	CR_1
Read from	LP_1	S_4	T_1	
Write to	LP_2	S_4	H_2	CR_2
Read from	LP_2	S_1	T_2	

4.2 BCR Protocols

The design of BCR is shown in **Figs. 3 and 4**. The first chain (CR_1:$H_1 \rightarrow S_2 \rightarrow S_3 \rightarrow T_1$), shown in **Fig. 3**, consists of a head (H_1 at S_1), two servers (S_2, S_3), and a tail (T_1 at S_4). Client requests targeting data in LP_1 are implemented as follows: each write-request is directed to the head (H_1) and is executed at its local store (LP_1), then the state changes are forwarded along the chain link to the next server (S_2) which updates its local data store (LP_1) and so on until the changes reach the tail (T_1). T_1 updates its local store (LP_1) and sends a write-notification to the client and an update-acknowledgement to S_3. The last acknowledgement is propagated down the chain until it reaches H_1. Each query-request is directed to T_1 which replies directly to the client.

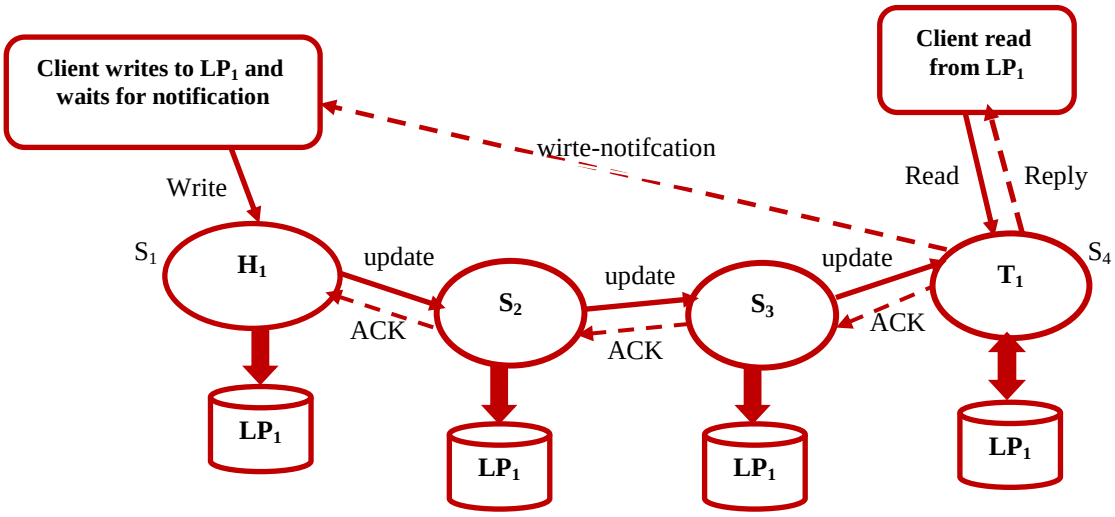

Fig. 3. The First Chain (CR_1)

Similarly, the second chain (CR_2:T_2←S_2←S_3←H_2), shown in **Fig. 4**, consists of a head (H_2 at S_4), two servers (S_3, S_2), and a tail (T_2 at S_1). Client requests targeting data in LP_2 are implemented as follows: each write-request is directed to the head (H_2) and executed at it local store (LP_2), then the state changes are forwarded to the next server (S_3) which updates its local data store (LP_2), and so on until the changes reach the tail (T_2). T_2 updates its local data store (LP_2) and sends a write-notification to the client and an update-acknowledgement to S_2. The last acknowledgement is propagated down the chain until it reaches H_2. Each query-request is directed to T_2 which replies directly to the client.

