The Costs and Limits of Availability for Replicated Services

HAIFENG YU
Intel Research Pittsburgh and Carnegie Mellon University
and
AMIN VAHDAT
University of California, San Diego

As raw system performance continues to improve at exponential rates, the utility of many services is increasingly limited by availability rather than performance. A key approach to improving availability involves replicating the service across multiple, wide-area sites. However, replication introduces well-known trade-offs between service consistency and availability. Thus, this article explores the benefits of dynamically trading consistency for availability using a *continuous consistency model*. In this model, applications specify a maximum deviation from strong consistency on a per-replica basis. In this article, we: i) evaluate the availability of a prototype replication system running across the Internet as a function of consistency level, consistency protocol, and failure characteristics, ii) demonstrate that simple optimizations to existing consistency protocols result in significant availability improvements (more than an order of magnitude in some scenarios), iii) use our experience with these optimizations to prove tight upper bound on the availability of services, and iv) show that maximizing availability typically entails remaining as close to strong consistency as possible during times of good connectivity, resulting in a communication versus availability trade-off.

Categories and Subject Descriptors: D.4.7 [**Operating Systems**]: Organization and Design—*Distributed systems*; H.2.4 [**Database Management**]: Systems—*Distributed databases*

General Terms: Algorithm, Experimentation, Measurement, Reliability

Additional Key Words and Phrases: Availability, continuous consistency, network services, replication, trade-off, upper bound

1. INTRODUCTION

As raw system performance continues to improve at exponential rates, the utility of many services is limited by availability rather than performance [Hennessy 1999]. Thus, many researchers are turning their attention to

Authors' addresses: H. Yu, 4720 Forbes Ave, Suite 410, Pittsburgh, PA 15213; email: yhf@cs.cmu.edu; A. Vahdat, Department of Computer Science and Engineering, University of California, 9500 Gilman Drive, La Jolla, CA 92093-0114; email: vahdat@cs.ucsd.edu.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 1515 Broadway, New York, NY 10036 USA, fax: +1 (212) 869-0481, or `permissions@acm.org`.
© 2006 ACM 0734-2071/06/0200-0070 $5.00

architectures that improve overall availability, for example, in the context of operating systems [Swift et al. 2004a; Swift et al. 2004b], cluster Web servers [Fox et al. 1997], cluster email servers [Saito et al. 1999], Internet auction system [Candea et al. 2004], and software RAID systems [Brown and Patterson 2000]. These studies all focus on the availability of a centralized service. However, given the pervasive use of the Internet to access remote services, even 100% local availability will not necessarily deliver high availability to end users. For example, one recent study shows that network failures prevent client access to a centralized service between 1.5–2% of the time [Dahlin et al. 2003].

Caching and replication are key approaches to improving the availability of Internet services in the face of such benign network failures (tolerating failures caused by malicious attacks such as denial of service attacks is beyond the scope of this article). Multiple replicas increase the probability that a client will be able to access a service despite individual failures. Unfortunately, strict consistency requirements can actually reduce the availability of a replicated service relative to a centralized one because consistency protocols often require synchronous access to a subset of replicas to achieve a uniform view of write ordering. Thus, if any one member of this required subset is unreachable, the entire service can be rendered unavailable.

To address this limitation, optimistic consistency models allow multiple updates to take place simultaneously at different replicas. A number of efforts propose trading reduced consistency for increased availability, including Bayou [Petersen et al. 1997; Terry et al. 1995], Ficus [Page et al. 1998], Coda [Kistler and Satyanarayanan 1992], Deno [Cetintemel et al. 2001], TACT [Yu and Vahdat 2002], Lazy Replication [Ladin et al. 1992] and Fluid Replication [Noble et al. 1999]. While optimism can dramatically increase availability, an unbounded rate of conflicting updates can quickly leave the system in a delusional state [Gray et al. 1996]. Thus, we explore the availability benefits of a *continuous* consistency model. In such a model, applications specify a maximum distance from strong consistency (where existing optimistic models leave this distance unbounded). Decreasing consistency results in a corresponding increase in overall availability. Thus, a continuous consistency model exposes a trade-off between consistency and availability that can be dynamically varied based on changing network and service characteristics. For example, a service might relax consistency to maintain 99.9% availability in the face of reduced network reliability.

While availability is extensively studied at the extreme of strong consistency [Amir and Wool 1996, 1998; Barbara and Garcia-Molina 1987, 1986; Coan et al. 1986; Diks et al. 1994; Garcia-Molina and Barbara 1984; Johnson and Raab 1991a, 1991b; Kumar and Segev 1993; Peleg and Wool 1995; Spasojevic and Berman 1994; Tong and Kain 1988], there is comparatively little understanding of the impact on availability of relaxed consistency models. This work is among the first to quantify the availability of a replicated service running across the wide area. Our prototype measures availability while varying the consistency level, the protocol used to enforce consistency, and the failure characteristics of the underlying network. Our findings reveal the inherent and continuous trade-off between consistency and availability and show that simple optimizations to

existing consistency protocols can significantly improve service availability. We hope our results will provide a framework for system developers to determine the degree of replication, the placement of replicas, and the consistency level required to achieve a target service availability. The long-term goal of our work is to enable services to tune their system availability as their workload changes and as network reliability changes. In the context of this goal, this article makes the following specific contributions.

—We implement a prototype replication infrastructure and a variety of popular consistency protocols. Using this prototype, we quantify service availability as a function of network failure characteristics, consistency level, and consistency protocol through both live wide-area deployment and network emulation. We find that existing consistency protocols are optimized for performance rather than availability and can actually deliver reduced availability when relaxing consistency. Simple modifications to these protocols greatly improve availability.
—For relaxed consistency, we find that maximizing availability entails maintaining as strong a consistency level as possible during times of full connectivity. This is required to build up a large cushion for the times when failures prevent communication. The need to maintain nearly strong consistency most of the time exposes an interesting trade-off between availability and communication when employing a continuous consistency model.
—Based on our experience with improving the availability of consistency protocols, we develop a theory to compute tight upper bounds on service availability as a function of network and end-host failure characteristics (faultload), workload, consistency level, and degree of replication. Because there is currently no convincing way to mathematically model a faultload (especially under arbitrary network topologies and potentially correlated failures), we compute the upper bound based on the faultload instead of a closed-form equation. This upper bound gives system designers an idea of the best possible availability their service can achieve, independent of the algorithms used to maintain consistency and the aggressiveness of their optimizations. We compare the availability characteristics of our consistency protocols to the calculated upper bound under a variety of consistency levels and faultloads. We find that with our simple optimizations, existing protocols can approach the availability upper bound under current Internet failure scenarios (more aggressive optimizations and even future knowledge is required to reach the upper bound in the general case).
—We study the effects of the degree of replication and the reliability of the underlying network on overall service availability. Thus, we quantify the tradeoff between the desire to widely replicate a service to maximize availability (in the limit, placing a replica on every client machine) and the desire to centralize a service to minimize consistency overheads (in the limit, updates would be applied to a single site and all client accesses would go through that site).

The rest of this article is organized as follows. Section 2 provides background and a brief review of the TACT continuous consistency model [Yu and Vahdat

2002]. Section 3 describes our replication and availability model. Section 4 studies the availability upper bound problem and derives a tight upper bound. Section 5 details our implementation and measurement methodology. Section 6 presents availability and upper bound results for replicated services under a variety of conditions through actual wide-area deployment and an emulation environment. Section 7 describes related work, and Section 8 presents our conclusions.

2. BACKGROUND

While research and development efforts typically focus on improving the performance of computer systems, improving overall service availability often receives relatively little attention. Currently however, the proliferation of companies that conduct significant portions of their business online has drawn increased interest to highly-available services. Thus, even a 1% improvement in availability is significant, corresponding to approximately 3.5 additional days of uptime per year. Put another way, each 0.1% improvement in service availability results in roughly 8 hours of additional uptime per year, corresponding to approximately $1 million in additional revenue for every $1 billion in annual revenue conducted online. The cost of downtime can be even more catastrophic in military and scientific scenarios (even when considering differences in availability of 0.001%–0.0001%, 5–50 minutes of additional uptime per year).

The general technique considered here of trading consistency for availability is well understood [Ladin et al. 1992]. Strong consistency ensures that concurrent updates will not conflict, but it limits system availability, throughput, and the practical degree of replication. Based on the observation that many applications do not always require strong consistency, optimistic consistency greatly improves system performance and availability but does not limit the number or severity of conflicting updates. Thus, application developers are forced to make a binary decision between strong and optimistic consistency models. At each of these two extremes, there is an associated trade-off between consistency, performance, and availability. A number of efforts [Adya et al. 2000; Krishnakumar and Bernstein 1994; Pu and Leff 1991; Yu and Vahdat 2002] argue for the benefits of a *continuous consistency model*. Here, optimistic and strong consistency are two extremes of a more general spectrum enabling applications to dynamically specify their availability/consistency requirements based on changing client, network, and service characteristics. A full discussion of the benefits and applicability of continuous consistency is beyond the scope of this article. For our purposes, a continuous model allows a more complete exploration of the problem space.

We focus our study on the TACT consistency model [Yu and Vahdat 2002]. However, we believe our findings reveal inherent aspects of the consistency versus availability trade-off for a wide range of underlying models. In general, TACT gradually reduces the amount of required synchronous communication among replicas in moving from strong to optimistic consistency. Intuitively, the model allows replicas to locally buffer a maximum number of updates before requiring remote communication. Each update can optionally carry an

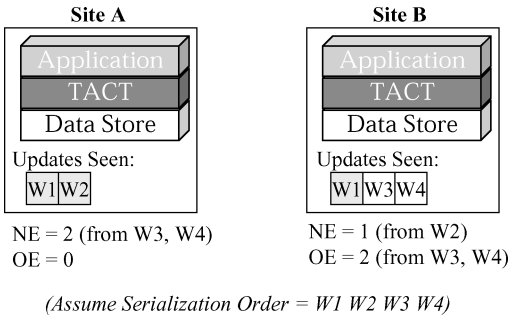

Fig. 1. Example system state with corresponding numerical and order error.

application-specific weight describing its importance. At any replica updates can be in either a tentative or committed state. Three per-replica metrics, *Numerical Error*, *Order Error*, and *Staleness*, specify system consistency. Numerical error is the maximum weight of writes not seen by a replica which can be bounded using the algorithm from Yu and Vahdat [2000]. Order error is the maximum weight of writes that have not established their commit order at the local replica (i.e., maximum weight of tentative writes in the local view). The commit order can be established using several existing protocols [Golding 1992; Keleher 1999; Petersen et al. 1997]. Finally, staleness is the maximum amount of time before a replica is guaranteed to observe a write accepted by a remote replica. Setting all three metrics to zero corresponds to strong consistency, while values of infinity correspond to optimistic consistency.

To provide some intuition behind the metrics, consider the application of TACT to a replicated airline reservation system [Yu and Vahdat 2002]. Here, numerical error corresponds to the maximum number of system-wide reservations that have not been propagated to the local replica. Order error corresponds to the maximum number of tentative reservations in a replica's local view (i.e., reservations that have not established their final commit order and that may have to be later rolled back). Finally, staleness is an upper bound on elapsed wall-clock time before an update is propagated to all replicas. Thus, roughly, numerical error bounds the maximum rate of conflicting reservations, order error bounds the rate of false negatives (where a user acts upon the presence of tentative reservations that are later rolled back), and staleness guarantees the maximum elapsed time before a reservation is seen system-wide (and hence can be confirmed).

Figure 1 depicts a simple example scenario with two replicas at sites A and B. In this example, replica A has accepted updates W_1 and W_2, and replica B has accepted updates W_3 and W_4. Update W_1 has been propagated from A to B. We assume that the final serialization order of the four writes is $W_1W_2W_3W_4$. Writes in shaded boxes are committed (e.g., W_1 at B), having established their final serialization order at the local replica. In this simple example, the numerical error at A is two because it has not seen two remote updates W_3 and W_4. The order error at A is zero because all local writes have established their final commit order. At B on the other hand, the order error is two because W_3 and W_4

cannot be committed (in the assumed serialization order) until W_2 is received locally. The rigorous definitions for all three metrics will be introduced when they are used in our upper bound theory.

In our previous work [Yu and Vahdat 2001], we have shown that TACT applies to a wide range of applications in addition to the airline reservation example previously mentioned. In the following, we describe two more example applications, a bulletin board service and QoS load balancing. The bulletin board application is a replicated message posting service modeled after more sophisticated services such as USENET. Messages are posted to individual replicas. Sets of updates are propagated among replicas, ensuring that all messages are eventually distributed to all replicas. This application is intended to be representative of interactive applications that often allow concurrent read/write access under the assumption that conflicts are rare, or can be resolved automatically.

Desirable consistency requirements for the bulletin board example include maintaining causal and/or total order among messages posted at different replicas. With causal order, a reply to a message will never appear before the original message at any replica. Total order ensures that all messages appear in the same order at all replicas, allowing the service to assign globally unique identifiers to each message. Order error in the TACT model captures the maximum number of out of order messages on each replica. Another interesting consistency requirement for interactive applications, including the bulletin board, is to guarantee that, at any time t, no more than k messages posted before t are missing from the local replica. Such semantics is readily captured by numerical error.

Our third application is a load distribution mechanism that provides Quality of Service (QoS) guarantees to a set of preferred clients. In this scenario, frontends (as in LARD [Pai et al. 1998]) accept requests on behalf of two classes of clients, standard and preferred. The frontends forward requests to backend servers with the goal of reserving some predetermined portion of server capacity for preferred clients. Thus, frontends allow a maximum number of outstanding requests (assuming homogeneous requests) at the backend servers. To determine the maximum number of standard requests that should be forwarded, each frontend must communicate current access patterns to all other frontends.

One goal of designing such a system is to minimize the communication required to accurately distribute such load information among frontends. This QoS application is intended to be representative of services that independently track the same logical data value at multiple sites such as a distributed sensor array, a load-balancing system, or an aggregation query. Such services are often able to tolerate some bounded inaccuracy in the underlying values they track (e.g., average temperature, server load, or employee salary) in exchange for reduced communication overhead or power consumption. Because the load information is numerical in nature, the numerical error metric conveniently expresses the allowed inaccuracy.

The three consistency metrics in the TACT model are largely (but not entirely) orthogonal. For example, even though numerical error and staleness

describe two different aspects of inconsistency, zero numerical error will necessarily imply zero staleness. In TACT, the consistency metrics are intentionally designed to be low-level and primitive. High-level application consistency semantics are captured by the concept of *consistency units* (or *conits*) [Yu and Vahdat 2001]. Conits may determine the granularity over which consistency is enforced, for example, a single flight, first class seats, coach seats, and so on. They may also be logical and export application semantics. See Yu and Vahdat [2001] for more discussion on the flexibility offered by conits and how to export application-level consistency semantics via conits. Finally, there have also been many previously proposed relaxed consistency models (e.g., Krishnakumar and Bernstein [1994]; Singla et al. [1997]; Torres-Rojas et al. [1999]). We have previously shown [Yu and Vahdat 2001] that many of these models can be expressed as special cases of the TACT model. Thus we believe that the results of our study will be general to a range of related consistency models.

