Performance of Approximate Causal Consistency for Partially Replicated Systems

Ta-yuan Hsu
Dept. of Electrical and Computer Engineering
University of Illinois at Chicago
Chicago, IL 60607-7053, USA
thsu4@uic.edu

Ajay D. Kshemkalyani
Dept. of Computer Science
University of Illinois at Chicago
Chicago, IL 60607-7053, USA
ajay@uic.edu

ABSTRACT

Causal consistency is one of the widely used consistency models in wide-area replicated systems due to highly scalable semantics. Partial replication is a replication mechanism that emphasizes a better network capacity utilization. However, it has a challenge of higher meta-data overhead and processing complexity in communication. Algorithm Approx-Opt-Track has been proposed to reduce meta-data size, however, by risking that causal consistency might be violated. In an effort to bridge this gap and reconcile the trade-off between them, we present the analytic data to show the performance of Approx-Opt-Track. We also give simulation results to show the potential benefits of Approx-Opt-Track, under almost the same guarantees as causal consistency, at a smaller cost. The results indicate that partial replication is a potentially viable alternative to full replication for implementing causal consistency.

CCS Concepts

•**Computing methodologies** → **Distributed algorithms;**
•**Networks** → *Network algorithms; Network simulations;*

Keywords

causal consistency; causality; cloud computing; distributed shared memory; partial replication

1. INTRODUCTION

Large-scale distributed data repositories are commonly used to provide efficient access to huge volumes of data. By replicating across different geographic sites for fault tolerance, this model can protect the data from systematic disasters resulting from correlated failures [9]. It not only reduces the access latency, but also achieves high availability. Reliable and scalable data consistency is one of the most important requirements for designing replication mechanisms and technologies. The CAP theorem [7] states that a distributed data system can achieve at most two of the following three properties: Consistency, Availability, and Partition tolerance. It has been shown that causal consistency is the strongest form of consistency that can also guarantee low latency [13].

Causal consistency for distributed shared memory was introduced by Ahamed et al. [1], who proposed a protocol for implementing it. Later, Baldoni et al. [3] gave an improved protocol. Both these works are Complete Replication and Propagation (CRP) based protocols and do not consider the case of partial replication. Some recent papers [2],[5],[6],[13],[14] have shown how to implement causal consistency in large geo-replicated data storage, also assuming full replication. Although partial replication can avoid taking unnecessary network capacity and hardware resources, it is a challenge to implement partial replication against full replication. This is primarily because of the higher complexity of tracking causal dependencies (e.g., additional communication cost and larger dependency meta-data).

Recently, the first protocol for causal consistency under partial replication, called the Opt-Track protocol, was introduced [16, 15]. This protocol used some of the ideas from causal ordering in message passing systems [12, 10, 4]. We showed that the Opt-Track protocol had the potential of making the meta-data size moderate under partial replication [8].

Approx-Opt-Track [11] has been proposed to further reduce the size of meta-data against Opt-Track. It introduced a parameter called *credits*, which denotes the available hop count associated with a meta-data entry. When the initial *credits* is infinite, Approx-Opt-Track is equivalent to Opt-Track. A small initial *credits* can make a meaningful reduction in the meta-data. However, it decreases the size of meta-data at the cost of violating causal consistency. It is a challenge to balance the trade-off between the meta-data size and causal consistency accuracy in Approx-Opt-Track. Approx-Opt-Track is the first work that uses the idea of credits for causal consistency.

Contributions

In this paper, we quantitatively evaluate the performance of Approx-Opt-Track for implementing causal consistency under partial replication. By controlling initial *credits*, we use simulations to analytically examine the trade-off between initial *credits* and the size of meta-data. We also study the impacts of varying the number of processes and the write rate. With a finite initial *credits* small enough, Approx-Opt-Track is seen to show significant gains over Opt-Track. In particular, for 40 processes, Approx-Opt-Track can lower

ARMS-CC'16, July 29 2016, Chicago, IL, USA

DOI: http://dx.doi.org/10.1145/2962564.2962572

the meta-data size by around 5% $\sim$ 20% without causal violations and from 40% $\sim$ 60% for up to 0.5% causal violations. Thus, we show that Approx-Opt-Track can provide less overhead than Opt-Track at little or no cost of violating causal consistency. It also follows that partial replication is a potentially viable alternative to full replication for providing causal consistency.

