Design and Evaluation of a Continuous Consistency Model for Replicated Services *

Haifeng Yu Amin Vahdat

Computer Science Department
Duke University
Durham, NC 27708
{yhf, vahdat}@cs.duke.edu
http://www.cs.duke.edu/˜{yhf, vahdat}

Abstract

The tradeoffs between consistency, performance, and availability are well understood. Traditionally, however, designers of replicated systems have been forced to choose from either strong consistency guarantees or none at all. This paper explores the semantic space between traditional strong and optimistic consistency models for replicated services. We argue that an important class of applications can tolerate relaxed consistency, but benefit from bounding the maximum rate of inconsistent access in an application-specific manner. Thus, we develop a set of metrics, *Numerical Error*, *Order Error*, and *Staleness*, to capture the consistency spectrum. We then present the design and implementation of TACT, a middleware layer that enforces arbitrary consistency bounds among replicas using these metrics. Finally, we show that three replicated applications demonstrate significant semantic and performance benefits from using our framework.

1 Introduction

Replicating distributed services for increased availability and performance has been a topic of considerable interest for many years. Recently however, exponential increase in access to popular Web services provides us with concrete examples of the types of services that would benefit from replication, their requirements and semantics. One of the primary challenges to replicating network services is consistency across replicas. Providing strong consistency (e.g., one-copy serializability [4]) imposes performance overheads and limits system availability. Thus, a variety of optimistic consistency models [14, 15, 18, 31, 34] have been proposed for applications that can tolerate relaxed consistency. Such models require less communication, resulting in improved performance and availability.

Unfortunately, optimistic models typically provide no bounds on the inconsistency of the data exported to client applications and end users. A fundamental observation behind this work is that there is a continuum between strong and optimistic consistency that is semantically meaningful for a broad range of network services. This continuum is parameterized by the maximum distance between a replica's local data image and some final image "consistent" across all replicas after all writes have been applied everywhere. For strong consistency, this maximum distance is zero, while for optimistic consistency it is infinite. We explore the semantic space in between these two extremes. For a given workload, providing a per-replica consistency bound allows the system to determine an expected probability, for example, that a write operation will conflict with a concurrent write submitted to a remote replica, or that a read operation observes the results of writes that must later be rolled back. No such analysis can be performed for optimistic consistency systems because the maximum level of inconsistency is unbounded.

The relationship between consistency, availability, and performance is depicted in Figure 1(a). In moving from strong consistency to optimistic consistency, application performance and availability increases. This benefit comes at the expense of an increasing probability that individual accesses will return inconsistent results, e.g., stale/dirty reads, or conflicting writes. In our work, we allow applications to bound the maximum probability/degree of inconsistent access in exchange for increased performance and availability. Figure 1(b) graphs different potential improvements in ap-

*This work is supported in part by the National Science Foundation (EIA-99772879,ITR-0082912). Vahdat is also supported by an NSF CAREER award (CCR-9984328). Additional information on the TACT project can be found at `http://www.cs.duke.edu/ari/issg/TACT/`.

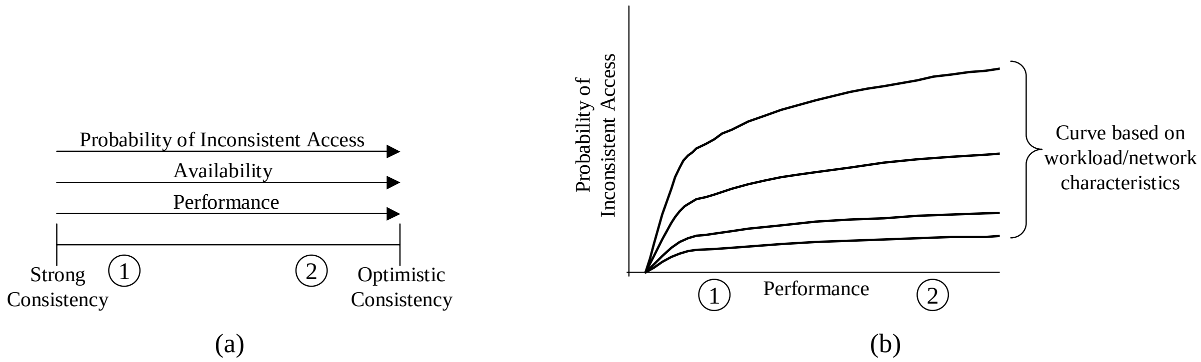

Figure 1: *a) The spectrum between strong and optimistic consistency as measured by a bound on the probability of inconsistent access. b) The tradeoff between consistency, availability, and performance depends upon application and network characteristics.*