3. SYSTEM MODEL AND ASSUMPTIONS

For simplicity, we refer to application data as a database, though the data can actually be stored in a database, file system, persistent object, and so on. The database is replicated in full at multiple *replicas*. Each replica may accept *reads* and *writes* from clients, both called *accesses*. Our reads and writes are query transactions and update transactions in database terminology. To simplify discussion, we assume a single conit in the system and that each write carries unit numerical weight and order weight for that conit. Nonunit numerical weight can be readily captured by our theory. Nonunit order weight can also be captured by considering a write with order weight of w as w writes with unit order weight. Here unit does not have to be 1.0, rather, it can be the common denominator of all order weights used by the system. We also assume that all replicas remain consistent at all times, that is, the numerical error, order error, and staleness on any replica are always within bounds.

The first step in carrying out a study on network service availability is determining the precise definition of availability. Today, we lack a widely-accepted definition for service availability especially when the service is accessed over a network. Storage vendors have performed ground-breaking work in building highly-available services. However, when they describe systems with four nines (99.99%) of availability, they typically refer to the uptime of the hardware and software. For replicated Internet services, a replica that is up would means that the hardware of the replica is up, the network connection is functioning, and the software is not misbehaving (e.g., not generating responses due to deadlock). However, this does not necessarily mean that the service is available to the clients. For example, some subset of clients may be unable to access the service as a result of failures in the network. Further, the network may be executing properly but running so slowly that individual requests take an unacceptable amount of time to complete (worse, the definition of unacceptable is entirely application and client specific). Finally, in the case of replicated services, a replica may be accessible to a client but the replica may not be able to proceed with individual requests because of consistency requirements.

In this study, we define availability over *submitted accesses* from the client to the network service. Each access is classified as: i) a *failed access* if the request cannot reach any replica because of network failures, ii) a *rejected access* if it is received by some replica but its acceptance would violate some consistency requirement, or iii) an *accepted access* otherwise. From the client perspective, availability is $Avail_{client} = accepted\ accesses/submitted\ accesses$. However, client availability ($Avail_{client}$) is affected by two factors: network availability ($Avail_{network}$), the percentage of accesses that can reach a replica, and service availability ($Avail_{service}$), the percentage of accesses reaching replicas that are actually accepted. Thus, $Avail_{client} = Avail_{network} \times Avail_{service}$. Network availability is determined by replication scale, client population, and Internet reliability (for client to server communication), while service availability is affected by replication scale, Internet reliability (for server to server communication), consistency level, and the consistency maintenance protocol. Replication improves network availability, but decreases service availability if there are consistency requirements. Relaxing consistency helps to improve service availability. In this article, we quantify availability using both service availability and client availability.

We assume that both replicas and the network may experience fail-stop failures. We model replica failures as singleton network partitions because it is usually impractical to distinguish between remote host failures and network failure or congestion. We assume that network failures are symmetric. Even though nonsymmetric behavior is common at the IP level [Paxson 1996], we believe our assumption is a close approximation for reliable application-level communication, for example, over TCP. For reliability, these protocols require acknowledgments on the reverse path; thus, IP-level failures on either direction will result in TCP-level failures in both directions. We leave the more complex cases of byzantine failures and nonsymmetric network failure behavior to future work. Considering these cases would add much complexity into our already involved theory.

For the purpose of proving the upper bounds, without loss of generality, we also assume reachability among machines is transitive. This assumption is without loss of generality because, even if reachability is not transitive, theoretically, a replica can communicate with another replica as long as they are connected (potentially indirectly) to each other. RON [Andersen et al. 2001], MONET [Andersen et al. 2005], and One-hop Source Routing [Gummadi et al. 2004] achieve such functionality to some extent in practice. For proving the upper bounds, we must consider the best possible scenario where techniques such as RON become so perfect that they are able to route around all failures which do not result in network partition. Thus transitive reachability is actually an inference rather than an assumption when we prove the upper bound.

With the symmetry and transitivity assumptions, network failures break the network into partitions. Notice that different from replica failures, these partitions may not be singleton network partitions. A *faultload*, which is a trace of timestamped failure events and recovery events for replicas and the network, fully specifies the failure pattern. For example, a failure event may cause the network to be partitioned, while a recovery event merges two network

partitions into one. We do not explicitly discuss network topology since it is already abstracted in the faultload. The number and location of replicas are also specified as part of the faultload. It may appear that some of the factors (such as number and locations of replicas) are design parameters and should not be part of the faultload. Whether we include these parameters as part of the faultload does not affect this study. Our faultload definition can simply be viewed as a projection of a global faultload onto whatever number and locations of replicas the system chooses to use. We choose to use our current definition only to simplify discussion.

Failure and recovery events divide a run into *intervals*. By definition, network connectivity does not change during an interval. Notice that this does not mean that there are no failures (i.e., network being partitioned) within an interval. It simply means that the failure pattern does not change within an interval. We assume that the faultload observed by an application is not significantly affected by the application's own communication. We leave the interaction of an application's communication pattern with observed faultloads to future work.

We assume that an explicit *workload* fully describes the timestamped accesses reaching any of the replicas, that is, when and which access reaches which replica. Thus, the workload does not include failed accesses, and it implicitly encompasses $Avail_{network}$.

Our faultload and workload approach significantly differs from traditional approaches of studying availability [Douceur and Wattenhofer 2001; Johnson and Raab 1991a] where failures are probabilistically modeled instead of deterministically described. We apply this approach because, unlike replica failures, there is currently no convincing way to mathematically model the occurrences of network partitions. It is also NP-hard [Rosenthal 1977] to derive the partition model from link and node failure models. With the faultload approach, the upper bound on availability depends on faultloads instead of failure models. One advantage of this approach is that the upper bound theory itself is independent of the faultloads, while studies based on specific probabilistic failure models are usually difficult to extend to other failure models. Also notice that our faultload definition is general, and our theory and methodology would apply to any possible faultload whether they are realistic/representative or not. As a result, here we do not discuss what faultloads are representative. In our later experimental evaluation, we will use both measured faultloads and simulated faultloads.

4. AVAILABILITY UPPER BOUND THEORY

In the first part of our research, we aim to derive an upper bound on service availability as a function of workload, faultload, and consistency. This upper bound allows system designers to reason about the utility of different optimizations in light of the best availability that any protocol can achieve. More precisely, we intend to derive the following relationship between availability, consistency, workload, and faultload:

$$Avail_{service} \leq \mathcal{F}(consistency, workload, faultload).$$

The function $\mathcal{F}$ returns the availability upper bound, which is independent of the consistency maintenance protocol, and demonstrates the inherent effects of consistency, workload, and faultload on availability. The availability achieved by any system will be less than or equal to this upper bound.

We will first show that the availability upper bound problem is NP-hard, implying that exponential complexity is likely to be unavoidable. Section 4.2 explains how we control the exponential complexity such that the problem becomes tractable in practice. The correctness and tightness of the upper bound is proved in Section 4.3.

4.1 NP-Hardness of the Availability Upper Bound Problem

This section shows that even a very simple version of the availability upper bound problem is NP-hard. This simple version only considers numerical error, and each replica has a numerical error bound of either 1 or ∞. All writes have unit numerical weight. Even though the problem is significantly simplified, we will still be able to reduce the Independent Set problem to the upper bound problem.

Definition 4.1 (*Independent Set*). An *independent set* of a graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertexes such that each edge in E is incident on at most one vertex in V'. The *Independent Set* problem is to find a maximum-size independent set in G.

THEOREM 4.2 [CORMEN ET AL. 1990]. *The Independent Set problem is NP-hard.*

THEOREM 4.3. *The availability upper bound problem is NP-hard even if we only consider numerical error.*

PROOF. See Appendix. □

4.2 Deriving A Tight Availability Upper Bound

Despite the NP-hardness of the problem, we will show that it is possible to optimize the problem such that its complexity becomes tractable for many practical cases. We first introduce some important concepts and notations later used in our theory.

Definition 4.4. The *evolution graph* of a faultload is a directed graph constructed as follows. For each interval in the faultload, add a node to the graph for each network partition in that interval. Let $node_{k,m}$ correspond to $interval_k$, $partition_m$. An edge from $node_{k,m}$ to $node_{k',m'}$ is added if $k' = k + 1$, and $partition_{m'}$ intersects with $partition_m$ at one or more replicas. A node in the evolution graph is an *ancestor* of another node if there is a path from the former to the latter. Figure 2 gives an example evolution graph.

Given that we assume the three consistency metrics are within bounds on all replicas at all times, reads can always be accepted. Thus, to compute the availability upper bound, we only need to focus on writes. Let $writes_{k,m}$ be the number of writes accepted by $partition_m$ during $interval_k$, and let $wsubmit_{k,m}$

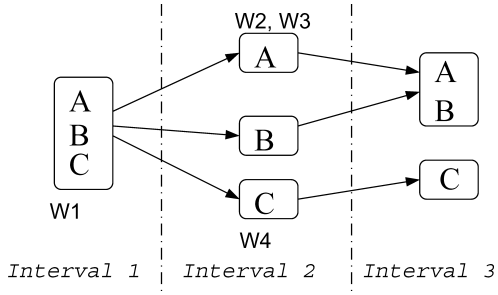

Fig. 2. An evolution graph with three replicas, A, B, and C. W_1, W_2, W_3, and W_4 are writes submitted by clients. For clarity, we do not plot client machines. Each box in the figure represents a network partition, and, beside each box, we annotate those accesses that can reach the replicas in each network partition.

$$\textbf{Maximize}: \sum_{k,m} writes_{k,m}$$
$$\textbf{Subject to}: \begin{cases} writes_{k,m} \leq wsubmit_{k,m} \quad \textbf{for all } k, m \\ \text{Additional consistency constraints} \end{cases}$$

Fig. 3. Baseline optimization problem.

be the number of writes submitted from clients. Our objective is to maximize $\sum_{k,m} writes_{k,m}$, subject to $writes_{k,m} \leq wsubmit_{k,m}$ and some additional constraints to be derived in the next section (Figure 3). More notation will be introduced later when they are first used as summarized in Table I.

4.2.1 *Constraints from Order Error.* Informally, order error is the number of writes that are out of order (according to the serialization order) at each replica. A *serialization order* is any total order among all accepted writes as long as it is agreed upon by all replicas. Consider the following example at the end of the second interval in Figure 2. Suppose the system has accepted four writes, W_1, W_2, W_3, and W_4, and replica A has seen and applied W_1, W_2, and W_3. With serialization order $W_1W_4W_2W_3$, the replica has two out of order writes, W_2 and W_3, since it does not see W_4, and the two writes are ordered after W_4 in the serialization order. Order error can be decreased by filling in holes in the serialization order. If A sees and applies W_4 later on, all writes will in order.

To formalize, a write is either *accepted* or *rejected* by the replica (and its corresponding partition) to which the write is submitted. The replica accepting a write is called the write's *originating replica*. After the write is accepted, the originating replica may *apply* the write to its local data store. At the same time, the originating replica may propagate the write to other replicas, and the other replicas may then apply the write to their local data stores as well. The applied write may be out of order. Finally, after the serialization order is determined, the write becomes *committed* if all writes before it in the serialization order have been seen and applied to the data store. Different replicas may commit the same write at different times based on when they become aware of the serialization order.

Table I. Notations Used in Our Theory (Suppose in the evolution graph, that the node containing $replica_i$ during $interval_k$ is $node_{k,m}$.)

Notation	Meaning
$writes_{k,m}$	Number of writes accepted by replicas in $partition_m$ during $interval_k$
$wsubmit_{k,m}$	Number of writes submitted by clients in $partition_m$ during $interval_k$
$unseen_{k,i}$	Set of writes accepted by all nodes $node_{k',m'}$ in the evolution graph where $k' \leq k$, $node_{k',m'} \neq node_{k,m}$ and $node_{k',m'}$ is not an ancestor of $node_{k,m}$
$seen_{k,i}$	The set of writes accepted by $node_{k,m}$ and all its ancestors
$applied_{k,i}$	The set of writes that $replica_i$ has applied to its local database by the end of $interval_k$
$inorder_{k,i}$	The set of writes in $seen_{k,i}$ that are in order (according to a serialization order)
$stale_k$	The set of writes accepted during $interval'_k$ such that (End time of $interval_k$ − end time of $interval'_k$) $\geq bound_{ST}$

Suppose the node containing $replica_i$ of $interval_k$ in the evolution graph is $node_{k,m}$. Let $seen_{k,i}$ be the set of writes accepted by $node_{k,m}$ and all its ancestors. Notice that once one replica sees a write, all replicas belonging to the same node (in the evolution graph) see the write. Also, we define $seen_{k,i}$ for each replica instead of just for each node (i.e., a network partition) so that we can argue about the consistency of each replica. Hence we define $seen_{k,i}$ instead of $seen_{k,m}$. Let $applied_{k,i}$ be the set of writes that $replica_i$ has applied to its local database by the end of $interval_k$, which can be any subset of $seen_{k,i}$. The writes in $applied_{k,i}$ may or may not have been committed. Notice that the replica has the freedom to choose the value for $applied_{k,i}$ and in the limit, $applied_{k,i}$ can be the empty set regardless of $seen_{k,i}$. Suppose the serialization order we consider is D which is an ordered list of writes. Let $inorder_{k,i}$ be D's longest prefix that is a subset of $seen_{k,i}$. Those are the writes in $seen_{k,i}$ that are in order according to D. In other words, we always have $inorder_{k,i} \subseteq seen_{k,i}$. The replica's order error then equals $|applied_{k,i} \setminus inorder_{k,i}|$, and it must be within specified bound $bound_{OE}$:

$$applied_{k,i} \subseteq seen_{k,i} \tag{1}$$

$$|applied_{k,i} \setminus inorder_{k,i}| \leq bound_{OE}. \tag{2}$$

Here the operator $\setminus$ means excluding all elements in $inorder_{k,i}$ from $applied_{k,i}$. Notice that setting $applied_{k,i}$ to $\emptyset$ will trivially satisfy the constraints on order error, but the constraint on numerical error in the next section will rule out this trivial solution.

In the inequalities, $inorder_{k,i}$ depends on the serialization order D. Enumerating all possible serialization orders is impractical, given that they can be any total order of accepted writes. Furthermore, we do not yet know the exact set of accepted writes. In the following, we will gradually distill better serialization orders, with the ultimate goal of reducing the set to a small size for practical problems. We further show that it is possible to enumerate those orders without even knowing which writes are accepted.

We first fix the set of accepted writes and define the better relationship among serialization orders. Directly defining better serialization orders that result in higher availability can be confusing. Instead, we define that with better serialization orders, more writes will be in order.

Table II. Hierarchy of Dominating Serialization Order Sets (Each serialization order set dominates the one below.)

