The Costs and Limits of Availability for Replicated Services

Haifeng Yu
Department of Computer Science
Duke University
yhf@cs.duke.edu

Amin Vahdat[*]
Department of Computer Science
Duke University
vahdat@cs.duke.edu

ABSTRACT

As raw system and network 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 tradeoffs between service consistency and availability. Thus, this paper 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 paper, we: i) evaluate 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 bounds 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 tradeoff.

1. INTRODUCTION

As raw system and network performance continues to improve at exponential rates, the utility of many services is limited by availability rather than performance [18]. Thus, many researchers are turning their attention to architectures that improve overall availability, for example in the context of cluster web servers [14], cluster email servers [33] and software RAID systems [6]. 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 [8].

[*]This research is supported in part by the National Science Foundation (EIA-99772879, ITR-0082912), IBM, Intel, and Microsoft. Vahdat is also supported by an NSF CAREER award (CCR-9984328).

Caching and replication are key approaches to improving the availability of Internet services in the face of such network failures. 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 orderings. 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 [31, 35], Ficus [29], Coda [21], Deno [7], TACT [36], Lazy Replication [24] and Fluid Replication [27]. While optimism can dramatically increase availability, an unbounded rate of conflicting updates can quickly leave the system in a "delusional" state [17]. 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 tradeoff 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 [2, 4, 9, 11, 19, 23], 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 tradeoff 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 paper 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 tradeoff between availability and communication in the face of 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. 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 paper is organized as follows. Section 2 provides background and describes our continuous consistency model. In Section 3, we describe our techniques for systematically determining the availability upper bounds for replicated services. Section 4 details our implementation and measurement methodology. Section 5 presents availability and upper bound results for replicated services under a variety of conditions through actual wide-area deployment and an emulation environment. Section 6 describes related work and Section 7 presents our conclusions.

2. BACKGROUND AND SYSTEM MODEL

2.1 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 [24]. Strong consistency ensures that concurrent updates will not conflict but 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 tradeoff between consistency, performance, and availability. A number of efforts [1, 22, 32, 36] 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 paper. For our purposes, a continuous model allows a more complete exploration of the problem space.

We focus our study on the TACT consistency model [36]. However, we believe our findings reveal inherent aspects of the consistency versus availability tradeoff 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 writes before requiring remote communication. TACT models updates as procedures with application-specific merge routines [35]. Each update can optionally carry an 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. 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). 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

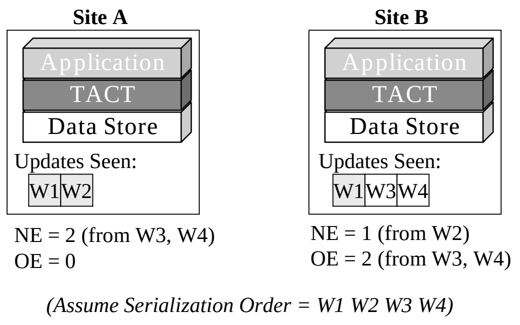

Figure 1: Example system state with corresponding numerical and order error.

optimistic consistency.

To provide some intuition behind the metrics, consider the application of TACT to a replicated airline reservation system [36]. In this example, 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 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.

2.2 Assumptions and Methodology

One 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. In a wide-area network setting however, a service may be "available" in the sense that all components are executing normally and the network connection is functional. However, 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, at least one replica may be accessible to all clients but some replicas may not be able to proceed with individual requests because of consistency requirements.

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 failures are symmetric. Even though non-symmetric behavior is common at the IP level [30], we believe our assumption is a close approximation for reliable communication protocols, such as 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. Note that we do not assume reachability among hosts is transitive. We assume that the CPU processing time and network delay is negligible compared to the duration of time where network connectivity does not change. Violating this assumption would only result in loosening the availability upper bounds derived in Section 3. Finally, TACT supports the notion of application-specific consistency units (or conits) that determine the granularity over which consistency is enforced [36], for example a single flight, first class seats, coach seats, etc. For simplicity, we assume a single conit in the system and that each write carries unit numerical weight and order weight for that conit. Extending our results to more general scenarios is straightforward.

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 under a fixed consistency level. Relaxing consistency helps to improve service availability. In this paper, we quantify availability using both service availability and client availability.

We investigate service availability using a workload and faultload approach. A workload is a trace of timestamped accesses, while a faultload is a trace of timestamped *fault events*. A fault event is either a failure that divides an existing network component into two components or a recovery that merges two existing components. The workload only contains the accesses that actually reach a replica, and thus implicitly encompasses $Avail_{network}$. The number and location of replicas are specified as part of the faultload. Fault events divide a run into *intervals*. By definition, network connectivity does not change during an interval. This workload and faultload approach has a number of advantages over the traditional approach of analytically modeling replicated systems [9, 11] because it can avoid assumptions about access patterns, failure patterns, failure correlations and network topology. However, we do 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.

3. DERIVING TIGHT AVAILABILITY UPPER BOUNDS

The principal result of this section is the ability to calculate tight upper bounds 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. The proofs for our derivation are available in a separate technical report [39]; here, we concentrate on the intuition behind the theory rather than the formal framework.

