Consistency or Latency? A Quantitative Analysis of Replication Systems Based on Replicated State Machines

Xu Wang, Hailong Sun, Ting Deng, Jinpeng Huai
School of Computer Science and Engineering
Beihang University
Beijing, China
Email: {wangxu, sunhl, dengting, huaijp}@act.buaa.edu.cn

***Abstract*—Existing theories like CAP and PACELC have claimed that there are tradeoffs between some pairs of performance measures in distributed replication systems, such as consistency and latency. However, current systems take a very vague view on how to balance those tradeoffs, e.g. eventual consistency. In this work, we are concerned with providing a quantitative analysis on consistency and latency for widely-used replicated state machines(RSMs). Based on our presented generic RSM model called RSM-d, probabilistic models are built to quantify consistency and latency. We show that both are affected by d, which is the number of ACKs received by the coordinator before committing a write request. And we further define a payoff model through combining the consistency and latency models. Finally, with Monte Carlo based simulation, we validate our presented models and show the effectiveness of our solutions in terms of how to obtain an optimal tradeoff between consistency and latency.**

Keywords*—consistency; write conflict; latency; replicated state machine.

I. Introduction

Practical distributed systems in Cloud and Big Data solutions often face many challenges such as high scalability and availability, which are built on massive commodity computers, disks, network devices and under complex management tasks [1,2]. To serve more users and provide high enough availability, services and data are typically replicated across multiple virtual machines(VMs), physical machines and even geographically-distributed clusters.

The CAP theorem [3] shows the impossibility of obtaining all three of consistency, availability and partition tolerance simultaneously. And PACELC [4] further interprets CAP, which claims two tradeoffs including consistency&availability and consistency&latency. On the one hand, people make a tradeoff between consistency and availability. In order to provide extremely high availability, many systems are always writable with eventual consistency, such as Dynamo [5] and Cassandra [6]; while others, like Chubby [7] and Zookeeper [8], must experience a short period of unavailability to recover from failures for guaranteeing strong consistency of different replicas. On the other hand, people focus on the tradeoff between consistency and latency. For example, eventually consistent reads, no matter how fast they are, can not bound the staleness as the newest versions will be eventually returned [9]; and strong consistency leads to higher latency for writes due to the write uniformity [10]. Although weak consistency has been used in some commercial systems and is acceptable by many practitioners for higher availability and lower latency, it is necessary to bound the inconsistency and to know how inconsistency the weak consistency is.

Since latency impacts the end-user experience of an Internet application, it has become an important system metric for modern service providers [11]. Some statistical results presented in [12] show that end-users are very sensitive to system latency. For instance, at Microsoft Bing, a 2-second slowdown reduces queries/user by 1.8% and revenue/user by 4.3%; a latency increase from 400ms to 900ms with Google search results in a 25% dropoff in page searches; and Amazom.com has a 1% drop in sales with a 100ms latency increase [13]. Therefore latency is an important factor in system design. In this work, we focus on the tradeoff between consistency and latency in distributed replication protocols.

Two popular fault-tolerance approaches in distributed systems and databases are replicated state machines(RSMs) [14] and quorum systems [15]. RSMs describe desirable replication semantics to make operations committed in total order. Quorum systems define sets of replicas W and R, where W is used for write and R for read. And they commit writes by vector clocks [16] with semantic reconciliation [5] in causal order. For quorum-like systems, several works [17,18] bound read staleness from the aspects of data versions, stale time and staleness probability to describe the inconsistency, and then discuss the tradeoff between consistency and latency. However, little has been done on quantifying inconsistency and latency for RSMs, while they are designed to provide relatively stronger consistency than quorum systems. For read operations of RSMs, we can borrow analysis techniques from the quantitative read staleness in quorum systems. But for write operations of RSMs, we need a new analytical and quantitative method to measure the degree of write inconsistency. Then we can rationally make a tradeoff between consistency and latency for RSMs based on the quantitative result of write inconsistency.

Although there are some existing alternative models to describe write inconsistency in the absence of crash failures, e.g., write consistency constraint [19,20] and limited write divergency of replicas [21,22], we want to know the degree of write inconsistency for RSMs when encountering crash

978-1-4799-0181-4/13/$31.00 ©2013 IEEE

failures, rather than how to satisfy constraints or bound the maximum deviation of a data item. Therefore we quantify write inconsistency for RSMs by the probability of write conflict which represents the probability of a write committed in an abnormal order.

In this paper, we present a quantitative probabilistic analysis approach to measuring the consistency and latency of RSMs, and further provide a solution for making tradeoff between consistency and latency to achieve the maximal system benefit. First, we have surveyed and analyzed three representative RSM implementations including non-uniform total order broadcast [23,24], distributed consensus(Paxos) [25,26] and uniform total order broadcast [23,24], as well as their variants [27-29]. Then we propose a generic system model RSM-d, where $d \in [1, n]$ (n is the number of replicas) represents the number of ACKs that must be received before committing a write. RSM-d can describe the three representative ones where $d = 1, [n/2]+1, n$ respectively. Second, we measure the write inconsistency of RSM-d based on a probabilistic model. The write inconsistency is quantified by the probability of write conflict which represents the probability of any write committed in an abnormal order. Our probabilistic model shows that (1) consistency increases when d rises if $d \in [1, [n/2]+1]$ and (2) strong consistency is always guaranteed if $d \in [[n/2]+1, n]$. Third, we evaluate the latency of RSM-d through calculating its expectation which shows that the latency strictly monotonically increases with respect to d. Fourth, combining the quantitative results of write inconsistency and latency of RSM-d, we provide a solution for tradeoff between consistency and latency to achieve the maximal system benefit. Finally, through Monte Carlo based event driven simulations, we validate our quantitative results, and show the effectiveness of our presented solution for tradeoff between consistency and latency from the aspect of overall system benefit.

We make the contributions as follows:

- We present a generic system model RSM-d for RSMs to give an unified description of major replication protocols using RSMs;
- On the basis of RSM-d, we build a probabilistic model for write conflict that is one of the key factors to cause inconsistency, and a probabilistic model for characterizing latency as well. We find out that both consistency and latency are affected by the common factor d.
- We further define a payoff model to make tradeoff between consistency and latency for achieving the maximum system benefit through combining the write conflict model and latency model from the perspective of a service provider;
- With Monte Carlo based event driven simulations, we validate our quantitative results and show the effectiveness of our presented solutions in terms of how to obtain an optimal tradeoff between consistency and latency.

The remainder of this paper is structured as follows. Section II introduces the background. Section III presents our system model. Section IV describes how to quantify the probability of write conflict. The calculation of latency is presented in Section V. Section VI shows the tradeoff of consistency and latency. Section VII provides our experimental results. Related work and discussion are presented in Section VIII and Section IX, respectively. Finally, Section X concludes.

II. Background

In this section, we present the background of RSMs.

RSMs are widely used in distributed systems with the target to tolerate failures. Failure modes fall into non-Byzantine (fail-stop, crash and message loss) and Byzantine (signed messages or not). With the complexity that at least $3f + 1$ nodes for tolerating f Byzantine failures and the low occurrence probability of Byzantine failures, common RSMs only consider non-Byzantine ones. And message loss failures can be easily eliminated by network protocols, so practitioners usually say that RSMs can tolerate crash failures.

Chandra and Toueg [30] have proved that the total order broadcast and the distributed consensus problems are solvable and equivalent to each other under $\diamond W$ failure detectors and at most $[n/2]$ simultaneous crash failures. In this work, we also follow these assumptions. It means that we discuss RSMs which tolerate up to f crash failures with $2f + 1$ replicas and can eventually detect the crash of nodes. However, practical failure detectors must output a result (crash or not) in a bounded time other than a long-waiting for eventual conclusions. [31] classifies the quality of service of practical failure detectors into two types: speed, *i.e.*, how fast a failure detector detects a crash; and accuracy, *i.e.*, how well it avoids false suspicion. Therefore practical failure detectors of RSMs have a probability to falsely suspect on a normal coordinator.

