OPCAM: Optimal Algorithms Implementing Causal Memories in Shared Memory Systems

Min Shen
University of Illinois at Chicago
mshen6@uic.edu

Ajay D. Kshemkalyani
University of Illinois at Chicago
ajay@uic.edu

Ta-yuan Hsu
University of Illinois at Chicago
thsu4@uic.edu

ABSTRACT

Data replication is commonly used for fault tolerance in reliable distributed systems. In this paper, we propose three optimal protocols for causal consistency in distributed shared memory systems. Our proposed optimal protocols are designed for partial replication across the distributed shared memory. Complete replication is a special case of our protocols and we also give the optimal implementation of causal consistency for the complete replication case. Algorithm Full-Track is optimal in the sense that it can update the local copy as soon as possible while respecting causal consistency. Algorithm Opt-Track is further optimal in the sense that the size of the local logs maintained and the amount of control information piggybacked on the update messages is minimal. Algorithm Opt-Track-CRP is a special case of algorithm Opt-Track for the full replication case. It is highly scalable, and significantly more efficient than the Baldoni et al. protocol for the complete replication case.

Categories and Subject Descriptors

C.2.4 [**Distributed systems**]: Distributed applications

General Terms

Theory, Design, Performance

Keywords

causal consistency, distributed shared memory, causality, replication

1. INTRODUCTION

Data replication is a common technique used for fault tolerance in reliable distributed systems. Besides providing fault tolerance, it also reduces access latency in the cloud. With data replication, consistency of data in the distributed shared memory becomes a core issue. There exists a spectrum of consistency models in distributed shared memory systems [11]: linearizability (the strongest), sequential consistency, causal consistency, pipelined RAM, slow memory, and eventual consistency (the weakest). These consistency models represent a trade-off between cost and convenient semantics for the application programmer.

ICDCN '15, January 04 - 07 2015, Goa, India
Copyright 2015 ACM 978-1-4503-2928-6/15/01 ...$15.00.
http://dx.doi.org/10.1145/2684464.2684483.

In this paper, we propose a suite of optimal protocols, OPCAM, for causal consistency in distributed shared memory systems. Causal memory was proposed by Ahamad et al. [1]. Later, Baldoni et al. gave an improved implementation of causal memory [2]. Their implementation is optimal in the sense that the protocol can update the local copy as soon as possible, while respecting causal consistency. Recently, consistency models have received attention in the context of cloud computing with data centers and geo-replicated storage, with product designs from industry, e.g., Google, Amazon, Microsoft, LinkedIn, and Facebook. The CAP Theorem by Brewer [8] states that for a replicated, distributed data store, it is possible to provide at most two of the three features: Consistency of replicas, Availability of writes, and Partition tolerance. In the face of this theorem, most systems such as Amazon's Dynamo [7] chose to implement eventual consistency [4], which states that eventually, all copies of each data item converge to the same value. Besides the above three features, two other desirable features of large-scale distributed data stores are: low Latency and high Scalability [12]. Causal consistency is the strongest form of consistency that satisfies low Latency, defined as the latency less than the maximum wide-area delay between replicas. Causal consistency has been implemented by Lloyd et al. [12], Mahajan et al. [13], Belaramani et al. [3], and Petersen et al. [14]. However, these implementations are not optimal in terms of the control information they use, and they do not even achieve optimality in the sense defined by Baldoni et al.. Further, with the exception of Lloyd et al., they do not provide scalability as they use a form of log serialization and exchange to implement causal consistency. Even further, all the works, including Baldoni et al., assume Complete Replication and Propagation (CRP) based protocols. These protocols assume full replication and do not consider the case of partial replication.