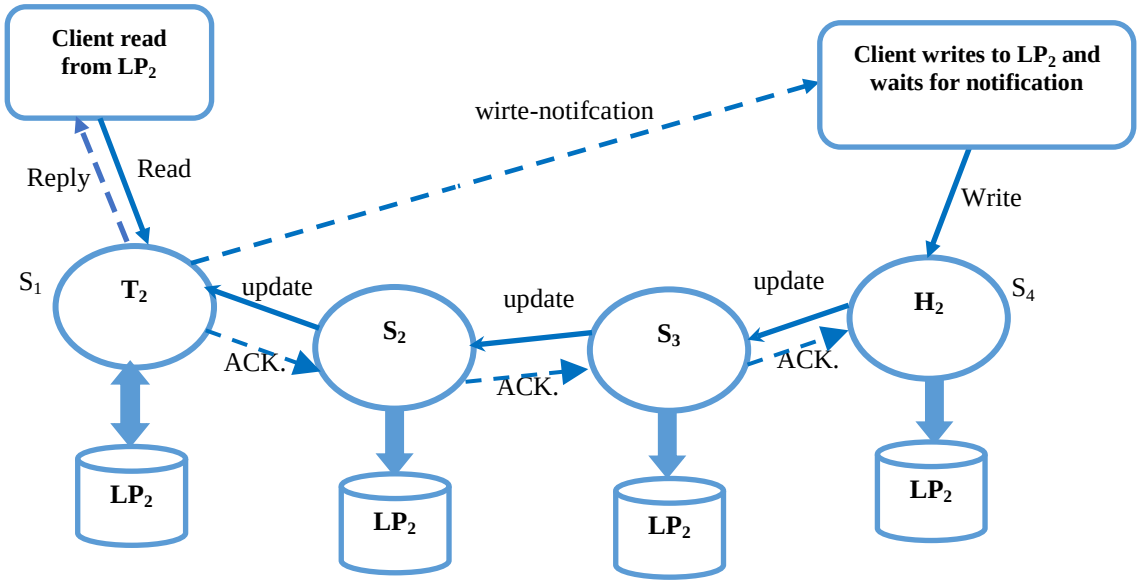

Fig. 4. The Second Chain (CR_2)

The protocol of the client side of BCR is shown in **Fig. 5**. Upon write request, the client has to determine the partition (LP_1 or LP_2) to which the target object belongs. If the object belongs

to LP_1, the request is directed to S_1, otherwise, it is directed to S_4. Similarly, upon read request, the client determines the partition to which the target object belongs. If the object belongs to LP_1, the request is directed to S_4, otherwise, it is directed to S_1. The code implementing the server side is similar the code of traditional CR with minor modifications. We enabled the read process at S_1 and the write process at S_4 to allow both servers to read and write. In addition, we modified the code on all servers to allow update notifications and acknowledgements to transfer in both directions (i.e., from S_1 to S_4 and vice versa).

```
Upon write k do
        Get LP where k belongs
        If k belongs to LP1 then send write request to S1
        Else send write request to S4
End

Upon write k do
        Get LP where k belongs
        If k belongs to LP1 then send write request to S4
        Else send write request to S1
End
```

Fig. 5. Client side read and write protocols in BCR

Finally, it is worth noticing that the mechanisms employed by BCR to recover from failure of a node are similar to those in the original chain replication. There are three types of failures and corresponding repairs [32]:1) Head Failure: when the head (S_1 for CR_1 or S_4 for CR_2) node fails, its successor (S_2 for CR_1 or S_3 for CR_2) takes over as the new head, as it contains most of the previous state of the head. All updates that were in head but were not propagated to it successor are retransmitted by the client proxy when the failure is detected. 2) Tail Failure: when tail node (S_4 for CR_1 or S_3 for CR_2) fails, it is easily recovered by replacing it with its predecessor, (S_3 for CR_1 or S_2 for CR_2). Because of the properties of the chain, the predecessor is guaranteed to have newer or equal state to the failing tail. 3) Failure of a middle node (S_2 or S_3): when a middle node (S_2) fails, the chain is repaired by connecting S_1 to S_3 without any state transfer, however, the two nodes (S_1, S_3) may have to exchange some pending PUT operations that were sent to S_2, but did not arrive to any of them. Similarly, when a middle node (S_3) fails, the chain is repaired by connecting S_4 to S_2 without any state transfer, however, the two nodes (S_4, S_2) may have to exchange some pending PUT operations that were sent to S_3, but did not arrive to any of them.

5. Experiments

In order to evaluate the throughput of BCR vs. the throughput of CR, we have conducted a set of experiments. The experiments were conducted on six Virtual Machines (VM) outsourced from the private cloud of King Saud University. Each VM is an eight-core machine with eight GB memory. Four VMs are dedicated for servers S_1, S_2, S_3, S_4, one VM is used as a master, and the last VM is used to generate client requests. Since we have up to 500 clients, we cannot provide a separate VM for each client program; hence, we created a thread for each client program on the same VM. The object store is replicated at each sever and contains 4000

objects which are evenly divided into two partitions, LP_1 and LP_2, each partition contains 2000 objects.

5.1 Experiments Setup

We have conducted four experiments as shown in **Table 2**. The number of clients in each experiment is 200, 300, 400, and 500 respectively where each client sends 10 requests. The total number of requests sent by the clients in each experiment is 2000, 3000, 4000, and 5000 respectively. Client requests are generated at a rate of 10% writes and 90% reads since this rate is very common in most large-scale applications such as Facebook and Twitter. Another reason for choosing this rate is that the original CR is more suitable for applications with high read rate. However, we have conducted experiments with different rates (i.e., 20% writes-80% reads, 30% writes-70% reads, and 40% writes-60% reads) and have obtained similar results. In all experiments, we measured throughput and execution time for both BCR and CR. The execution time is measured empirically by registering the VM local time at the instant of issuing the first request from a client till the execution of the last request and take the difference between them.