As surveyed in [23-29], RSMs are typically implemented by non-uniform total order broadcast, distributed consensus and uniform total order broadcast. As shown in Figure 1, the three representative RSM implementations work as follows:

(1)In non-uniform total order broadcast, once a coordinator receives a write, it assigns the write a unique sequence number and then directly commits it to all replicas. When the write arrives at a replica, the replica will record the write, and commit it according to its sequence number and then respond to the coordinator. Once the coordinator has received the first response, it will reply to the client. Note that any time a coordinator is suspected as crashed by failure detectors, the new coordinator will be elected [32] and recovers failures based on the historical commit records.

(2)For distributed consensus based RSMs, once a coordinator receives a write, at first it propagates the write with the given sequence number to all replicas. After logging the write, every replica will send an acknowledgement(ACK) message to the coordinator. Until at least $[n/2] + 1$ ACKs (including the coordinator itself) are received by the coordinator, it will commit the write. When the coordinator is suspected as crashed by failure detectors, the new coordinator will be elected and recover failures based on the historical log records (not the historical commit records).

(3)Uniform total order broadcast is very similar to distributed consensus, but the coordinator must wait for all n ACKs to be returned.

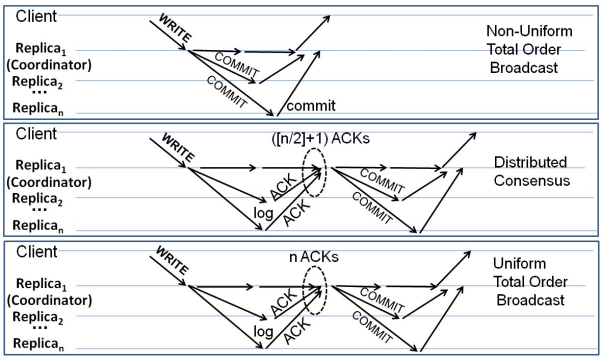

Fig. 1. Three Representative RSM Implementations

Assuming without loss of generality the coordinator sends itself an ACK message, we can discover that an obvious difference among the three representative RSM implementations is the number of ACKs (denoted by d) that the coordinator waits for before committing a write. And the values of d for non-uniform total order broadcast, distributed consensus and uniform total order broadcast are 1,$\lfloor n/2 \rfloor + 1$ and n, respectively. Thus we can easily conclude that the latencies of them from low to high are non-uniform total order broadcast, distributed consensus and uniform total order broadcast. In terms of consistency, distributed consensus(Paxos) has been showed its safety(strong consistency) in [33]. That is, RSMs are always consistent if $\lfloor n/2 \rfloor + 1 \leq d \leq n$. However, for non-uniform total order broadcast, if a committing write is lost in historical commit records due to crash failures(*i.e.*, all committed nodes including the coordinator crash) or inaccurate failure detection on the coordinator happens, the sequence number held by the write or used by the falsely suspected coordinator will be reused to keep system liveness. These will lead to write conflict and then result in write inconsistency.

III. RSM-d: A Generic RSM Model

In this section, we will present our system model RSM-d, which is the abstraction of RSM implementations. Since our desired system model should cover the three representative RSMs, we have surveyed them and try to derive the system model.

Although we present (non-uniform and uniform) total order broadcast algorithms in section II, they are only one of five classes. [23] provides a comprehensive survey for total order broadcast, considering all five classes of ordering mechanisms and both non-uniform and uniform algorithms. The first three ordering mechanisms: fixed sequencer (shown in section II), moving sequencer and privilege-based are built based on a fixed sequencer or a moving token; The other two: communication history and destination agreement are implemented based on a total order logical clock [16]. And the solution of distributed consensus for RSM is the same with the destination agreement, where Paxos is used to make agreement on the order for a write, and multi-paxos [25] for all writes. Essentially we add a 'sequencer' which can compare the orders of any two writes to causal order, and then gain total order.

Based on the analysis above, we are concerned with the total order broadcast with the fixed sequencer and the Paxos algorithm. The reasons are: (1)the fixed sequencer algorithm has the lowest latency among all total order broadcast ones (on point-to-point networks) [24]; (2)Paxos is proved to be optimal in general consensus algorithms [34];(3)both the token in the moving sequencer and privilege-based algorithms, and the logical clock in the communication history and destination agreement algorithms, can be seen as "another fixed sequencer".

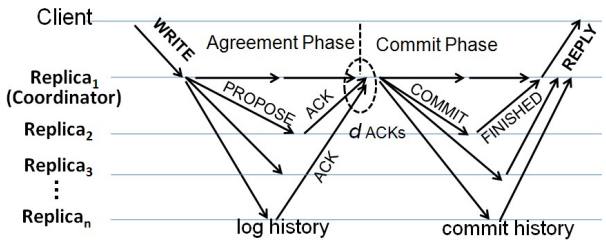

Fig. 2. RSM-d Model

Then we combine total order broadcast with the fixed sequencer and the Paxos algorithm to an unified system model RSM-d, which is feasible since total order broadcast and distributed consensus are essentially equivalent to each other [30]. As depicted in Figure 2, RSM-d includes the agreement phase and the commit phase. In the agreement phase, any write must be broadcast to at least d replicas and be logged, where $d \in [1, n]$. These d replicas include the coordinator itself as well, that is, the coordinator can send "COMMIT" messages for a write once it receives $d-1$ ACKs. In the commit phase, each commit will be recorded to the commit history, and the coordinator will reply to the client when the first result is returned. If $d = 1$, RSM-d degrades to non-uniform total order broadcast, where the coordinator can directly enter the commit phase; if $d = \lfloor n/2 \rfloor + 1$, RSM-$d$ represents the Paxos algorithm; and if $d = n$, RSM-d changes to uniform total order broadcast. Not only the three representative RSMs are studied, but we will also go deep into other RSMs ($d \neq 1, \lfloor n/2 \rfloor + 1$ or n) which are ignored in past.

RSM-d, as the abstraction of RSM implementations, also follows the assumptions of RSMs. It assumes that the number of simultaneous crash nodes (denoted by f) is no more than [n/2], *i.e.*, $1 \leq f \leq \lfloor n/2 \rfloor$, which guarantees its solvability. In addition, once a failure detector suspects the crash of the coordinator, it will start to elect a new coordinator and recover failures only based on historical records (log and commit history). Note that multiple coordinators are allowed in RSM-d such as Multi-Ring Paxos [29] since it is compatible with Paxos.

IV. Probability of Write Conflict

In RSM-d, if ACK number $d \geq \lfloor n/2 \rfloor + 1$, all writes will be committed in their normal and sequential orders. Even there are $f (f \leq \lfloor n/2 \rfloor)$ crash failures occur, after a short time of coordinator election and failure recovery based on historical records on all nodes alive, replicas will continuously keep consistency. However, if $d < \lfloor n/2 \rfloor + 1$, a committing write may disappear in historical records because of crash failures, or inaccurate failure detection on the coordinator happens,

TABLE I. SYMBOLS IN SECTION IV

Symbols	Definition
d	the number of ACK messages
n	the number of all replicas
f	the number of simultaneous crash nodes
P_{wc}	the probability of write conflict for a write
P_{wl}	the probability of write lost in historical records (log and commit history)
P_{wd}	the probability of write duplication produced by inaccurate failure detection on the coordinator
P_c	the crash probability of any replica
T	the stochastic variable of message delay between any two replicas
$f(t)$	the probability density function of T
$F(T)$	the cumulative distribution function of T

then the sequence number held by the write or used by the falsely suspected coordinator will be reused to keep system liveness. These will lead to write conflict. In this section, we will analyze when and how RSM-d produces write conflict and then calculate the probability of write conflict which represents the probability of any write committed in an abnormal order.

First of all, we provide some definitions in Table I, where crash events of different nodes and message delays between replicas are independent.

It is obvious that $P_{wc} = 0$ due to the safety of Paxos if $d \geq [n/2] + 1$. But something is complicated about write conflict if $d < [n/2] + 1$. Thus we give three assumptions to simplify the computation of P_{wc} as follows:

(a) failure detectors are always 100% accurate;
(b) "PROPOSE" messages in RSM-d are sent to exact d fastest nodes (including the coordinator);
(c) coordinator election and failure recovery do not consider the commit history in the commit phase.

