ACRE: A Method for Supporting Strong Consistency and Adaptivity in Replicated Data Storage

Scott J. Denman and Kyoung-Don Kang
Department of Computer Science
State University of New York at Binghamton
{*sdenman1, kang*}*@binghamton.edu*

Abstract

As most key-value stores partition and replicate data to support high availability with no strong consistency guarantee among replicas, users may suffer from data inconsistency. Although previous research has been done to support strong consistency among replicated data, most existing approaches suffer from potential hotspots and load imbalance. Neither do they consider dynamic data access patterns that may largely vary over time. In this paper, we propose a new approach, called ACRE (Adaptive Chain REplication), to support strong consistency among data replicas, hotspot avoidance and load balancing, and adaptivity to dynamic data access patterns.

1 Introduction

Replicated data stores are a fundamental building block for advanced online applications such as scientific computing and social networking. In these systems, data are partitioned and replicated multiple times to manage vast amounts of data in a highly available manner [5]. However, the CAP theorem [1] states that it is impossible for replicated data stores to support strong data consistency, availability, and tolerance to network partitions simultaneously. Thus, to support the high availability of service, a number of key-value stores, such as the ones used at Google [3] and Amazon [2], only support eventual consistency, providing no strong consistency among replicas. As a result, users of online applications may suffer from a nuisance or even inconsistent results. Also, developing advanced applications without strong data consistency support is cumbersome and time consuming.

A novel method called chain replication [7] has been developed to support strong consistency, while improving the availability and throughput of fail-stop storage servers. In chain replication, all nodes storing shared data objects are organized as a chain. The head node in a chain handles all write requests, while the tail processes all read requests. Writes are propagated down from the head to the tail of the chain of replicated data stores before acknowledging the client. Thus, total ordering among the versions of a data item in the chain is supported. Also, strong consistency is supported with respect to the version of the data that is successfully written (i.e., committed) at the tail. As the chain replication method is simple and lacks multi-round protocols, it supports consistency, high availability, and easy recovery. However, all read requests for a data object must be processed at the tail. Thus, the tail may become a hot spot and performance bottleneck.

CRAQ (Chain Replication with Apportioned Queries) [6] extends the chain replication method [7] to support strong consistency as well as lower latency and higher throughput for *read* requests. In CRAQ, not only the tail but any node in a chain can process reads. When a write commits at the tail, an acknowledgment (ACK) for the committed version of the data object is passed backwards, ultimately reaching the head. The head and intermediate nodes, if any, process read requests using the latest version acknowledged by the tail. If the head or an intermediate node receives a read request for a dirty version awaiting an ACK, it simply waits for the ACK or sends a version query to the tail that returns the latest version number. In this way, CRAQ enhances the read throughput, while supporting strong consistency with respect to the tail. However, CRAQ has a drawback too. A node has to query the tail first, if it wants to process a read request without waiting for the ACK. Thus, the tail may be flooded by excessive version queries and become a

bottleneck especially in the presence of write-heavy workloads.

Although chain replication and CRAQ support strong consistency with acceptable throughput and availability, neither of them is effective enough to deal with *dynamic data access patterns*. This can be a serious problem, because it is known that data access patterns often vary largely over time [4]. Hence, optimizing a replication scheme for a specific data access pattern may result in undesirable performance and resource waste, while not necessarily enhancing the availability.

To shed light on this problem, we propose a new approach called ACRE (Adaptive Chain REplication). In ACRE, any node can process a read request by returning the latest acknowledged version, similar to CRAQ [6]. However, ACRE is different from chain replication and CRAQ; *ACRE autonomously adapts chain replication based on the dynamic data access patterns* that can be observed in the life cycle of data, such as new scientific data or news articles read a lot when created and gradually become dormant, while supporting *strong consistency*. More specifically, ACRE extends an existing chain by adding another node to the chain to process read requests efficiently, if the frequency of read requests increases significantly, for example, due to the increased popularity of the data.