Our proposed optimal protocols for causal consistency are designed for partial replication across the distributed shared memory. Complete replication is a special case of our protocols and we also give the optimal implementation of causal consistency for the complete replication case. We argue that partial replication is more natural for some applications. Consider the following example. A user A's data is replicated in a shared memory system across multiple data centers located in different regions. Since user A's connections are mostly located in the Chicago region and the US West coast, the majority of views of user A's data will be coming from those two regions. In such a case, it is an overkill to replicate user A's data in data centers outside these two regions, and partial replication has very small impact on the overall latency in this scenario. With p replicas placed at some p of the total of n data centers, each Write operation that would have triggered an update broadcast to the n data centers now becomes a multicast to just p of the n data centers. This is a direct savings in the number of messages and p is a tunable parameter.

Contributions

1. We present an optimal algorithm Full-Track implementing causal memory in a partially replicated shared memory system. The optimality achieved is in the sense defined by Baldoni et al., viz., the ability of the protocol to update the local copy as soon as possible while respecting causal consistency.
2. Algorithm Full-Track can be made optimal in terms of the size of the local logs maintained and the amount of control information piggybacked on the update messages. The resulting algorithm, Opt-Track, for partially replicated shared memory systems is thus optimal not only in the sense of Baldoni et al. but also in the amount of control information used in local logs and on update messages by achieving minimality.
3. As a special case of Algorithm Opt-Track, we present algorithm Opt-Track-CRP that is optimal in a fully replicated shared memory system. This algorithm is optimal not only in the sense of Baldoni et al. but also in the amount of control information used in local logs and on update messages, which is much less than for algorithm Opt-Track, making it highly scalable. The algorithm is significantly more efficient than the Baldoni et al. protocol for the full replication case.

2. SYSTEM MODEL

The system model used in this paper is based on those proposed by Ahamad et al. [1] and Baldoni et al. [2]. We consider such a system which consists of n application processes $ap_1, ap_2, \ldots, ap_n$ interacting through a shared memory $\mathcal{Q}$ composed of q variables $x_1, x_2, \ldots, x_q$. Each application process ap_i can perform either a *read* or a *write* operation on any of the q variables. A *read* operation performed by ap_i on variable x_j which returns value v is denoted as $r_i(x_j)v$. Similarly, a *write* operation performed by ap_i on variable x_j which writes the value u is denoted as $w_i(x_j)u$. Each variable has an initial value $\perp$.

By performing a series of *read* and *write* operations, an application process ap_i generates a local history h_i. If a local operation o_1 precedes another operation o_2, we say o_1 precedes o_2 under *program order*, denoted as $o_1 \prec_{po} o_2$. The set of local histories h_i from all n application processes form the global history H. Operations performed at distinct processes can also be related using the *read-from order*, denoted as $\prec_{ro}$. Two operations o_1 and o_2 from distinct processes ap_i and ap_j resp. have the relationship $o_1 \prec_{ro} o_2$ if there are variable x and value v such that $o_1 = w(x)v$ and $o_2 = r(x)v$, meaning that *read* operation o_2 retrieves the value written by the *write* operation o_1. It is shown in [1] that

- for any operation o_2, there is at most one operation o_1 such that $o_1 \prec_{ro} o_2$;
- if $o_2 = r(x)v$ for some x and there is no operation o_1 such that $o_1 \prec_{ro} o_2$, then $v = \perp$, meaning that a read with no preceding write must read the initial value.

With both the *program order* and *read-from order*, the *causality order*, denoted as $\prec_{co}$, can be defined on the set of operations O_H in a history H. The causality order is the transitive closure of the union of local histories' program order and the read-from order. Formally, for two operations o_1 and o_2 in O_H, $o_1 \prec_{co} o_2$ if and only if one of the following conditions holds:

1. $\exists ap_i$ s.t. $o_1 \prec_{po} o_2$ (program order)
2. $\exists ap_i, ap_j$ s.t. o_1 and o_2 are performed by ap_i and ap_j respectively, and $o_1 \prec_{ro} o_2$ (read-from order)
3. $\exists o_3 \in O_H$ s.t. $o_1 \prec_{co} o_3$ and $o_3 \prec_{co} o_2$ (transitive closure)

Essentially, the causality order defines a partial order on the set of operations O_H. For a shared memory to be causal memory, all the write operations that can be related by the causality order have to be seen by all application process in the same order defined by the causality order.