3.1 Overview

At a high level, any consistency maintenance protocol must answer a number of *questions* to achieve a target level of consistency among replicas. First, a protocol must determine which writes to accept (or reject) from clients. Greedily accepting all writes that do not violate consistency requirements is not always optimal as it could preclude the acceptance of a larger number of writes in the future. Next, a protocol must determine when and where to propagate writes. Write propagation decreases numerical error but can increase order error temporarily if the propagated write cannot be locally committed. Finally, the protocol must decide the serialization order—which writes to commit and in what order. Different serialization orders enable writes to be committed at different rates, affecting the rate at which order error can be reduced. We discuss these issues in more detail below. For now, we divide these questions into two disjoint sets: $Q_{offline}$ is the set of questions with optimal answers that can be pre-determined offline, while Q_{online} contains all remaining questions whose optimal answers depend on consistency level, workload, and faultload. To prove tight upper bounds on the availability of a service, it is necessary to search for the optimal answers to these questions. Of course, the set of possible answers is exponential. Thus, a key challenge is to make the search of the set of possible answers tractable by proving that certain types of answers will always result in better availability than others.

Using the pre-determined optimal answers to $Q_{offline}$, we construct a *dominating algorithm*, which is also a consistency protocol. The dominating algorithm is an abstract algorithm not intended for implementation (though we apply some of these concepts to optimize existing algorithms in Section 5). By definition, the dominating algorithm makes strictly "better" decisions than other protocols for $Q_{offline}$. The dominating algorithm does not specify the answer of any question from Q_{online}, rather it takes some *inputs* that specify these answers. For example, one input may specify whether to accept or reject a particular write W. Obviously, such inputs will affect the behavior of the dominating algorithm. For a given workload and faultload, we say that a consistency protocol P_1 *dominates* protocol P_2, if i) P_1 achieves the same or higher level of availability as P_2 and ii) P_1 maintains the same or higher level of consistency as P_2. The upper bound is the availability achieved by the protocol $\mathcal{P}$ that dominates all protocols. If we take all potential inputs of the dominating algorithm into account, we can see that some inputs to the dominating algorithm exist such that the dominating algorithm will dominate an arbitrary consistency protocol. The upper bound is then obtained by searching the set of possible inputs to the dominating algorithm (i.e., the answers to Q_{online}).

The construct of the dominating algorithm does not necessarily guarantee a tractable approach to calculating the upper bound on availability. For example, in the extreme, a trivial dominating algorithm may specify nothing ($Q_{offline}$ being empty) and take all answers from its input. Thus, the key to making the computation of the upper bound tractable is to i) maximize the set $Q_{offline}$ and ii) optimize the search for optimal answers to Q_{online}.

To provide a flavor of our approach, we first sketch the derivation for calculating upper bounds on system availability as a function of numerical error and staleness (with order error unbounded), assuming that reads are always accepted by a replica. We then present the results for calculating upper bounds on availability when all three metrics are bounded. The potential for improving availability by selectively rejecting reads is discussed elsewhere [39].

3.2 Availability Upper Bound as a Function of Numerical Error and Staleness

Before we begin the derivation, it is important to note that considering only numerical error and staleness greatly simplifies the problem and the dominating algorithm concept is not strictly required. However, to prepare for the general case, we still describe the approach using the general framework.

A consistency protocol for numerical error and staleness needs to answer questions regarding write propagation, such as "when/where to propagate writes", and questions regarding write acceptance, such as "whether to accept a particular write". There are trivial optimal answers (i.e., propagating writes whenever possible) to questions regarding write propagation, since pushing writes to remote replicas always helps to reduce numerical error and staleness. Thus, these questions form $Q_{offline}$ in this case. Based on these answers, the dominating algorithm uses *aggressive write propagation*: i) whenever a replica accepts a write, it pushes the write to all reachable replicas immediately, ii) whenever a recovery event merges two network components, all replicas immediately propagate unseen writes to all reachable replicas. Here we would like to emphasize that aggressive write propagation is not practical nor necessarily intended for implementation, it is just a concept used to derive the best availability a consistency protocol can potentially achieve. Section 5 shows the delivered availability for real protocols *not* using aggressive write propagation relative to the calculated upper bound.

Questions regarding write acceptance do not have offline optimal answers and thus belong to Q_{online}. Numerical error and staleness constrain the set of writes that can be accepted, which in turn, limits availability. To find the availability upper bound, we must perform an exhaustive search of all possible sets of accepted writes with the goal of maximizing availability while ensuring that numerical error and staleness bounds are not violated. However, a naive enumeration will incur exponential complexity in terms of the number of received writes. Properties of aggressive write propagation allow us to optimize our search by using a single logical write to represent all writes accepted by a partition during an interval. We can then "simulate" the execution of the dominating algorithm for a given faultload as follows. We maintain n write logs, one for each replica. For each $partition_m$ in $interval_k$, the simulator creates a variable $writes_{k,m}$ representing the total number of writes accepted by $partition_m$ during $interval_k$. The "simulator" then attaches this variable to the logs corresponding to the replicas in $partition_m$, which means that all replicas in $partition_m$ see all such writes at the end of the interval. Next, at the beginning of an interval, the simulator updates all logs according to aggressive write propagation rules. To ensure that numerical error is properly maintained on $replica_i$, the total number of writes accepted by the system but not seen by $replica_i$ must be smaller than the numerical error bound on $replica_i$:

$$\sum\{writes_{k,m} | k \leq \text{current interval index and } writes_{k,m} \notin replica_i\text{'s write log}\} \leq bound_{NE}$$

Staleness can be similarly incorporated into the theory by setting some of the above variables to zero. For example, if a variable $writes_{k,m}$ is not in $replica_i$'s write log and it is accepted before the acceptance time allowed by $replica_i$'s staleness bound, it must be zero. To preserve the tightness of the bounds for staleness, an interval is split in two when the allowed acceptance time falls in the middle of the interval. Using this technique, the allowed acceptance time will always fall on interval boundaries [39].