Table 2. Experiment Setup

Exp. ID	No. of clients	No. of requests	No. of write requests	No. of read requests
Exp. 1	200	2000	200	1800
Exp. 2	300	3000	300	2700
Exp. 3	400	4000	400	3600
Exp. 4	500	5000	500	4500

5.2 Experimental Results

Tables 3, 4, 5 and **6** and **Figs 6, 7, 8,** and **9** show the load distribution in BCR and CR in Exp1,2,3, and 4 respectively. From these tables, we can see that in CR, S_1 (the head) executes write requests only which represent 10% of all requests, while S_4 (the tail) executes read requests only which represent 90% of all requests. On the other hand, in BCR, both servers S_1 and S_4 can executes write and read requests and they approximately share these requests evenly. The tables also show that in BCR, the two chains approximately execute the same number of requests.

Table 3. Load Distribution (no. of requests) in CR and BCR (Exp.1)

CR				**BCR**			
S_1		**S_4**		**S_1**		**S_4**	
R	**W**	**R**	**W**	**R**	**W**	**R**	**W**
0	200	1800	0	904	98	896	102
200		1800		1002		998	
CR_1		**CR_2 (NA)**		**CR_1**		**CR_2**	
2000		0		994		1006	

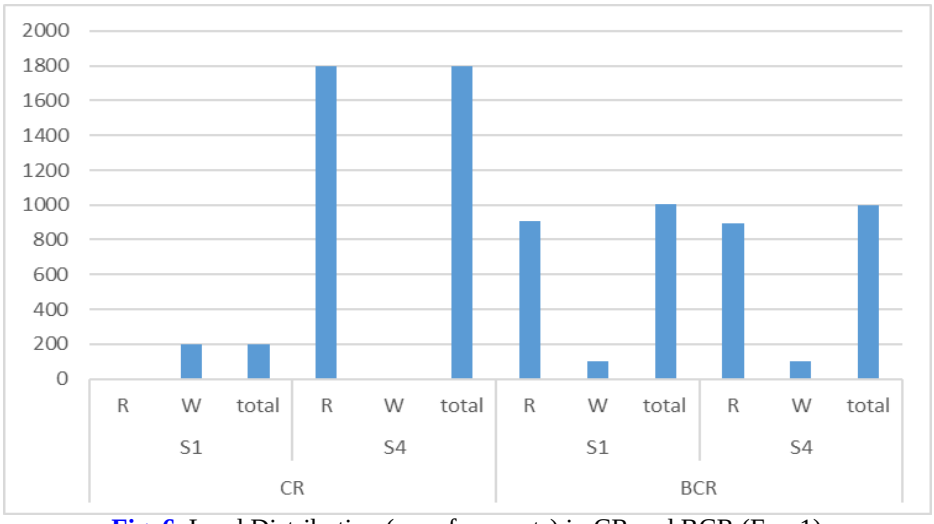

Fig. 6. Load Distribution (no. of requests) in CR and BCR (Exp.1)

Table 4. Load Distribution (no. of requests) in CR and BCR (Exp.2)

CR				**BCR**			
S_1		**S_4**		**S_1**		**S_4**	
R	**W**	**R**	**W**	**R**	**W**	**R**	**W**
0	300	2700	0	1377	156	1323	144
300		2700		1533		1467	
CR_1		**CR_2 (NA)**		**CR_1**		**CR_2**	
3000		0		1479		1521	

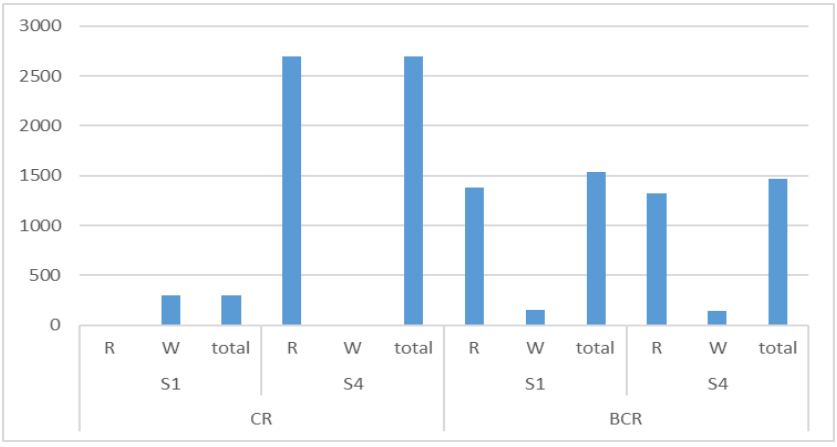

Fig. 7. Load Distribution (no. of requests) in CR and BCR (Exp.2)

Table 5. Load Distribution (no. of requests) in CR and BCR (Exp.3)