After the simplification, we can easily get a basic probability of write conflict. Then we relax these three assumptions step by step. Every relaxing will result in a new but more accurate value of P_{wc}, thus we can achieve the final computing result till above three assumptions are all discarded.

A. Write Log Loss

Under assumptions (a),(b) and (c), a write conflict occurs only if the d nodes with write log crash. Because the coordinator is one of these d nodes, its crash will be detected by completely accurate failure detector of other nodes. Then the following coordinator election and failure recovery can not see the write in log history if all d nodes with its log crash, otherwise the write will be found. The sequence number held by the unseen write will be reused in a coming new write, thus these two writes employ a same sequence number and have a conflict.

Figure 3 shows the scenario of write log loss, when at least the d nodes that receive "PROPOSE" messages and log write fastest have crashed. Let P_{wll} be the probability of all log loss of a write. If at least the fastest d nodes with write log crash, considering assumptions (b), thus $P_{wll} = (P_c)^d$.

Since historical records only include log history, P_{wll} is also the probability of a write lost in historical records. And false suspicion on the coordinator does not exist because of

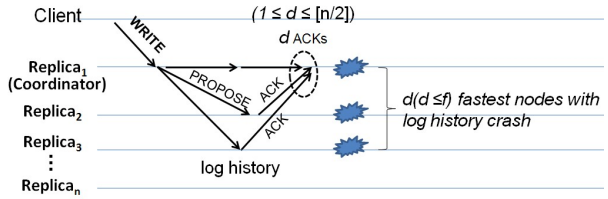

Fig. 3. Write Log Loss

the 100% accurate failure detectors. So at present:

$$P_{wc} = P_{wll} = (P_c)^d \tag{1}$$

B. Non-uniform Write

Now we give up assumption (c) but assumptions (a) and (b) are still reserved. This means that we permit committed writes produce new information (commit history) in historical records, not only the log history. Under this circumstance, only when all committed and all logged nodes crash simultaneously, the write will disappear in historical records. The scenario of all committed writes crashed has been long studied [23] in non-uniform total order broadcast, so we call it "non-uniform write". And P_{wnu} denotes the probability of non-uniform write.

As shown in Figure 4, if $d(d \leq f)$ fastest nodes with log history and $(f - d)$ fastest committed nodes with commit history crash, a similar write conflict will occur with write log loss. P_{wnu} can be calculated by:

$$P_{wnu} = \sum_{k=0}^{[n/2]-d} (P_c)^k (1 - P_c)^{n-d-k}$$

where k denotes the number of fastest $(f-d)$ committed writes (excluding the d logged replicas) before crashes and its upper bound is $[n/2] - d$. Since write conflict requires all committed and all logged nodes crash simultaneously, a new formula of P_{wc} is :

$$P_{wc} = P_{wll} P_{wnu} = (P_c)^d \sum_{k=0}^{[n/2]-d} (P_c)^k (1 - P_c)^{n-d-k} \tag{2}$$

C. Expanding Logged Writes

At present we further give up assumption (b) and only keep assumption (a). Thus we should consider asynchronously

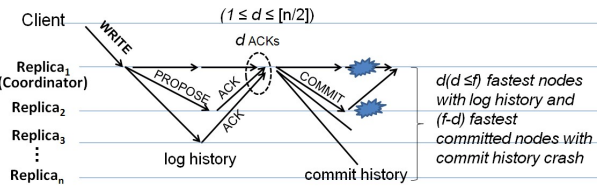

Fig. 4. Non-uniform Write

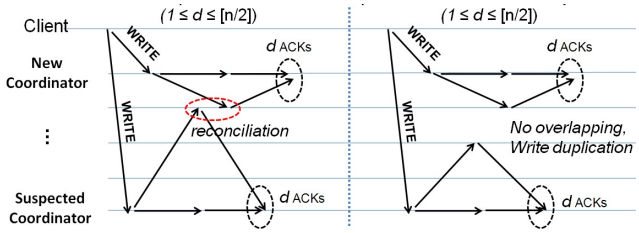

Fig. 5. Write Duplication

propagating "PROPOSE" messages and maybe more than d replicas have logged writes before crashes. And the expansion of the logged writes set may be accelerated by anti-entropy processes [35]. Therefore, we employ an empirical method to detect the number of logged writes prior to crash failures.

Stochastic variable D denotes the number of logged writes when crashes occur, and its empirical probability density function is $P_{elw}(D)$ by stochastic statistics. For convenience, we define a function:

$$Q(x) = (P_c)^x \sum_{k=0}^{[n/2]-x} (P_c)^k (1-P_c)^{n-x-k}$$

where $x \in [d, n]$. Obviously, equation (2) can be rewritten as $P_{wc} = Q(d)$. Similarly, for each $D = m(d \leq m \leq [n/2])$, the conditional probability of write conflict when $D = m$ is $Q(m)$. Thus we can get P_{wc} by summing the conditional probability for each possible $D = m$:

$$\begin{aligned} P_{wc} &= \sum_{m=d}^{[n/2]} P_{elw}(m) Q(m) \qquad (3) \\ &= \sum_{m=d}^{[n/2]} P_{elw}(m) (P_c)^m \sum_{k=0}^{[n/2]-m} (P_c)^k (1-P_c)^{n-m-k} \end{aligned}$$

As described above, equations (1),(2) and (3) actually represent the probability of that a write is lost in historical records (all of d log history lost, commit history lost and asynchronously propagating write log lost, denoted by P_{wl}) during coordinator election and failure recovery under different assumptions. We view them as the probability of write conflict because a write loss will yield a write conflict, for the consideration of system liveness and reuse of sequence numbers. That is:

$$\begin{aligned} P_{wc} &= P_{wl} \\ &= \sum_{m=d}^{[n/2]} P_{elw}(m) Q(m) \qquad (4) \end{aligned}$$

D. Write Duplication

If we continue to give up assumption (a), failure detectors act closely to practical ones. They will have a probability to falsely suspect the normal coordinator, and then elect a new coordinator to reuse a sequence number which may has been used even the suspected coordinator are normally committing a write with the same sequence number. This will turn out the duplication of writes and result in inconsistency.

Figure 5 shows the scenario of write duplication. When an inaccurate failure detector have a false suspicion on the coordinator, the new coordinator will propose new writes with sequence numbers which may have been used by the suspected one. This leads to that two coordinators simultaneously propose two different writes in a same order. [25] tells that, if nodes which receive both writes exist, these nodes can reconcile them through a piggyback message on the second ACK like this "The sequence number has been used, please try another!". Apparently that at least one reconciling node exists if $d \geq [n/2] + 1$. However, if $d \leq [n/2]$, the two nodes sets that respond ACKs may have no overlapping. Then both coordinators will think their own write having an unique order and commit them, which incurs write duplication and conflict.

Let P_{no} be the probability of no overlapping of two nodes sets responding ACKs. Besides both coordinators themselves, other two random $(d-1)$ nodes sets may not overlap and are all different nodes. Thus, P_{no} is the number of two random $(d-1)$ nodes sets that have no overlapping and exclude both coordinators divided by the number of all two random $(d-1)$ nodes sets:

$$P_{no} = \frac{\binom{n-2}{d-1}\binom{n-d-1}{d-1}}{\binom{n-1}{d-1}\binom{n-1}{d-1}}$$

Now we will discuss why failure detectors make mistakes. Although some works [31,36] propose more adaptive and faster failure detection algorithms, a common failure detector, which is used in many practical systems, as depicted in Figure 6, works as follows:

1) at interval T_e a coordinator propagates heartbeat messages to a replica;
2) when the replica receives a heartbeat message, it trusts that the coordinator is working normally and starts a timer with a timeout value of T_o;
3) if the replica has not received a new heartbeat message before the timer expires, it begins to suspect the coordinator and to consider the coordinator has crashed.