Serialization Order Set	Size of the Set for Faultloads we Study in Section 6	Proof of this Set Dominating the Next	Key Transformation
ALL	$\sim 10^{10,000}$	—	—
ANCESTOR	Varies	Lemma 4.6	$S_1W_2S_2W_1S_3 \Rightarrow S_1W_1W_2S_2S_3$
CLUSTER	$<5,000$	Lemma 4.7	$S_1W_1S_2W_2 \ldots S_nW_nS_{n+1}$ $\Rightarrow S_1W_1W_2 \ldots W_nS_2 \ldots S_nS_{n+1}$
Logical serialization orders	$<5,000$	Lemma 4.8	$Y_1Y_2 \ldots Y_n$ $\Rightarrow Z_1X_1Z_2X_2 \ldots Z_mX_mZ_{m+1}$

Definition 4.5. Given an evolution graph and the set of accepted writes, serialization order D_1 *dominates* serialization order D_2 if the $inorder_{k,i}$ based on D_1 is a superset of the $inorder_{k,i}$ based on D_2, for all valid k and i. In other words, with the given evolution graph and the given set of accepted writes, any writes that can be committed based on D_1 must also be committable based on D_2.

Notice that, trivially, the domination relation is transitive. We can see that this definition does result in better serialization orders in terms of availability since any execution that uses D_2 can also use D_1 without violating any of the consistency constraints. The rigorous proof (Lemma 4.9), however, has to be delayed until we derive all the constraints in the optimization problem.

Next we will perform three distillation steps to reduce the set of serialization orders we need to consider. These three steps are summarized in Table II. The three steps will distill the set of all serialization orders to the set *ANCESTER*, and then distill *ANCESTER* to the set *CLUSTER*, and finally distill *CLUSTER* to the set of logical serialization orders. The number of logical serialization orders (i.e., the size of the last set) is below 5, 000 for all the faultloads that we will study in Section 6, and thus it is computationally tractable to enumerate all logical serialization orders.

Our first step of distilling serialization orders is based on the ancestor partial order among writes. A write W_1 is an *ancestor* of another write W_2 if either i) the node in the evolution graph accepting W_1 is an ancestor of the node accepting W_2, or ii) W_1 and W_2 are accepted by the same node, and W_1 is accepted before W_2. Intuitively, since the ancestors of a write are always seen before the write itself, it is better to place the ancestors before the write in the serialization order. We construct *ANCESTER* which is the set of serialization orders that are compatible with the ancestor partial order. In Figure 2, a serialization order for W_1, W_2, W_3, and W_4 can be any of the 24 possible permutations of the four writes while *ANCESTER* only contains $W_1W_2W_3W_4$, $W_1W_2W_4W_3$, and $W_1W_4W_2W_3$. The following lemma shows why *ANCESTER* dominates all serialization orders.

LEMMA 4.6. *For any serialization order D, there exists a serialization order* $D' \in$ *ANCESTOR such that* D' *dominates D.*

PROOF. We rearrange D step-by-step until it becomes compatible with ancestor order, while making sure that the new serialization order dominates D.

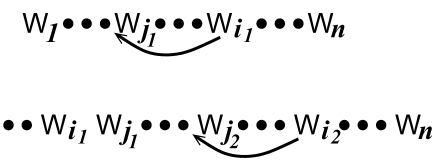

Fig. 4. Adjust writes to make the serialization order compatible with ancestor order.

Consider any two writes W_1 and W_2, where W_1 is an ancestor of W_2. Suppose W_2 precedes W_1 in $D = S_1W_2S_2W_1S_3$, where S_1, S_2 and S_3 are arbitrary write sequences. We construct another serialization order $D' = S_1W_1W_2S_2S_3$ and show that for any k and i, $inorder_{k,i}$ based on D' is a superset of $inorder_{k,i}$ based on D. For any write $W \in inorder_{k,i}$ based on D, we show that W also belongs to $inorder_{k,i}$ based on D' in all the following cases.

(1) $W \in S_1$. Since $W \in inorder_{k,i}$ based on D, we know that $seen_{k,i}$ at least includes those writes in S_1 that are before W. Since all preceding writes of W in D' already belong to $seen_{k,i}$, W belongs to $inorder_{k,i}$ based on D'.

(2) $W \in S_3$. Similar to the first case.

(3) $W = W_2$. Because $W \in inorder_{k,i}$ based on $D = S_1W_2S_2W_1S_3$, we know that $S_1 \subseteq seen_{k,i}$ and $W_2 \in seen_{k,i}$. Because W_1 is an ancestor of W_2, we also have $W_1 \in seen_{k,i}$. Thus, we know $(S_1 \cup \{W_1, W_2\}) \subseteq seen_{k,i}$. Since $D' = S_1W_1W_2S_2S_3$, we know $W_2 \in inorder_{k,i}$ based on D'.

(4) $W \in S_2$. Similar to the previous case.

(5) $W = W_1$. Because $W_1 \in inorder_{k,i}$ based on $D = S_1W_2S_2W_1S_3$, we know that $S_1 \subseteq seen_{k,i}$ and $W_1 \in seen_{k,i}$. Thus $W_1 \in inorder_{k,i}$ based on D'.

Thus, from the definition of domination, we know D' dominates D.

Next, for every pair of W_1 and W_2 in D, we use the previous technique to rearrange D so that the ordering between W_1 and W_2 no longer conflict with ancestor order. This global rearrangement is possible because the ancestor order, is a partial order, and there exists a topological ordering for the writes. Without loss of generality, suppose $D = W_1W_2 \ldots W_n$, and the topological ordering of the writes based on ancestor order is $W_{i_1}W_{i_2} \ldots W_{i_n}$. We first rearrange W_{i_1}. Let j_1 be the smallest index of the writes in D of which W_{i_1} is an ancestor. If $j_1 < i_1$, we move W_{i_1} to the place immediately before W_{j_1} (Figure 4). Otherwise, we leave W_{i_1} in its original place. After this step, the serialization order does not violate any ancestor relations involved with W_{i_1}. Next use the same technique on W_{i_2} and, if necessary, move it forward to the place immediately before W_{j_2}, where j_2 is the smallest index of the writes of which W_{i_2} is an ancestor. Since W_{i_2} comes after W_{i_1} in the topological ordering, we know that either i) W_{i_1} and W_{i_2} have no ancestor relation, or ii) W_{i_1} is an ancestor of W_{i_2}. In the first case, we know that, after this step, the serialization order does not violate any ancestor relations involved with W_{i_1} or W_{i_2}. In the second case, since W_{i_1} precedes W_{i_2} and W_{i_2} precedes W_{j_2}, we know W_{i_1} precedes W_{j_2}. Thus we have $j_1 \leq j_2$. In other words, W_{i_2} will always be moved to some place after W_{i_1}. So we know in both cases, now the serialization order does not violate any ancestor relations involved with W_{i_1} or W_{i_2}. A simple induction on the number of writes rearranged can prove that we

can always rearrange writes without introducing new conflicts with ancestor order. Finally, within a finite number of steps, we will get a serialization order that dominates D and is compatible with ancestor order. □

For our next step of distillation, define *CLUSTER* to be the set of serialization orders that are in *ANCESTER* and where writes accepted by the same node in the evolution graph cluster together. In our previous example with four writes, *CLUSTER* only contains $W_1W_2W_3W_4$ and $W_1W_4W_2W_3$, since W_2 and W_3 are accepted by the same node and thus cluster together. The intuition of this step is that for writes accepted by the same node either all or none of them belong to $seen_{k,i}$ (for any k and i), thus it never hurts to cluster them together. The following lemma shows that *CLUSTER* dominates *ANCESTER*.

LEMMA 4.7. *For any serialization order $D \in$ ANCESTOR, there exists a serialization order $D' \in$ CLUSTER such that D' dominates D.*

PROOF. We will again adjust D step-by-step until it is in *CLUSTER*, while making sure the new serialization order dominates D. We process the nodes in the evolution graph, one-by-one. Without loss of generality, suppose $D = S_1W_1S_2W_2 \ldots S_nW_nS_{n+1}$, where $W_1, W_2, \ldots, W_n$ are the writes accepted by the node we are processing, and $S_1, S_2, \ldots, S_{n+1}$ are arbitrary write sequences. We construct another serialization order $D' = S_1W_1W_2 \ldots W_nS_2S_3 \ldots S_nS_{n+1}$. We next show the adjustment does not introduce any new conflicts with ancestor order so that D' is also compatible with ancestor order. Since D' is constructed by pushing $S_2, S_3, \ldots, S_n$ toward the end of the serialization order, it suffices to show that any write $W \in \{S_2 \cup S_3 \cup \ldots \cup S_n\}$ is not an ancestor of $W_1, W_2, \ldots,$ or W_n. Since $W_1, W_2, \ldots W_n$ are accepted by the same node in the evolution graph, if W is an ancestor of W_j, for some j between 1 and n, it must be true that W is an ancestor of W_1 also. Because D is compatible with ancestor order, W should belong to S_1 rather than $\{S_2 \cup S_3 \cup \ldots \cup S_n\}$. Thus we know the adjustment will not introduce conflicts with ancestor order, and D' is compatible with ancestor order also.

Next we prove that D' dominates D. For any W, k and i, such that $W \in inorder_{k,i}$ based on D, we show that W also belongs to $inorder_{k,i}$ based on D' in all the following cases.

(1) $W \in S_1$ or $W \in S_{n+1}$. Because the set of writes preceding W does not change when we switch from D to D', if W can be committed in D, then it can also be committed in D'.

(2) $W = W_j$, $1 \leq j \leq n$. When we switch from D to D', the set of writes proceeding W does not expand. As a result, if W can be committed in D, then it can also be committed in D'.

(3) $W \in S_j$, $2 \leq j \leq n$. Because any write in S_j, $2 \leq j \leq n$, comes after W_1 in D, we must have $W_1 \in seen_{k,i}$. Furthermore, since $W_1, W_2, \ldots, W_n$ are accepted by the same node in the evolution graph, from the definition of $seen_{k,i}$, they must all belong to $seen_{k,i}$. If W belongs to $inorder_{k,i}$ based on D, then all preceding writes of W in D' already belong to $seen_{k,i}$ and thus W belongs to $inorder_{k,i}$ based on D'.

We will show that we will never split the sequence $W_1W_2 \ldots W_n$ in the serialization order when processing other nodes in the evolution graph. The reason is that, during the transformation from $D = S_1W_1S_2W_2 \ldots S_nW_nS_{n+1}$ to $D' = S_1W_1W_2 \ldots W_nS_2S_3 \ldots S_nS_{n+1}$, none of the S_i's are ever split. After a finite number of steps, we will obtain a serialization order in *CLUSTER* that dominates D. □

Hence, a small set of serialization orders, *CLUSTER*, dominates all serialization orders. However, the set *CLUSTER* still depends on which writes are actually accepted. For example, if we know that W_1, W_2, and W_3 are accepted (and no other writes are accepted), then *CLUSTER* is uniquely determined. But if W_1, W_2, and W_4 are accepted, then *CLUSTER* will be different. To avoid considering all possible combinations regarding which writes are accepted and which are not, we use the concept of *logical writes* and *logical serialization orders* (a sequence of logical writes). We let each node $node_{k,m}$ in the evolution graph accept a logical write that stands for $writes_{k,m}$ number of physical writes. A replica's order error is computed to be the total number of physical writes in out-of-order logical writes. If all logical writes stand for one or more physical writes, this is equivalent to using physical serialization orders. However, since some of the logical writes may stand for zero physical writes, false holes may appear in the sequence. Fortunately, we can show that this does not affect correctness:

LEMMA 4.8. *Given a logical serialization order $D = Y_1Y_2 \ldots Y_n \in$ CLUSTER, suppose logical serialization order $D_1 = X_1X_2 \ldots X_m$ is obtained by deleting those logical writes in $Y_1Y_2 \ldots Y_n$ that stand for zero physical writes. Then there exists a logical serialization order $D_2 \in$ CLUSTER such that D_2 dominates*[1] *D_1.*

PROOF. We scan D from Y_1 to Y_n. For each Y_j that corresponds to X_i, let Z_i be the sequence of $Y_{k_1}Y_{k_2} \ldots$, $1 \le k_1 \le k_2 \le \ldots < j$, such that all these variables i) stand for zero writes, ii) are ancestors of Y_j, and iii) have not been included in $Z_1, Z_2, \ldots, Z_{i-1}$. Construct $D_2 = Z_1X_1Z_2X_2 \ldots Z_mX_mZ_{m+1}$. Intuitively, D_2 is constructed from $Y_1Y_2 \ldots Y_n$ by pushing all variables in $Y_1Y_2 \ldots Y_n$ standing for zero writes, as far to the right as possible without violating the ancestor order. We will show that $D_2 \in CLUSTER$, and D_2 dominates D_1.

Because D_2 is a logical serialization order and each node in the evolution graph only accepts one logical write to prove $D_2 \in CLUSTER$, it suffices to show $D_2 \in ANCESTOR$. For any two logical writes, Y_i and Y_j such that Y_i is an ancestor of Y_j, we consider the following cases.

(1) Y_i and Y_j both stand for one or more physical writes. When we construct D_2 from D, the relative position of Y_i and Y_j is not changed. Since D is compatible with ancestor order, we know that Y_i precedes Y_j in D_2 also.
(2) Y_i stands for zero writes, while Y_j stands for one or more physical writes. Y_i will not be pushed past Y_j when we construct D_2.

[1]Here we slightly abuse the definition of dominate, since D_1 and D_2 contain different sets of logical writes. However, the meaning should be clear from the context.

(3) Y_i stands for one or more physical write, while Y_j stands for zero writes. Since $i < j$ and Y_j is never moved toward the beginning of the serialization order, Y_i is still before Y_j in D_2.
(4) Y_i and Y_j both stand for zero writes. Suppose in D, Y_j belongs to Z_k. We know that Y_j must be an ancestor of X_k, and thus Y_i is also an ancestor of X_k. So Y_i either belongs to Z_k, or one of $Z_1, \ldots, Z_{k-1}$ and is therefore before Y_j in D_2.

PROOF. Next, we show that D_2 dominates D_1. Consider any physical write W represented by logical write X_j. If $W \in inorder_{k,i}$ based on D_1 for any k, i, it must be true that $\{X_1, X_2, \ldots, X_j\} \in seen_{k,i}$. Given that Z_1 is an ancestor of X_1, Z_2 is an ancestor of $X_2, \ldots, Z_j$ is an ancestor of X_j, $Z_1, Z_2, \ldots, Z_j$ must also belong to $seen_{k,i}$. Thus we have $W \in inorder_{k,i}$ based on D_2. □

With the previous lemma, we know that it is safe to use logical serialization orders in place of physical ones, and the set *CLUSTER* is actually the set of all topological orderings of the evolution graph as a DAG.