CR				**BCR**			
S_1		**S_4**		**S_1**		**S_4**	
R	**W**	**R**	**W**	**R**	**W**	**R**	**W**
0	400	3600	0	1768	203	1832	197
400		3600		1971		2029	
CR_1		**CR_2 (NA)**		**CR_1**		**CR_2**	
4000		0		2035		1965	

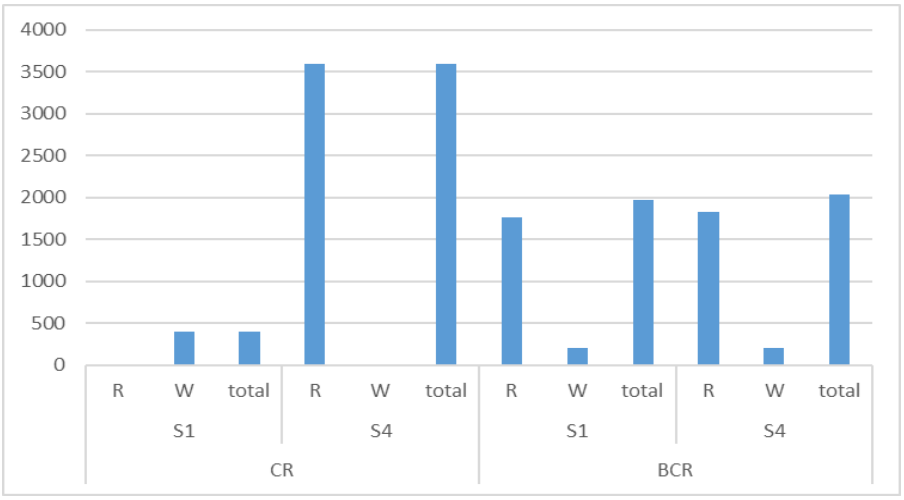

Fig. 8. Load Distribution (no. of requests) in CR and BCR (Exp.3)

Table 6. Load Distribution (no. of requests) in CR and BCR (Exp.4)

CR				**BCR**			
S_1		**S_4**		**S_1**		**S_4**	
R	**W**	**R**	**W**	**R**	**W**	**R**	**W**
0	500	4500	0	2312	244	2188	256
500		4500		2556		2444	
CR_1		**CR_2 (NA)**		**CR_1**		**CR_2**	
5000		0		2432		2568	

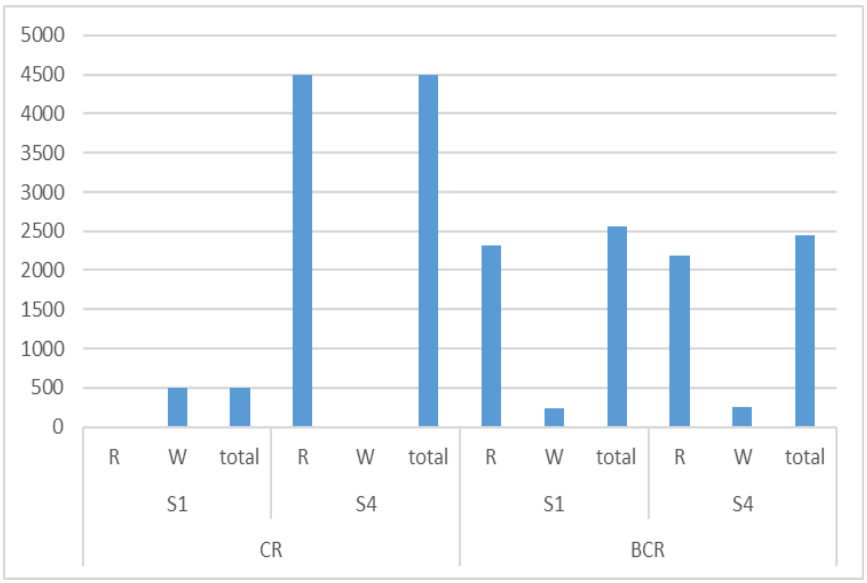

Fig. 9. Load Distribution (no. of requests) in CR and BCR (Exp.4)

Fig. 10 compares the load distribution in BCR and CR, the results depict that while in CR there is a significant difference between the loads on S_1 and S_4, in BCR, these servers have approximately the same load. Moreover, **Fig. 11** shows that in BCR, the load on CR_1 and CR_2 are approximately the same.