Recall that stochastic variable T denotes the message delay between the coordinator and the replica. Its probability density function and cumulative distribution function are $f(t)$ and $F(t)$, respectively. And T_{hb2} and T_{hb1} are its two samples, which represent the message delays of two successive heartbeat messages. "The replica has not received a new heartbeat message before the timer expires" means that $T_{hb2} + T_e > T_{hb1} + T_o$. Therefore, even the coordinator works, if $T_{hb2} + T_e > T_{hb1} + T_o$, the replica still thinks it has crashed. The probability of such false suspicion between two nodes is denoted by P_{fs}:

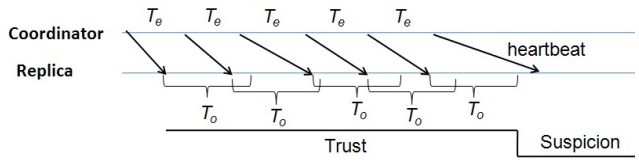

Fig. 6. Failure Detection

$$\begin{aligned} P_{fs} &= Pr(T_{hb2} + T_e > T_{hb1} + T_o) \\ &= Pr(T_{hb2} - T_{hb1} > T_o - T_e) \end{aligned}$$

Let stochastic variable $T' = T_{hb2} - T_{hb1}$. As proved in [37], the probability density function of T' is an even function. With Chebysev Inequality, we can find:

$$P_{fs} \leq \frac{var(T)}{(T_o - T_e)^2}$$

where $var(T)$ denotes the variance of T. The above inequation can be used to estimate the upper bound of P_{fs} if we do not know the distribution of T.

In a group membership, if at least one of other $n-f-1$ replicas which are alive detects the crash of the coordinator (even a mistaking detection), a new coordinator will be elected and may duplicate writes. Assume the coordinator sends heartbeat messages to all other replicas and each heartbeat is independent. The probability of false suspicion on a coordinator in a group P_{gfs} can be calculated by summing the probability of $n-f-1$ replicas alive multiplied by that of at least one replica alive falsely suspecting the coordinator:

$$P_{gfs} = \sum_{f=0}^{[n/2]} \binom{n-1}{f} {P_c}^f (1-P_c)^{n-1-f} (1-(1-P_{fs})^{n-1-f})$$

However, for gossip-style failure detectors [38], the coordinator propagates heartbeat to only one replica chosen at random other than all other ones for better scalability. Thus P_{gfs} of gossip-style failure detectors (denoted by another symbol P_{gs-gfs}) have a little different:

$$P_{gs-gfs} = \sum_{f=0}^{[n/2]} \binom{n-1}{f} {P_c}^f (1-P_c)^{n-1-f} \binom{n-1-f}{1} P_{fs}$$

Let P_{wd} be the probability of write duplication. Write duplication occurs if (1)the coordinator is normally working but (2)other replicas that are alive have false suspicion on it and (3)two nodes sets responding ACKs for suspected and new coordinators do not overlap, that is, P_{wd} equals product of the probability of these three events:

$$P_{wd} = (1-P_c) P_{gfs} P_{no} \tag{5}$$

Note that if we also consider asynchronously propagating "PROPOSE" messages when calculating P_{no}, the value of P_{no} will decrease. But it is difficult to accurately compute P_{no} since it depends on the duration of specific coordinator election, failure recovery algorithms and even the time when the new coordinator receives the possible duplicated write. In addition, some coordinator election and recovery algorithms [32] may discover the historical records of the original write, which further lowers the value of P_{wd}. So equation (5) is a conservative upper bound on P_{wd}.

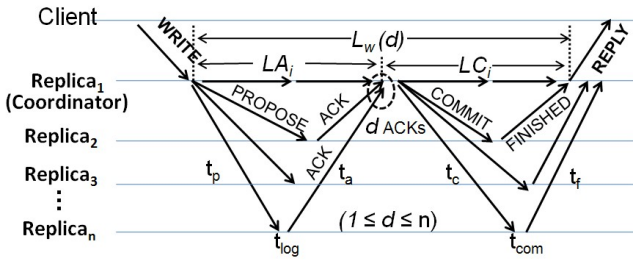

Fig. 7. Latency

E. Write Conflict

As described above, now we have given up all three assumptions which restrain RSM-d. Generally, write conflict of RSM-d can be divided to two classes: *write lost* and *write duplication*. The former happens when all historical records of a write are lost due to crash failures. Note that the coordinator is included in those crashed nodes; and the latter occurs when the coordinator is alive but false suspicion(s) and no overlapping of two nodes sets responding ACKs happen. Obviously write lost and write duplication are mutually exclusive events. Therefore:

$$\begin{aligned} P_{wc} &= P_{wl} + P_{wd} \\ &= \sum_{m=d}^{[n/2]} P_{elw}(m) Q(m) + (1-P_c) P_{gfs} P_{no} \end{aligned} \tag{6}$$

Although a final result of P_{wc} is given, we should adapt it in different cases. For instance, Zookeeper [8] still adheres to our assumption (c) which only considers the log history. Although it configures d as $[n/2]+1$, we should remove the factor of P_{wnu} in equation (6) once d decreases.

V. Latency

PACELC [4] have claimed the tradeoff between consistency and latency. In this section, we will calculate the value of latency for RSM-d.

As shown in Figure 7, we have defined some symbols in Table II which will be used. Obviously, $LA_i = t_p(i) + t_{log}(i) + t_a(i)$. Particularly, for the coordinator, $t_p = t_a = t_c = t_f = 0$. Excluding the coordinator, we can get $(n-1)$ values of LA_i. Then we arrange them in increasing order such as:

$$LA_{(1)} \leq LA_{(2)} \leq ... \leq LA_{(n-1)}$$

where $n \geq 2$.

According to the definition of RSM-d in section III, the latency of a write elapses from the coordinator receiving the write, at least $d-1$ ACKs (excluding the coordinator) responded, to at least one committed result responded. Thus:

$$L_w(d) = LA_{(d-1)} + min(LC_i) \tag{7}$$

where $d \geq 2$. If $d = 1$, $L_w(1) = t_{log}(coodinator) + min(LC_i) = t_{log}(coodinator) + min(t_c(i) + t_{com}(i) + t_f(i))$.

TABLE II. SYMBOLS IN SECTION V

Symbols	Definition
$L_w(d)$	the latency of a write with a given value of d
LA_i	the duration of the agreement phase for replica i
$LA_{(k)}$	the kth smallest value of LA_i (excluding the coordinator)
LA	any value of LA_i
LC_i	the duration of the commit phase for replica i
$t_p(i)$	the consumed time of the "PROPOSE" message from the coordinator to replica i
$t_{log}(i)$	the consumed time of logging a write for replica i
$t_a(i)$	the consumed time of the "ACK" message from replica i to the coordinator
$t_c(i)$	the consumed time of the "COMMIT" message from the coordinator to replica i
$t_{com}(i)$	the consumed time of committing a write for replica i
$t_f(i)$	the consumed time of the "FINISHED" message from replica i to the coordinator

Here we focus on the message delays and ignore the log time $t_{log}(i)$ (or can be seen a constant). Since $t_p(i)$ and $t_a(i)$ are both samples of random variable T (defined in section IV), $LA_i = t_p(i) + t_a(i)$ follows a probability density function $g(t) = \int_{-\infty}^{+\infty} f(x)f(t-x)dx$ ($f(t)$ has been defined in section IV) and a cumulative distribution function $G(t)$.