Organization

Section 2 outlines the Approx-Opt-Track protocol. Section 3 presents the communication framework for simulating Approx-Opt-Track based on different initial *credits*. Section 4 shows all the experimental results. Section 5 illustrates the trade-off between initial *credits* and the meta-data size. Section 6 gives the conclusion.

2. APPROX-OPT-TRACK PROTOCOL

Opt-Track protocol has a moderately-sized message meta-data [8] in partially replicated systems. Exploiting implicit knowledge about messages delivered in causal order, it makes the meta-data manageable and linear in the number of processes. Approx-Opt-Track protocol has been proposed [11] to further reduce the size of meta-data by deleting older dependencies, introducing the notion of *credits*, *cr*. Whenever a dependency is created, it is assigned a certain number of initial *credits*. Every read and write operation leads to a decrement of the available number of credits. When the available *cr* becomes zero, it means that the associated dependency becomes aged enough and can be reasonably deleted. If the initial *cr* is set to be ∞ in Approx-Opt-Track, the algorithm is equivalent to Opt-Track [15]. By setting the initial *cr* to a smaller finite value, we can remove older dependencies earlier to prune meta-data. However, it imposes cost by risking that those dependencies deleted might not be satisfied, resulting in unexpected outcomes violating causal consistency. The parameter *credits* can help us approximate causal consistency to the accuracy needed.

The data structures in Approx-Opt-Track [11] are very similar to the ones in Opt-Track [15]. The major difference between them is a new parameter *cr* in each entry in LOG_i. Each site s_i needs to maintain the following four data structures.

1. $clock_i$: local counter for write operations performed by application process ap_i.
2. $Apply_i[1{\sim}n]$: an array of integers. $Apply_i[j] = c$ means that a total number of c updates written by application process ap_j have been applied at site s_i.
3. $LOG_i = \{\langle j, clock_j, Dests, cr\rangle\}$: the local log (initially set to empty). Each entry indicates a write operation in the causal past. $Dests$ is the destination list for that write operation. Only necessary destination information is stored. *cr* is the remaining amount of credits.
4. $LastWriteOn_i\langle$variable id h, $LOG\rangle$: This hash map stores the piggybacked LOG from the most recent update applied at site s_i for locally replicated variable x_h.

Formally, we define *credits* of a meta-data entry as the hop count available before the entry gets old enough and is deleted. When a message passes through one hop, it is transmitted along a logical channel from one site/process to another. In the following cases, the hop count *cr* is decremented.

- A piggybacked meta-data for an existing dependency is received.
- A new dependency is identified by a receiving site/process.
- The data is remotely fetched by a read operation.

3. SIMULATION SYSTEM MODEL

We consider an asynchronous distributed system, which is composed of a finite set of asynchronous processes interconnected through a communication network. Formally, they run on multiple sites which are distributed over a wide area in a practical distributed system. To simplify and without loss of generality, we assume that there is only one process on each site. Each site has a local memory and can communicate by asynchronous message passing through TCP channels of the underlying network. The communication network is reliable and transmits messages in FIFO order without omissions or duplications.

3.1 Process Model

A process consists of two major subsystems viz., the application subsystem and the message receipt subsystem. The application subsystem deals with the functionality of scheduling operation events (write/read) including two major functions, viz., $Write$ and *Read*. The message receipt subsystem takes charge of responding to remote request service, including two major functions, viz., $ApplyingMulticast$ and $RespondingFetch$. It also maintains a floating point local clock to generate event patterns based on a temporal schedule.

The simulation system core is based on Algorithm Approx-Opt-Track [11]. For a write operation $w(x_h)v$, the application subsystem delivers the message $m(w(x_h)v)$, the corresponding meta-data $-$ local log L_w, and hop count *credits* to other replicas. For a read operation $r(x_h)$, the application subsystem returns the local variable x_h's value or sends a fetch message $fetch(x_h)$ to a predesignated site to get the remote variable x_h's value as well as the corresponding meta-data $LastWriteOn\langle h\rangle$ and hop count *credit*. The message receipt subsystem monitors two distinct types of incoming messages. First, on receiving a *multicast message* $m(w(x_h)v)$, the receipt subsystem determines when to apply a new value v to the variable x_h in causal consistency and then update the meta-data $LastWriteOn\langle h\rangle$ and hop count *credit*. Second, for a *remote fetch message* $m(fetch(x_h))$, it simply replies with the local value of the variable x_h, the corresponding meta-data $LastWriteOn\langle h\rangle$, and hop count *credit* to the requesting site.