plication performance versus the probability of inconsistent access, depending on workload/network characteristics. Moving to the right in the figure corresponds to increased performance, while moving up in the figure corresponds to increased inconsistency. To achieve increased performance, applications must tolerate a corresponding increase in inconsistent accesses. The tradeoff between performance and consistency depends upon a number of factors, including application workload, such as read/write ratios, probability of simultaneous writes, etc., and network characteristics such as latency, bandwidth, and error rates. At the point labeled "1" in the consistency spectrum in Figure 1(b), a modest increase in performance corresponds to a relatively large increase in inconsistency for application classes corresponding to the top curve, perhaps making the tradeoff unattractive for these applications. Conversely, at point "2," large performance increases are available in exchange for a relatively small increase in inconsistency for applications represented by the bottom curve.

Thus, the goals of this work are: i) to explore the issues associated with filling the semantic, performance, and availability gap between optimistic and strong consistency models, ii) to develop a set of metrics that allow a broad range of replicated services to conveniently and quantitatively express their consistency requirements, iii) to quantify the tradeoff between performance and consistency for a number of sample applications, and iv) to show the benefits of dynamically adapting consistency bounds in response to current network, replica, and client-request characteristics. To this end, we present the design, implementation, and evaluation of the TACT toolkit. TACT is a middleware layer that accepts specifications of application consistency requirements and mediates read/write access to an underlying data store. If an operation does not violate pre-specified consistency requirements, it proceeds locally (without contacting remote replicas). Otherwise, the operation blocks until TACT is able to synchronize with one or more remote replicas (i.e., push or pull some subset of local/remote updates) as determined by system consistency requirements.

We propose three metrics, *Numerical Error*, *Order Error*, and *Staleness*, to bound consistency. Numerical error limits the total weight of writes that can be applied across all replicas before being propagated to a given replica. Order error limits the number of tentative writes (subject to reordering) that can be outstanding at any one replica, and staleness places a real-time bound on the delay of write propagation among replicas. Algorithms are then designed to bound each metric: Numerical error is bounded using a push approach based solely on local information; a write commitment algorithm combined with compulsory write pull enforces order error bound; and staleness is maintained using real-time vector. To evaluate the effectiveness of our system, we implement and deploy across the wide area three applications with a broad range of dynamically changing consistency requirements using the TACT toolkit: an airline reservation system, a distributed bulletin board service, and load distribution front ends to a Web server. Relative to strong consistency techniques, TACT improves the throughput of these applications by up to a factor of 10. Relative to weak consistency approaches, TACT provides strong semantic guarantees regarding the maximum inconsistency observed by individual read and write operations.

The rest of this paper is organized as follows. Section 2 describes the three network services implemented in the TACT framework to motivate our system architecture. Section 3 presents the system model and design we adopt for our target services. Next, Section 4 details the TACT architecture and Section 5 evaluates the performance of our three applications in the TACT framework. Finally, Section 6 places our work in the context of related work and Section 7 presents our conclusions.

2 Applications

2.1 Airline Reservations

Our first application is a simple replicated airline reservation system that is designed to be representative of replicated E-commerce services that accept inquiries (searches) and purchase orders on a catalog. In our implementation, each server maintains a full replica of the flight information database and accepts user reservations and inquiries about seat availability. Consistency in this application is measured by the percentage of requests that access inconsistent results. For example, in the face of divergent replica images, a user may observe an available seat, when in fact the seat has been booked at another replica (false positive). Or a user may see a particular seat is booked when in fact, it is available (false negative). Intuitively, the probability of such events is proportional to the distance between the local replica image and some consistent final image.