The shared memory abstraction and its causal consistency model is implemented on top of the underlying distributed message passing system which also consists of n sites, with each site s_i hosting an application process ap_i. Since we assume a non-fully replicated system, each site holds only a subset of variables $x_h \in \mathcal{Q}$. For application process ap_i, we denote the subset of variables kept on the site s_i as X_i. If the replication factor of the shared memory system is p and the variables are evenly replicated on all the sites, then the average size of X_i is $\frac{pq}{n}$.

To facilitate the read and write operations in the shared memory abstraction, the underlying message passing system provides several primitives to enable the communication between different sites. The read and write operations performed by the application processes also generate certain *events* in the underlying message passing system.

To implement the causal memory in the shared memory abstraction, each time an update message m corresponding to a write operation $w_j(x_h)v$ is received at site s_i, a new thread is spawned to check when to locally apply the update. The condition that the update is ready to be applied locally is called activation predicate in [2]. This predicate, $A(m_{w_j(x_h)v}, e)$, is initially set to $false$ and becomes $true$ only when the update $m_{w_j(x_h)v}$ can be applied after the occurrence of local event e. The thread handling the local application of the update will be blocked until the activation predicate becomes $true$, at which time the thread writes value v to variable x_h's local replica. The key to implement the causal memory is the activation predicate.

Baldoni et al. designed an optimal activation predicate in [2]. Their activation predicate cleanly represents the causal memory's requirement: a write operation shall not be seen by an application process before any causally preceding write operations. It is optimal because the moment this activation predicate becomes true is the earliest instant that the corresponding update can be applied.

3. ALGORITHMS

The algorithm implementing causal memories given in [2] is for a fully-replicated system. However, it might not always be possible to assume full replication, as the incurred cost might be too high. It is thus important to design algorithms that implement causal memories even in a non-fully replicated system.

Several algorithms that aim at achieving a causal message ordering have been previously proposed [15, 9, 10]. Different from the algorithms proposed in [2], these algorithms are for message passing systems where application processes communicate with each other via sending and receiving messages. Putting aside this difference, none of these causal message ordering algorithms assume messages being broadcast each time application processes communicate with each other. This is similar to non-fully replicated shared memory systems, where an individual application process writing a variable does not write to all sites in the system. In both cases, the changes in one application process do not get propagated to the entire system. Thus, we take inspiration from these causal message ordering algorithms and design two algorithms implementing causal memories in a non-fully replicated shared memory system, both of which adopt the optimal activation predicate.

The first algorithm, Algorithm Full-Track, is adapted from the causal message ordering algorithm proposed by Raynal et al. [15]. Since the system is non-fully replicated, each application process performing a write operation will only write to a subset of all the sites in the system. The tracking of the writes required to implement the optimal activation predicate conceptually uses the notion of the matrix clock that is traditionally used for tracking logical time.

Algorithm Full-Track achieves optimality in terms of the activation predicate. However, in other aspects, it can still be further optimized. We notice that, each message corresponding to a write operation piggybacks an $O(n^2)$ matrix, and the same storage cost is also incurred at each site s_i. Kshemkalyani and Singhal proposed an optimal algorithm based on necessary and sufficient conditions [9, 10] that aim at reducing the message size and storage cost for causal message ordering algorithms in message passing systems (hereafter referred to as the KS algorithm). The ideas behind the KS algorithm exploit the transitive dependency of causal deliveries of messages. In the KS algorithm, each site keeps a record of the most recently received message from each other site. The list of destinations of the message are also kept in each record (the KS algorithm assumes multicast communication). With each outgoing message, these records are also piggybacked. The KS algorithm achieves another optimality in the sense that no redundant destination information is recorded. Although the KS algorithm is for message passing systems, its idea of deleting unnecessary dependency information still applies to shared memory systems. We adapt the KS algorithm to a non-fully replicated shared memory system to implement causal memory there. The resulting algorithm is Algorithm Opt-Track.