The pdf and cdf of $LA_{(k)}$ are denoted by $h_{(k)}(t)$ and $H_{(k)}(t)$ $(1 \leq k \leq n-1)$, respectively. Intuitively, for any positive real number ε that is small enough, $LA_{(k)} \in [t, t+\varepsilon]$ if and only if one $LA_i \in [t, t+\varepsilon]$ and exact $(k-1)$ values of random LA_i are smaller than t and other $(n-1-k)$ values of LA_i are more than $t+\varepsilon$. That is:

$$\begin{aligned}
& Pr(LA_{(k)} \in [t, t+\varepsilon]) \\
= \; & \binom{n-1}{1} Pr(LA \in [t, t+\varepsilon]) \binom{n-2}{k-1} Pr(LA < t)^{k-1} \\
& \cdot Pr(LA > t+\varepsilon)^{n-1-k} \\
= \; & (n-1) Pr(LA \in [t, t+\varepsilon]) \binom{n-2}{k-1} Pr(LA < t)^{k-1} \\
& \cdot Pr(LA > t+\varepsilon)^{n-1-k} \\
= \; & (n-1) g(t) \varepsilon \binom{n-2}{k-1} Pr(LA < t)^{k-1} \\
& \cdot Pr(LA > t+\varepsilon)^{n-1-k} \\
= \; & (n-1) g(t) \varepsilon \binom{n-2}{k-1} (G(t))^{k-1} Pr(LA > t+\varepsilon)^{n-1-k} \\
= \; & (n-1) g(t) \varepsilon \binom{n-2}{k-1} (G(t))^{k-1} (1 - G(t+\varepsilon))^{n-1-k}
\end{aligned}$$

Thus, we can have:

$$\begin{aligned}
h_{(k)}(t) &= \lim_{\varepsilon \to 0} \frac{Pr(LA_{(k)} \in [t, t+\varepsilon])}{\varepsilon} \\
&= (n-1) g(t) \binom{n-2}{k-1} (G(t))^{k-1} (1 - G(t))^{n-1-k}
\end{aligned}$$

By equation (7), we can get the expectation of $L_w(d+1) - L_w(d) (d \geq 1)$:

$$\begin{aligned}
& E(L_w(d+1) - L_w(d)) \\
= \; & E(LA_{(d)} + min(LC_i)) - E(LA_{(d-1)} + min(LC_i)) \\
= \; & E(LA_{(d)} - LA_{(d-1)})
\end{aligned}$$

where both $min(LC_i)$ are offset since the commit phase have no change, and $LA_{(0)} = 0$.

Note that $f(t)$ is the distribution of the message delay, therefore $f(t) = 0 (t \leq 0)$, as well as $g(t) = 0 (t \leq 0)$, $G(0) = 0$ and $G(+\infty) = 1$. If $d = 1$, then:

$$\begin{aligned}
& E(L_w(2) - L_w(1)) = E(LA_{(1)}) \\
= \; & \int_{-\infty}^{+\infty} t(n-1) g(t) (1 - G(t))^{n-2} dt \\
= \; & \int_{0}^{+\infty} (-t) d(1 - G(t))^{n-1} \\
= \; & \int_{0}^{+\infty} (1 - G(t))^{n-1} dt - \lim_{t \to +\infty} t(1 - G(t))^{n-1} \\
= \; & \int_{0}^{+\infty} (1 - G(t))^{n-1} dt - \lim_{t \to +\infty} t(\int_{t}^{+\infty} g(x)dx)^{n-1}
\end{aligned}$$

If the expectation of $g(t)$ exists, that is, $E(g(t)) = \int_0^{+\infty} tg(t)dt$ is a constant. In this condition, easily proved that:

$$\lim_{t \to +\infty} t(\int_{t}^{+\infty} g(x)dx)^{n-1} = 0$$

Such that:

$$E(L_w(2) - L_w(1)) = \int_{0}^{+\infty} (1 - G(t))^{n-1} dt$$

If $d \geq 2$, then:

$$\begin{aligned}
& E(L_w(d+1) - L_w(d)) = E(LA_{(d)} - LA_{(d-1)}) \\
= \; & \int_{-\infty}^{+\infty} t(n-1) g(t) \binom{n-2}{d-1} (G(t))^{d-1} \\
& \cdot (1 - G(t))^{n-1-d} dt - \int_{-\infty}^{+\infty} t(n-1) g(t) \\
& \cdot \binom{n-2}{d-2} (G(t))^{d-2} (1 - G(t))^{n-d} dt \\
= \; & \binom{n-1}{d-1} \int_{0}^{+\infty} (-t) d(G(t))^{d-1} (1 - G(t))^{n-d}) \\
= \; & \binom{n-1}{d-1} (\int_{0}^{+\infty} (G(t))^{d-1} (1 - G(t))^{n-d} dt \\
& - \lim_{t \to +\infty} t((\int_{t}^{+\infty} g(x)dx)^{n-d}))
\end{aligned}$$

Similarly, if $E(g(t))$ exists we can prove that:

$$\lim_{t\to+\infty} t((\int_t^{+\infty} g(x)dx)^{n-d}) = 0$$

Thus:

$$\begin{aligned} &E(L_w(d+1) - L_w(d)) \\ &= \binom{n-1}{d-1}(\int_0^{+\infty}(G(t))^{d-1}(1-G(t))^{n-d}dt) \end{aligned}$$

where $d \geq 2$. Let $\Delta LA(d) = \binom{n-1}{d-1}(\int_0^{+\infty}(G(t))^{d-1}(1 - G(t))^{n-d}dt)$. Finally we show that:

$$E(L_w(d+1) - L_w(d)) = \Delta LA(d)(d \geq 1) \tag{8}$$

Obviously $\Delta LA(d) > 0$. So equation (8) means that the expectation of latency strictly monotonically increases with respect to d given the distribution of message delays $f(x)$ and the number of replicas n.

VI. Tradeoff of Consistency and Latency

In the system model of RSM-d, we have quantified the inconsistency by the probability of write conflict and the expectation of latencies. For long running RSMs, we consider that:(1)the system consistency means the rate of sequentially committed writes, which can be measured by $1 - P_{wc}$; and (2)the system latency means the average value of write latencies, which can be estimated by $E(L_w(d))$. In this section, we will discuss tradeoff between the consistency and latency.

Equation (6) shows the result of P_{wc} if $d \in [1, n/2]$, and $P_{wc} = 0$ has been proved in [33] when $d \in [\lceil n/2 \rceil + 1, n]$. If $d \in [1, n/2]$ and the values of $n, P_c, f(x), T_e$ and T_o are given, when d increases, $P_{gfs}, P_{elw}(m), Q(m)$ will not change for the same m, and P_{no} decreases, so P_{wc} will decrease. That is:

$$P_{wc}(1) > ... > P_{wc}(\lceil n/2 \rceil) > P_{wc}(\lceil n/2 \rceil + 1) = ... = P_{wc}(n)$$

Equation (8) presents a conclusion that $E(L_w(d))$ rises if d increases. Thus:

$$E(L_w(1)) < E(L_w(2)) < ... < E(L_w(n))$$

The above two inequalities validate the tradeoff between consistency and latency. But what we want further is how to quantitatively select consistency and latency to benefit us best. Here we provide a possible solution.

Assume B is the sum of system benefit (e.g., income, use experience), B_c and B_l are factors that consistency and latency imposed on the benefit. For example, B can be seen as the overall income of an online store, and transactional purchases with stronger consistency will lead to more income, and higher latency will result in few users and less income. Since consistency is positive to the benefit, we set $B_c = \alpha(1 - P_{wc}(d)) + \beta(\alpha > 0)$, where α represents the benefit produced by one unit of consistency; and oppositely latency is passive, we set $B_l = \gamma - \theta E(L_w(d))(\theta > 0)$, where θ represents the loss incurred by one unit of latency. Define $B = B_c + B_l + \delta$, where δ represents the value of other factors influencing the overall benefit. That is:

$$\begin{aligned} B(d) &= B_c + B_l + \delta \\ &= \alpha(1 - P_{wc}(d)) + \beta + \gamma - \theta E(L_w(d)) + \delta \\ &= (\alpha + \beta + \gamma + \delta) - (\alpha P_{wc}(d) + \theta E(L_w(d))) \\ &= \eta - (\alpha P_{wc}(d) + \theta E(L_w(d))) \\ &= (\eta - \theta E(L_w(1))) \\ &\quad -(\alpha P_{wc}(d) + \theta(E(L_w(d)) - E(L_w(1)))) \\ &= (\eta - \theta E(L_w(1))) - v(d) \end{aligned}$$

where $\eta = \alpha + \beta + \gamma + \delta$ and $v(d) = \alpha P_{wc}(d) + \theta(E(L_w(d)) - E(L_w(1)))$. Although α, θ, η and $E(L_w(1))$ may change for different contexts, e.g., a search service and an e-commerce service, a book shop and a house sales, Christmas Day and common days, we can consider they are stable during long enough time for a specific application if we only change the value of d. Thus only $v(d)$ is the variable part of $B(d)$, and the minimal $v(d)$ will result in the maximal $B(d)$. Such that:

$$\begin{aligned} B(d) &= (\eta - \theta E(L_w(1))) - v(d) \\ &\leq (\eta - \theta E(L_w(1))) - v(d') \\ &= B(d') = B_{max} \end{aligned}$$

where $d' \in [1, n]$ so as to $\alpha P_{wc}(d') \approx \theta(E(L_w(d')) - E(L_w(1)))$ and make $v(d')$ minimal. Since $P_{wc}(d')$ is the result of equation (6) and $(E(L_w(d')) - E(L_w(1)))$ can be calculated by equation (8), we can determine the value of d' and get B_{max} if α, θ and η are known.