The workload requires that each variable $writes_{k,m}$ be smaller than the number of writes submitted to $partition_m$ during $interval_k$:

$$writes_{k,m} \leq \text{number of writes submitted to } partition_m \text{ during } interval_k$$

The above inequalities for all (replica, interval) pairs compose a system of linear constraints for all $writes_{k,m}$. Hence, maximizing availability is equivalent to maximizing the target $\sum writes_{k,m}$ under the above constraints (assuming that reads are always accepted), a standard linear programming problem.

We also prove that the calculated upper bound is tight [39]. However, the tightness of the bounds does not necessarily mean there exists a protocol that can always achieve the upper bounds for an arbitrary workload, faultload and consistency level. In fact, reaching the upper bound generally requires future knowledge. To understand this, note that a replica must decide whether a write can be accepted upon receiving the write. However, an adversary can always construct a workload and a faultload after this point such that the decision was sub-optimal [39].

3.3 Availability Upper Bound as a Function of Order Error

An order error bounding protocol needs to answer three kinds of questions: questions regarding write propagation, questions regarding write acceptance and questions regarding write commitment. To commit a write, a replica must see all preceding writes in the *serialization order*. The *serialization order* is the global total write order that an order error bounding protocol tries to maintain across all replicas. For example, consider a system with two replicas that are partitioned from each other. Suppose $replica_1$ receives W_1 then W_2, while $replica_2$ receives W_3 then W_4. A serialization order here can be any permutation of the four writes. If the serialization order is $S = W_1W_2W_3W_4$, then a replica can only commit W_3 after it sees and commits W_1 and W_2.

Aggressive write propagation always maximizes the writes seen by a replica at any given time and expedites write commitment. However, it may also increase the number of uncommitted writes on a replica. In our previous example, if $replica_2$ propagates W_3 to $replica_1$ before $replica_1$ accepts W_2, then $replica_1$ cannot commit W_3 and its order error is increased. Thus, while aggressive write propagation always serves to reduce numerical error and staleness, in certain cases it can actually increase order error. We use the following approach to address this issue with aggressive write propagation. When remote writes are first seen by a replica, they are not applied to the data store immediately, and thus do not count toward order error. We apply them to the data store only when they can be committed, so that remote writes never increase order error (although they do increase numerical error by definition). Local writes are always applied to the data store immediately as required by the semantics of the consistency model. This design and aggressive write propagation form the dominating algorithm for order error.

Questions regarding write acceptance and write commitment belong to Q_{online}. The key to all write commitment questions is the serialization order. As long as a serialization order is determined, the dominating algorithm will behave deterministically on all write commitment activities. However, the number of possible serialization orders is factorial with the number of writes, making an exhaustive search impossible. Further, with arbitrary serialization orders, we can no longer answer write acceptance questions through linear programming because two writes accepted by a partition during an interval may not be committed at the same time. This prevents us from using a single logical write to represent all writes accepted by a partition during an interval.

To optimize our search on serialization orders, we would like to find a small set of serialization orders that are strictly "better" than all other serialization orders. Here, we define a *domination* relationship among serialization orders (similar to the domination relationship defined earlier for consistency protocols) to facilitate the following derivation. In order to be useful, the definition of domination between serialization orders must imply domination between protocols, as follows: Serialization order S dominates serialization order S' if, for a given workload and faultload, the dominating algorithm using S can commit a write W whenever the dominating algorithm using S' can commit that write. In other words, S dominates S' if S allows the commitment of any write that could be committed using S'. In our previous example with two replicas and four writes, serialization order $S = W_1W_2W_3W_4$ dominates $S' = W_2W_1W_3W_4$. This is because whenever W_2 can be committed using S', the replica must have already seen W_1 (which is accepted before W_2), and thus can also commit W_2 in S. The same is true for W_1, W_3 and W_4. Notice that S' does not dominate S because immediately after accepting W_1, $replica_1$ can commit W_1 using S, but it cannot commit W_1 in S'. Using this definition of "domination", we prove that if S dominates S', then the dominating algorithm using S dominates the dominating algorithm using S'.

Next, we construct three sets of serialization orders as follows:

ALL All possible serialization orders.

$CAUSAL$ Serialization orders compatible with causal order.

$CLUSTER$ Serialization orders where writes accepted by the same partition during a particular interval cluster together.

In our previous example with two replicas, ALL contains all possible permutations of the four writes, that is, 24 serialization orders. $CAUSAL$ contains the six orderings where W_1 precedes W_2 and W_3 precedes W_4. $CLUSTER$ only contains $W_1W_2W_3W_4$ and $W_3W_4W_1W_2$, because W_1 and W_2 are accepted by the same partition during the interval and thus cluster together (the same is true for W_3 and W_4).

For our faultloads, ALL may contain up to 10^{1000} serialization orders, while the size of $CLUSTER$ is smaller than 1000. If we can prove that $CAUSAL$ dominates ALL and $CLUSTER$ dominates $CAUSAL$, the upper bound computation becomes tractable by restricting our search scope to $CLUSTER$. To prove $CAUSAL$ dominates ALL, we only need to show that if write W_1 causally precedes write W_2, then it is always "better" to place W_1 before W_2 in the serialization order. This is true because by the time any replica sees W_2, it must have already seen W_1. Thus, any serialization order can be pair-wise adjusted (according to causal order) to be a new serialization order that dominates the original one. The proof of $CLUSTER$ dominating $CAUSAL$ is the key to the whole theory. The intuition is that it does not expedite write commitment on any replica if the writes accepted by the same partition during a particular interval are allowed to split into multiple sections in the serialization order. The actual proof is intricate and is available elsewhere for brevity [39].