Algorithm Opt-Track can be directly applied to fully-replicated shared memory systems. However, due to the full replication case, several optimizations can be applied, including that there is no need to keep a list of the destination information with each write operation. In this way, we bring the cost of representing a write operation from potentially $O(n)$ down to $O(1)$. This improves the algorithm's scalability when the shared memory is fully replicated. The resulting algorithm is Algorithm Opt-Track-CRP.

Four metrics are used in the complexity analysis (See Table 1):

- message count: count of the total number of messages generated by the algorithm.
- message size: the total size of all the messages generated by the algorithm. It can be formalized as $\sum_i$*(# type i messages * size of type i messages).*
- space complexity: the space complexity at each site s_i for storing the various local logs.
- time complexity: the time complexity at each site s_i for performing the write and read operations.

The following parameters are used in the analysis:

- n: number of sites in the system
- q: number of variables in the shared memory system
- p: replication factor, i.e., the number of sites at which each variable is replicated
- w: number of write operations performed in the shared memory system
- r: number of read operations performed in the shared memory system
- d: number of write operations stored in local log (used only in the analysis of Opt-Track-CRP)

Although the total message size complexity of the Opt-Track algorithm is $O(np^2w+rp^2)$, this is only the asymptotic upper bound. As shown by Chandra et al. [5, 6], on average, each message's record in the KS algorithm maintains only a constant size of destination list. This means that, due to the optimal condition that only necessary destination information is kept, the local log at each site will only incur an amortized storage cost of $O(n)$. This also applies to the Opt-Track algorithm. Thus, the amortized message size complexity of the Opt-Track algorithm is $O(npw + rp)$. The space complexity of the Opt-Track algorithm is $O(p^2q)$. Still, this is only the asymptotic upper bound. The amortized space complexity will be $O(pq)$, since the size of the destination list is constant on average [5, 6].

4. DISCUSSION

Compared with the existing causal memory algorithms, our suite of algorithms has advantages in several aspects. Similar to the complete replication and propagation causal memory algorithm, *OptP*, proposed by Baldoni et al., our algorithm also adopts the optimal activation predicate. However, compared with the Opt-Track-CRP algorithm, the *OptP* algorithm incurs a higher cost in the message size complexity, the time complexity for read operations, and the space complexity. This is due to the fact that the *OptP* algorithm requires each site to maintain a $Write$ clock of size n, and does not take advantage of the optimization techniques in the KS algorithm. Compared with other causal memory algorithms [12, 13, 3, 14], our algorithms have the additional ability to implement causal memories in non-fully replicated shared memory systems.

The benefit of partial replication compared with full replication lies in multiple aspects. First, it reduces the number of messages sent with each write operation. Although the read operation may incur additional messages, the overall number of messages can still be lower than the case of full

Table 1: Complexity measures of causal memory algorithms.

Metric	Full-Track	Opt-Track	Opt-Track-CRP	*OptP* [2]
Message count	$O(pw + \frac{rp}{n})$	$O(pw + \frac{rp}{n})$	$O(nw)$	$O(nw)$
Message size	$O(n^2pw + nrp)$	$O(np^2w + rp^2)$ amortized $O(npw + rp)$	$O(nwd)$	$O(n^2w)$
Time Complexity	write $O(p)$ read $O(n^2)$	write $O(np^2)$ read $O(n)$	write $O(1)$ read $O(1)$	write $O(1)$ read $O(n)$
Space Complexity	$O(npq)$	$O(p^2q)$ amortized $O(pq)$	$O(n)$	$O(n^2)$