On the other hand, if write requests dominate and read requests shrink, we shorten the chain by removing a node from the chain as long as the required minimum chain length, i.e., the number of the replicated data stores in the chain, for fault tolerance is maintained. In this way, the total number of cascading writes from the head to the tail along the chain can be decreased without affecting the required level of fault tolerance. Also, the wait time for the ACK of a write request and the frequency of version queries sent to the tail decrease. Thus, ACRE enhances not only the read performance but also the cost-effectiveness of strong consistency as well as fault tolerance considering dynamic data access patterns.

In addition, ACRE naturally supports *load balancing among the chained storage nodes*. Because any node can process reads, it is less likely for the tail to become a bottleneck. For load balancing, read requests can be evenly distributed (in an approximate sense) to the chained nodes via, for example, random distribution of reads to the nodes. Also, a write is performed by every node in the chain to support strong consistency and fault tolerance. Relatively little prior work has been done to support flexible data replication with strong consistency as well as adaptivity to dynamic data access patterns, while supporting hotspot avoidance and load balancing [7, 6, 5].

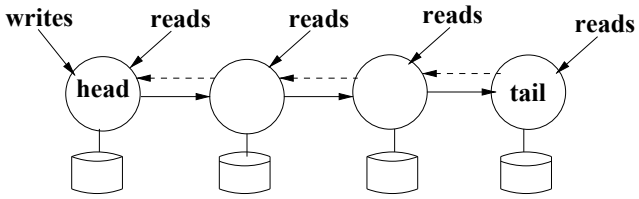
For performance evaluation, we have implemented ACRE and CRAQ and evaluated them using synthetic read/write workloads, similar to [6]. Our initial results show that ACRE decreases the average delay for data access by approximately 30 - 50% compared to the service delay provided by CRAQ by dynamically adapting the replication chain considering different data access patterns.

The remainder of this paper is organized as follows. A description of ACRE is given in Section 2. Performance evaluation results are presented in Section 3. Related work is discussed in Section 4. Finally, Section 5 concludes the paper and discusses future work.

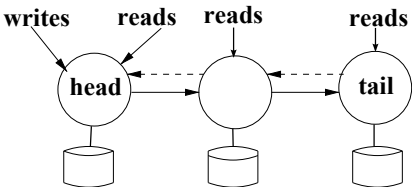
2 Adaptive Chain Replication

In this section, the design and implementation issues of ACRE are discussed in sequence.

2.1 Design of ACRE



(a) An Example Chain for Read-Heavy Workloads



(b) An Example Minimum Length Chain for Write-Heavy Workloads

Figure 1. Adaptive Chain Replication

In ACRE, any node can process read requests as shown in Figure 1. The solid horizontal lines in Figure 1 indicate a write request performed in a

cascading manner, while a dotted line represent an ACK packet carrying the version information that acknowledges a successful write of a data item at the tail. The head or an intermediate node (that is neither the head nor the tail of a chain) has to either wait for the corresponding ACK from the tail or send the tail a version query, if it receives a read request for a dirty data object. For a read request, a node in ACRE returns the latest acknowledged version.

The key advantage that sets ACRE apart from the existing approaches is the *flexibility* and *adaptation* to dynamic changes in data access patterns. In ACRE, if the frequency of read requests increases by a certain threshold, a chain is *expanded*. For example, assume that a chain initially consists of 3 nodes, but extends to 4 nodes as shown in Figure 1(a). To extend a chain, ACRE performs the following procedure:

1. ACRE appends a new node to the current tail, if necessary, to extend the chain considering dynamic data access patterns.
2. ACRE requires the current tail to forward the most up-to-date versions of the data, S, to the new node. Let t_1 indicate the time at which the tail starts sending S to the new node.
3. The new node writes S to its storage. While the new node is doing the writes, the current tail continues to work as the tail of the chain.
4. Once the new node finishes all writes, the new node announces itself as the new tail to all the other nodes in the chain via a reliable multicast. When the current tail receives the announcement at time t_2, it stops working as the tail and becomes the previous tail.
5. If some data ΔS have been written by the previous tail between t_1 and t_2, the previous tail forwards ΔS to the new tail, which treats ΔS as regular data writes to its storage and acknowledges the other nodes after finishing the writes.