Using $CLUSTER$ not only enables exhaustive enumeration, it also helps us to search for optimal answers for write acceptance questions. Recall that without using $CLUSTER$, linear programming cannot model the write acceptance problem because we cannot use one logical write to represent all writes accepted by a partition during an interval. Now since we are only concerned with the serialization orders in $CLUSTER$, we can again use this approach. Thus, for each serialization order enumerated, we use the previous "simulation" approach (attaching variables to interval/partition combinations) to determine the constraints and to derive the upper bound by solving a linear programming problem. We also prove that the bound is tight [39].

3.4 General Case Availability Upper Bound

The upper bound derivation for the general case is based on the derivation for order error. Numerical error, staleness and order error interact in subtle ways. Applying remote writes to the local data store will decrease a replica's numerical error and staleness. But if these writes cannot be committed, order error is increased. For the order error dominating algorithm, remote writes are only applied to the data store when they can be committed. However, this approach may not be optimal when considering the interactions between order error, numerical error, and staleness as described earlier. Thus, besides the three types of questions an order error bounding protocol needs to answer, the general case dominating algorithm must answer a forth type of question regarding when and how to apply remote writes to the data store. These questions also belong to Q_{online}.

Now we need to search for optimal answers on three kinds of questions: questions on write acceptance, questions on serialization order and questions on applying remote writes. In order to apply the results developed before on dominating serialization orders, we prove that in this general case if serialization order S dominates another serialization order S', then the dominating algorithm using S dominates the one using S'. Thus, all results developed for order error apply here. We then develop properties of the dominating algorithm to allow the incorporation of applying remote writes into the linear programming problem. Finally, the same approach is used to obtain the upper bound as before through "simulation" and linear programming.

3.5 Complexity and Experience

Our algorithm to compute the availability upper bound in the general case is exponential in complexity in terms of the number of intervals. Such exponential 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, completing execution within three hours on a Sun Ultra-5 Workstation.

The algorithm to compute availability upper bounds 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 "simulates" aggressive write propagation and records log contents at the end of each interval. Next, it enumerates all serialization orders as described above, and uses a linear programming solver (PCx [10]) to calculate upper bounds on availability based on the specified constraints.

4. 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.

Faultload	Description	Avg. Fail. Rate
SAMPLED1	First day of the RMI-ping trace	0.17%
SIM0_10	Simulated trace	0.11%
SIM1_00	Simulated trace	1.05%
SIM5_00	Simulated trace	4.12%

Table 1: This table summarizes the characteristics of faultloads.

4.1 Sample Faultloads

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 [30, 34] uses fairly coarse-grained inter-arrival times for successive measurements between two sites, with a minimum inter-arrival of approximately 10 minutes. Such a granularity cannot unambiguously capture the duration of short network failures, which make up a majority of failure events [30]. Thus, we set out to collect a sample of Internet connectivity with average measurement intervals of 3 seconds on each path.

Because measuring Internet failure characteristics is not a focus of this work, we only present a summary of our measurement methodology here. Full details are available separately [39]. We measure interconnectivity among 8 sites: Verio (CA), Rackspace (TX), Rackspace (U.K.), University of California, Berkeley, University of California, San Diego, University of Utah, University of Texas, Austin, and Duke University. All sites have good network connectivity and no competing CPU activity. The total duration of the trace is 6 days in February 2001, with over 12 million samples. The faultload has an average failure time on all paths of 0.046%. We assume that failures last for the entire duration of failed samples and that no failures take place between two successive successful samples. The fine-granularity of our trace ensures that the inaccuracy introduced by these assumptions is relatively small. With these assumptions, we generate a connectivity matrix for the system as a function of time. We then calculate the transitive closure of the matrix, to allow for the ability to mask failures by "routing" through other replicas [34]. Using the transitive closure serves to isolate the availability effects of continuous consistency from the effects of application-level routing.

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. 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.

We do not claim that our measured faultload is representative. In fact, our measurement sites tend to be well-connected with relatively little network congestion. Because we are interested in understanding how the availability of Internet services is affected by a broad range of failure characteristics, we construct a number of synthetic faultloads with varying characteristics. We use a simple event-driven simulator to obtain diverse faultloads based on a sample Internet-like topology generated by the Internet topology generator [40]. 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. We use exponential distributions for both failure duration and failure inter-arrival time. By varying the parameters of the distributions, we obtain a series of simulated faultloads. For example, "SIM1_00", with an average failure rate of 1%, is obtained with a failure inter-arrival mean of 1 day and failure duration mean of of 120 seconds for nodes, and failure inter-arrival mean of 14 days and failure duration mean of 150 seconds for links. Table 1 summarizes the characteristics of the one-day faultloads we use in this study; results for additional faultloads are presented elsewhere [39].

4.2 WAN Prototype Details

We now describe our modifications to the TACT prototype [36] to study the replicated service availability as a function of consistency and faultload. The prototype is written in Java based on RMI, and write propagation is performed through anti-entropy [31]. It supports three simple replicated services: an airline reservation system, a bulletin board system, and a load-balancing request distributor [36]. For this paper, we run our availability experiments using the bulletin board service. In addition to faultload and consistency level, the availability of a replicated service will also depend upon the specifics of the consistency maintenance protocol—whether it be strong consistency, optimistic consistency or somewhere in between. We thus extend the prototype to use a variety of consistency protocols, as described below.