Next, we will show how to compute the value of α, θ and η. In a distributed system using RSMs, at first we record the sum of benefit as $B(1), B(\lceil n/2 \rceil + 1)$ and $B(n)$ when we tune the value of d to $1, \lceil n/2 \rceil + 1$, and n, respectively. Therefore:

$$\begin{cases} B(1) = \eta - (\alpha P_{wc}(1) + \theta E(L_w(1))) \\ B(\lceil n/2 \rceil + 1) = \eta - \theta E(L_w(\lceil n/2 \rceil + 1)) \\ B(n) = \eta - \theta E(L_w(n)) \end{cases}$$

With above three equations, we can get α, θ and η.

However, if we only want to make $B(d)$ maximal but do not have to know its exact value, the value of η is not necessary. In this condition, α is the increment of benefit when consistency increases one unit, and θ is the decrement of benefit when latency increases one unit. Thus we say that α is the benefit ratio of consistency and θ is the benefit ratio of latency, and we can obtain them by statistical methods.

In any case, the value of d' is determined for the maximal $B(d)$. Then the consistency $1 - P_{wc}(d')$ and latency $E(L_w(d'))$ will also be determined. In this way, we make the tradeoff between consistency and latency for the maximal system benefit.

VII. Experiments

As discussed in Section IV and V, the probability of write conflict depends on the failure rate of servers, the distribution of message delay and the configuration of heartbeat interval and timeout of failure detection for a distributed RSM system; and write latency depends on the distribution of message delay, the duration of logging and write committing. In addition, the possible usage of anti-entropy and gossip-style failure

detectors may make system complicated and non-deterministic. In this section, we focus on validating our presented results, while making the minimum assumptions on RSM-d for decreasing the number of experimental variables and expanding the scope of application. Thus anti-entropy and gossip-style failure detectors are not considered.

A. Event-driven Simulation

Considering the complexity of equations (1)-(8) and verifiability for experimental results, we implement RSM-d using Monte Carlo based Event Driven Simulations. For every ten million writes, we detect write loss by checking whether a write is lost in historical records (log and commit history) of all replicas which are alive, and write duplication through checking whether there are two writes employing a same sequence number. The numbers of write lost and write duplication are recorded as N_{wl} and N_{wd}. Thus we estimate P_{wc} as $(N_{wl} + N_{wd})/10,000,000$. At the same time, we calculate the average value of latencies to estimate the expectation of $L_w(d)$.

B. Validating Probability of Write Conflict

In this experiment, we compare our prediction results calculated by equation (6) with the observed experimental values to validate it. The parameters we use are tried to be selected as more reasonable and practical ones. Failure rate P_{wc} is set to 2%,3% and 4%, according to the statistics of Google [2]; We think the distribution of message delay $f(x)$ as an exponential one with a rate λ, where λ may be $0.01(100ms)$,$0.02(50ms)$,$0.05(20ms)$,$0.1(10ms)$ and $0.2(5ms)$, referencing to [7,8,26]; and heartbeat interval T_e and timeout T_o of failure detectors are set to $(1000ms, 2000ms)$ and $(500ms, 1000ms)$.

With above input parameters, we repeat experiments for $n \in [2, 9]$ and $d \in [1, [n/2]]$ and record the values of N_{wl} and N_{wd} to estimate P_{wc}. At the same time, we use equation (6) to predict P_{wc}. Comparing observed values with predicted ones of P_{wc} in all cases, our average RMSE=0.0009% and std. dev.=0.0052%. This validates our predicted results. When $d \geq [n/2] + 1$, we observe that N_{wl} and N_{wd} are always 0, which matches our analysis.

C. Latency Validation

To validate equation (8) about the expectation of write latency, we collect all latencies, calculate their average value and compare with the result of equation (8). The message delay distribution $f(x)$ also follows $\lambda = 0.01, 0.02, 0.05, 0.1, 0.2$, which are introdued into corresponding messages. And we perform a $null$ write (which has no operation) to guarantee that the duration of logging and write committing can be ignored.

For each $n \in [2, 9]$ and $d \in [1, n]$, we run experiments to get the average value of latencies and predict the expectation of latency by equation (8). Comparing observed values with predicted ones of $E(L_w(d)) - E(L_w(1))$ in all cases, our average RMSE=0.013ms and std. dev.=0.019ms, which validates our predicted latencies.

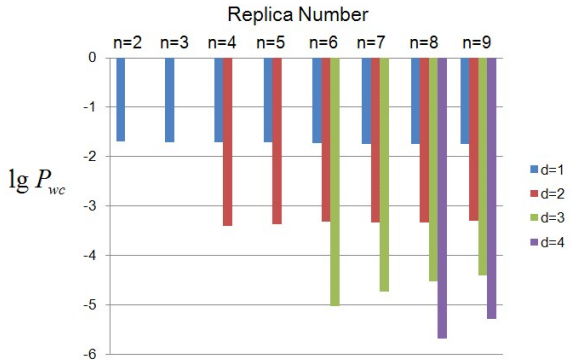

Fig. 8. Impact of d on Write Conflict

D. Impact of d on Write Conflict

This experiment is to show the trend of write conflict with respect to d. Without loss of generality, parameters are configured such as $\lambda = 0.01$,$P_c = 2\%$,$T_e = 1000ms$ and $T_o = 2000ms$.

As shown in Figure 8, the x-axis represents the number of replicas n $(2 \leq n \leq 9)$ and corresponding number of ACKs d $(1 \leq d \leq [n/2])$, and the y-axis shows the probability of write conflict. For highlighting the variances of different P_{wc}, we use $\lg P_{wc}$ to replace P_{wc}. We can see that the probability of write conflict P_{wc} decreases as d is increased for any fixed replica number n, and declines in steps that are at least one order of magnitude. Note that when $d > [n/2]$, the probability of write conflict P_{wc} is 0. So they are not shown in the figure.

E. Impact of d on Latency

Equation (8) reveals the relation of write latency and d. In this experiment, we want to show the change of write latency with respect to d more intuitively. The message delay distribution $f(x)$ is set as $\lambda = 0.01$. And we also perform a $null$ write to eliminate the influences of log and commit time. However, even the times of log and commit exist, they do not interfere with our results, because $L_w(1)$ comprises all their values which will be removed.

Figure 9 indicates the impact of number of ACKs d on Latency. As shown, the x-axis represents the number of replicas n $(2 \leq n \leq 9)$ and corresponding number of ACKs d $(2 \leq d \leq [n/2] + 1)$, and the y-axis shows the latency $L_w(d)$ which has removed the value of $L_w(1)$. The figure shows that $L_w(d)$ increases as d rises. Here we focus on $d \in [2, [n/2]+1]$, in fact $L_w(d)$ increases as d rises for any $d \in [1, n]$. Another interesting observation is that latency with a fixed d decreases while n is increased. The reason is that the coordinator has more alternative ACKs to select the fastest d ones.