3.2 Simulation Parameters

We examine the effects of the following system parameters on the performance of the Approx-Opt-Track protocol.

- Number of Processes (n): The limitation of the number of processes in the system depends on the processor model and memory allocation of the benchmark device running the simulation. On an Intel Core 2 Duo computer with 16 GB DDR2 and the simulation framework being implemented in JDK8, we can simulate up to 40 processes.

- Number of Variables (q): In a real cloud storage, q is unbound. Due to the memory limitation, q we used in the benchmark experiment is one hundred.
- Replication Factor (p): The number of sites at which each variable is replicated is set to be $0.3 \times n$.
- Write Rate (w_{rate}): It is defined as the ratio of the number of write operations to the total number of operations. Binding by a variety of write rates, we can study performance for read-intensive and write-intensive application workloads.
- Hop Count Credit (cr): Credits of a meta-data entry denote the hop count available before the entry ages out and is removed.
- Message Count (m_c): The total number of messages generated by all the processes.
- Message Meta-Data Size (m_s): The total size of all the meta-data transmitted over all the processes.

3.3 Process Execution

All the processes in the system are symmetric. The operation events are triggered based on a event schedule randomly generated in advance. Due to hardware limitation, the time interval T_e between two events is given from $5ms$ to $2005ms$. The propagation time T_t is from $100ms$ to $3000ms$. (In reality, T_e is initially set to be $10ms \sim 200ms$ and T_t is given $5ms \sim 300ms$ in order to reflect the more realistic propagation time in the world. However, due to our hardware limitation, the shorter time interval and propagation time cannot be realized in a larger number of processes. To seek the next-best thing, we formulate isomorphic communication patterns with the above longer T_e and T_t. The simulation results of these two different time ranges in smaller numbers of processes are not obviously distinct.)

The processes in the distributed system execute concurrently. However, simulating each process as an independent process at a site invoked inter-process communication. When a process gets initialized, it first invokes the message receipt subsystem. Then, the system executes $Scheduled - ExecutorService$ in JDK to drive the application subsystem which extends $TimerTask$ class − a JDK scheduling service to dispose of the scheduling operation events. In the simulation, the system relies on TCP channels to deliver messages. An application subsystem stops generating operation events once it runs out of all the scheduling events and flags its status as finished. The simulation is done when all the application subsystems have their status set to 'finished'.

4. SIMULATION RESULTS

This section presents results of simulations performed to study the trade-off among initial credits cr, message meta-data size m_s, and causal consistency accuracy rate. The performance metrics used are as follows:

- The causal consistency violation error rate.
- The average size of the message meta-data transmitted for different initial credits and *write rates*.
- The message meta-data size saving rate.

Table 1: Critical Initial Credits.

	w_{rate}	the number of processes				
		5	10	20	30	40
$R_e \sim 0.5\%$	0.2	3	3	3	4	4
	0.5	3	3	3	3	3
	0.8	3	3	4	4	4
$R_e = 0$	0.2	5	6	7	8	8
	0.5	3	5	7	7	9
	0.8	4	5	7	8	8

Table 2: Critical Average Message Meta-Data Size m_{ave} (KB).

R_e	w_{rate}	the number of processes				
		5	10	20	30	40
∼0.5%	0.2	0.277	0.330	0.430	0.820	1.037
	0.5	0.345	0.425	0.495	0.562	0.720
	0.8	0.401	0.445	0.640	0.759	0.840
0	0.2	0.312	0.481	0.927	1.566	2.146
	0.5	0.345	0.524	0.899	1.190	1.572
	0.8	0.426	0.558	0.864	1.140	1.361

In order to clarify the relative contribution of these metrics, multivariate analyses were conducted. We report three types of experiments, in each of which we vary the three parameters n, w_{rate}, and cr. For each combination of parameters in each experiment, three runs were performed. The number of processes was varied from 5 up to 40. The w_{rate} was set to be 0.2 (lower write rate), 0.5 (medium write rate), and 0.8 (higher write rate), respectively. The initial hop count credit cr was specified from one to a critical value, with which there is no message transmission violating causal consistency in the corresponding simulation. The variation in the simulation results for the three runs of each combination was less than 2%. The mean of the three runs is represented for each combination. Each simulation execution runs $600n$ operation events totally. Experimental data was stored after the first 15% operation events to eliminate the side effect in startup.