ACRE repeats this process, if necessary, to support acceptable read performance, while supporting the increased availability as data become more popular. As a result, the throughput and availability of data enhance as more replicas are created for more read requests. In this paper, a node failure is handled in a similar manner to the original chain replication protocol [7].

On the other hand, the chain is *shortened* as shown in Figure 1(b), if the data access pattern becomes write-heavy. We take this approach, because a long chain is subject to large delays and overheads for write-heavy workloads, providing little opportunities to improve the performance by processing many reads in parallel. Specifically, ACRE *cuts the tail*, if necessary, to efficiently handle increasing writes. Notably, removing the tail from the chain is the simplest and fastest way to shorten a chain, since the the data stored by all the other nodes are at least as fresh as the tail's. Thus, we avoid overheads for maintaining data consistency among the nodes by cutting the tail. The leaving tail only has to process pending version queries, if any, and then announce its departure, while declaring its predecessor as the new tail to the other nodes in the chain. ACRE repeats the tail cutting process as writes become more frequent until the chain cannot be shortened any further to maintain the minimum specified number of replicas as shown in Figure 1(b). For example, the Google File System maintains three replicas for each data object by default [3].

In ACRE, adaptation is performed by adding or removing a node at the end of the chain for coordinated control with little overhead. Depending on read/write workloads, a node can be removed from and added back to the tail later; however, the newly added node only needs the latest versions acknowledged by the current tail. When a new node is added to the tail, the current tail continues to serve the data requests, while sending the latest data to the new tail in the background to minimize the impact on the performance.

2.2 Chain Adaptation

In this paper, we assume that clients are given the list of the data stores currently in the chain. In the specific implementation of ACRE evaluated in Section 3, the head node disseminates the list whenever it is updated. Also, a client submits a read request to a (pseudo) randomly selected data store for load balancing purposes, while always submitting a write request to the head node. In addition, ACRE has a number of tunable parameters, such as the chain adaptation threshold that triggers the chain to adapt to the data access pattern. One metric, two thresholds, and two constraints used for adaptation in ACRE are described in this subsection.

For performance enhancement via adaptation, we use the *read ratio* metric that measures the ratio of

read requests to writes tracked at each node in the chain. This ratio is updated following every request reception at a given node. It should be emphasized that the scope of the ratio is limited to a single node but all writes to the chain will eventually touch each node in the chain. (However, this is not true for read requests). Using the read ratio measured at each node, the chain is adapted by ACRE, if all the following thresholds are exceeded and the constraints are met.

- *Extend threshold:* This is the upper bound used to determine when the chain will benefit from adding a new node. When the read ratio exceeds this threshold at any node that individual node sends the tail a request to extend the chain (provided the following adaptation constraints 1 and 2 are met).
- *Reduce threshold:* This is the lower bound used to determine when the chain will benefit from cutting the tail. When the reduce threshold exceeds the read ratio at a node, the node sends the tail a request to shorten the chain (provided the following two constraints are met).

In addition to defining the read ratio and the thresholds for adaptation, we allow adaptation only if the following two constraints are met to support *hysteresis* and avoid potential oscillations.

- *Constraint 1. Minimum request bound:* The minimum request bound imposes a lower limit on the number of requests a node has processed before allowing a chain adaptation request to be sent. This bound allows ACRE to avoid too frequent adaptation, which can result in oscillatory and unstable system behaviors.
- *Constraint 2. Minimum adaptation gap:* The minimum adaptation gap imposes a minimum wait time between two consecutive adaptation requests processed by the chain. This allows time for the previous request to take effect prior to requesting a similar adaptation. As any adaptation happens at the tail, the tail tracks the amount of time since the last adaptation. No additional adaptation will be allowed until this minimum amount of time has passed.