Our prototype uses the only protocol we are aware of to bound numerical error [37]. Each replica ensures that the error bound on other replicas is 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 bound on other replicas is not violated.

A number of different write commitment algorithms can be used for bounding order error. We implement three popular such protocols in our prototype, primary copy [31], Golding's algorithm [16], and voting [15, 20]. Essentially, all three write commitment algorithms determine a total order on all writes. For completeness, we present a brief summary of each protocol below.

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. Each replica maintains a version vector [16] to determine whether or not it has seen all writes with logical time 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 version 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 [20], 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.

4.3 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 re-executing 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 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 [28].

To validate emulation results, we deploy our prototype system running the replicated bulletin board service on all the sites used for measuring faultload, 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, and thus it increases 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 2 summarizes 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) [5] 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, because ACPs and availability under strong consistency are not the focus of this work and because different ACPs may result in very different levels of service availability, we choose to simulate the best-case service availability at the strong consistency extreme of the spectrum. Note that all other data points are derived from our prototype and that our methodology for determining the upper bound on 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.

5. 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 bounds. Finally, we explore the effects of the degree of replication on service availability.

Because of space limitations, we omit an evaluation of the staleness metric and focus on numerical error and order error. Preliminary evaluations and the theory in Section 3 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.

5.1 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 4.3. 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 700-800 Mhz 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 bounds for service availability using the methodology from Section 3. 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 2) as the availability metric. We also assume that replicas accept all reads and reject the writes that would violate global consistency requirements. Thus, we can re-define service availability to be the percentage of writes that are accepted by the replicas.

Consistency	# Writes (WAN)	# Rejected (WAN)	Avail. (WAN)	# Writes (emulation)	# Rejected (emulation)	Avail. (emulation)	Accuracy
NE=6, OE=∞	120,703	1,699	98.6%	120,703	1,762	98.5%	96.3%
NE=∞, OE=1	60,439	293	99.5%	60,439	298	99.5%	98.3%

Table 2: Wide-area deployment and emulation verification results.

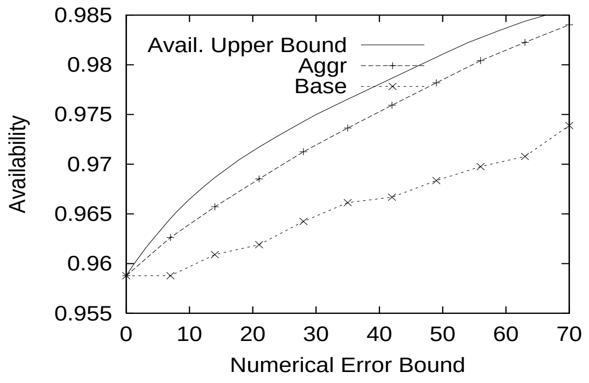

Figure 2: Availability as function of numerical error (SIM1_00).

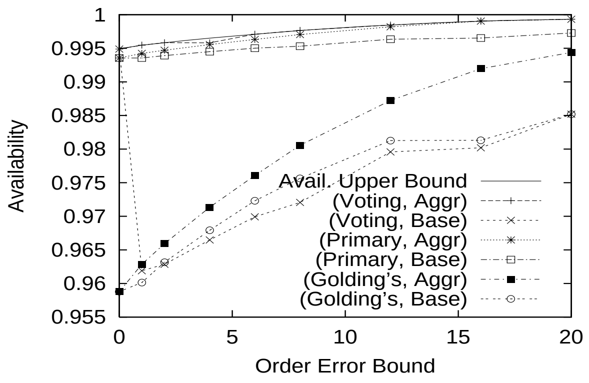

Figure 3: Availability as function of order error (SIM1_00).

Figure 2 plots availability as a function of numerical error for the SIM1_00 faultload (see Table 1). 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 algorithm 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 3 show that the best overall availability is achieved through aggressive write propagation. Thus, we modify our numerical error bounding algorithm 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 Aggr in Figure 2 depicts the results of this modification. Using this technique, achieved availability closely approaches the upper bound.

Figure 3 shows availability as a function of order error for the SIM1_00 faultload using our order error bounding protocols. 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 there is roughly equal failure rates among the replicas). Thus, primary copy should deliver 8 times better availability than Golding's algorithm; this can be verified in Figure 3. 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 our assumption that reads are never rejected. Worse yet, once the system enters such a state, it must wait for the partition to be repaired before

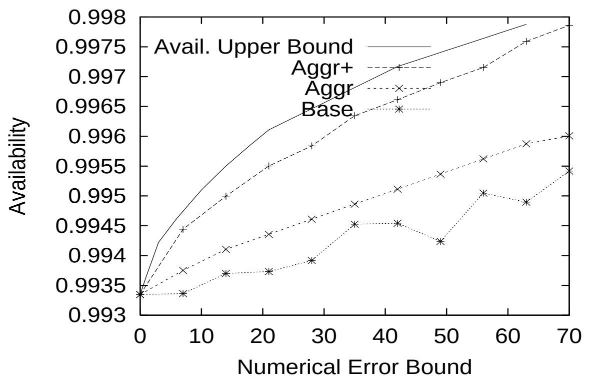

Figure 4: Availability as function of numerical error (SAMPLED1).