4.1 Violation Error Rate (R_e)

We define n_e as the number of messages applied by the remote replicated sites with a violation of causal consistency. R_e is the ratio of n_e to the total number of transmitted messages m_c. The results for R_e versus different initial hop count credits are shown in Figures 1, 2, and 3. Each of them corresponds to a different w_{rate}.

With increasing cr (less than 4), R_e rapidly decreases. For the same initial cr and w_{rate}, the larger the n, the higher the R_e. The larger the w_{rate}, the lower the R_e. Table 1 highlights the results for two types of critical initial credits (cr_c) when R_e is around 0.5 % (exactly, $0.4\% \sim 0.6\%$) and $R_e = 0$ (no causal consistency violation). When n is larger, the critical initial cr is basically also larger.

4.2 Average Message Meta-Data Size (m_{ave})

Figures 4, 5, and 6 visualize the experimental data of m_{ave}. With increasing initial credit cr, m_{ave} linearly increases. The findings also indicate that m_{ave} becomes smaller with a higher w_{rate} in more peers. Table 2 lists the critical m_{ave} corresponding to the numerical data in Table 1.

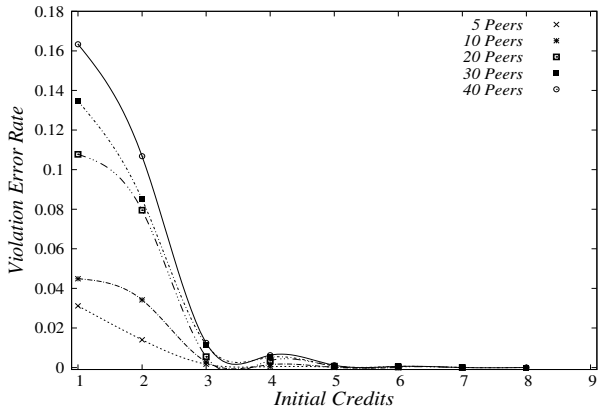

Figure 1: The Violation Error Rate with w_{rate} 0.2.

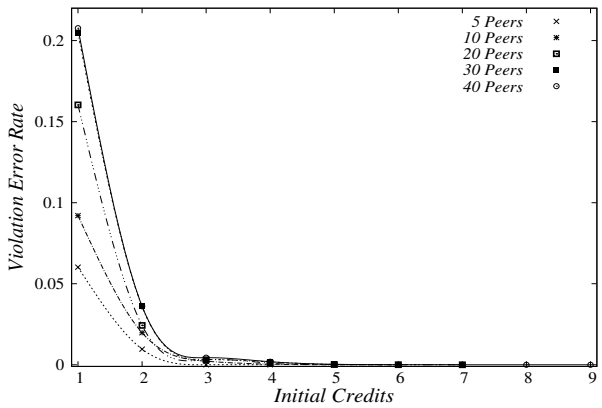

Figure 2: The Violation Error Rate with w_{rate} 0.5.

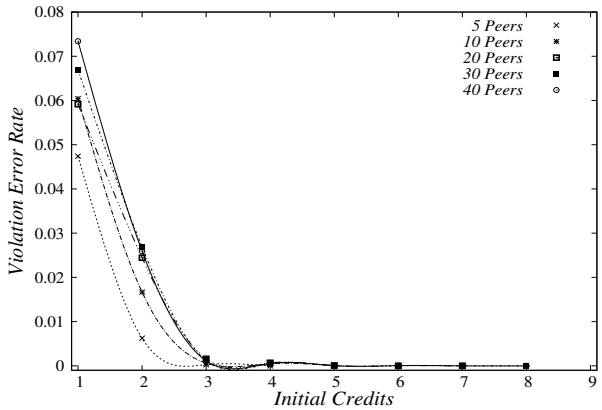

Figure 3: The Violation Error Rate with w_{rate} 0.8.