We observe that it is possible for the nodes in the chain may make conflicting requests for chain adaptation in an extreme case. For example, one node may request ACRE to extend the chain, while another node may request ACRE to shorten the chain

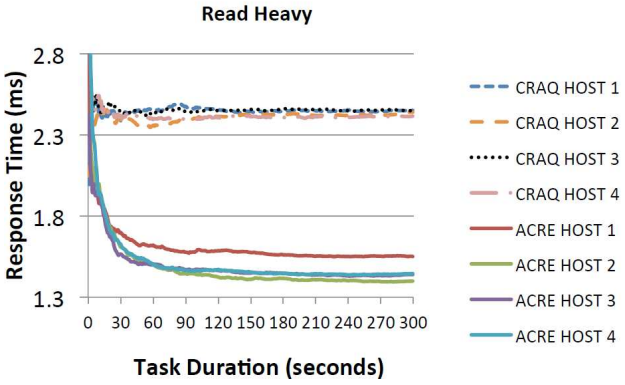


Figure 2. Response Time for Test 1

when the read load is largely imbalanced among the nodes in the chain. To address this problem, every adaptation request is sent to the tail that makes a final decision and actually adapts the chain, if necessary, to decrease the data access delay. In this paper, the tail makes a decision for adaptation based on the read ratio measured by itself, if two or more nodes send conflicting adaptation requests to the tail and the two constraints are satisfied. We take this approach, because the tail is aware of the progress of all writes and receives a roughly fair share of read requests, since clients randomly distribute read/write requests to the nodes for load balancing purposes in this paper.

3 Performance Evaluation

In this section, we implement ACRE and CRAQ using TCP/IP and evaluate their performance. We describe our experimental settings for synthetic workload generation followed by the performance results.

3.1 Experimental Settings

In this paper, we aim to evaluate CRAQ and ACRE for different synthetic read/write workloads, similar to [6]. More specifically, three tests are performed for workloads that model read-heavy, write-heavy, and mixed request streams, respectively.

- *Test 1:* In this test, data are initially written once but only read later on. Hence, the read to write ratio approaches infinity to model a read-heavy workload.
- *Test 2:* In this test, data are initially read but only written later on. Thus, the read to write ratio approaches zero, modeling a write-heavy workload.

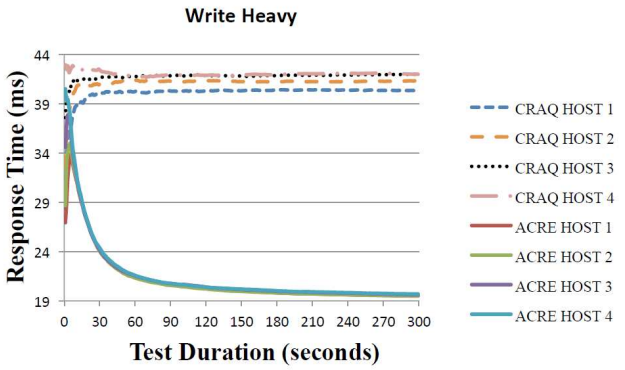


Figure 3. Response Time for Test 2

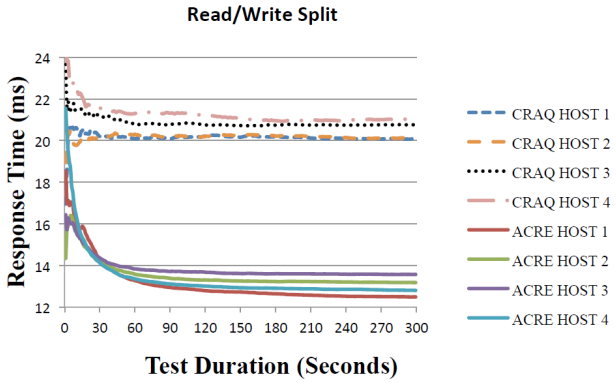


Figure 4. Response Time for Test 3

- *Test 3:* The read to write ratio set to 1 in this test. Hence, an equal number of read and write requests are issued to model a balanced read/write split.