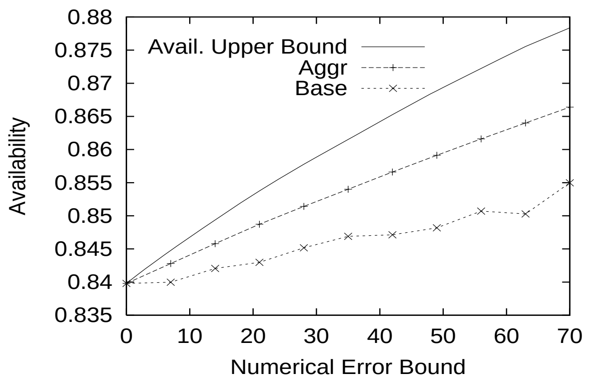

Figure 6: Availability as function of numerical error (SIM5_00).

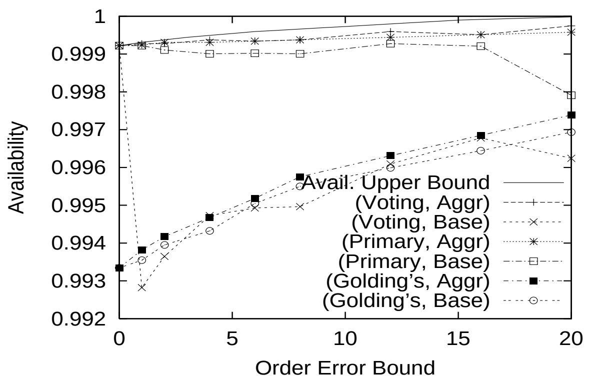

Figure 5: Availability as function of order error (SAMPLED1).

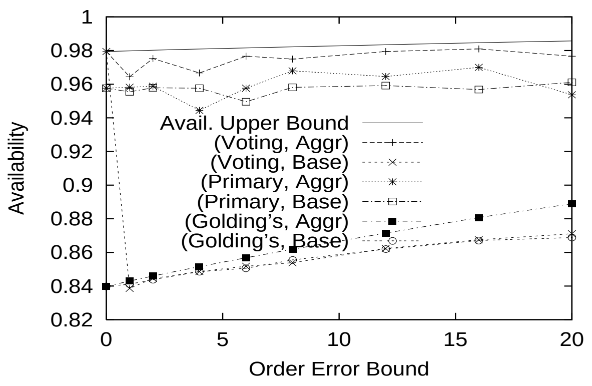

Figure 7: Availability as function of order error (SIM5_00).

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 above) 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.

Figures 4 and 5 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 algorithms demonstrate much higher availability. For Figure 4, 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 6 and 7 show similar results for a faultload with a much higher failure rate (over 4%). It is interesting to note the importance of using aggressive 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 reliability of the Internet.

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.

Of course, we do not claim all faultloads have such properties. However, the above observations are consistent with expected faultloads given the Internet's hierarchical topology and the power-law distribution of node degrees [12]. For faultloads that do not have these properties, the availability upper bounds cannot be easily approached. We have verified this through a manually-constructed faultload [39].

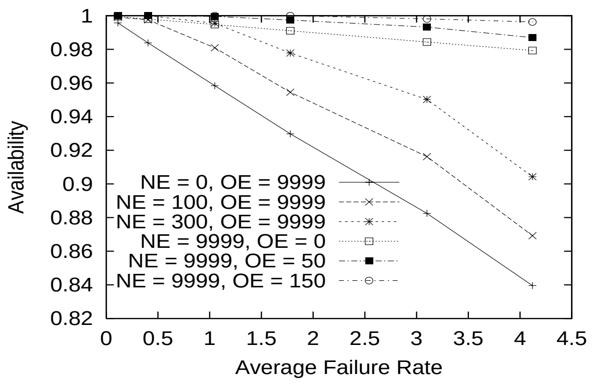

Figure 8: Availability with different average failure rate.

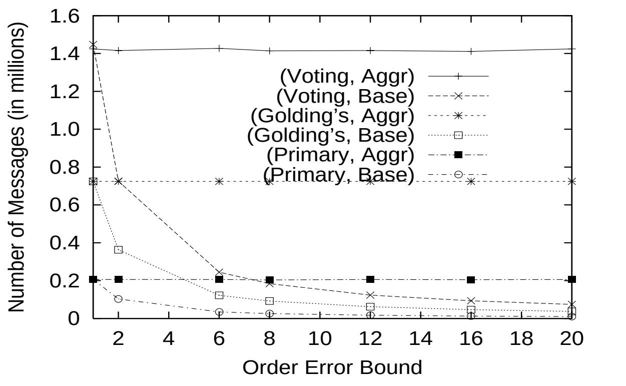

Figure 10: Number of messages to maintain order error.

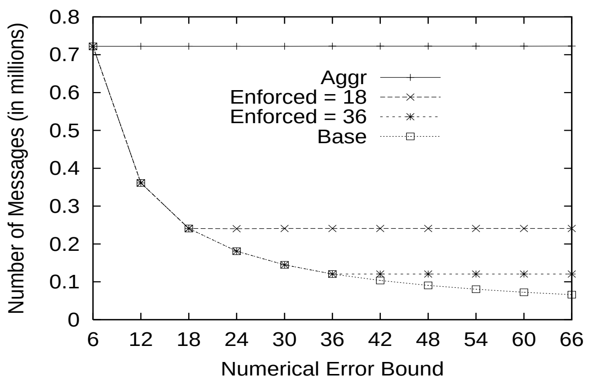

Figure 9: Number of messages to maintain numerical error.