4.2.2 *Effects of Numerical Error and Staleness.* Informally, numerical error is the number of missing writes on a replica. Formally, suppose the node containing $replica_i$ of $interval_k$ in the evolution graph is $node_{k,m}$. Let $unseen_{k,i}$ be the set of writes accepted by all nodes $node_{k',m'}$ such that i) $k' \leq k$, ii) $node_{k',m'} \neq node_{k,m}$, and iii) $node_{k',m'}$ is not an ancestor of $node_{k,m}$. In other words, we partition all writes accepted by the system by the end of $interval_k$ into two disjoint sets, $unseen_{k,i}$ and $seen_{k,i}$, for any valid i. Along with our previous notations, the numerical error of the $replica_i$ at the end of $interval_k$ is $|unseen_{k,i}| + |seen_{k,i} \setminus applied_{k,i}|$. Notice that if a write W has been seen by a replica, the replica may or may not choose to apply W to the data store. If it applies W to the data store, then it reduces numerical error but potentially increases order error (if W cannot be immediately committed). As a result, $applied_{k,i}$ can be any subset of $seen_{k,i}$, and we must add $|seen_{k,i} \setminus applied_{k,i}|$ to the numerical error of the replica.

Suppose $bound_{NE}$ is the maximal allowed numerical error, we have:

$$|unseen_{k,i}| + |seen_{k,i} \setminus applied_{k,i}| \leq bound_{NE}. \tag{3}$$

The third metric in the consistency model, staleness, bounds the age of missing writes. For any write W accepted at wall-clock time t, a replica must see W by time $t + bound_{ST}$ in order to be consistent, where $bound_{ST}$ is the bound on staleness. For any $interval_k$, define $endtime(interval_k)$ to be the wall-clock time that $interval_k$ ends. Then a consistent replica in $interval_k$ must see all writes accepted before time $endtime(interval_k) - bound_{ST}$. Without loss of generality, we assume $endtime(interval_k) - bound_{ST}$ falls on the boundary of two intervals: We can always split the interval if it falls in the middle of an interval. Define $stale_k$ to be the set of all writes accepted during $interval_{k'}$ such that $endtime(interval_k) - endtime(interval_{k'}) \geq bound_{ST}$. Given that for $replica_i$ in $interval_k$, the set of missing writes is $unseen_{k,i} \cup (seen_{k,i} \setminus applied_{k,i})$, staleness

$$\textbf{Maximize:} \sum_{k,m} writes_{k,m}$$
$$\textbf{Subject to:} \begin{cases} writes_{k,m} \leq wsubmit_{k,m} \quad \textbf{for all } k, m \\ applied_{k,i} \subseteq seen_{k,i} \quad \textbf{for all } k, i \\ |applied_{k,i} \setminus inorder_{k,i}| \leq bound_{OE} \quad \textbf{for all } k, i \\ |unseen_{k,i}| + |seen_{k,i} \setminus applied_{k,i}| \leq bound_{NE} \quad \textbf{for all } k, i \\ |(unseen_{k,i} \cup (seen_{k,i} \setminus applied_{k,i})) \cap stale_k| = 0 \quad \textbf{for all } k, i \end{cases}$$

Fig. 5. Complete optimization problem.

imposes the following constraint.

$$|(unseen_{k,i} \cup (seen_{k,i} \setminus applied_{k,i})) \cap stale_k| = 0. \tag{4}$$

Inequalities 1, 2, 3, and 4 are the new constraints introduced by consistency requirements, and we write our complete optimization problem in Figure 5. The optimization problem, at this point, is not an IP (integer linear programming) problem yet, and we will show how to transform it to IP in the next section. To compute the availability upper bound, we first enumerate all logical serialization orders based on the evolution graph. We then construct an optimization problem for each order enumerated and choose the highest availability across all logical serialization orders as the upper bound.

4.3 Correctness and Tightness of the Upper Bound

4.3.1 *Correctness of Upper Bound.* Based on the optimization problem in Figure 5, we first show that our definition of domination among serialization orders does result in better serialization orders in terms of availability.

LEMMA 4.9. *Given an execution that results in serialization order D_1 and satisfies all constraints in Figure 5, if serialization order D_2 dominates D_1, then there must exist another execution that results in D_2 and also satisfies all constraints in Figure 5.*

PROOF. The only constraint that depends on serialization order is $|applied_{k,i} \setminus inorder_{k,i}| \leq bound_{OE}$. Switching from D_1 to D_2 will always expand the set $inorder_{k,i}$ for any k and i. Thus, this constraint will not be violated. □

Next, to present a rigorous form of the upper bound theorem, we translate Inequalities 1, 2, 3, and 4 to linear inequalities[2] so that the IP problem can be fully specified. When we fix the logical serialization order, for any k and i, $unseen_{k,i}$, $seen_{k,i}$ and $inorder_{k,i}$ are already uniquely determined. The only free set is $applied_{k,i}$. We will isolate all terms containing $applied_{k,i}$ in the inequalities and try to substitute $|applied_{k,i}|$ for $applied_{k,i}$. Doing this will allow us to use a single numerical variable to represent the value of $|applied_{k,i}|$ in the IP problem. We introduce two functions used in the transformation. Given a logical serialization order and a set of writes S, define the function $IO(S)$ to be the set of writes that are in order according to the serialization order.

[2]Lemma 4.9 can also be proved after this transformation, but it is much clearer to prove it based on the original constraints.

For example, with our definition for $seen_{k,i}$ and $inorder_{k,i}$, we actually have $inorder_{k,i} = IO(seen_{k,i})$. We also define function $OO(S) = S \setminus IO(S)$, that is, the set of writes in S that are out of order.

We trivially weaken Inequality 1 to:

$$\begin{aligned} |IO(applied_{k,i})| &\leq |IO(seen_{k,i})| \\ |OO(applied_{k,i})| &\leq |OO(seen_{k,i})|. \end{aligned}$$

We then transform Inequality 2:

$$\begin{aligned} bound_{OE} &\geq |applied_{k,i} \setminus inorder_{k,i}| \\ &= |(IO(applied_{k,i}) \cup OO(applied_{k,i})) \setminus inorder_{k,i}| \\ &= |OO(applied_{k,i})|. \end{aligned}$$

For Inequality 3, since $applied_{k,i}$ is always a subset of $seen_{k,i}$, the following inequality is equivalent:

$$|unseen_{k,i}| + |seen_{k,i}| - |IO(applied_{k,i})| - |OO(applied_{k,i})| \leq bound_{NE}.$$

Inequality 4 needs to be decomposed into two inequalities:

$$\begin{aligned} |unseen_{k,i} \cap stale_k| &= 0, \\ |(seen_{k,i} \setminus applied_{k,i}) \cap stale_k| &= 0. \end{aligned}$$

The first inequality can already be expressed linearly. For the second one, notice that:

$$\begin{aligned} 0 &= |(seen_{k,i} \setminus applied_{k,i}) \cap stale_k| \\ &= |(seen_{k,i} \cap stale_k) \setminus (applied_{k,i} \cap stale_k)| \\ &\geq |(seen_{k,i} \cap stale_k) \setminus applied_{k,i}|. \end{aligned}$$

Because the size of a set is always nonnegative, we must have $|(seen_{k,i} \cap stale_k) \setminus applied_{k,i}| = 0$. This equality is then equivalent to the following two:

$$\begin{aligned} |IO(seen_{k,i} \cap stale_k) \setminus IO(applied_{k,i})| &= 0, \\ |OO(seen_{k,i} \cap stale_k) \setminus OO(applied_{k,i})| &= 0. \end{aligned}$$

We can further weaken them to:

$$\begin{aligned} |IO(seen_{k,i} \cap stale_k)| - |IO(applied_{k,i})| &\leq 0, \\ |OO(seen_{k,i} \cap stale_k)| - |OO(applied_{k,i})| &\leq 0. \end{aligned}$$

Now we can use two numerical variables to represent $|IO(applied_{k,i})|$ and $|OO(applied_{k,i})|$, and Inequalities 1, 2, 3, and 4 will all become linear equalities. Next we can prove that the upper bound is correct.

THEOREM 4.10. *Given a workload, faultload, and required consistency level (i.e., bounds on numerical error, order error, and staleness), we compute an availability upper bound in the following way. We construct the evolution graph for the faultload and enumerate all topological orderings of the evolution graph. For each ordering enumerated, we solve the IP problem in Figure 6. Finally, we*

$$\textbf{Maximize}: \sum_{k,m} writes_{k,m}$$
$$\textbf{Subjectto}: \begin{cases} writes_{k,m} \le wsubmit_{k,m} \quad \textbf{for all } k, m \\ |IO(applied_{k,i})| \le |IO(seen_{k,i})| \quad \textbf{for all } k, i \\ |OO(applied_{k,i})| \le |OO(seen_{k,i})| \quad \textbf{for all } k, i \\ |OO(applied_{k,i})| \le bound_{OE} \quad \textbf{for all } k, i \\ |unseen_{k,i}| + |seen_{k,i}| - |IO(applied_{k,i})| - |OO(applied_{k,i})| \\ \qquad\qquad \le bound_{NE} \quad \textbf{for all } k, i \\ |unseen_{k,i} \cap stale_k| = 0 \quad \textbf{for all } k, i \\ |IO(seen_{k,i} \cap stale_k)| - |IO(applied_{k,i})| \le 0 \quad \textbf{for all } k, i \\ |OO(seen_{k,i} \cap stale_k)| - |OO(applied_{k,i})| \le 0 \quad \textbf{for all } k, i \end{cases}$$

Fig. 6. Complete IP problem.

pick the maximal optimal solution among all optimal solutions obtained. No system can achieve a higher availability than the availability we obtain.

PROOF. First, if we fix the serialization order, our approach clearly gives us the maximal availability among all possible executions that will result in the given serialization order. This is true because any execution must satisfy all the constraints in our optimization problem in Figure 5. From the way we translate the constraints in Figure 5 to linear inequalities, we know all the constraints in our IP problem (Figure 6) must also be satisfied by the execution.

Next, we prove the upper bound for all serialization orders by contradiction. Suppose there exists an execution that can achieve higher availability than what we obtain. Obviously, the execution must still satisfy all the constraints in Figure 5. Let D be the serialization order of the execution. From Lemma 4.6, 4.7, and 4.8, we know that there exists another serialization order $D' \in CLUSTER$, such that D' dominates D. From Lemma 4.9, we know there must exist another execution that results in D' and also satisfies all constraints in Figure 5 (and, in turn, all the constraints in Figure 6). Since we already solved the IP problem based on D', we know the availability we obtain must be higher than or equal to the original execution's availability. Contradiction. □

4.3.2 *Tightness of the Bound.*

THEOREM 4.11. *The upper bound on availability obtained in Theorem* 4.10 *is tight.*

PROOF. We need to show that there exists an execution reaching the bound. We already have a serialization order D and an optimal solution of the IP problem that can achieve the maximal availability. In the execution constructed, for $replica_i$ in $partition_m$ during $interval_k$, we have it accept $write_{k,m}$ writes among those writes submitted to $partition_m$ during $interval_k$.

The only thing we need to prove now is that all replicas are consistent (i.e., all three metrics are within bounds) all the time. We first show that $replica_i$ is consistent at the end of $interval_k$, for any k and i, by carefully assigning writes to the set $applied_{k,i}$. We know that the following constraints are satisfied:

$$|IO(applied_{k,i})| \le |IO(seen_{k,i})|,$$
$$|OO(applied_{k,i})| \le |OO(seen_{k,i})|,$$

$$|IO(seen_{k,i} \cap stale_k)| \leq |IO(applied_{k,i})|,$$
$$|OO(seen_{k,i} \cap stale_k)| \leq |OO(applied_{k,i})|.$$

We first set $IO(applied_{k,i})$ to $IO(seen_{k,i} \cap stale_k)$, and then add writes in $IO(seen_{k,i} \setminus stale_k)$ to $IO(applied_{k,i})$, until the size of the set reaches $|IO(applied_{k,i})|$ in the optimal solution. The set $OO(applied_{k,i})$ is constructed in a similar way. Note that such construction is always possible because the previous inequalities are satisfied. The sets $IO(applied_{k,i})$ and $OO(applied_{k,i})$ uniquely determine $applied_{k,i}$. With the set $applied_{k,i}$ constructed in this manner, because all the constraints in the optimization problem are satisfied, the replica is consistent with respect to the given $bound_{OE}$, $bound_{NE}$, and $bound_{ST}$ at the end of $interval_k$.

We still need to prove that the replica is consistent not only at the end of $interval_k$, but also throughout the duration of $interval_k$. We already know the set of writes that the replica has applied to the database by the end of $interval_k$ (which is $applied_{k,i}$). We now construct how the replica should apply writes to its database in the beginning and middle of $interval_k$. Let S be the set of writes accepted by $partition_m$ during $interval_k$. At the beginning of $interval_k$, the replica applies all writes in the set $applied_{k,i} \setminus S$ to its database. As new writes become accepted by the partition, the replica applies them to the database upon acceptance, if they are in $applied_{k,i}$.

In order to show the replica is consistent throughout the interval, it suffices to prove that numerical error, order error, and staleness never decrease during the interval since the replica is already consistent at the end of the interval. Numerical error and staleness can only increase within an interval when the network connectivity does not change. For order error, note that among all writes in $OO(applied_{k,i} \setminus S)$, none will become in order during $interval_k$. This can be proved using a simple contradiction. Suppose a write W, $W \in OO(applied_{k,i} \setminus S)$, becomes in order during $interval_k$. The write W must be seen by the replica before $interval_k$ since W does not belong to S. Thus W must be an ancestor of any write in S. Given that the serialization orders we enumerate are all compatible with ancestor order, W must precede S in the serialization order. Thus accepting the writes in S will not fill in the holes in the serialization order for W. If W will be in order by the end of $interval_k$, it must already be in order at the beginning of $interval_k$ which means W does not belong to $OO(applied_{k,i} \setminus S)$ in the first place. Contradiction. Thus we know that order error will not decrease either and the replica must be consistent throughout the interval.

We have shown that the execution constructed is consistent with respect to the given order error, numerical error, and staleness bounds. Since the execution also achieves the availability upper bound, the bound is tight. □

4.3.3 *Complexity and Experience.* Our algorithm to compute the availability upper bound is implemented in approximately 2,000 lines of C++ code. Numerical error, order error, staleness, and a file describing faultload and workload serve as inputs to the algorithm. The code first constructs the evolution graph. Next, it enumerates all serialization orders as described previously and uses a

linear programming solver (PCx [Czyzyk et al.]) to calculate the upper bound on availability based on the specified constraints.

The algorithm has exponential complexity in terms of the number of intervals. Such complexity comes from both integer linear programming and enumeration of serialization orders. We avoid the exponential complexity of integer linear programming by approximating through real-number linear programming. The complexity from serialization order enumeration has been tractable for all our faultloads described in the next section, and our algorithm terminates within three hours on a Sun Ultra-5 Workstation for any of our faultloads. The reason for the short execution time is that, in realistic faultloads, complex network partition scenarios are rare. On the other hand, the total number of serialization orders that need enumerating grows with the number of failures and increases significantly only when the failure scenarios are complex. Some of our simulated faultloads already have higher average failure rates (i.e., 5%) than what was reported (i.e., 2%) for today's Internet [Dahlin et al. 2003]. Thus, we believe our algorithm will be computationally tractable for most realistic faultloads.

5. IMPLEMENTATION

In this section, we discuss details of our prototype replication system, implementation of various consistency protocols, an approach for measuring a sample faultload and creating additional synthetic faultloads with varying characteristics, and the emulation environment used to conduct sensitivity analysis to various network failure conditions.