One interesting aspect of this application is that its consistency requirements change dynamically based on client, network, and application characteristics. For instance, the system may wish to minimize the rate of inquiries/updates that observe inconsistent intermediate states for certain preferred clients. Requests from such clients may require a replica to update its consistency level (by synchronizing with other replicas) before processing the request or may be directed to a replica that maintains the requisite consistency by default. As another example, if network capacity (latency, bandwidth, error rate) among replicas is abundant, the absolute performance/availability savings may not be sufficient to outweigh the costs associated with weaker consistency models. Finally, the desired consistency level depends on individual application semantics. For airline reservations, the cost of a transaction that must be rolled back is fairly small when a flight is empty (one can likely find an alternate seat on the same flight), but grows as the flight fills.

2.2 Bulletin Board

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. 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.

2.3 QoS Load Distribution

The final application implemented in our framework is a load distribution mechanism that provides Quality of Service (QoS) guarantees to a set of preferred clients. In this scenario, front-ends (as in LARD [27]) accept requests on behalf of two classes of clients, standard and preferred. The front ends forward requests to back end servers with the goal of reserving some pre-determined portion of server capacity for preferred clients. Thus, front ends allow a maximum number of outstanding requests (assuming homogeneous requests) at the back end servers. To determine the maximum number of "standard" requests that should be forwarded, each front end must communicate current access patterns to all other front ends.

One goal of designing such a system is to minimize the communication required to accurately distribute such load information among front ends. 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.

3 System Design

In this section, we first describe the basic replication system model we assume, and then elaborate on the model and metrics we provide to allow applications to continuously specify consistency level.

3.1 System Model

For simplicity, we refer to application data as a data store, though the data can actually be stored in a database, file system, persistent object, etc. The data store is replicated in full at multiple sites. Each replica accepts requests from users that can be made up of multiple primitive read/write operations. TACT mediates application read/write access to the data store. On a single replica, a read or write is isolated from other reads or writes during execution. Depending on the specified consistency requirements, a replica may need to contact other replicas before processing a particular request.

Replicas exchange updates by propagating writes. This can take the form of gossip messages [22], anti-entropy sessions [13, 28], group communication [5], broadcast, etc. We chose anti-entropy exchange as our write propagation method because of its flexibility in operating under a variety of network scenarios. Each write bears an accept stamp composed of a logical clock time [23] and the identifier of the accepting replica. Replicas deterministically order all writes based on this accept stamp. As in Bayou [28, 34], updates are procedures that check for conflicts with the underlying data store before being applied in a *tentative* state. A write is tentative until a replica is able to determine the write's final position in the serialization order, at which point it becomes *committed* through a write commitment algorithm (described below).

Each replica maintains a logical time vector, similar to that employed in Bayou and in Golding's work [13, 28, 34]. Briefly, each entry in the vector corresponds to the latest updates seen from a particular replica. The *coverage property* ensures that a replica has seen all updates (remote and local) up to the logical time corresponding to the minimum value in its logical time vector. This means the serialization positions of all writes with smaller logical time than that minimal value can be determined and thus those writes can be committed. Anti-entropy sessions update values in each replica's logical time vector based on the logical times/replicas of the writes exchanged. Note that writes may have to be reordered or rolled back before as dictated by serialization order.

While TACT's implementation of anti-entropy is not particularly novel, a primary aspect of our work is determining when and with whom to perform anti-entropy in order to guarantee a minimum level of consistency. Replicas may propagate writes to other replicas at any time through *voluntary anti-entropy*. However, we are more concerned with write propagation required for maintaining a desired level of consistency, called *compulsory anti-entropy*. Compulsory anti-entropy is necessary for the correctness of the system, while voluntary anti-entropy only affects performance.