Figure 8 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 rate. 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 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.

5.2 Availability/Communication Tradeoffs

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 [3, 26], 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 service availability. Figures 9 and 10 show the corresponding increase in the number of messages incurred by our consistency protocols using the faultload generated during WAN verification (described in Section 4.3). 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 9 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). For order error (as depicted in Figure 10), 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 repli-

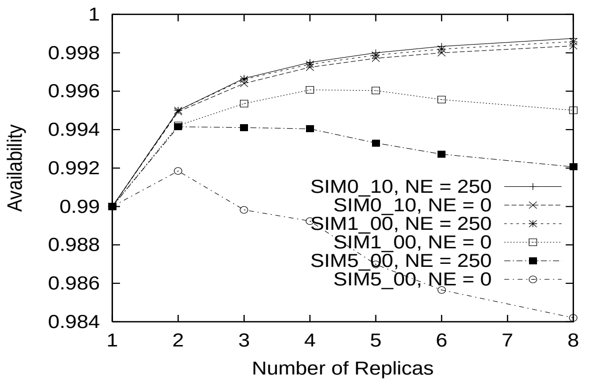

Figure 11: $Avail_{client}$ **as function of replication scale with** $Avail_{network} = 1 - 1\%/n$**.**

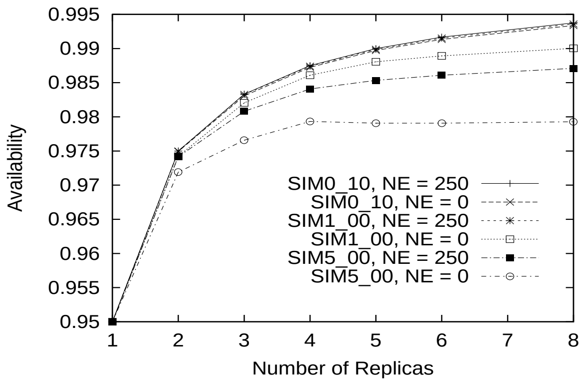

Figure 12: $Avail_{client}$ **as function of replication scale with** $Avail_{network} = 1 - 5\%/n$**.**

cas, primary copy requires about 1/7th of the messages required by voting. Note that our prototype 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 tradeoffs are representative.

5.3 Effects of Replication Scale

To this point, our evaluation of service availability considered a fixed number (eight) 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 2). 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 subsection are for calculated upper bounds. Note that we earlier demonstrated the ability of real protocols to approach these upper bounds for the studied faultloads. We restrict our attention to relaxing numerical error because of space limitations.

Figure 11 shows the effect of varying the degree of replication on service availability. 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 low failure rate model is assumed (SIM0_10), availability continues to increase to eight replicas (and beyond). Figure 12 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 other cases, the marginal benefit of each additional replica on $Avail_{network}$ is large enough to overcome the cost of replication (reduction on $Avail_{service}$).

5.4 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 tight upper bounds on availability derived in Section 3. 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 tradeoff between service 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.

6. RELATED WORK

This work uses the continuous consistency model developed in our earlier efforts [36, 38], which describe the motivation for continuous consistency and quantifies the performance and semantic benefits for a range of Internet services.

This paper quantifies the inherent costs, in terms of consistency and communication, of 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 these bounds for our set of workloads and faultloads.

Fox and Brewer [13] discuss the potential tradeoffs 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 [1, 22, 32]. We believe our methodology and results are generally applicable to a broad range of consistency models [38]. Availability of a replication system is also related to the "availability" of distributed consensus protocols. Relative to protocols [25] 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).

Chandra et.al.[8] 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 [2, 4, 9, 11, 19, 23] explore service availability under strong consistency, typically in the context of quorum systems. Amir et.al. [2] evaluate the availability of a quorum system running at two Internet sites. Coan et.al. [9] derive tight availability upper bounds in the case of two-way partitions. Another study [19] shows that replication provides little availability benefits 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.

7. 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 tradeoffs between strong and optimistic consistency models, this paper 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 paper 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 bounds 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.

8. ACKNOWLEDGMENTS

We would like to thank our shepherd, Brian Noble, and the rest of the anonymous reviewers for their insightful feedback. We would also 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.

9. REFERENCES

[1] A. Adya, B. Liskov, and P. O'Neil. Generalized Isolation Level Definitions. In *Proceedings of the IEEE International Conference on Data Engineering*, March 2000.

[2] Y. Amir and A. Wool. Evaluating Quorum Systems over the Internet. In *Proceedings of the Annual International Symposium on Fault-Tolerant Computing*, June 1996.

[3] M. Baker, J. Hartman, M. Kupfer, K. Shirriff, and J. Ousterhout. Measurements of a Distributed File System. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, pages 198–212, Oct. 1991.

[4] D. Barbara and H. Garcia-Molina. The Vulnerability of Vote Assignments. *ACM Transactions on Database Systems*, August 1986.

[5] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, 1987.

[6] A. Brown and D. Patterson. Towards Maintainability, Availability, and Growth Benchmarks: A Case Study of Software RAID Systems. In *Proceedings of the 2000 USENIX Annual Technical Conference*, June 2000.

[7] U. Cetintemel, P. Keleher, and M. Franklin. Support for Speculative Update Propagation and Mobility in Deno. In *Proceedings of the 21st IEEE International Conference on Distributed Computing Systems*, April 2001.