5.1 Measuring a Sample Faultload

One of the key factors influencing the availability of Internet services is the rate of failures in the underlying network. Existing work on network measurement [Paxson 1996; Savage et al. 1999] uses fairly coarse-grained interarrival times for successive measurements between two sites with a minimum interarrival of approximately 10 minutes. Such a granularity cannot unambiguously capture the duration of short network failures which make up a majority of failure events [Paxson 1996]. Thus, we set out to collect a sample of Internet connectivity with average measurement intervals of 3 seconds on each path. The total duration of the trace is 6 days in February 2001 with over 12 million samples. The interval of 3 seconds is a balancing point between accuracy and incurred network traffic. The samples are carefully laid out to avoid simultaneous measurements to multiple destinations. Table III summarizes the characteristics of the sites used for our measurements.

One issue in measuring the faultload is determining the proper network layer to conduct our experiments. IP (layer 3), TCP (layer 4), and RPC (layer 7) would each produce different failure characteristics. Because our sample replication system communicates using Java's remote method invocation (RMI), we use an RMI-based measurement utility to determine a sample faultload to best match the failures experienced by our prototype. Using the same faultload for deriving an upper bound on availability as well as for our deployed prototype

Table III. Sites Used in Measuring Faultload

Site	Location
128.121.247.90	Verio (CA)
212.100.224.106	Rackspace (U.K.)
155.101.132.13	Univ. of Utah
132.239.10.32	UC San Diego
128.83.121.4	Univ. of Texas, Austin
128.32.44.137	UC Berkeley
152.3.144.58	Duke Univ.
209.61.189.44	Rackspace (TX)

will enable a direct comparison between achieved availability and the upper bound. If we instead performed the measurement at the IP layer, then the failures would not be the same as observed by our prototype system. On the other hand, if the system indeed used direct IP for communication, it would experience a potentially different faultload from our measurement and achieve different availability.

A difficulty with using Java to measure a faultload is the blocking nature of Java's communication primitives. If an RMI experiences a network failure, it simply blocks until the network recovers. Such blocking is problematic because retries are limited by the retransmission rate of the underlying TCP implementation which varies from platform to platform. We address this limitation by killing threads that do not return from an RMI call within 500 ms. Note that the specification of a failure depends upon the determination of a timeout. Since there is no correct value for this timeout, we arbitrarily choose 500ms for our trace.

For our experiments, each site runs an autonomous Java program with 32 worker threads, 4 for each destination path (each site also probes itself). The program invokes a null RMI to every other site every 3 seconds. We use multiple threads for the same path so that if one is delayed, we can still maintain our target 3-second sample interval. If each RMI incurs one outgoing and one incoming TCP packet, our tool incurs less than 10KB/sec of bandwidth in both directions. Given that our site's network connections are well provisioned, this rate should be small enough to avoid excessive network perturbation. We have dedicated accesses to the machines, and the program consumes approximately 1% CPU time on each site, greatly reducing the possibility of mistaking an overloaded site as a network failure.

Since an RMI is based on round-trip communication, a failure in either direction results in an observed failure in both directions. Thus, we merge samples taken from any site A to another site B with samples from B to A. This technique decreases the average interarrival of samples to 1.5 seconds and makes the failures in our faultload symmetric. To derive a faultload, we assume that failures last for the entire duration of failed samples and also that no failures take place between two successive successful samples. These are standard assumptions made in previous studies [Dahlin et al. 2003] on network availability, and the granularity of our trace ensures that the inaccuracy is relatively small. We generate a connectivity matrix for the system as a function of time and then

calculate the transitive closure of the matrix to allow for the ability to mask failures by routing through other replicas [Savage et al. 1999]. Using the transitive closure serves to isolate the availability effects of continuous consistency from the effects of application-level routing.

The faultload we obtain has an average failure time on all paths of 0.046%. To make our evaluation computationally tractable, we focus on the first day of this trace (called SAMPLED1) which has the highest failure rate of 0.17%. We report network failure rate as the average unavailability of all paths in a given faultload after taking the transitive closure. Note that the average failure rate cannot abstract many important aspects of a faultload. For example, the timing and nature of failures can mean that two faultloads with the same average failure rates can result in very different levels of service availability.

5.2 Simulated Faultloads

Since we are interested in understanding how the availability of Internet services is affected by a broad range of faultloads, we use our experience with our measured faultload to construct a number of synthetic faultloads with varying characteristics. This also alleviates the concern of whether our measured faultload is representative. In fact, our measurement sites tend to be well-connected with relatively little network congestion. In general, we believe our measured faultload underreports failures relative to the Internet in general. Having simulated faultload thus enables us to perform a more extensive study.

We use a simple event-driven simulator to obtain diverse faultloads based on a sample Internet-like topology generated by the Internet topology generator[Zegura et al. 1997]. The target topology is a hierarchical 600-router transit-stub topology with a per-node degree of 4.09, a 14-hop network diameter, an average of 10.8 hops between nodes, and 188 biconnected components. There are 24 backbone routers in the sample topology. For each of these routers, we choose a stub router among the stub domains that is directly connected to the backbone and attach a replica to that stub router. Thus, we can vary the number of replicas from 1 to 24, modeling the case where replicas are widely dispersed and placed at well-connected points in the network topology.

Next, for each link and node in the graph, we use a failure arrival and failure duration model to create failure and recovery events. Based on previous Internet trace analysis results [Dahlin et al. 2003], we use exponential distribution for both failure interarrival and failure duration. For router failures, we also use a correlation model to avoid the pitfall of assuming independent failures [Amir and Wool 1996]. Whenever a node fails, we assume that the entire domain where the node resides completely fails (because of correlated failures) with 1% probability and that adjacent nodes fail with 5% probability. We have also experimented with other correlation values, and the results are qualitatively similar. A variety of faultloads can then be generated by computing the connected components of the graph as a function of time. Table IV presents the details of the faultloads used in our availability study, and Table V summarizes the details of our simulated faultloads.

Table IV. Characteristics of Faultloads

Faultload	Description	Avg. Fail. Rate
SAMPLED1	First day of the RMI-ping trace	0.17%
SIM0_10	Simulated trace	0.10%
SIM0_50	Simulated trace	0.55%
SIM1_00	Simulated trace	1.10%
SIM5_00	Simulated trace	4.12%
STRESS	Constructed	0.16%

Table V. Simulated Faultloads and Simulation Parameters

Faultload	Average Fail. Rate	MTTF for Node	MTTR for Node	MTTF for Link	MTTR for Link
SIM0_01	0.010%	10 days	12 sec	20 weeks	15 sec
SIM0_10	0.10%	3 days	36 sec	6 weeks	45 sec
SIM0_50	0.55%	1 day	60 sec	2 weeks	75 sec
SIM1_00	1.10%	1 day	120 sec	2 weeks	150 sec
SIM2_00	2.19%	1 day	240 sec	2 weeks	300 sec
SIM3_00	3.29%	1 day	363 sec	2 weeks	454 sec
SIM5_00	4.12%	1 day	545 sec	2 weeks	681 sec

5.3 WAN Prototype Details

We now describe our modifications to the TACT prototype [Yu and Vahdat 2002] to study the replicated service availability as a function of consistency and fault-load. The prototype is written in Java based on RMI, and write propagation is performed through anti-entropy [Petersen et al. 1997]. It supports three simple replicated services: an airline reservation system, a bulletin board system, and a load-balancing request distributor [Yu and Vahdat 2002]. For this article, we run our availability experiments using the bulletin board service. In addition to fault-load and consistency level, the availability of a replicated service will also depend upon the specifics of the consistency maintenance protocol—whether it is strong consistency, optimistic consistency or somewhere in between. We thus extend the prototype to use a variety of consistency protocols, as described in the following.

Our prototype uses the only protocol we are aware of to bound numerical error [Yu and Vahdat 2000]. Each replica ensures that the error bounds on other replicas are not violated. If necessary, the replica may push writes to other replicas before accepting a new write. For example, if numerical error is uniformly set to twenty at all replicas and there are eleven total replicas, each replica may buffer at most two unseen writes before propagating those writes to other replicas. If this write propagation cannot be performed, future writes must be rejected (decreasing service availability) to ensure that the numerical error bounds on other replicas are not violated. In the case where the total number of replicas is not known beforehand, the allowed numerical error can be considered as a certain number of tokens, and the tokens are then distributed among the replicas via gossiping.

A number of different write commitment algorithms can be used for bounding order error. We implement three such popular protocols in our prototype,

primary copy [Petersen et al. 1997], Golding's algorithm [Golding 1992], and voting [Keleher 1999]. Essentially, all three write-commitment algorithms determine a total order on all writes. For completeness, we present a brief summary of each protocol.

First, in the primary copy protocol, a write is committed when it reaches the primary replica, and the serialization order is the write order seen by the primary replica. A replica that needs to reduce order error commits writes by first pushing its tentative writes to the primary and then pulling any other unseen updates from the primary.

Next, in Golding's algorithm, each write is assigned a logical timestamp that determines the serialization order. The logical timestamp is simply the Lamport logical time [Lamport 1978] when the write is accepted. Each replica maintains a logical time vector [Golding 1992] to determine whether or not it has seen all writes (some of which are potentially accepted by other replicas) with logical timestamp less than t. If so, it is able to commit all writes with logical timestamp smaller than t. To reduce order error in Golding's algorithm, a replica pulls writes from other replicas to advance its logical time vector (and hence commit additional writes).

Finally, a voting protocol conducts a series of elections to determine a serialization order. During a round, each replica casts a vote for the first uncommitted write in their write log. The write with the most votes wins and is committed next (in serialization order) across all replicas. While the weight of votes cast by each replica may be varied dynamically, our evaluation considers only unit weight (though our implementation is more general). Votes are also propagated through anti-entropy. We use special techniques in our prototype for multiple rounds of elections to be in progress simultaneously [Keleher 1999], greatly improving system performance. To reduce order error with voting, a replica first pushes writes to remote sites. These sites then cast their votes, and the results are pulled in subsequent anti-entropy sessions to allow write commitment.

5.4 Emulation Environment and Verification

A goal of this work is to study service availability while varying system characteristics, including consistency level, consistency protocol, and faultload. Unfortunately, it is not possible to fix Internet failure characteristics while reexecuting our prototype with different consistency levels and protocols. Thus, we perform the majority of our evaluation using a local area emulation environment. Emulation accuracy is verified through live wide-area deployment. Our goal is to run our prototype on a LAN while subjecting it to one of a number of sample faultloads. We instrument our prototype replication system to check a connectivity matrix that varies as a function of time (as determined by a given faultload). An RMI between two nodes is allowed to proceed only if the nodes are connected according to the faultload at the time of the operation. Because we focus on service availability rather than performance, we do not emulate variable latency, bandwidth, or drop rate [Noble et al. 1997].

To validate emulation results, we deploy our prototype system running the replicated bulletin board service on all 8 sites used for measuring faultload

Table VI. Wide-Area Deployment and Emulation Verification Results Under $NE = 6$ and $OE = \infty$

	# Writes	# Rejected	Availability	Accuracy
WAN	120,703	1,699	98.6%	–
Emulation	120,703	1,762	98.5%	96.3%

Table VII. Wide-Area Deployment and Emulation Verification Results Under $NE = \infty$ and $OE = 1$

	# Writes	# Rejected	Availability	Accuracy
WAN	60,439	293	99.5%	–
Emulation	60,439	298	99.5%	98.3%

except for the Rackspace machine in Texas which was not available. We run two separate 24-hour experiments at two different target consistency levels using Golding's algorithm to bound order error. In the first experiment, we set numerical error to six (recall that there are seven replicas total) and leave order error unbounded, while in the second experiment, numerical error is unbounded and order error is set to one. These two consistency levels are intended to stress the emulation environment. With small numerical error and order error, there will be more communication and more events, thus increasing the possibility that the real-time emulator may miss some deadlines or process events in different orders. Our system logs all writes with timestamps. These two runs produce two sample faultloads that we play back to our emulation environment with writes injected at the same rate as the wide-area deployment based on our timestamp logs. Table VI and VII summarize the accuracy of our emulation environment relative to the wide-area deployment. A number of separate smaller-scale deployments yield similar accuracy. While there is room to improve the accuracy of our emulation environment, we believe that the general availability trends revealed by our emulation environment for different consistency levels and faultloads are accurate.

Finally, when either numerical error or order error is set to zero in the presence of failures, an atomic commit protocol (ACP) [Bernstein et al. 1987] is required to determine whether a write can be accepted. ACPs are inherently blocking, meaning that a failure may force an ACP to block until the failure is repaired. However, ACPs and availability under strong consistency are not the focus of this work and different ACPs may result in very different levels of service availability. Thus, for measuring the service availability under strong consistency, we choose to use simulation instead of emulation. Note that all other data points (i.e., when the consistency is relaxed) are derived from our prototype and that our upper bound theory for service availability is general for all combinations of numerical and order error. Upon receiving a write, our code determines the necessary subset of replicas that must be connected for a particular consistency protocol to accept the write (e.g., for voting, it checks if the partition where the write originated forms a majority partition). If the proper subset is not available, the write is rejected.

6. SYSTEM AVAILABILITY

In this section, we quantify our prototype's availability as a function of various faultloads, consistency levels, and consistency protocols. We also compare the achieved availability of our prototype software relative to the tight upper bound. Finally, we explore the effects of the degree of replication on service availability.

We omit an evaluation of the staleness metric and focus on numerical error and order error. Preliminary evaluations and the theory in Section 4.2 show that staleness exhibits behavior very similar to numerical error. For staleness, consistency is bounded by the passage of real time rather than by the acceptance of a given number of writes system-wide; for a fixed update rate, numerical error and staleness are directly related.

6.1 Achieved Availability vs. Upper Bound

The goal of this section is to develop an understanding of the availability characteristics of replicated network services as a function of consistency and failure characteristics. Unless otherwise specified, the results are generated in the LAN emulation environment described in Section 5.4. Thus, data points are from repeated runs of the TACT software while varying: i) numerical and order error, ii) consistency protocols, and iii) faultloads. Our software runs on a cluster of seventy 700-800Mhz Pentiums, running Linux and Solaris, with each machine playing back a target workload and faultload in real time over a 24-hour period for each data point. We compare these data points against the tight upper bound for service availability using the theory from Section 4. The workload is a uniform update rate of one write per 10 seconds on each replica, resulting in 0.8 writes per second system-wide for the eight emulated replicas.

For our initial set of results, we use service availability ($Avail_{service}$ from Section 3) as the availability metric. Because all replicas remain consistent all the time (Section 3), the replicas must always be able to process reads. Thus we only need to focus on writes especially those writes whose acceptance would violate global consistency requirements. So we can redefine service availability to be the percentage of writes that are accepted by the replicas.