replication if the replication factor is low and readers tend to read variables from local replica instead of remote ones. Hadoop HDFS and MapReduce is one such example. The HDFS framework usually chooses a small constant number as the replication factor even when the size of the cluster is large. Furthermore, the MapReduce framework tries its best to satisfy data locality, i.e., assigning tasks that read only from the local machine. In such a case, partial replication generates much less messages than full replication. Partial replication can also help to reduce the total size of messages transmitted within the system. Although the two partial replication causal memory algorithms proposed in this paper (Full-Track algorithm and Opt-Track algorithm) might have a higher message size complexity compared with their counterparts for full replication, e.g., Full-Track vs. *OptP* and Opt-Track vs. Opt-Track-CRP, this measure is only for the control messages and does not take into consideration the size of the data that is actually being replicated. In modern social networks, multimedia files like images and videos are frequently shared. The size of these files is much larger than the control information piggybacked on them. Full replication might improve the latency for accessing these files from different locations, however it also incurs a large overhead on the underlying system for transmitting and storing these files across different sites. Furthermore, in the scenario in Section 1, where most accesses to a user's file are located within certain geographical regions, the improvement in the latency brought by full replication is less significant compared to the cost it imposes on the underlying system. Thus, partial replication can help to reduce the transmission and storage overhead in the underlying system when read operations tend to be performed on the local replica and the files to be replicated are much larger than the control information.

The full version of the paper is available as [16].

5. REFERENCES

[1] M. Ahamad, G. Neiger, J. Burns, P. Kohli, and P. Hutto. Causal memory: definitions, implementation and programming. *Distributed Computing, 9, 1*, pages 37–49, 1995.

[2] R. Baldoni, A. Milani, and S. Piergiovanni. Optimal propagation-based protocols implementing causal memories. *Distributed Computing, 18, 6*, pages 461–474, 2006.

[3] N. Belaramani, M. Dahlin, L. Gao, A. Venkataramani, P. Yalagandula, and J. Zheng. Practi replication. *In NSDI*, 2006.

[4] P. Bernstein and S. Das. Rethinking eventual consistency. *Proc. of the 2013 ACM SIGMOD International Conf. on Management of Data*, 2013.

[5] P. Chandra, P. Gambhire, and A. D. Kshemkalyani. Performance of the optimal causal multicast algorithm: A statistical analysis. *IEEE Transactions on Parallel and Distributed Systems, 15(1)*, pages 40–52, January 2004.

[6] P. Chandra and A. D. Kshemkalyani. Causal multicast in mobile networks. *Proc. of the 12th IEEE/ACM Symposium on Modelling, Analysis, and Simulation of Computer and Communication Systems*, pages 213–220, 2004.

[7] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. *Proc. of the 19th ACM SOSP*, pages 205–220, 2007.

[8] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 2002.

[9] A. Kshemkalyani and M. Singhal. An optimal algorithm for generalized causal message ordering. *Proc. of the 15th ACM Symposium on Principles of Distributed Computing (PODC)*, page 87, 1996.

[10] A. Kshemkalyani and M. Singhal. Necessary and sufficient conditions on information for causal message ordering and their optimal implementation. *Distributed Computing, 11, 2*, pages 91–111, 1998.

[11] A. Kshemkalyani and M. Singhal. *Distributed Computing: Principles, Algorithms, and Systems.* Cambridge University Press, 2008.

[12] W. Lloyd, M. Freedman, M. Kaminsky, and D. Andersen. Don't settle for eventual: Scalable causal consistency for wide-area storage with COPS. *Proc. of the 23rd ACM SOSP*, pages 401–416, 2011.

[13] P. Mahajan, L. Alvisi, and M. Dahlin. Consistency, availability, and convergence. *Tech. Rep. TR-11-22, Univ. Texas at Austin, Dept. Comp. Sci.*, 2011.

[14] K. Petersen, M. Spreitzer, D. Terry, M. Theimer, and A. Demers. Flexible pdate propagation for weakly consistent replication. *In SOSP*, 1997.

[15] M. Raynal, A. Schiper, and S. Toueg. The causal ordering abstraction and a simple way to implement it. *Information Processing Letters, 39, 6*, pages 343–350, 1991.

[16] M. Shen, A. Kshemkalyani, and T. Hsu. OPCAM: Optimal algorithms implementing causal memories in shared memory systems. *Tech. Rep., Univ. Illinois at Chicago, Dept. Comp. Sci.*, 2014.