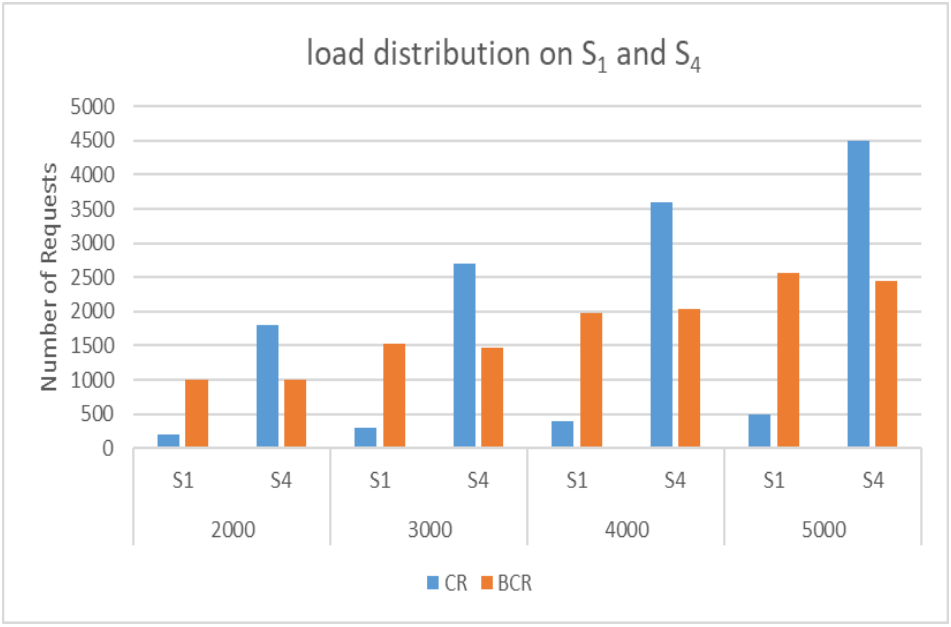

Fig. 10. Load distribution on S_1 and S_4 in BCR and CR

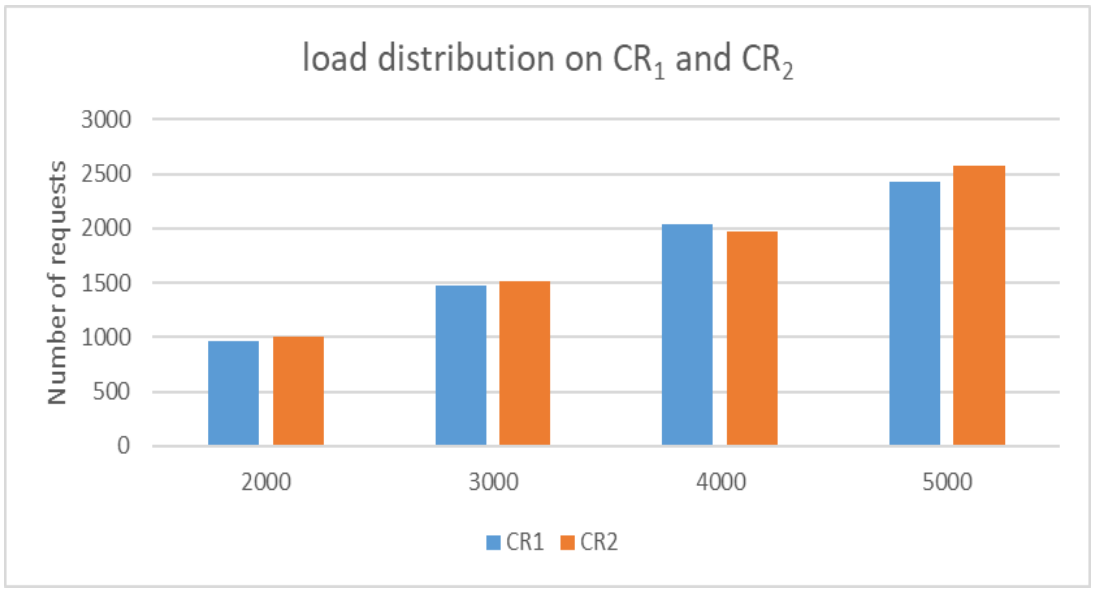

Fig. 11. Load distribution on CR_1 and CR_4 in BCR

Fig. 12 compares throughput in BCR and CR, a quick inspection to this figure releases that BCR outperforms CR in terms of read, write, and total throughput by approximately 85%. The reason for this result is that BCR utilizes two chains in parallel to execute all requests. This result is also shown in **Fig. 13**, which depicts that in all experiments, BCR execution time is less than CR execution time by a factor of approximately two.