Figure 7 plots availability as a function of numerical error for the SIM1_00 faultload (see Table IV). Order error is not bounded in these experiments. The solid curve represents the calculated upper bound on availability as a function of numerical error. This baseline numerical error bounding protocol achieves approximately 1% less availability than the upper bound. Recall that a 1% improvement in availability is significant, corresponding to approximately 3.5 additional days of uptime per year. The baseline protocol enforces numerical error lazily. That is, it does not push writes until required to maintain a given level of consistency, typically resulting in the lowest communication costs. However, results from Section 4.3 show that the best overall availability is achieved through aggressive write propagation so that the replica can actually see all writes in $seen_{k,i}$. Thus, we modify our numerical error-bounding protocol to push writes more aggressively. Here, the system always attempts to maintain a numerical error of zero asynchronously but continues to operate if it fails as long as the target numerical error bounds are not violated. The curve labeled

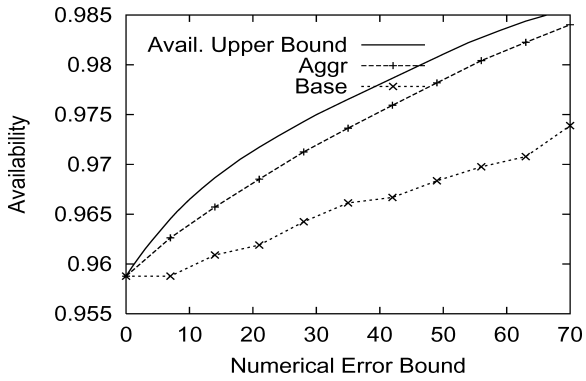

Fig. 7. Availability as a function of numerical error (SIM1_00). We do not use the standard number-of-nines metric for availability because sometime the availability upper bound is 1.0 which translates to an infinite number of nines. Also, the y-axis of the graphs in this section have different ranges since a consistent range would make some of the graphs unreadable.

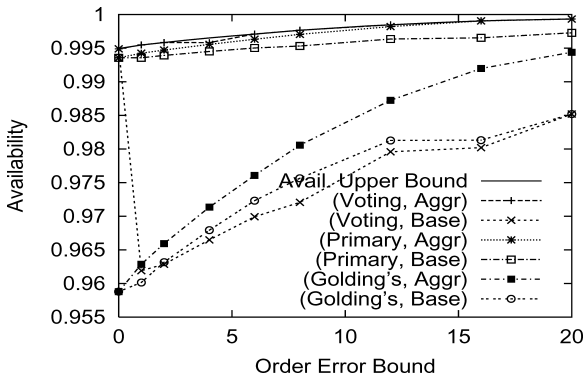

Fig. 8. Availability as function of order error (SIM1_00).

Aggr in Figure 7 depicts the results of this modification. Using this technique, achieved availability closely approaches the upper bound.

Figure 8 shows availability as a function of order error for the SIM1_00 faultload using our order error-bounding protocols; numerical error is not bounded in these experiments. Notice that, when we only bound order error, the system actually admits a trivial solution which is to always use the initial database state and discard all writes accepted. This is possible only because numerical error is not bounded at all. To still be able to study order error in isolation, here we require that a replica applies to its database all writes that it has accepted. We also make a corresponding change in our IP problem to require that $applied_{k,i} = seen_{k,i}$ for all valid k and i.

The graph depicts curves for two versions of each algorithm, one for the baseline implementation that bounds order error lazily and one for an aggressive version that attempts to maintain order error as close to zero as possible. The solid curve shows the computed tight upper bound on availability. Even with aggressive order error bounding, Golding's algorithm achieves low availability

in general because committing a write usually requires the advancement of all entries in a replica's version vector. This operation typically requires full connectivity to pull writes from remote replicas, often forcing the system to reject writes during periods of disconnectivity (SIM1_00 achieves full connectivity only 96% percent of the time).

Primary copy typically delivers the highest level of availability. Our sample faultload contains largely singleton partitions. Thus, with 8 replicas in our experiments, there is a 1/8th probability that the primary copy is inaccessible by the rest of the network (since the failure rates are roughly equal among the replicas). Thus, primary copy should deliver 8 times better availability than Golding's algorithm; this can be verified in Figure 8. Note that the relative advantage of primary copy will increase with the number of replicas. One unique characteristic of the primary copy approach is that its achieved availability is relatively insensitive to aggressive order error bounding. With primary copy, inconsistency accumulated at the beginning of a connectivity interval generally does not adversely impact overall service availability because the primary is still accessible for most partition scenarios (allowing writes to be committed even after the partition occurs).

With baseline order error bounding, voting delivers the lowest overall level of service availability. Interestingly, with aggressive order error-bounding, voting achieves the highest level of availability, closely tracking the upper bound. This difference results from the inherent properties of the voting algorithm. Consider the case where order error is bound lazily at one. Here, a replica can buffer one uncommitted local write and will cast a vote for the write before any other replica sees the write or can cast a vote for it. Then potentially, each replica in the system casts a vote for a different uncommitted write. In this case, a replica must collect votes from all remote replicas to determine the winner because each write holds exactly one vote, and any unknown vote could be the deciding one. In the presence of a partition, no replica will be able to commit any write. While retiring the minority partition would allow the system to commit the write, it would violate the requirement that all replicas are consistent all the time. Worse yet, once the system enters such a state, it must wait for the partition to be repaired before any additional writes can be committed. Aggressive order error bounding (or order error set to zero) greatly improves voting's availability because each replica needs to commit writes immediately. Write commitment is accomplished by pushing the write to all replicas to allow them to cast their vote. This operation reduces the window of vulnerability (described previously) where each replica may cast a vote for distinct writes. The achieved availability can be better than primary copy since the primary may be partitioned from the rest of the network, whereas with voting, the system can always adapt to choose the majority partition.

It is worth emphasizing that there has been much previous work on the availability of quorum systems, and it is well known [Barbara and Garcia-Molina 1987] that the majority quorum scheme [Thomas 1979] achieves better availability than the primary copy scheme under independent replica failures where the failure probability is smaller than 0.5. Our results may appear contradictory since we show that the voting protocol [Keleher 1999] performs worse than

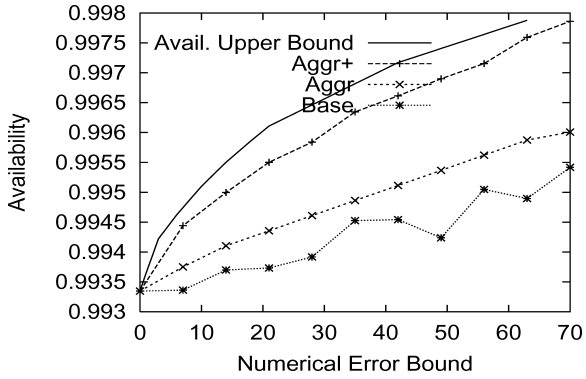

Fig. 9. Availability as function of numerical error (SAMPLED1).

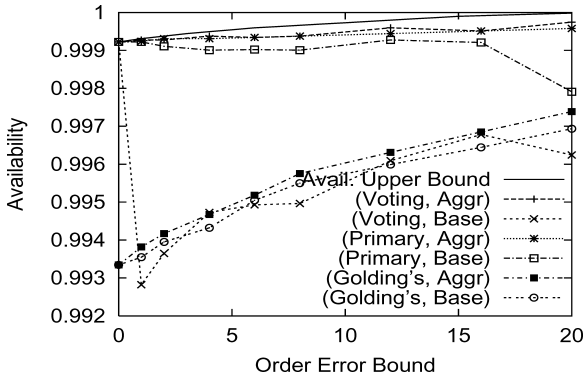

Fig. 10. Availability as function of order error (SAMPLED1).

the primary copy protocol [Petersen et al. 1997]. The truth is that, despite the similarities among the names, the two protocols (voting protocol [Keleher 1999] and primary copy protocol [Petersen et al. 1997]) for bounding order error are not quorum protocols. For example, in the voting protocol [Keleher 1999], each replica casts a vote for some write, and the system will pick a winning write based on the number of votes (with some tie-breaking scheme). Here, even the failure of a single replica may block the progress, if the vote of that replica could potentially affect the decision. In the majority quorum scheme [Thomas 1979], on the other hand, the system is guaranteed to make progress as long as a majority of the replicas are available.

Figures 9 and 10 plot service availability using a one-day sample of our measured faultload. The results are qualitatively similar to availability for the SIM1_00 faultload. However, with the lower rate of underlying failures (0.2% versus 1.0%), all of the protocols demonstrate much higher availability. For Figure 9, the curve labeled Aggr+ is a modification to our protocol that repeatedly attempts to set numerical error to zero even if the first attempt fails. With this change, achieved availability more closely tracks the upper bound. Figures 11 and 12 show similar results for a faultload with a much higher failure rate (over 4%). It is interesting to note the importance of using aggressive

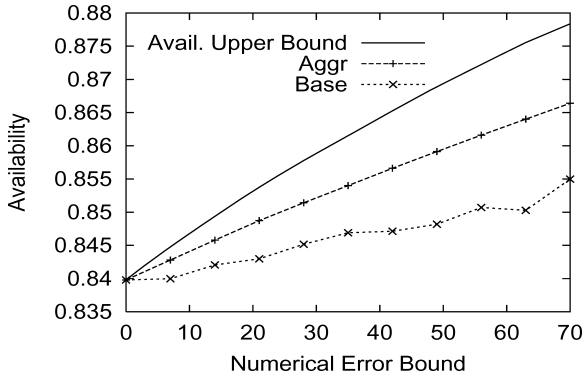

Fig. 11. Availability as function of numerical error (SIM5_00).

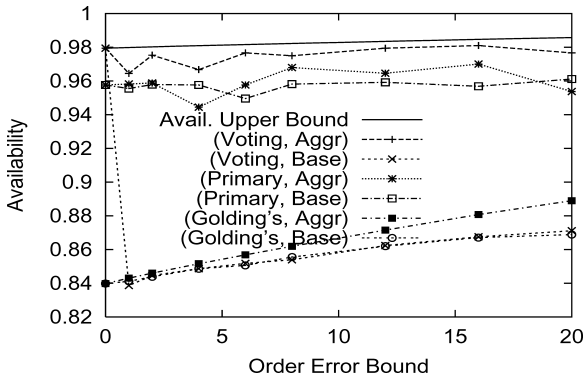

Fig. 12. Availability as function of order error (SIM5_00).

order error bounding; this optimization alone improves service availability from approximately 84% to more than 96% under voting. Also, for SIM5_00, the benefits of relaxing consistency are not as significant because, with a high failure rate, the replication system needs much lower consistency levels to achieve high availability. Similar to Amdahls's law governing performance, system availability will be limited by the weakest link in the chain. In this scenario, optimization efforts are likely better directed toward improving network reliability.

The results under all our remaining faultloads, except STRESS, are qualitatively similar to the previous results for SAMPLED1 and SIM5_00. For all faultloads studied, we observe that the theoretical availability upper bound can be closely approached by our simple protocols. This results from the following characteristics of both our measured and simulated faultloads.

(1) All partitions are singleton partitions. That is, a replica is partitioned away from the rest of the network. However, it is possible that multiple singleton partitions are present simultaneously.
(2) For most failures, the system transitions from fully-connected to a singleton partition scenario and then back to fully-connected.

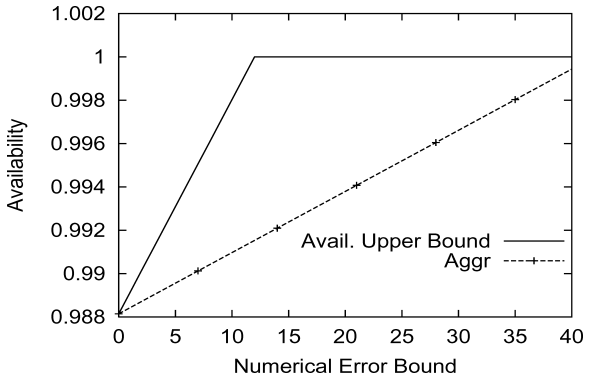

Fig. 13. Availability as function of numerical error (STRESS).

If we bound numerical error with aggressive write propagation, then at the point when the system transitions from fully-connected to a singleton partition scenario, the buffer on each replica for unpropagated writes should be close to empty. Also, our numerical error-bounding protocol distributes the allowed numerical error evenly among the replicas as buffer space. Thus the available buffer space to use for each replica during the partition scenario is roughly the same. Because the write injection rate on different replicas are also roughly the same, during the partition scenario, writes will only be rejected when almost all buffers have been fully utilized. As a result, we approach the optimal availability. For order error, notice that for the primary copy protocol, unless the primary is in a singleton partition, the serialization order we obtain will be optimal. On the other hand, the probability of the primary being partitioned is only $1/n$ (n being the number of replicas), assuming the primary is chosen randomly. Thus at least we are optimal $(n-1)/n$ of the time.

Of course, we do not claim that all faultloads have such properties. However, these observations are consistent with expected faultloads given the Internet's hierarchical topology and the power-law distribution of node degrees [Faloutsos et al. 1999]. For faultloads that do not have these properties, the availability upper bound may or may not be approached using the simple protocols presented. In the limit, approaching the upper bound in the general case requires completely accurate future knowledge which is impossible. In the following, we use a manually-constructed faultload to demonstrate this.

Figure 13 and 14 show the achieved availability versus the upper bound under a manually-constructed faultload with an average failure rate of 0.16%. This faultload is not intended to be representative, rather, we construct it to expose the low availability potentially achieved by real protocols. There is only one failure scenario in STRESS, during which 8 replicas are partitioned into 4 partitions, with 2 replicas each. The first 6 replicas do not receive any writes during the interval, while the last 2 receive 6 writes each. For the rest of the faultload, the 8 replicas are fully-connected and receive altogether 1000 writes. For primary copy, we use $replica_1$ as the primary. To achieve an availability as high as possible, we only consider aggressive versions of the protocols in our experiments. Also, to understand the achieved availability, we only need

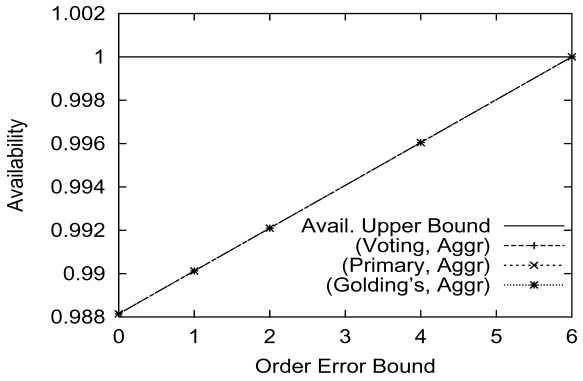

Fig. 14. Availability as function of order error (STRESS).

to concentrate on writes submitted during the failure scenario. As we can see from Figure 13 and 14, the achieved availability does not come close to the upper bound. For numerical error, the protocol splits allowed numerical error equally among replicas, but the first 6 replicas cannot utilize those allowed slots because they do not receive any writes. The last 2 replicas, on the other hand, quickly reach the allowed error and have to reject further writes. The dominating algorithm with optimal inputs, on the other hand, will allocate all allowed error to the last two replicas. For order error, no replica may commit any writes when the network is partitioned using Golding's algorithm. Voting cannot commit any writes either since the writes can only collect at most 2 votes. For primary copy, $replica_1$ and $replica_2$ may commit writes, but they do not receive any writes. The dominating algorithm with optimal inputs here will allow $replica_7$ and $replica_8$ to commit writes, thus always achieving 100% availability.