[8] B. Chandra, M. Dahlin, L. Gao, and A. Nayate. End-to-End WAN Service Availability. In *Proceedings of the 3rd Usenix Symposium on Internet Technologies and Systems*, January 2001.

[9] B. Coan, B. Oki, and E. Kolodner. Limitations on Database Availability When Networks Partition. In *Proceedings of the 5th ACM Symposium on Principle of Distributed Computing*, pages 187–194, August 1986.

[10] J. Czyzyk, S. Mehrotra, M. Wagner, and S. Wright. PCx: Software for Linear Programming. Available at: `http://www-fp.mcs.anl.gov/otc/Tools/PCx/`.

[11] K. Diks, E. Kranakis, D. Krizanc, B. Mans, and A. Pelc. Optimal Coteries and Voting Schemes. *Information Processing Letters*, pages 1–6, 1994.

[12] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On Power-law Relationships of the Internet Topology. In *SIGCOMM*, 1999.

[13] A. Fox and E. Brewer. Harvest, Yield, and Scalable Tolerant Systems. In *Proceedings of HotOS-VII*, March 1999.

[14] A. Fox, S. Gribble, Y. Chawathe, and E. Brewer. Cluster-Based Scalable Network Services. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, Saint-Malo, France, October 1997.

[15] D. K. Gifford. Weighted Voting for Replicated Data. In *Proceedings of the 7th SOSP*, 1979.

[16] R. Golding. A Weak-Consistency Architecture for Distributed Information Services. *Computing Systems*, 5(4):379–405, Fall 1992.

[17] J. Gray, P. Helland, P. E. O'Neil, and D. Shasha. The Dangers of Replication and a Solution. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, June 1996.

[18] J. Hennessy. The Future of Systems Research. *IEEE Computer*, 32(8):27–33, August 1999.

[19] D. B. Johnson and L. Raab. 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*, May 1991.

[20] P. Keleher. Decentralized Replicated-Object Protocols. In *Proceedings of the 18th Annual ACM Symposium on Principles of Distributed Computing*, April 1999.

[21] J. J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems*, 10(1):3–25, Feb. 1992.

[22] N. Krishnakumar and A. Bernstein. Bounded Ignorance: A Technique for Increasing Concurrency in a Replicated System. *ACM Transactions on Database Systems*, 19(4), December 1994.

[23] A. Kumar and A. Segev. Cost and Availability Tradeoffs in Replicated Data Concurrency Control. *ACM Transactions on Database Systems*, March 1993.

[24] R. Ladin, B. Liskov, L. Shrira, and S. Ghemawat. Providing High Availability Using Lazy Replication. *ACM Transactions on Computer Systems*, 10(4):360–391, November 1992.

[25] B. Lampson. How to Build a Highly Available System Using Consensus. In *Distributed Algorithms, Lecture Notes in Computer Science 1151*. Springer, 1996.

[26] L. Mummert. *Exploiting Weak Connectivity in a Distributed File System*. PhD thesis, Carnegie Mellon University, December 1996.

[27] B. Noble, B. Fleis, and M. Kim. A Case for Fluid Replication. In *Proceedings of the 1999 Network Storage Symposium (Netstore)*, October 1999.

[28] B. Noble, M. Satyananarayanan, G. Nguyen, and R. Katz. Trace-based Mobile Network Emulation. In *Proceedings of SIGCOMM*, September 1997.

[29] T. Page, R. Guy, J. Heidemann, D. Ratner, A. Goel, G. Kuenning, and G. Popek. Perspectives on Optimistically Replicated Peer-to-Peer Filing. *Software–Practice and Experience*, 28(2):155–180, February 1998.

[30] V. Paxson. End-To-End Routing Behavior in the Internet. In *Proceedings of the ACM SIGCOMM '96 Conference on Communications Architectures and Protocols*, August 1996.

[31] K. Petersen, M. Spreitzer, D. Terry, M. Theimer, and A. Demers. Flexible Update Propagation for Weakly Consistent Replication. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP-16)*, October 1997.

[32] C. Pu and A. Leff. Replication Control in Distributed System: An Asynchronous Approach. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, May 1991.

[33] Y. Saito, B. Bershad, and H. Levy. Manageability, Availability and Performance in Porcupine: A Highly Scalable Internet Mail Service. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, December 1999.

[34] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson. The End-to-End Effects of Internet Path Selection. In *SIGCOMM*, 1999.

[35] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, Dec. 1995.

[36] H. Yu and A. Vahdat. Design and Evaluation of a Continuous Consistency Model for Replicated Services. In *Proceedings of Operating Systems Design and Implementation (OSDI)*, October 2000.

[37] H. Yu and A. Vahdat. Efficient Numerical Error Bounding for Replicated Network Services. In *Proceedings of the 26th International Conference on Very Large Databases (VLDB)*, September 2000.

[38] H. Yu and A. Vahdat. Combining Generality and Practicality in a Conit-Based Continuous Consistency Model for Wide-Area Replication. In *The 21st IEEE International Conference on Distributed Computing Systems (ICDCS)*, April 2001.

[39] H. Yu and A. Vahdat. The Costs and Limits of Availability for Replicated Services. Technical Report CS-2001-03, Duke University, July 2001. Available from `http://www.cs.duke.edu/~vahdat/ps/tr-cs-2001-03.pdf`.

[40] E. W. Zegura, K. Calvert, and M. J. Donahoo. A Quantitative Comparison of Graph-Based Models for Internet Topology. *IEEE/ACM Transactions on Networking*, 5(6), December 1997.