3.2 A Continuous Consistency Model

In our consistency model, applications specify their application-specific consistency semantics using *conits*. A conit is a physical or logical unit of consistency, defined by the application. For example, in the airline reservation example, individual flights or blocks of seats on a flight may be defined as a conit. An interesting issue beyond the scope of this paper is setting the granularity of conits. The required per-conit accounting overhead (described below) argues for coarse conit granularity. Conversely, coarse-grained conits may introduce issues of false sharing as updates to one data item in a conit may reduce performance/availability for accesses to logically unrelated data items in the same conit.

For each conit, we quantify consistency continuously along a three-dimensional vector:

$$\begin{aligned}&\mathbf{Consistency} = \\ &\qquad(\mathbf{Numerical\ Error}, \mathbf{Order\ Error}, \mathbf{Staleness})\end{aligned}$$

Numerical Error bounds the discrepancy between the value of the conit relative to its value in the "final image." For applications that maintain numerical records, the semantics of this metric are straightforward. For other applications, however, application-specific weights (defaulting to one) can be assigned to individual writes. The weights are the relative importance of the writes, from the application's point of view. Numerical error then becomes the total weighted unseen writes for a conit. Based on application semantics, two different kinds of numerical error, absolute numerical error and relative numerical error, can be defined. *Order Error* measures the difference between the order that updates are applied to the local replica relative to their ordering in the eventual "final image." *Staleness* bounds the difference between the current time and the acceptance time of the oldest write on a conit not seen locally.

Figure 2 illustrates the definition of order error and numerical error in a simple example. Two replicas, A and B, accept updates on a conit containing two data items, x and y. The logical time vector for A is $(24, 5)$. The coverage property implies that all writes in its log with logical time less than or equal to five are committed (indicated by the shaded box), leaving three tentative writes. Similarly, the logical time vector for B is $(0, 17)$, meaning that both writes in its log are tentative. Order error bounds the maximum number of tentative writes at a replica, i.e., the maximum number of writes that may have to be reordered or rolled back because of activity at other replicas. In general, a lower bound on order error implies a lower probability that a read will observe an inconsistent intermediate state. In this example, if $A's$ order error is bounded to three, A must invoke the write commitment algorithm—performing compul-

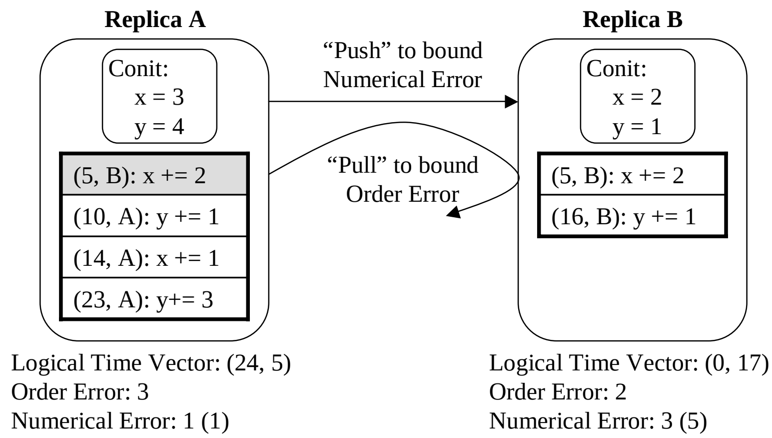

Figure 2: Example scenario for bounding order error and numerical error with two replicas.

sory anti-entropy to pull any necessary updates from B to reduce its number of tentative writes—before accepting any new writes.

Figure 2 also depicts the role of numerical error. Numerical error is the weight of all updates applied to a conit at *all* replicas not seen by the local replica. In the example, the weight of a write is set to be the update amount to either x or y, so that a "major" update is more important than a "minor" update. The replica A has not seen one update (with a weight of one) in this example, while B has not seen three updates (with a total weight of five). Note that order error can be relaxed or tightened using only local information. Bounding numerical error, on the other hand, relies upon the cooperation of all replicas. Thus, dynamically changing numerical error bounds requires the execution of a consensus algorithm.