We want to emphasize that STRESS only demonstrates some of the factors that can make achieving the upper bound difficult. Even though some modifications to the previously presented protocols may help them to achieve better availability under STRESS, approaching the upper bound under an arbitrary faultload is very difficult. Counterexamples can be easily constructed by understanding how the upper bound is derived. In the limit, how close we can approach the upper bound will depend upon the extent that we can predict network failures and optimize for the predicted failure-pattern. Such failure-prediction techniques are beyond the scope of this work.

Figure 15 isolates the effects of network failure rate on the upper bound of service availability for 8 replicas subjected to a variety of simulated faultloads with different average failure rates. As expected, service availability degrades with increasing failure rates. Thus, a given level of relaxed consistency can only mask network failures to a certain point. For example, when numerical error is zero, service availability quickly degrades from near 100% for a reliable network (0.1% failure rate) down to 84% when network failures occur 4% of the time (recall that average failure rate does not necessarily correspond to periods of full connectivity). Availability degrades much more gracefully for bounded order error and unbounded numerical error because a total serialization order

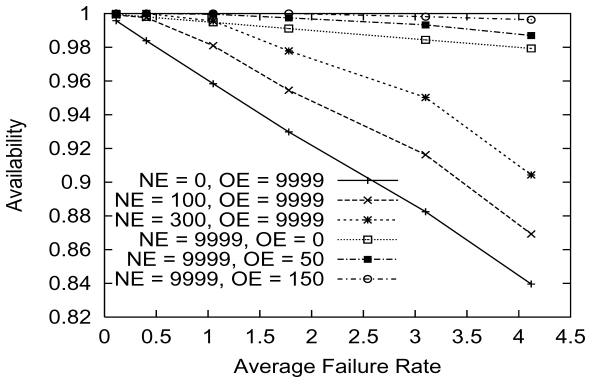

Fig. 15. Availability with different average failure rates.

can be determined even in the face of some network partitions. On the other hand, the nature of numerical error dictates that the system will eventually block (rejecting writes) for a partition of sufficient duration.

6.2 Availability/Communication Trade-offs

An interesting result of our work is that achieving maximum service availability with a relaxed consistency model can entail increased communication overhead. As discussed earlier, the communication costs of maintaining consistency can be reduced by waiting as long as possible to propagate writes. This approach allows the system to potentially combine multiple updates (depending on application semantics) and to amortize communication costs (packet header overheads, packet boundaries, and ramping up to the bottleneck bandwidth in TCP) over multiple logical writes. For example, studies of file system workloads show that file and individual write lifetimes are short [Baker et al. 1991; Mummert 1996], meaning that waiting to transmit an update may obviate the need to ever transmit the update. For other applications, such as load balancing or resource allocation, multiple numerical updates to a single data item can often be combined by waiting a threshold period of time. Thus, waiting as long as possible (while still maintaining consistency bounds) to propagate writes has the potential for reducing communication costs and improving overall service throughput (by requiring less update processing at each node). However, our results show that maximizing availability requires aggressive write propagation so that the system stays as close to strong consistency as possible during periods of good connectivity. The essential insight is that numerical error, order error, and staleness provide each replica with a window of writes that need not be propagated to remote replicas. Because the system cannot predict when a failure will take place or how long it might last, striving to keep the window empty during periods of good connectivity ensures that the system will maximize the duration of failures that it can mask from end users before being forced to reject accesses.

In this section, we quantify this communication versus availability tradeoff. Previously, we showed how aggressive consistency maintenance improves

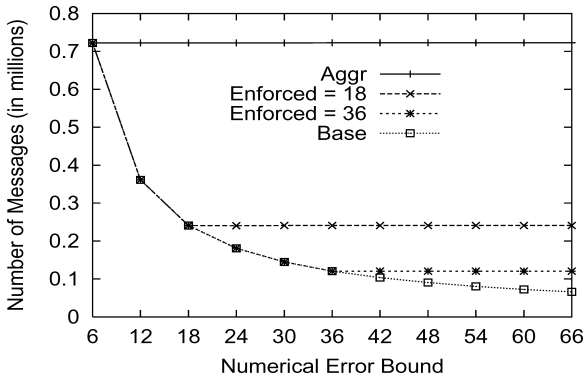

Fig. 16. Number of messages to maintain numerical error.

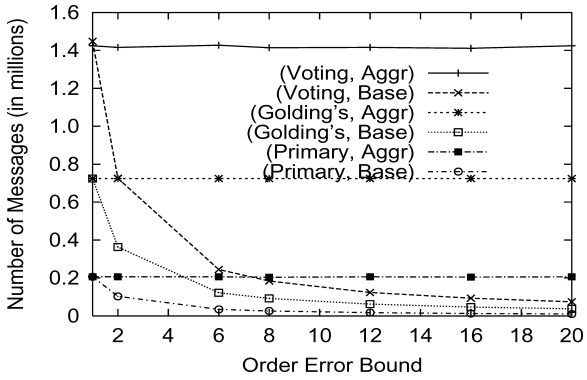

Fig. 17. Number of messages to maintain order error.

service availability. Figures 16 and 17 show the corresponding increase in the number of messages incurred by our consistency protocols using the faultload generated during WAN verification (described in Section 5.4). Results for other faultloads are similar. The update rate here is 1.4 updates per second system-wide which is evenly distributed across all 7 replicas. Each curve in Figure 16 corresponds to a different aggressively enforced numerical error. As expected, as we reduce the enforced numerical error, communication costs increase (with a corresponding increase in availability). Quantitatively, the communication cost roughly inceases inversely with the enforced numerical error. For example, decreasing the enforced numerical error by half (from 36 to 18), doubles the communication cost (from 0.12 million to 0.24 million) since we can now only combine 18 updates into a single message instead of 36 updates.

For order error (as depicted in Figure 17), our target protocols incur different communication costs. Voting requires the most messages because, to commit a write, a replica must potentially push its writes to all replicas followed by a pull of votes from all replicas. Golding's algorithm requires approximately half the messages required by voting because Golding's only needs to pull updates from remote replicas to reduce order error. Finally, with 7 replicas, primary copy requires about 1/7th of the messages required by voting. Note that our prototype

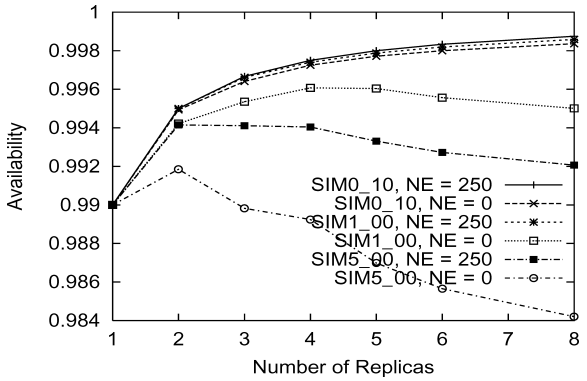

Fig. 18. $Avail_{client}$ as function of replication scale with $Avail_{network} = 1 - 1\%/n$.

is not optimized. Thus, our reported communication cost may be somewhat higher than required. However, we believe the relative communication costs for the various protocols and the revealed trade-offs are representative.

6.3 Effects of Replication Scale

To this point, our evaluation of service availability considered a fixed number (8) of replicas. We now attempt to isolate the effect of the number of replicas on client availability which is defined as the percentage of end user accesses that are accepted by the service ($Avail_{client}$, see Section 3). Our goal is to investigate the tension to replicate widely, to maximize the reach of a service (maximize $Avail_{network}$), and to centralize a service to minimize consistency overhead (maximize $Avail_{service}$). Carrying out this study requires knowledge of $Avail_{network}$ as the number of replicas changes. Since we do not have service reachability measurements to derive such a function (in general, measuring this requires access to a large client population), we arbitrarily pick the percentage of the client population that cannot reach any replica to be $1\%/n$ and $5\%/n$, where n is the number of replicas (e.g., 5 replicas corresponds to an $Avail_{network}$ value of 99.8% in the first case and 99% in the second). In both cases, each additional replica results in diminishing returns with respect to additional clients able to access the service. Finally, we consider a read to write ratio of 10 : 1. All results in this section are for calculated upper bounds. Note that we earlier demonstrated the ability of real protocols to approach this upper bound for the studied faultloads. We restrict our attention to relaxing numerical error, though the results for other metrics are similar.

Figure 18 shows the effect of varying the degree of replication on service availability, from 1 to 8 replicas. Because of the high cost of replication for read/write Internet services, we do not consider more than 8 replicas. Each curve represents a different faultload and different numerical error. For a $1\%/n$ growth rate and a high network failure rate (SIM5_00 curves), the optimal number of replicas tops out at 2. With a lower failure rate (SIM1_00), and zero numerical error, the best availability is delivered with 4 replicas. Once consistency is relaxed sufficiently (numerical error at 250 with SIM1_00) or a

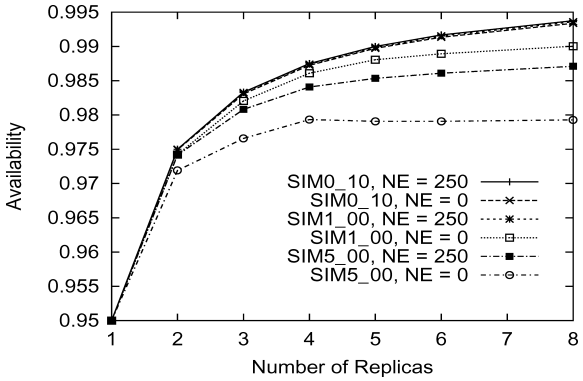

Fig. 19. $Avail_{client}$ as function of replication scale with $Avail_{network} = 1 - 5\%/n$.

low failure rate model is assumed (SIM0_10), availability continues to increase to 8 replicas (and beyond). Figure 19 shows similar results when the marginal benefit of each replica is larger. Here, even with a high failure rate (SIM5_00) and zero numerical error, availability increases to 4 replicas before starting to decrease. For this case, the marginal benefit of each additional replica on $Avail_{network}$ is large enough to overcome the cost of replication (reduction on $Avail_{service}$).

6.4 Discussion

In summary, our evaluation shows that simple optimizations to existing consistency protocols can greatly improve the availability of replicated services. For our target faultloads, we find that staying as close to strong consistency as possible during times of good connectivity allows services to approach the upper bound on availability. Of the order-error bounding algorithms considered, voting and primary copy generally achieve the best availability (using our optimizations) with voting achieving slightly better availability, while primary copy incurs significantly less communication overhead. Our results on availability as a function of the number of replicas quantifies the intuition that additional replicas will not always improve service availability and can in fact reduce it. Thus, system designers must carefully balance the marginal availability benefit of additional replicas against the costs of maintaining consistency for a given faultload and consistency level.

The results in this section also indicate a trade-off between read and write availability. Increasing the number of replicas will increase read availability since a larger number of clients will be able to access the service. However, write availability is potentially reduced because of the increased probability that the system will not be able to accept writes in the face of network partitions. One way to address this tension is to dynamically retire minority partitions that are unable to enforce consistency bounds for a threshold period of time. However, improving availability through retiring minority partitions requires an accurate prediction of future workloads and faultloads.

7. RELATED WORK

This work uses the continuous consistency model developed in our earlier efforts [Yu and Vahdat 2002] which describe the motivation for continuous consistency and quantifies the performance and semantic benefits for a range of Internet services. This article quantifies the inherent costs in terms of consistency and communication for increasing the availability of replicated services. Further, we derive tight upper bounds on the availability of services for a given set of environmental conditions and show that simple optimizations to existing consistency protocols allow services to approach this bound for our set of workloads and faultloads.

Fox and Brewer [1999] discuss the potential trade-offs between consistency and availability in the context of a cluster-based Internet service. Relative to their efforts, we focus on wide-area service availability and are able to quantify service availability as a function of consistency. A number of other efforts explore continuous consistency models [Adya et al. 2000; Krishnakumar and Bernstein 1994; Pu and Leff 1991]. We believe that our results are generally applicable to a broad range of consistency models [Yu and Vahdat 2002]. Availability of a replication system is also related to the availability of distributed consensus protocols. Relative to protocols [Lampson 1996] for qualitatively increasing the availability of distributed consensus, our work provides a framework for quantifying availability improvements under a variety of relaxed conditions (where relaxed consistency might correspond to bounded disagreement in consensus).

Dahlin et al. [2003] derive an analytical failure model for an average Internet path and then simulate the effects of caching and replication to mask Internet failures. For dynamic data, they assume optimistic consistency where service availability is solely limited by the time to create a replica or prefetch appropriate state. Many efforts [Amir and Wool 1996, 1998; Barbara and Garcia-Molina 1987, 1986; Coan et al. 1986; Diks et al. 1994; Garcia-Molina and Barbara 1984; Johnson and Raab 1991a; Kumar and Segev 1993; Peleg and Wool 1995; Spasojevic and Berman 1994; Tong and Kain 1988] explore service availability under strong consistency, typically in the context of quorum systems. Amir and Wool [1996] evaluate the availability of a quorum system running at two Internet sites. Coan et al. [1986] derive tight availability upper bounds in the case of two-way partitions. Another study [Johnson and Raab 1991a] shows that replication provides little availability benefit relative to an optimally-placed centralized service under strong consistency. This confirms our argument that, in many cases, higher availability can only be achieved by relaxing consistency. In general, the above efforts focus on availability at strong consistency through simulation and analysis, whereas we study availability along the entire consistency spectrum through both wide-area deployment and local area emulation.

8. CONCLUSIONS

The development of the Internet and the increasing popularity of mobile communication make networked access to remote resources the common case for computing. In these scenarios, service utility is often determined by its

availability rather than the traditional metric of raw performance. Replication is a key approach for improving the availability of network services. Given the well-known trade-offs between strong and optimistic consistency models, this article explores the benefits of a continuous consistency model for improving service availability. At a high level, this model allows applications to bound their maximum distance from strong consistency. The long-term goal of this work is to allow applications to dynamically set their consistency level, degree of replication, and placement of replicas based on changing network and service characteristics to achieve a target level of service availability.

In support of this goal, this article makes the following contributions. First, we quantify the availability of a prototype replicated service running across the Internet. Our prototype system measures service availability, while varying the consistency level, the protocol used to enforce consistency, and the failure characteristics of the underlying network. Our results show that simple optimizations to existing consistency protocols can significantly improve service availability (e.g., going from 99% to 99.9% in one scenario) and reveal that relaxed consistency cannot simultaneously maximize availability and minimize communication. We also develop a theory to derive tight upper bound on service availability as a function of workload, failure characteristics, and consistency level, enabling system designers to reason about the absolute merits of various consistency protocols and optimizations. Finally, we show that simple optimizations to existing consistency protocols enable services to approach our calculated availability upper bounds in our target scenarios.