We initially use 4, 10, and 10 storage nodes in Test 1, Test 2, and Test 3, respectively. ACRE may extend or shrink the chain between 4 and 10 nodes based on the data access pattern. Each test is run for 5 minutes. These tests were intended to demonstrate the benefits of ACRE over a short burst of extreme read to write ratios. In each test, 100 clients running on 4 physical machines submit read/write requests to the storage nodes in the chain. A write request is always sent to the head to maintain the strong consistency of the chained data nodes. A read request is submitted to one of the available nodes in the chain that is selected using a pseudo random number generator for load balancing. After sending a read/write request, a client waits for a random inter-request delay selected in the range of [0ms, 100ms]. Hence, on average, 2,000 read/write requests are submitted to the chain every second. Each read/write request reads/writes 1024 bytes. For ACRE, the minimum request bound for adaptation and minimum adaptation gap are set to 500 requests and 30s, respectively.

3.2 Performance Evaluation Results

Each data point in Figure 2 - Figure 4 shows the average response time of read/write requests processed per second measured in the four physical machines hosting the clients. (In ACRE, a chain can be extended up to 10 nodes. However, the response time in the other nodes remain similar to the ones shown in the figures.) From these figures, we observe that the response time of ACRE is considerably shorter than that of CRAQ. The response time gap becomes pronounced after 30s. As shown in the figures, the response time of ACRE is 30% - 50% shorter than that of CRAQ after 30s. By dynamically adapting the chain based on the data access patterns, ACRE reduces the service delay, while supporting strong data consistency and data availability. This contrasts to the popular data stores, e.g., [3, 2], which sacrifice strong data consistency and rely on eventual consistency, providing no guarantee on data consistency. These results show the viability of adaptive replication that provides a strong consistency guarantee, while adapting to dynamic read/write patterns.

4 Related Work

Data are partitioned and replicated to support high performance and availability. Popular key-value stores, e.g., [1, 2], only support eventual consistency among replicas. As a result, inconsistent data can be exposed to distributed clients accessing shared data.

Chain replication [7] supports strong consistency by using a chain of replicated data stores. However, the tail node in a chain can be a performance bottleneck, because only the tail has to process all reads. CRAQ [6] enhances the read performance by allowing not only the tail but also all the other nodes in a chain to process reads. However, the tail can be a bottleneck, if write-heavy workloads are given.

ACRE addresses this problem by dynamically adapting the chain in a cost-effective manner, if necessary, to improve the performance by considering data access patterns unlike static approaches, such as [3, 2, 7, 6], which support either weak consistency or strong consistency in a fixed chain. The need for strong consistency and high availability with load balancing

is increasing. Also, many real world applications have time-varying data access patterns. However, it is challenging to support strong consistency, high availability, and adaptivity to dynamic data access patterns. ACRE takes an initial step to address these challenges.

5 Conclusions and Future Work

Due to the lack for strong consistency among replicas, replicated key-value stores may suffer from data inconsistency. In this paper, we present a new approach, called ACRE, to support strong consistency, load balancing, and adaptivity to dynamic data access patterns to enhance the performance. In the future, we will investigate more efficient approaches to supporting strong consistency with further enhanced performance.

Acknowledgement

This work was supported, in part, by the NSF grant CNS-117352.

References

[1] E. Brewer. Towards Robust Distributed Systems. In *PODC Keynote*, 2000.

[2] G. Decandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, D. Hastorun, G. Decandia, and W. Vogels. Dynamo: Amazon's Highly Available Key-Value Store. *Symposium on Operating Systems Principles*, 2007.

[3] S. Ghemawat, H. Gobioff, and S. Leung. The Google File System. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, 2003.

[4] R. T. Kaushik and M. Bhandarkar. GreenHDFS: Towards An Energy-Conserving, Storage-Efficient, Hybrid Hadoop Compute Cluster. In *HotPower*, 2010.

[5] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. Don't Settle for Eventual: Scalable Causal Consistency for Wide-Area Storage with COPS. In *ACM Symposium on Operating Systems Principles*, 2011.

[6] J. Terrace and M. J. Freedman. Object Storage on CRAQ: High-throughput chain replication for read-mostly workloads. In *USENIX Annual Technical Conference*, 2009.

[7] R. van Renesse and F. B. Schneider. Chain Replication for Supporting High Throughput and Availability. In *Operating Systems Design and Implementation*, 2004.