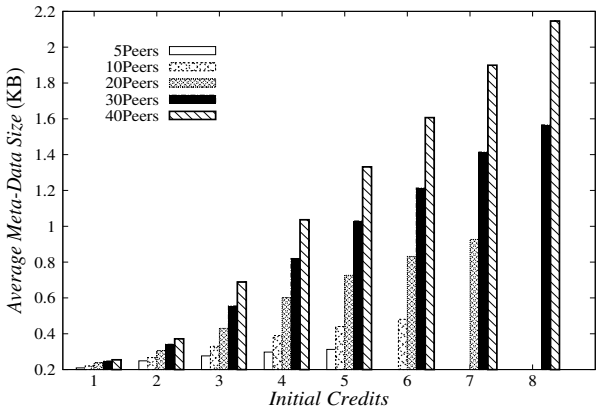

Figure 4: The Average Meta-Data Size (m_{ave}) with w_{rate} 0.2.

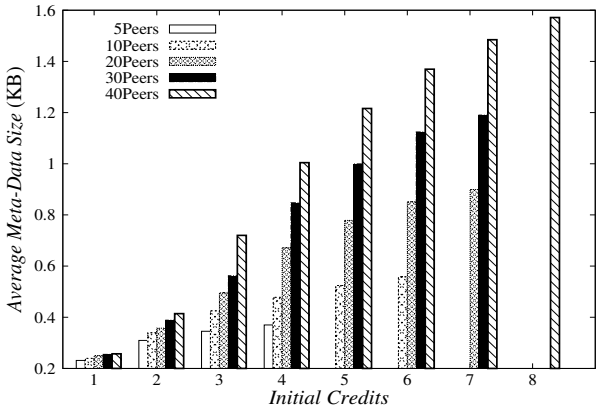

Figure 5: The Average Meta-Data Size (m_{ave}) with w_{rate} 0.5.

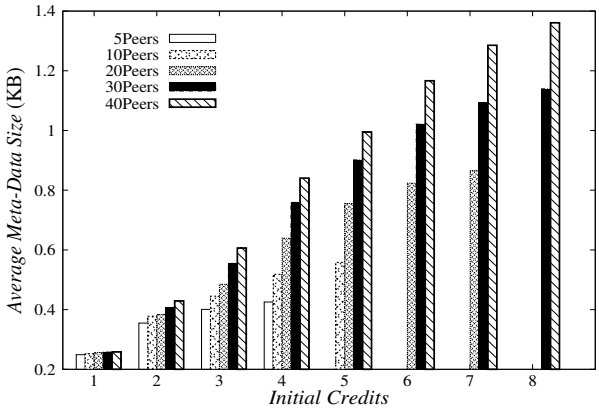

Figure 6: The Average Meta-Data Size (m_{ave}) with w_{rate} 0.8

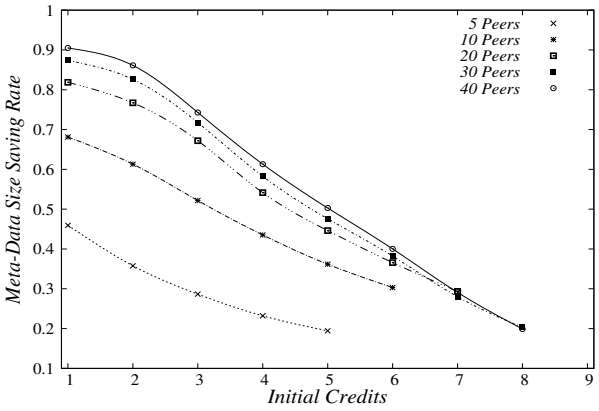

Figure 7: The Meta-Data Size Saving Rate (R_s) with w_{rate} 0.2.

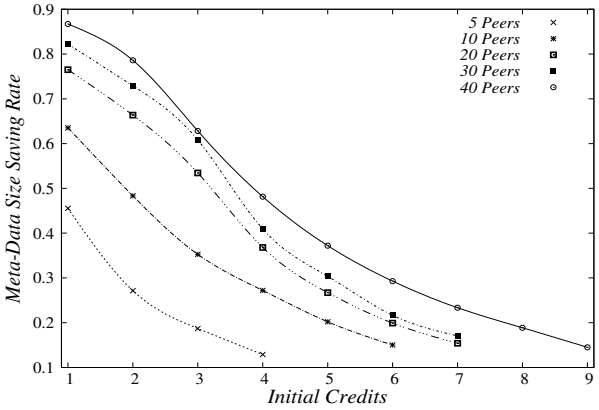

Figure 8: The Meta-Data Size Saving Rate (R_s) with w_{rate} 0.5.