One benefit of our model is that conit consistency can be bounded on a per-replica basis. Instead of enforcing a system-wide uniform consistency level, each replica can have its own independent consistency level for a conit. A simple analysis can show that as a replica relaxes its consistency while other replicas' consistency levels remain unchanged, the total communication amount of that replica is reduced. For relaxed numerical error, it means other replicas can push writes to that replica less frequently, resulting in fewer incoming messages. Outgoing communication amount remains unchanged since that is determined by the consistency levels of other replicas. However, since numerical error is bounded using a push approach, if the replica is too busy to handle the outgoing communication, writes submitted to it will be delayed. Similar, if the replica relaxes order error and staleness, incoming communication amount will be decreased. Thus, one site may have poor network connectivity and limited processing power, making more relaxed consistency bounds appropriate for that replica. Conversely, it may be cheap (from a performance and availability standpoint) to enforce stronger consistency at a replica with faster links and higher processing capacity. One interesting aspect of this model is that it potentially allows the system to route client requests to replicas with appropriate consistency bounds on a per-request basis. For instance, in the airline reservation system, requests from "preferred" clients may be directed to a replica that maintains higher consistency levels (reducing the probability of an inconsistent access).

When all three metrics are bounded to zero, our continuous consistency model reaches the strong consistency extreme of the spectrum, which is serializability [4] and external consistency [1, 12]. If no bounds are set for any of the metrics, there will be no consistency guarantees, similar to optimistic consistency systems. In moving from strong to optimistic consistency, applications bound the maximum logical "distance" between the local replica image and the (unknown) consistent image that contains all writes in serial order. This distance corresponds directly to the percentage chance that a read will observe inconsistent results or that a write will introduce a conflict. In the next section, we will demonstrate how our three applications employ these metrics to capture their consistency requirements. Based on our experience with TACT, we believe that the above metrics allow a broad range of applications to conveniently express their consistency requirements. Of course, the exact set of metrics is orthogonal to our goal of exporting a flexible, continuous, and dynamically tunable spectrum of consistency models to replicated services.

Application	Consistency Semantics	Conit Definition	Weight Definition	Metrics Capturing the Semantics
Bulletin Board	A. Message Ordering B. Unseen Messages C. Message Delay	A Newsgroup	(Subjective) Importance of a News Message	A. Order Error B. Absolute Numerical Error C. Staleness
Airline Reservation	A. Reservation Conflict Rate R B. Inconsistent Query Results	Seats on a Flight	Reservation: 1	A. Relative Numerical Error $R_{max} = 1 - 1/(1+\gamma)$ $R_{avg} = (1 - 1/(1+\gamma))/2$ B. Order Error and Staleness
QoS Load Distribution	A. Accuracy of Resource Consumption Info	Resource Consumption Info	Request Forward: 1 Request Return : -1	A. Relative Numerical Error

Table 1: Expressing high-level application-specific consistency semantics using the TACT continuous consistency model.

3.3 Expressing Application-specific Consistency Semantics through Conits

One important criteria for the evaluation of any consistency model is whether it captures the semantic requirements of a broad range of applications. Thus, in this section we describe the ways different consistency levels affect the semantics of the three representative applications described in Section 2 and explain how these semantics are captured by our model. Table 1 summarizes these application-specific consistency semantics and their expression using TACT, as detailed in the discussion below.

For the distributed bulletin board, one consistency requirement is the ordering of messages, which is captured by the order error metric. More specifically, for this application order error is the number of messages that may appear out of order at any replica. However, it is possible that such a bound is overly restrictive for unrelated messages, e.g., for two messages posted to different newsgroups. In this case, a conit can be defined for each newsgroup to more precisely specify ordering requirements. Another possible consistency requirement is the maximum number of remotely posted messages that are unseen by the local replica at a particular time. Our numerical error metric serves to express this type of semantics. Our model allows application-specific weights to be assigned to each write, allowing users to (subjectively) force the propagation of certain writes. A third consistency requirement for this application is message delay, that is, the delay between the time a message is posted and the time it is seen by all replicas. This requirement can be translated to staleness in a straight-forward manner.