APPENDIX

PROOF FOR THEOREM 4.3. We will reduce the *Independent Set* problem to the availability upper bound problem. Consider an arbitrary graph G with n nodes for which we would like to solve the *Independent Set* problem. Obviously, G has at most n^2 edges. We construct the corresponding replication system as follows. The system has n^2 *edge replicas* corresponding to the potential edges in G, and $n^2 \times n$ *node replicas* for the nodes in G. Each node i in G corresponds to the set S_i of n^2 node replicas, $1 \leq i \leq n$. Both edge replicas and node replicas are simply ordinary replicas from the replication system's perspective. We call them edge replica and node replicas only to map them to the edges and nodes in G. We need total n^3 node replicas because otherwise we will not observe sufficient numerical error after the transformation. The replicas are all disconnected unless we explicitly connect them.

We want to construct the workload and faultload so that we inject one write for each node in G, and any two writes corresponding to adjacent nodes in G cannot both be accepted. In other words, if they were both accepted, numerical error would be violated. To achieve this, we construct two intervals in the faultload for each node in G. In the first interval, a single write in injected. During the second interval, all edge replicas will see and apply the write except the edge replica corresponding to the edge incident on the node.

Formally, consider any odd integer i between 1 and $2n - 1$. During $interval_i$, two node replicas are connected if and only if they correspond to the same node

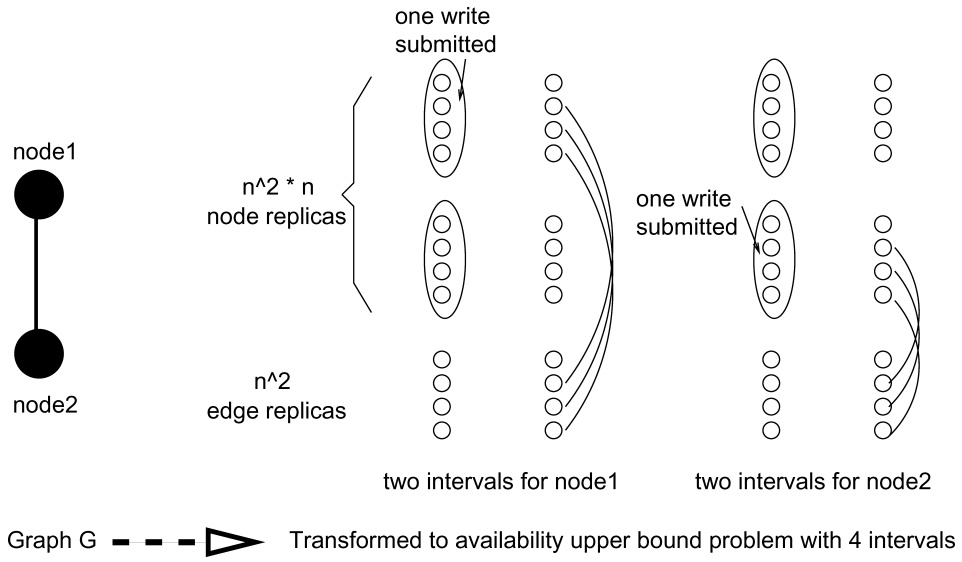

Fig. 20. Example of transforming a graph G (with $n = 2$) to the availability upper bound problem.

in G. In other words, the node replicas will form n network partitions with size of n^2 each. A write with unit numerical weight will be injected to the $(i+1)/2$th partition (which contains $S_{(i+1)/2}$). During $interval_{i+1}$, all n^3 node replicas are disconnected from each other. The kth edge replica will connect to the kth node replica in $S_{(i+1)/2}$ unless the kth edge in G is incident on the $(i+1)/2$th node, $1 \leq k \leq n^2$. Figure 20 gives an example of transforming a simply graph to the availability upper bound problem. Finally, the node replicas have numerical error bound of ∞, while the edge replicas have numerical error bound of 1. Thus we only need to care about the consistency of the edge replicas.

With the faultload and workload constructed above, it is not hard to see that unseen writes will only result when an edge is incident on a node. Each node in G corresponds to a write. Since an edge is incident on two nodes, it means that the two writes cannot both be accepted. It is then easy to see that finding a maximum independent set is equivalent to determining the maximum number of writes that can possibly be accepted without violating numerical error bound. Since *Independent Set* is NP-hard, the availability upper bound problem is also NP-hard. □

ACKNOWLEDGMENTS

We would like to thank Mike Dahlin, Jay Lepreau (whose Emulab infrastructure is supported by NSF award ANI-0082493), and Geoff Voelker for hosting some of our measurement software at various universities. We also would like to thank the anonymous reviewers for their detailed comments which significantly improved this paper.

REFERENCES

ADYA, A., LISKOV, B., AND O'NEIL, P. 2000. Generalized isolation level definitions. In *Proceedings of the IEEE International Conference on Data Engineering*.

AMIR, Y. AND WOOL, A. 1996. Evaluating quorum systems over the Internet. In *Proceedings of the Annual International Symposium on Fault-Tolerant Computing*.

AMIR, Y. AND WOOL, A. 1998. Optimal availability quorum systems: Theory and practice. *Inform. Proces. Letters*, 223–228.

ANDERSEN, D., BALAKRISHNAN, H., KAASHOEK, F., AND MORRIS, R. 2001. Resilient overlay networks. In *Proceedings of the 18th Symposium on Operating Systems Principles*.

ANDERSEN, D. G., BALAKRISHNAN, H., KAASHOEK, M. F., AND RAO, R. 2005. Improving Web availability for clients with MONET. In *Proceedings of the Symposium on Networked Systems Design and Implementation*.

BAKER, M., HARTMAN, J., KUPFER, M., SHIRRIFF, K., AND OUSTERHOUT, J. 1991. Measurements of a distributed file system. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles*. 198–212.

BARBARA, D. AND GARCIA-MOLINA, H. 1986. The vulnerability of vote assignments. *ACM Trans. Comput. Syst.*

BARBARA, D. AND GARCIA-MOLINA, H. 1987. The reliability of voting mechanisms. *IEEE Trans. Comput. 36*, 10 (Oct.), 1197–1208.

BERNSTEIN, P. A., HADZILACOS, V., AND GOODMAN, N. 1987. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley.

BROWN, A. AND PATTERSON, D. 2000. Towards maintainability, availability, and growth benchmarks: A case study of software RAID systems. In *Proceedings of the 2000 USENIX Annual Technical Conference*.

CANDEA, G., KAWAMOTO, S., FUJIKI, Y., FRIEDMAN, G., AND FOX, A. 2004. Microreboot—A technique for cheap recovery. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*.

CETINTEMEL, U., KELEHER, P., AND FRANKLIN, M. 2001. Support for speculative update propagation and mobility in Deno. In *Proceedings of the 21st IEEE International Conference on Distributed Computing Systems*.

COAN, B., OKI, B., AND KOLODNER, E. 1986. Limitations on database availability when networks partition. In *Proceedings of the 5th ACM Symposium on Principle of Distributed Computing*. 187–194.

CORMEN, T., LEISERSON, C., AND RIVEST, R. 1990. *Introduction to Algorithms*. The MIT Press.

CZYZYK, J., MEHROTRA, S., WAGNER, M., AND WRIGHT, S. PCx: Software for linear programming. Available at: http://www-fp.mcs.anl.gov/otc/Tools/PCx/.

DAHLIN, M., CHANDRA, B., GAO, L., AND NAYATE, A. 2003. End-to-end WAN service availability. *ACM/IEEE Trans. Network. 11*, 2 (April).

DIKS, K., KRANAKIS, E., KRIZANC, D., MANS, B., AND PELC, A. 1994. Optimal coteries and voting schemes. *Inform. Proc. Letters*, 1–6.

DOUCEUR, J. R. AND WATTENHOFER, R. P. 2001. Competitive hill-climbing strategies for replica placement in a distributed file system. In *Proceedings of the 15th International Symposium on Distributed Computing (DISC)*. 48–62.

FALOUTSOS, M., FALOUTSOS, P., AND FALOUTSOS, C. 1999. On power-law relationships of the Internet topology. In *SIGCOMM*.

FOX, A. AND BREWER, E. 1999. Harvest, yield, and scalable tolerant systems. In *Proceedings of HotOS-VII*.

FOX, A., GRIBBLE, S., CHAWATHE, Y., AND BREWER, E. 1997. Cluster-based scalable network services. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*. Saint-Malo, France.

GARCIA-MOLINA, H. AND BARBARA, D. 1984. Optimizing the reliability provided by voting mechanisms. In *Proceedings of the 4th International Conference on Distributed Computing Systems*.

GOLDING, R. 1992. A weak-consistency architecture for distributed information services. *Comput. Syst. 5*, 4 (Fall), 379–405.

GRAY, J., HELLAND, P., O'NEIL, P., AND SHASHA, D. 1996. The dangers of replication and a solution. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*.

GUMMADI, K. P., MADHYASTHA, H. V., GRIBBLE, S. D., LEVY, H. M., AND WETHERALL, D. 2004. Improving the reliability of Internet paths with one-hop source routing. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*.

HENNESSY, J. 1999. The future of systems research. *IEEE Comput. 32*, 8 (Aug.), 27–33.

JOHNSON, D. B. AND RAAB, L. 1991a. A tight upper bound on the benefits of replication and consistency control protocols. In *Proceedings of the 10th ACM Symposium on Principles of Database Systems*.

JOHNSON, D. B. AND RAAB, L. 1991b. Effects of replication on data availability. *Int. J. Comput. Simul. 1*, 4.

KELEHER, P. 1999. Decentralized replicated-object protocols. In *Proceedings of the 18th Annual ACM Symposium on Principles of Distributed Computing*.

KISTLER, J. J. AND SATYANARAYANAN, M. 1992. Disconnected operation in the coda file system. *ACM Trans. Comput. Syst. 10*, 1 (Feb.), 3–25.

KRISHNAKUMAR, N. AND BERNSTEIN, A. 1994. Bounded ignorance: A technique for increasing concurrency in a replicated system. *ACM Trans. Datab. Syst. 19*, 4 (Dec).

KUMAR, A. AND SEGEV, A. 1993. Cost and availability trade-offs in replicated data concurrency control. *ACM Trans. Datab. Syst*.

LADIN, R., LISKOV, B., SHIRIRA, L., AND GHEMAWAT, S. 1992. Providing availability using lazy replication. *ACM Trans. Comput. Syst. 10*, 4, 360–391.

LAMPORT, L. 1978. Time, clocks, and the ordering of events in a distributed system. *Comm. ACM 21*, 7 (July) 558–565.

LAMPSON, B. 1996. How to build a highly available system using consensus. In Distributed Algorithms, Lecture Notes in Computer Science Vol. 1151. Springer.

MUMMERT, L. 1996. Exploiting weak connectivity in a distributed file system. Ph.D. thesis, Carnegie Mellon University.

NOBLE, B., FLEIS, B., AND KIM, M. 1999. A Case for Fluid Replication. In *Proceedings of the 1999 Network Storage Symposium (Netstore)*.

NOBLE, B., SATYANANARAYANAN, M., NGUYEN, G., AND KATZ, R. 1997. Trace-based mobile network emulation. In *Proceedings of SIGCOMM*.

PAGE, T., GUY, R., HEIDEMANN, J., RATNER, D., GOEL, A., KUENNING, G., AND POPEK, G. 1998. Perspectives on optimistically replicated peer-to-peer filing. *Softw. Practice Exper. 28*, 2 (Feb), 155–180.

PAI, V. S., ARON, M., BANGA, G., SVENDSEN, M., DRUSCHEL, P., ZWAENEPOEL, W., AND NAHUM, E. 1998. Locality-aware request distribution in cluster-based network servers. In *8th International Conference on Architectural Support for Programming Languages and Operating Systems*.

PAXSON, V. 1996. end-to-end routing behavior in the Internet. In *Proceedings of the ACM SIGCOMM'96 Conference on Communications Architectures and Protocols*.

PELEG, D. AND WOOL, A. 1995. The availability of quorum systems. *Inform. Computat. 123*, 2 (Dec), 210–223.

PETERSEN, K., SPREITZER, M., TERRY, D., THEIMER, M., AND DEMERS, A. 1997. Flexible update propagation for weakly consistent replication. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP-16)*.

PU, C. AND LEFF, A. 1991. Replication control in distributed system: An asynchronous approach. In *Proceedings of the ACM SIGMOD Conference on Management of Data*.

ROSENTHAL, A. 1977. Computing the reliability of a complex network. *SIAM J. App. Math. 32*, 384–393.

SAITO, Y., BERSHAD, B., AND LEVY, H. 1999. Manageability, availability and performance in porcupine: A highly scalable Internet mail service. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles*.

SAVAGE, S., COLLINS, A., HOFFMAN, E., SNELL, J., AND ANDERSON, T. 1999. The end-to-end effects of Internet path selection. In *SIGCOMM*.

SINGLA, A., RAMACHANDRAN, U., AND HODGINS, J. 1997. Temporal notions of synchronization and consistency in Beehive. In *Proceedings of the 9th ACM Symposium on Parallel Algorithms and Architectures*.

SPASOJEVIC, M. AND BERMAN, P. 1994. Voting as the optimal static pessimistic scheme for managing replicated data. *IEEE Trans. Parall. Distrib. Syst.* 64–73.

SWIFT, M. M., ANNAMALAI, M., BERSHAD, B. N., AND LEVY, H. M. 2004a. Recovering device drivers. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*.

SWIFT, M. M., BERSHAD, B. N., AND LEVY, H. M. 2004b. Improving the reliability of commodity operating systems. *ACM Trans. Comput. Syst. 22*, 4 (Nov.).

TERRY, D. B., THEIMER, M. M., PETERSEN, K., DEMERS, A. J., SPREITZER, M. J., AND HAUSER, C. H. 1995. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*.

THOMAS, R. H. 1979. A majority consensus approach to concurrency control for multiple copy databases. *ACM Trans. Datab. Syst. 4*, 2 (June), 180–209.

TONG, Z. AND KAIN, R. Y. 1988. Vote assignments in weighted voting mechanisms. In *Proceedings of the 7th IEEE Symposium on Reliable Distributed Systems*. 138–143.

TORRES-ROJAS, F., AHAMAD, M., AND RAYNAL, M. 1999. Timed consistency for shared distributed objects. In *Proceedings of the 18th ACM Symposium on Principle of Distributed Computing*.

YU, H. AND VAHDAT, A. 2000. Efficient numerical error bounding for replicated network services. In *Proceedings of the 26th International Conference on Very Large Databases (VLDB)*.

YU, H. AND VAHDAT, A. 2001. Combining generality and practicality in a conit-based continuous consistency model for wide-area replication. In *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS)*.

YU, H. AND VAHDAT, A. 2002. Design and evaluation of a conit-based continuous consistency model. *ACM Trans. Comput. Syst*.

ZEGURA, E. W., CALVERT, K., AND DONAHOO, M. J. 1997. A quantitative comparison of graph-based models for Internet topology. *IEEE/ACM Trans. Network. 5*, 6 (Dec.).

Received February 2004; revised April 2005; accepted June 2005