4.3 Message Meta-Data Size Saving Rate (R_s)

Note that Approx-Opt-Track with $cr = \infty$ is equivalent to Opt-Track in [15]. We define R_s as the following.

$$R_s = 1 - \frac{m_s(cr \neq \infty)}{m_s(\mathsf{Opt-Track})} \quad (1)$$

Figures 7, 8, and 9 reflect the results for R_s versus different initial credits in different (low, medium, and high) write rates, respectively. Note that although cr is unbounded in Approx-Opt-Track, it does not make sense for simulation with $cr > cr_c$, in which case there is no message delivery violating causal consistency.

With increasing n, R_s increases. Corresponding to the same n and initial credit cr, the higher w_{rate}, the lower R_s seems to be. Table 3 lists the critical R_s corresponding to the numerical data in Table 1. It can be seen that R_s is significantly negatively related to w_{rate}.

5. DISCUSSION

We expect that if the initial allocation of hop count cr is a small finite value but enough, it not only reduces message meta-data size, but also maintains the desired causal consistency accuracy. In other words, it is expected with very high probability that when cr reaches zero so as to delete the

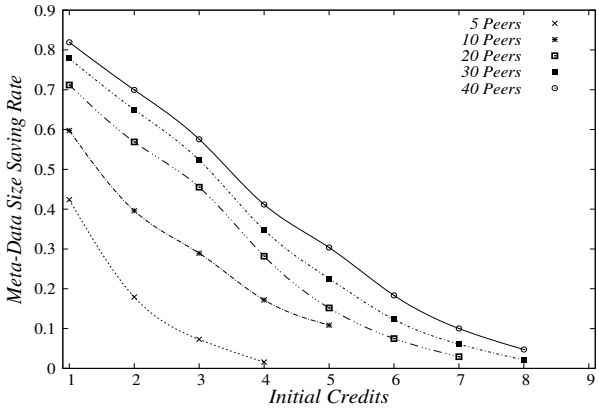

Figure 9: The Meta-Data Size Saving Rate (R_s) with w_{rate} 0.8.

Table 3: Message Meta-Data Size Saving Rates R_s in R_e close to or equal to zero.

R_e	w_{rate}	the number of processes				
		5	10	20	30	40
∼0.5%	0.2	0.287	0.521	0.672	0.582	0.613
	0.5	0.187	0.352	0.534	0.608	0.628
	0.8	0.073	0.289	0.282	0.348	0.412
0	0.2	0.194	0.303	0.294	0.203	0.198
	0.5	0.187	0.202	0.154	0.171	0.145
	0.8	0.016	0.108	0.029	0.021	0.047

corresponding entry, the corresponding message has reached its destination.

5.1 Impact of initial cr on R_e

With increasing the initial cr, R_e rapidly decreases, especially when $cr < 4$. The smaller the initial cr, the earlier the meta-data entry is deleted. Thus, when an entry is removed earlier, the message associated with the deleted entry might not reach its destination more likely. It causes that the corresponding dependency may not be satisfied, resulting in higher R_e. Table 1 shows the major and minor critical initial credits – cr_0 ($R_e = 0$) and $cr_{\sim 0.5\%}$ ($R_e = 0.4\% \sim 0.6\%$) – for different numbers of processes.

Furthermore, $cr_{\sim 0.5\%}$ seems not to significantly increase as the number of processes n. It implies that by setting the initial cr to a small finite value but enough value, most of the dependencies associated with the meta-data will become aged and can be removed without risking causal violations after being transmitted across a few hops, even in a large number of processes. On the other hand, in $R_e = 0$ (no causal violations), the resulting correlation coefficients of cr_0 and n for different w_{rate} are around $0.94 \sim 0.95$. It means that the more the number of processes n, the larger the initial cr is to avoid causal violations.