Moving to the airline reservation example, one important manifestation of system consistency is the percentage of conflicting reservations. An interesting aspect of this application is TACT's ability to limit reservation conflict rate by bounding relative numerical error based on the application's estimate of available seats. To simplify the discussion, we assume single seat reservations (though our model and implementation are more general) and define a conit over all seats on a flight with each reservation carrying a numerical weight of -1. Initially, the value of the conit is the total number of seats on the flight. As reservations come in, the value of the conit is the number of available seats in each replica's data store. Suppose reservations are randomly distributed among all available seats. For a reservation accepted by one replica, the probability that it conflicts with another remote (unseen) reservation is U/V, where U is the number of unseen reservations, and V is the number of available seats as seen by the local replica. Suppose V_{final} is the accurate count of available seats, such that $V_{final} = V - U$. Thus, the rate of conflicting reservations, R, equals $1 - V_{final}/V$. If γ bounds the maximum relative numerical error of the conit then, by definition, we have $-\gamma \leq 1 - V/V_{final} \Rightarrow V_{final} \geq 1/(1+\gamma) \times V$. Thus, the upper bound on R, $R_{max} = 1 - V_{final}/V = 1 - 1/(1+\gamma)$. Since in this example, V_{final} is always smaller than or equal to V, the average value of V_{final} should then be $(1 + 1/(1+\gamma))/2 \times V_i$ Thus, the average rate of conflicting reservations, R_{avg}, equals to $1 - V_{final}/V = 1 - (1 + 1/(1+\gamma))/2 = (1 - 1/(1+\gamma))/2$. In Section 5, we present experimental results to verify this analysis. Non-random reservation behavior will result in a higher conflict rate than the above formula. However, applications can still reduce the expected/maximum conflict rate by specifying multiple conits per flight, e.g., multiple conits for first class versus coach seats or aisle versus window seats.

Other consistency semantics for the airline reservation example can be expressed using order error or staleness. For example, the system may wish to limit the percentage of queries that access an inconsistent image, i.e., see a multi-seat reservation that must later be rolled back because of a conflicting single-seat reservation at another replica. Such consistency semantics can be enforced by properly bounding the limit on order error (an analysis is omitted for brevity).

In our third application, QoS load distribution, front ends estimate the total resource consumption for standard clients as the total number of outstanding standard requests on the back ends. This value also serves as the definition of a conit for this application. Front ends increase this value by 1 upon forwarding a request from a standard client and decrease it by 1 when the request returns. If this value exceeds a pre-determined resource consumption limit, front ends will not forward new standard client requests until resource consumption drops below this limit. The relative numerical error of each front end's estimate of resource consumption captures this application's consistency semantics — each front end is guaranteed that its estimate of resource consumption is accurate within a fixed bound. Note that this load balancing application is not concerned with order error (writes are interchangeable) or staleness (no need to synchronize if the mix of requests does not change).

3.4 Bounding Consistency Metrics

Given the assumed system model, we now describe in turn our algorithms for bounding numerical error, order error, and staleness. Note that the details and correctness proofs for our numerical error algorithms available separately [38].

The first algorithm, *Split-weight AE*, employs a "push" approach to bound absolute numerical error. It "allocates" the allowed positive and negative error for a server evenly to other servers. Each $server_i$ maintains two local variables x and y for $server_j, j \neq i$. Intuitively, the variable x is the total weight of negatively-weighted writes that $server_i$ accepts but has not been seen by $server_j$. $server_i$ has only conservative knowledge (called its *view*) of what writes $server_j$ has seen. The variable x is updated when $server_i$ accepts a new write with a negative weight or when $server_i$'s view is advanced. Similarly, the variable y records the total weight of positively-weighted writes. Suppose the absolute error bound on $server_j$ is α_j. In other words, we want to ensure that $|V_{final} - V_j| \leq \alpha_j$, where V_{final} is the consistent value and V_j is the value on $server_j$. To achieve this, $server_i$ makes sure that at all times, $x \geq -\alpha_j/(n-1)$ and $y \leq \alpha_j/(n-1)$, where n is the total number of servers in the system. This may require $server_i$ to push writes to $server_j$ before accepting a new write.