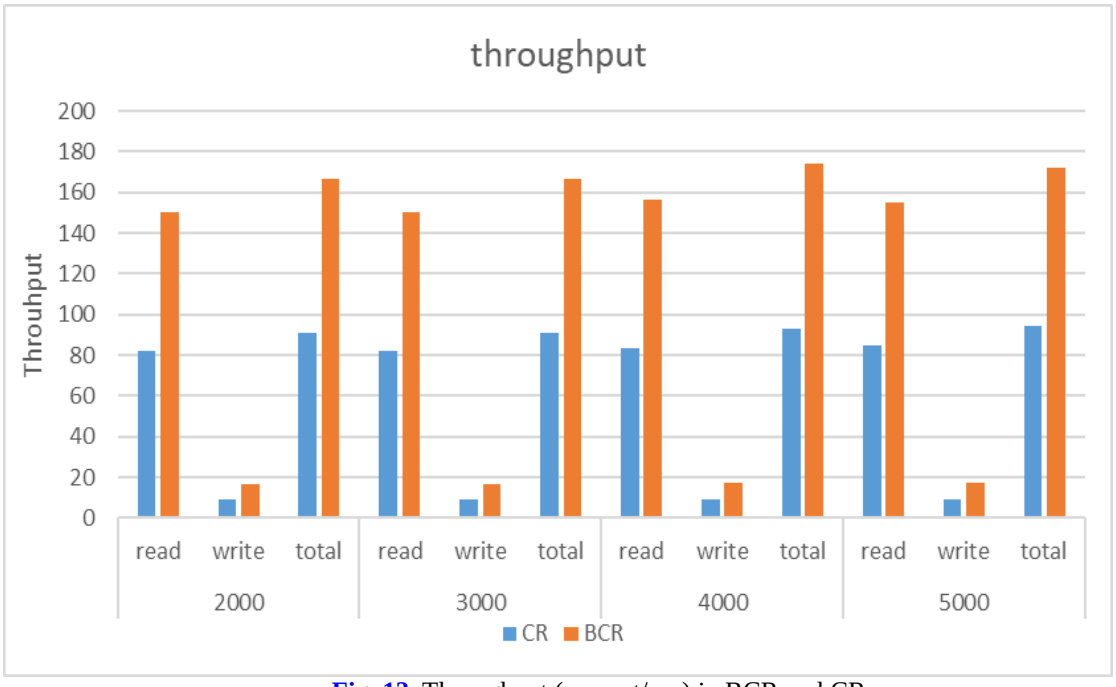

Fig. 12. Throughput (request/sec) in BCR and CR

Fig. 13. Execution time in BCR and CR

6. Conclusions and Future Work

In this paper, we proposed a novel approach called Bidirectional Chain Replication (BCR) to improve throughput in traditional Chain Replication (CR) through better utilization of computing and communication resources of the chain. Unlike CR where the whole replicated data store is treated as a single unit, in BCR the replicated data at each server in the chain are split into two disjoint Logical Partitions (LP_1, LP_2). This forms two chains that can work concurrently on the same hardware in two opposite directions and share the workload without conflict since the first chain (CR_1) exclusively manipulates data objects in LP_1 and the second chain (CR_2) exclusively manipulates data objects in LP_2. Experimental performance evaluation of BCR showed an improvement of approximately 85% in throughput over traditional CR. One limitation on BCR is that its performance depends on how requests are distributed on the logical partitions. In the ideal case, both types of requests are evenly distributed over the two partitions. Practically, some data objects belonging to a single partition are requested more frequently than others belonging to the other partition which results in unfair distribution of the requests on the partitions. The challenge is how to dynamically redistribute the popular data objects between the two logical partitions such that requests are uniformly distributed over partitions. This challenge is left for future work. Another future research direction is how to expand BCR to duplicate the performance of existence practical systems such as ChainReaction [34] and CRAQ [35].

References

[1] Charron-Bost, B., Pedone, F. & Schiper, A., "Replication: Theory and Practice," *Springer*, 2010. Article (CrossRef Link)

[2] Furat F. and Almetwally M., "Callenges and New Avenues in Existing Replication Techniques," in *Proc. of proceeding of the 6th International Conference on Cloud Computing and Service Science (CLOSER2016)*, vol. 1, pp. 147-154, Rome, Italy, April 2016.

[3] Safa Albasam and Almetwally M., "Dynamic Health-based Object Ownership Distributed Protocol," in *Proc. of proceeding of 6th International Conference on Digital Information Processing and Communications (ICDIPC)*, Beirut, Lebanon, April 2016.