5.2 Impact of w_{rate} on R_e

This section evaluates how different write rates influence the violation error rates R_e across a variety of n and cr. Figures $1 \sim 3$ show the results of R_e among different w_{rate} in smaller initial *credits* over different process numbers. Table 1 summarizes the values of cr_0 and $cr_{0.5\%}$ from the results of R_e in Figures $1 \sim 3$ and shows that w_{rate} does not have

an apparent impact on cr_0 and $cr_{\sim 0.5\%}$. However, when initial $cr < 4$, the higher the w_{rate}, the lower the R_e. Causal consistency follows *read-from order* $\prec_{ro}$. Two operations o_1 and o_2 have the relationship $o_1 \prec_{ro} o_2$ if there exists $o_1 = w(x)v$ (write a value v into variable x) and $o_2 = r(x)$ (read the value from variable x) such that operation o_2 retrieves the value stored by operation o_1. When the initial cr is a smaller value, dependencies might not be satisfied with higher probability. The higher the read rate r_{rate} (i.e., w_{rate} is lower), the more likely *read-from* relation occurs.

5.3 Impact of initial cr on m_{ave}

Figures 4 $\sim$ 6, each of which reflects a certain w_{rate}, illustrate experimental results for average message meta-data size m_{ave} in different initial *credits* with respect to varying n. m_s is linearly proportional to initial cr, because we only carried out the experiments under initial *credits* $\leq$ critical cr_0.

For a certain combination of n and initial cr, m_s decreases as w_{rate} increases. The reason is the same as shown in [8], due to fewer $MERGE$ and more $PURGE$ operations in Opt-Track [15] and Approx-Opt-Track [11]. A read operation will invoke a $MERGE$ function to merge the piggybacked log of the corresponding write to that variable with the local log LOG. Thus, a higher read rate may increase the likelihood that the size of explicit information becomes larger. Furthermore, although a write operation results in the increase of explicit information, it comes with a $PURGE$ function to prune the redundant information, so that the size of LOG could be decreased. Therefore, a higher write rate with a corresponding lower read rate causes fewer MERGE and more PURGE operations generated.

Table 2 lists the analytic data about m_{ave} in cr_0 and $cr_{\sim 0.5\%}$. For the case of 10 processes, $m_s(10)$ is around 0.48KB $\sim$ 0.56KB for $R_e = 0$. For the case of 20 processes, $m_s(20)$ is around 0.86KB $\sim$ 0.93KB for $R_e = 0$. For the case of 40 processes, $m_s(40)$ is around 1.36KB $\sim$ 2.15KB for $R_e = 0$. Consider $w_{rate} = 0.5$ and $w_{rate} = 0.8$. Using cross-comparison analyses, $m_s(40)/\ m_s(20)/\ m_s(40)$ is less than $m_s(10)\times 4/\ m_s(10)\times 2/\ m_s(20)\times 2$. The results reflect the better scalability of Approx-Opt-Track in higher w_{rate} under no risk of violating causal consistency.

5.4 Impact of initial cr on m_s

This section reports the effect of Approx-Opt-Track to reduce the meta-data overheads under causal consistency. As mentioned before, the meta-data for dependencies could be reduced at the cost of some violations of causal consistency. We intend to study the exact nature of the trade-off between m_s and R_e. We expect to find a finite initial cr small enough that can not only reduce the meta-data size, but also have the system fully follow the causal consistency.

According to equation (1), R_s decreases as m_s increases, which positively depends on initial credits cr. Figures 7 $\sim$ 9 present the linear relationships between m_s and initial cr under initial credits $< cr_0$ (or $cr_{\sim 0.5\%}$, called 'unsatisfied state'). R_s is determined by cr, n, and w_{rate} in the unsatisfied state.

For the same number of processes, the curves of R_s versus cr shift downward as w_{rate} increases. It implies that, in unsatisfied states, m_s increases more slowly than $m_s(cr = \infty)$ does as r_{rate} rises. This is because there are more $MERGE$ operations to delete meta-data entries in higher r_{rate} (lower w_{rate}).

Table 3 summarizes the details of numerical data about R_s in the major and minor critical initial *credits*. For the case of 40 processes, R_s is around 40% $\sim$ 60% at a very slight cost of violation of causal consistency ($R_e \sim 0.5\%$). R_s reaches around 5% $\sim$ 20% without violating causality order in terms of different write rates. This evidence proves that if the initial allocation of cr is a small but enough digit, the message about which the corresponding meta-data is deleted would already have reached its destination very likely. From the above comprehensive analyses, Approx-Opt-Track shows a better network capacity utilization than Opt-Track without causing additional causal violations.

6. CONCLUSIONS

We considered the problem of providing causal consistency protocols in partially replicated systems. Approx-Opt-Track has been proposed and showed theoretically its potential to improve the meta-data size of Opt-Track. However, there is a trade-off between m_s (or initial cr) and R_e. This paper conducted a performance trade-off analysis of R_e, m_s, and R_s using multi-scale simulations.