F. Consistency v.s. Latency

As defined in section VI, consistency=$1 - P_{wc}$. Now we combine the results of above two experiments and show the tradeoff between consistency and latency. As shown in Figure 10, the x-axis represents the latency $L_w(d)$ which has removed the value of $L_w(1)$, and the y-axis shows the consistency $1 -$

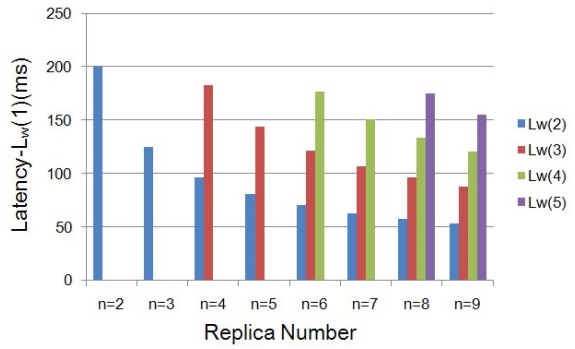

Fig. 9. Impact of d on Latency

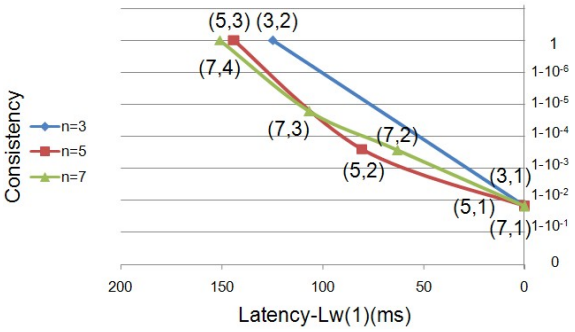

Fig. 10. Consistency v.s. Latency

$P_{wc}(d)$. In addition, we select three popular values of n as 3, 5 and 7 and the corresponding $d(1 \leq d \leq \lceil n/2 \rceil + 1)$, and mark the points by (n, d) such as $(3, 1)$. It is obvious that consistency and latency are competitive (we all think higher latency is worse). Given a fixed replica number n and $d \in [1, \lceil n/2 \rceil + 1]$, it shows that stronger consistency must lead to worse latency and better latency must result in weaker consistency.

G. An Example for Tradeoff of Consistency and Latency

Assume that there is an online store like Amazon. And it has used RSMs in its sales system. Although we try to collect more practical data about α, θ and η (defined in section VI), it is difficult for business problems. However, if we do not want to know the exact value of system benefit $B(d)$, we only need the benefit ratio of consistency α and the benefit ratio of latency θ by statistics methods, and then determine d' for the maximal system benefit $B(d')$.

Here we show how to estimate α and θ. We suppose the online store averagely sells S products per hour, and their average price is V. On the one hand, if every normal(consistent) purchase can bring about 20% benefit of the price, and every abnormal(inconsistent) purchase incurs double loss of the price(such as the compensation for order conflicts). Then the benefit factor produced by consistency (*i.e.*, B_c) is $20\%SV(1-P_{wc}) - 2SV(1-(1-P_{wc}))$, therefore the benefit ratio of consistency $\alpha = 2.2SV$. On the other hand, users respond to latency, e.g., an additional 100ms of latency results in 1% drop in sales at Amazon [13]. That is, the benefit ratio of latency θ is $0.01\%SV$. Therefore, we have that $\alpha = 2.2SV$ and $\theta = 0.01\%SV$.

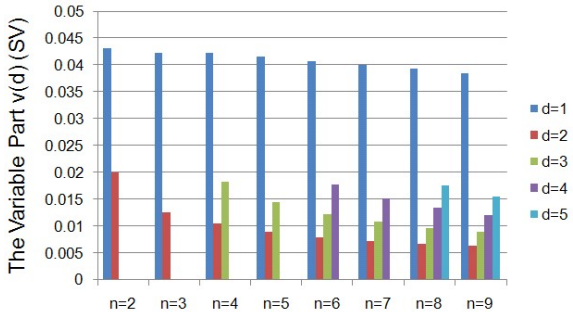

Fig. 11. The Variable Part of System Benefit

Section VI shows that the overall benefit $B(d)$ will be maximal if its variable part $v(d) = SV(2.2P_{wc}(d) + 0.01\%(E(L_w(d) - E(L_w(1)))))$ has the minimal value. For any $d \in [1, \lceil n/2 \rceil + 1]$ and $n \in [2, 9]$, we have calculated the values of $v(d)$ with the above experimental results. Figure 11 depicts the variable part $v(d)$ of the overall benefit $B(d)$. For example, if we set $n = 3$ and $d = 2$, $v(d)$ will be $0.125SV$. We can see that $v(d)$ has the minimal value and $B(d)$ has the maximal value when $d = 2$ if the replica number is 4 or 5, which is neither non-uniform total order broadcast nor Paxos implementation. So it shows that common practical RSM systems (with $d = 1$ and $d = \lceil n/2 \rceil + 1$) may not the best selection from the aspect of overall system benefit. We should consider more intermediate states in the consistency spectrum.

VIII. Related Work

We have surveyed many replicated state machine techniques [23-26] and their variants such as [27-29]. Their three representative implementations are non-uniform total order broadcast, Paxos and uniform total order broadcast. Although RSMs has been long studied, in this work we do not want to design better algorithms but try to quantitatively analyze these works from the aspects of consistency and latency. In addition, prior analysis works [24,33] on RSMs confine in the three representative implementations of RSMs and their variants, and few considers consistency and latency simultaneously. Our analysis covers more than the three typical ones and quantitatively reveals the tradeoff between consistency and latency.

Consistency, which has been long studied [39], attracts more eyes in recent years. With the consideration of CAP [3] and PACELC [4], people develop a lot of consistency models, from weakest eventual consistency [9], timeline consistency [40] to causal consistency [41]. Instead of developing a new consistency model or performance optimization for existing models, we discuss when and how the inconsistency occurs, and further measure its degree with a probabilistic model in the context of RSMs.

There are several works [17,18] presenting bounded read staleness from data versions, stale time and staleness probability to evaluate the inconsistency of quorum like [15], anti-entropy [35] systems, where writes are committed by vector

clocks [16] with semantic reconciliation [5] in causal order as opposed to total order of RSMs. In this work, we focus on write inconsistency for RSMs, since read serializability can be easily guaranteed in total order to eliminate read staleness.

Some previous works have been done on write inconsistency. [19,20] provide techniques to make writes satisfying some constraints, which use strong and weak consistency to ensure the constraints are not violated. Write divergency of replicas are limited in [21,22]. For example, [21] provides some metrics such as numerical error and order error for data items, and then presents a variety of algorithms for writes to guarantee predefined metrics. These write inconsistency models aim to improve the performance in the absence of crash failures, but we want to know how much inconsistency incurred by crash failures a replication system may encounter, rather than confine writes to some constraints, or bound the maximum deviation of a value from its newest version.

In the context of RSMs, in this work we quantify write inconsistency by the probability of write conflict while considering crash failures. Furthermore, we calculate write latency and describe how to quantitatively make consistency and latency tradeoff from the aspect of system benefit.

IX. Discussion

In this section, we provide several problems that should be noted or considered in future work.

A. Quorum Systems

Quorum systems define sets of replicas W written to and R read from, which can be configured such as ONE(W=1), QUORUM($W = \lceil n/2 \rceil + 1$) and ALL($W = n$) alternatives or even any other values. This is very like the three representative implementations of RSMs, but they are essentially different. The reason is that writes of quorum systems are ordered causally and those of RSMs are totally ordered. Thus some conditions result in write conflict for RSMs but not for quorum systems. For instance, write duplication may occur when the coordinator is falsely suspected. In contrast, it is meaningless to discuss write conflict for quorum systems because we can not decide the original write and possible duplicated one are in the same order or not for the partial orders of writes. In the practical quorum systems at present, they mostly view the second write as a new write.

However, our partial work can be useful for analyzing quorum systems. The formula on the expectation of latency(equation (8)) are also applicable to quorum systems if replacing W to d.

B. Failure Mode and Failure Detection

As discussed in section II, our work is based on the assumption of non-Byzantine failures. If we enlarge failure modes to Byzantine ones, our work should be made corresponding modification. In future work, we will consider this extension.

For non-Byzantine failures, we have modeled a common and popular heartbeat based but not adaptive and fast failure detector. Some optimized failure detectors [31,36] use synchronized clocks or adjust heartbeat intervals adaptively to achieve faster failure detection. This means that the expression $T_{hb2} - T_{hb1} > T_o - T_e$ in the formula of the probability of false suspicion $P_{fs} = Pr(T_{hb2} - T_{hb1} > T_o - T_e)$ should change according to specific failure detectors.

C. Coordinator Election and Failure Recovery

While presenting the formula of write duplication, we have noted that coordinator election and failure recovery [32] may influence its result.