[4] F.P. Junqueira and M. Serafini, "On barriers and the gap between active and passive replication," in *Proc. of Proceeding of the 27th International Symposium on Distributed Computing (DISC2013)*, Springer, vol. 8205, pp. 299-313, October 14-18, Jerusalem, Israel, 2013. Article (CrossRef Link)

[5] Schneider, F. B. & Zhou, L., "Implementing Trustworthy Services Using Replicated State Machines," in *Proc. of IEEE Symposium on Security & Privacy*, vol. 3, pp. 34-43, Oakland, California, USA, 2005. Article(CrossRefLink)

[6] Sousa, J. & Bessani, A. "From Byzantine Consensus to BFT State Machine Replication: A Latency-Optimal Transformation," in *Proc. of proceeding of the 9th IEEE European Dependable Computing Conference (EDCC2012)*, pp. 37-48, Sibiu, Romania, 2012. Article(CrossRefLink)

[7] Dettoni, F., Lung, L. C., Correia, M. & Luiz, A. F. "Byzantine Fault-Tolerant State Machine Replication with Twin Virtual Machines," in *Proc. of IEEE Symposium On Computers and Communications (ISCC2013)*, Split, Croatia, July 2013. Article(CrossRefLink)

[8] Cecchet, E., Candea, G. & Ailamaki, A. "Middleware Based Database Replication: The Gaps Between Theory and Practice," in *Proc. of Proceedings of the 2008 ACM Sigmod International Conference On Management of Data*, pp. 739-752, 2008. Article(CrossRefLink)

[9] Lang, W., Patel, J. M. & Naughton, J. F. "On Energy Management, Load Balancing and Replication," *ACM SIGMOD Record*, vol. 38, pp. 35-42, 2010. Article(CrossRefLink)

[10] Effatparvar, M., Yazdani, N., Effatparvar, M., Dadlani, A. & Khonsari, A. "Improved Algorithms for Leader Election in Distributed Systems," in *Proc. of proceeding of the 2nd IEEE International Conference on Computer Engineering and Technology (ICCET2010)*, V2-6-V2-10, 2010.

[11] Budhiraja, N., Marzullo, K., Schneider, F. B. & Toueg, S. "The Primary-Backup Approach," *Distributed Systems*, vol. 2, pp. 199-216, 1993.

[12] Mostafa, A. M. & Youssef, A. E. A, "Primary Shift Protocol for Improving Availability in Replication Systems," *International Journal of Computer Applications*, vol. 72, pp. 37-44, 2013.

[13] Mostafa, A. M. & Youssef, A. E. "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, 2014. Article(CrossRefLink)

[14] Mostafa, A. M. & Youssef, A. E. "PRP: A Primary Replacement Protocol Based On Early Discovery of Battery Power Failure in MANETS," *Multimedia Tools and Applications*, vol. 74, no. 16, pp. 6243-6254, 2015. Article(CrossRefLink)

[15] Bolosky, W. J., Bradshaw, D., Haagens, R. B., Kusters, N. P. & Li, P. "Paxos Replicated State Machines as The Basis of a High-Performance Data Store," in *Proc. of Proceeding of 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI2011)*, pp.141-154, Boston, MA, April 2011.

[16] Lamport, L. "The Part-Time Parliament", *ACM Transactions On Computer Systems (ToCS)*, vol. 16, pp. 133- 169, 1998. Article(CrossRefLink)

[17] Lamport, L. "Paxos Made Simple," *ACM SIGACT News*, vol. 32, pp. 18-25, 2001.

[18] Lampson, B. "The ABCD's of Paxos,", *Proceedings of the twentieth annual ACM symposium on Principles of Distributed Computing (PODC2001)*, pp. 13, Newport, Rhode Island, USA, 2001. Article(CrossRefLink)