The simulation results showed that by controlling initial cr, we can trade-off the level of potential causal consistency inaccuracy by the meta-data size. By setting a small finite initial cr, most of dependencies turn out to be aged after being transmitted across a few hops. For various numbers of processes, varying from 5 to 40, the minor critical initial *credits* ($cr_{\sim 0.5\%}$) are around 3 $\sim$ 4 with a very low R_e, which is close to 0.5%. It concludes that if the initial allocation of cr is made as a finite single-digit, by the time the cr reaches zero, the message corresponding to the meta-data would reach its destination with very high probability.

With a finite initial cr small enough, Approx-Opt-Track was also seen to show significant gains over Opt-Track. In particular, as for 40 processes, Approx-Opt-Track can lower the meta-data size by around 5% $\sim$ 20% without causal violations and from 40% $\sim$ 60% for up to 0.5% causal violations. Thus, we showed that Approx-Opt-Track can provide less overhead than Opt-Track at little or no cost of violating causal consistency. Coupled with the analysis [11] about the net message sizes (payload + meta-data size) under partial replication versus full replication, our results also suggest that partial replication is a viable alternative to full replication for causal consistency in distributed systems.

Our results assumed a system model of distributed shared memory similar to that used by Ahamad et al. [1], Baldoni et al. [3], [15], [8], and [11]. Currently deployed geo-replicated cloud platforms, which do only full replication and no partial replication, use a two-level storage architecture which is a different model. Future work would entail adapting the Opt-Track and Approx-Opt-Track algorithms for causal consistency under partial replication to this architecture.

7. REFERENCES

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

[2] S. Almeida, J. a. Leitão, and L. Rodrigues. Chainreaction: A causal+ consistent datastore based

on chain replication. In *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, pages 85–98, New York, NY, USA, 2013. ACM.

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

[4] 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):40–52, 2004.

[5] J. Du, S. Elnikety, A. Roy, and W. Zwaenepoel. Orbe: Scalable causal consistency using dependency matrices and physical clocks. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 11:1–11:14, New York, NY, USA, 2013. ACM.

[6] J. Du, C. Iorgulescu, A. Roy, and W. Zwaenepoel. Gentlerain: Cheap and scalable causal consistency with physical clocks. In *Proceedings of the ACM Symposium on Cloud Computing, Seattle, WA, USA, November 03 - 05, 2014*, pages 4:1–4:13, 2014.

[7] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, June 2002.

[8] T. Y. Hsu and A. D. Kshemkalyani. Performance of causal consistency algorithms for partially replicated systems. In *IPDPS Workshops*, pages 525–534. IEEE, 2016.

[9] I. Iliadis, D. Sotnikov, P. Ta-Shma, and V. Venkatesan. Reliability of geo-replicated cloud storage systems. In *Dependable Computing (PRDC), 2014 IEEE 20th Pacific Rim International Symposium on*, pages 169–179, Nov 2014.

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

[11] A. D. Kshemkalyani and T.-Y. Hsu. Approximate causal consistency for partially replicated geo-replicated cloud storage. In *Proceedings of the Fifth International Workshop on Network-Aware Data Management*, NDM '15, pages 3:1–3:8, New York, NY, USA, 2015. ACM.

[12] A. D. Kshemkalyani and M. Singhal. An optimal algorithm for generalized causal message ordering. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '96, pages 87–, New York, NY, USA, 1996. ACM.

[13] 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 *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 401–416, New York, NY, USA, 2011. ACM.

[14] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. Stronger semantics for low-latency geo-replicated storage. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13, pages 313–328, Berkeley, CA, USA, 2013. USENIX Association.

[15] M. Shen, A. D. Kshemkalyani, and T. Y. Hsu. Causal consistency for geo-replicated cloud storage under partial replication. In *IPDPS Workshops*, pages 509–518. IEEE, 2015.

[16] M. Shen, A. D. Kshemkalyani, and T. Y. Hsu. OPCAM: optimal algorithms implementing causal memories in shared memory systems. In *Proceedings of the 2015 International Conference on Distributed Computing and Networking, ICDCN 2015, Goa, India, January 4-7, 2015*, pages 16:1–16:4, 2015.