RSMs run coordinator election and failure recovery after suspicions on coordinators for synchronizing the states of both committed writes and historical records for replicas. This synchronization may enlarge the range of historical records perceived by the new coordinator. Thus the historical records incurred by writes sent by the suspected coordinator may be caught by the new coordinator during coordinator election and failure recovery. This will avoid the reuse of sequence numbers and lower the probability of write duplication. Although we can also provide a formula to measure the impact of coordinator election and failure recovery on write duplication, it depends on the specific processes of them, which can be very different.

D. Independence of Crash Failures and Message Delays

Although we have released some assumptions in the computation of probability of write conflict and latency, we still assume that crash events of different nodes and message delays between replicas are independent. In practice, they do not always hold, since replica nodes and network links in modern data centers usually share physical machines, routers and power devices and so on. However, the correlations among failures and delays depend on the specific hardware and software implementations. A reasonable solution is to acquire the correlation factors among failures and delays by statistics for established replication systems, and then append them to the probability of crash failures and the message delays for more accurate results.

X. Conclusion

In this paper, we are concerned with providing a quantitative analysis of the relationship between consistency and latency. Given that replicated state machines are widely used in modeling replication systems and there are several variants of RSM implementations, we first propose a generic model called RSM-d to give a unified description of major replication protocols using RSMs. Second, on the basis of RSM-d, we build a probabilistic model for write conflicts that is one of the key factors to cause inconsistency, and a probabilistic model for characterizing latency as well. Third, we further define a payoff model through combining the write conflict model and latency model from the perspective of a service provider. Finally, with Monte Carlo based simulation, we validate our presented results and show the effectiveness of our solution to make tradeoff between consistency and latency to achieve the maximum system benefit.

ACKNOWLEDGMENT

The authors would like to thank Richong Zhang and Yu Tang for their discussions and feedback improved this work.

This work was supported partly by National Natural Science Foundation of China (No. 61103031, No. 61272165), partly by China 863 program (No. 2012AA011203), partly by A Foundation for the Author of National Excellent Doctoral Dissertation of PR China, partly by Beijing Nova Program and partly by Program for New Century Excellent Talents in University.

REFERENCES

[1] Amazon.com. Summary of the Amazon EC2 and Amazon RDS Service Disruption in the US East Region. April 2011

[2] J. Dean. Designs, lessons, and advice from building large distributed systems. Keynote from LADIS 2009.

[3] E. A. Brewer. Towards Robust Distributed Systems. In PODC, pages 7C7, 2000

[4] D. J. Abadi. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. IEEE Computer, 45(2):37C42, 2012.

[5] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazons highly available key-value store. In SOSP 2007.

[6] The Apache Cassandra Project. http://cassandra.apache.org/

[7] M. Burrows, "The Chubby lock service for loosely-coupled distributed systems," in OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation, 2006, pp. 335-350.

[8] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, ZooKeeper: Wait-free coordination for Internet-scale systems, in USENIX ATC10: Proceedings of the 2010 USENIX Annual Technical Conference. USENIX Association, 2010.

[9] W. Vogels. Eventually consistent. Commun. ACM, 52:40C44, January 2009.

[10] K. Birman. Reliable Distributed Systems Technologies, Web Services and Applications. Springer, 2005

[11] E. Schurman and J. Brutlag. Performance related changes and their user impact. Presented at Velocity Web Performance and Operations Conference, June 2009.

[12] Velocity and the Bottom Line. http://radar.oreilly.com/2009/07/velocity-making-your-site-fast.html

[13] G. Linden. Make data useful. https://sites.google.com/site/glinden/Home/StanfordDataMining.2006-11-29.ppt. 29 November 2006

[14] F. Schneider. Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Computing Surveys, 22(4), 1990

[15] Merideth, Michael G., and Michael K. Reiter. Selected results from the latest decade of quorum systems research. Replication, Springer Berlin Heidelberg, 2010, 185-206.

[16] Lamport, L. Time, clocks, and the ordering of events in a distributed system. ACM Communications, 21(7), pp. 558- 565, 1978

[17] D. Malkhi, M. Reiter, A. Wool, and R. Wright. Probabilistic quorum systems. Information and Communication, (170):184C206, 2001

[18] Peter Bailis, Shivaram Venkataraman, Joseph M. Hellerstein, Michael Franklin, Ion Stoica. Probabilistically Bounded Staleness for Practical Partial Quorums. In VLDB 2012

[19] S. S. Chawathe, H. Garcia-Molina, and J. Widom. Flexible Constraint Management for Autonomous Distributed Databases. IEEE Data Eng. Bull., 17(2):23C27, 1994

[20] A. Gupta and S. Tiwari. Distributed constraint management for collaborative engineering databases. In CIKM, pages 655C664, 1993

[21] H. Yu and A. Vahdat. Design and Evaluation of a Continuous Consistency Model for Replicated Services. In OSDI, pages 305C318, 2000

[22] S. Shah, K. Ramamritham, and P. J. Shenoy. Resilient and Coherence Preserving Dissemination of Dynamic Data Using Cooperating Peers. IEEE Trans. Knowl. Data Eng., 16(7):799C812, 2004

[23] Xavier Defago, Andre Schiper, and Peter Urban. Total order broadcast and multicast algorithms: Taxonomy and survey. ACM Comput. Surv. 36, 4 (December 2004)

[24] Xavier Defago. Comparative performance analysis of ordering strategies in atomic broadcast algorithms. IEICE Trans. on Information and Systems. December 2003

[25] L. Lamport. Paxos Made Simple. ACM SIGACT News, 32(4):18C25, December 2001

[26] P. Marandi, M. Primi, N. Schiper, and F. Pedone. Ring Paxos: A high-throughput atomic broadcast protocol. International Conference on Dependable Systems and Networks (DSN), 2010, pp. 527 -536

[27] L. Lamport. Generalized Consensus and Paxos. Technical Report MSR-TR-2005-33, Microsoft Research, 2005

[28] B. Kemme and G. Alonso. Database replication: a tale of research across communities. PVLDB, 2010

[29] Parisa Jalili MarandiMarco PrimiFernando Pedone. Multi-Ring Paxos. In DSN 2012

[30] Chandra, T., Toueg, S.: Unreliable Failure Detectors for Reliable Distributed Systems. Journal of the ACM 43(2), 225C267 (1996)

[31] W. Chen. On the Quality of Service of Failure Detectors.IEEE Transactions on Computer. May 2002

[32] Diogo Becker, Flavio Junqueira, and Marco Serafini. Leader Election for Replicated Services Using Application Scores. In Middleware 2011

[33] B. Lampson. The ABCDs of Paxos. In Proceedings of the 20th ACM Symposium on Principles of Distributed Computing (PODC01), page 13. ACM Press, 2001

[34] Idit Keidar and Sergio Rajsbaum. On the cost of fault-tolerant consensus when there are no faults-a tutorial. TechnicalReport MIT-LCS-TR-821, Laboratory for Computer Science, Massachusetts Institute Technology, Cambridge, MA, 02139, May 2001. also published in SIGACT News 32(2) (June 2001)

[35] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In PODC 1987.

[36] Marin Bertier, Olivier Marin, Pierre Sens, Implementation and Performance Evaluation of an Adaptable Failure Detector, Proceedings of the 2002 International Conference on Dependable Systems and Networks, p.354-363, June 23-26, 2002

[37] M. Raynal and F. Tronel. Group Membership Failure Detection: A Simple Protocol and Its Probabilistic Analysis. Distributed Systems Eng. J., vol. 6, no. 3, pp. 95-102, 1999

[38] R. van Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. In Proceedings of International Conference and Distributed Systems Platforms and Open Distributed Processing (IFIP),1998

[39] J. Gray, P. Helland, P. ONeil, and D. Shasha. The dangers of replication and a solution. In SIGMOD 1996

[40] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. PNUTS: Yahoo!s Hosted Data Serving Platform. PVLDB, 1:1277C1288, August 2008

[41] W. Lloyd, M. J. Freedmand, M. Kaminsky, and D. G. Andersen. Dont settle for eventual: Scalable causal consistency for wide-area storage with COPS. In SOSP 2011