[19] Tan, Z., Dang, Y., Sun, J., Zhou, W. & Feng, D. "Paxstore: A Distributed Key Value Storage System," in *Proc. of Proceeding of International Conference on Network and Parallel Computing (NPC2014)*, Springer, Lecture Notes in Computer Science, LNCS-8707, pp. 471-484, Ilan, Taiwan, 2014. Article(CrossRefLink)
[20] Burrows, M. "The Chubby Lock Service for Loosely Coupled Distributed System," in *Proc. of Proceedings of the 7th USENIX Symposium On Operating Systems Design and Implementation (OSDI2006)*, pp. 335-350, Seattle, WA, Nov, 2006.
[21] Corbett, J. C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J. J., Ghemawat, S., Gubarev, A., Heiser, C. & Hochschild, P. "Spanner: Google's Globally Distributed Databas," *ACM Transactions On Computer Systems (ToCS)*, vol. 31, no. 8, 2013. Article(CrossRefLink)
[22] Hunt, P., Konar, M., Junqueira, F. P. & Reed, B. "Zookeeper: Wait-Free Coordination for Internet-Scale Systems," in *Proc. of USENIX Annual Technical Conference*, 2010.
[23] Shapiro, Marc, et al. "Conflict-free replicated data types," in *Proc. of Symposium on Self-Stabilizing Systems*, Springer Berlin Heidelberg. 2011. Article(CrossRefLink)
[24] S. Ghemawat, H. Gobioff, and S.-T. Leung. "The google file system," in *Proc. of Symposium on Operating Systems Principles (SOSP)*, Oct. 2003. Article(CrossRefLink)
[25] B. Fitzpatrick. "Memcached: a distributed memory object caching syste," 2009. Article(CrossRefLink)
[26] T. D. Chandra, R. Griesemer, and J. Redstone. "Paxos made live: an engineering perspective," in *Proc. of 26th ACM SOSP, PODC '07*, pages 398–407, New York, NY, USA, 2007. Article(CrossRefLink)
[27] L. Lamport. Fast Paxos, 2006. Available: Article(CrossRefLink)
[28] Y. Mao, F. P. Junqueira, and K. Marzullo. "Mencius: building efficient replicated state machines for WANs," in *Proc. of 8th USENIX OSDI*, pages 369–384, San Diego, CA, Dec. 2008.
[29] L. Lamport. "Generalized consensus and Paxos," 2005. Article(CrossRefLink)
[30] Iulian Moraru, David G. Andersen, Michael Kaminsky. "There Is More Consensus in Egalitarian Parliaments," in *Proc. of Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP '13)*, pp. 358-372, Farminton, Pennsylvania — November 03 - 06, 2013. Article(CrossRefLink)
[31] Phillipe Ajoux, Nathan Bronson, Sanjeev Kumar, Wyatt Lloyd†, and Kaushik Veeraraghavan. "Challenges to adopting stronger consistency at scale," in *Proc. of 15th Workshop on Hot Topics in Operating Systems*, 2015.
[32] Van Renesse, Robbert, and Fred B. Schneider. "Chain Replication for Supporting High Throughput and Availability," *USENIX Symposium On Operating Systems Design and Implementation (OSDI04)*, vol. 4, pp:91-104, 2004.
[33] Van Renesse, Robbert, Chi Ho, and Nicolas Schiper. "Byzantine chain replication," in *Proc. of International Conference On Principles of Distributed Systems*, Springer Berlin Heidelberg, 2012. Article(CrossRefLink)
[34] Almeida, Sérgio, João Leitão, and Luís Rodrigues. "ChainReaction: a causal+ consistent datastore based on chain replication," in *Proc. of Proceedings of the 8th ACM European Conference on Computer Systems*, ACM, 2013. Article(CrossRefLink)
[35] Terrace, Jeff, and Michael J. Freedman. "Object Storage on CRAQ: High-Throughput Chain Replication for Read-Mostly Workloads," in *Proc. of USENIX Annual Technical Conference*. 2009.
[36] Fritchie, Scott Lystig. "Chain replication in theory and in practice," in *Proc. of Proceedings of the 9th ACM SIGPLAN workshop on Erlang*, 2010. Article(CrossRefLink)
[37] W. Lloyd, M. Freedman, M. Kaminsky, and D. Andersen. "Don't settle for eventual: scalable causal consistency for wide- area storage with cops," in *Proc. of ACM SOSP*, pages 401–416, 2011.
[38] P. Mahajan, L. Alvisi, and M. Dahlin. "Consistency, availability, and convergence," *Technical Report TR-11-22*, Univ. Texas at Austin, 2011.
[39] D. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. "FAWN: a fast array of wimpy nodes," *Comm. ACM*, vol. 54, no. 7, pp:101–109, 2011. Article(CrossRefLink)

[40] R. Escriva, B. Wong, and E. G. Sirer. “HyperDex: A distributed, searchable key-value store for cloud computing,” *Technical report*, CSD, Cornell University, 2011.

Dr. Almetwally Mostafa is currently an assistant professor at KSU. He received his M.Sc. from Al-Azhar University, and received his Ph.D. in computers and systems from a channel program between Université catholique de Louvain Belgium and Al-Azhar University. He also obtained his B.Sc. degree in Computers and systems Engineering from Al-Azhar University, Egypt. His research interest includes cloud computing, distributed systems protocols, mobile and pervasive Computing, fault- tolerance, and Security of Information systems. Before he joined KSU, he has worked for two years at CETIC Centre d’Excellence en Technologies de l’Information et de la Communication.

Dr. Ahmed Youssef is currently an associate professor at KSU. He received his Ph.D. and M.Sc. degrees in Computer Science & Engineering from University of Connecticut (UConn), CT, USA. He also obtained his M.Sc. and B.Sc. degrees in Electronics & Electrical Communications Engineering from Helwan University,Egypt. His research interest includes Cloud Computing, Mobile and Pervasive Computing, Big Data, and Information Security. Before he joined KSU, he worked in several universities including Helwan University (HU), British University in Egypt (BUE), Misr International University (MIU), Misr University for Science &Technology (MUST), Canadian International College (CIC), October 6th University, and 6th Oct. University for Modern Sciences and Arts (MSA).

Yazeed Ali Aljarbua is a teaching assistant at the department of information systems, College of Computer and Information Sciences, King Saud University (KSU). He obtained his bacholer degree in information systems from KSU and he is currently pursuing for his Master degree.