Split-Weight AE is pessimistic in the sense that $server_i$ may propagate writes to $server_j$ when not actually necessary. For example, the algorithm does not consider the case where negative weights and positive weights may offset each other. We developed another optimal algorithm, *Compound-Weight AE*, to address this limitation at the cost of increased space overhead. However, simulations indicate that potential performance improvements do not justify the additional computational complexity and space overhead [38].

A third algorithm, *Inductive RE*, provides an efficient mechanism for bounding the relative error in numerical records. The algorithm transforms relative error into absolute error. Suppose the relative error bound for $server_j$ is γ_j, that is, we want to ensure $|1 - V_j/V_{final}| \leq \gamma_j$, equivalent to $|V_{final} - V_j| \leq \gamma_j \times V_{final}$. A naive transforming approach would use $\gamma_j \times V_{final}$ as the corresponding absolute error bound, requiring a consensus algorithm to be run to determine a new absolute error bound each time V_{final} changes.

Our approach avoids this cost by conservatively relying upon local information as follows. We observe that the current value V_i on any $server_i$ was properly bounded before the invocation of the algorithm and is an approximation of V_{final}. So $server_i$ may use V_i as an approximate norm to bound relative error for other servers. More specifically, for $server_i$, we know that $V_{final} - V_i \geq -\gamma_i \times V_{final}$, where γ_i is the relative error bound for $server_i$, which transforms to $V_{final} \geq V_i/(1+\gamma_i)$. Using this information to substitute for V_{final} on the right-hand side in the inequality in the last paragraph produces:

$$|V_{final} - V_j| \quad \leq \quad \gamma_j \times \frac{V_i}{1+\gamma_i}$$

Thus, to bound relative error, $server_i$ only needs to recursively apply Split-Weight AE, using $\gamma_j \times V_i/(1+\gamma_i)$ as α_j. Note that while this approach greatly increases performance by eliminating the need to run a consensus algorithm among replicas, it uses local information ($V_i/(1+\gamma_i)$) to approximate potentially unknown global information (V_{final}) in bounding relative error. Thus it behaves conservatively (bounding values more than strictly necessary) when relative error is high as will be shown in our evaluation of these algorithms in Section 5.

To bound order error on a per-conit basis, a replica first checks the number of tentative writes on a conit in its write log. If this number exceeds the order error limit, the replica invokes a write commitment algorithm to reduce the number of tentative writes in its write log. This algorithm operates as follows. The replica pulls writes from other replicas by performing compulsory anti-entropy sessions to advance its logical time vector, allowing it to commit some set of its tentative writes. In doing so, the replica ensures that it remains within a specified order error bound before accepting new tentative writes.

To bound the staleness of a replica, each server maintains a *real time vector*. This vector is similar to the logical time vector, except that real time instead of logical time is used. A similar coverage property is preserved between the writes a server has seen and the real time vector. If A's real time vector entry corresponding to B is t, then A has seen all writes accepted by B before real time t. To bound staleness within l, a server checks whether $current\ time - t < l$ holds for each entry in the real time vector.[1] If the inequality does not hold for some entries, the server performs compulsory anti-entropy session with the corresponding servers, pulling writes from them, and advances the real time vector. This pull approach may appear to be less efficient than a push approach because of unnecessary polling when no updates are available. However, a push approach cannot bound staleness if there is no upper limit on network delay or processing time.

4 System Architecture

The current prototype of TACT is implemented in Java 1.2 using RMI for communication (e.g., for accepting read/write requests and for write propagation). TACT replicas are multi-threaded, thus if one write incurs compulsory write propagation, it will not block writes on other conits. We implemented a simple custom database for storing and retrieving data values, though our design and implementation is compatible with a variety of storage mechanisms.

Each TACT replica maintains a write log, and allows redo and undo on the write log. It is also responsible for all anti-entropy sessions with remote replicas. The system supports parallel anti-entropy sessions with multiple replicas, which can improve performance significantly for compulsory anti-entropy across the wide area. For increased efficiency, we also implement a one-round anti-entropy push. With standard anti-entropy, before a replica pushes writes to another replica, it first obtains the target replica's logical time vector to determine which writes to propagate. However, we found that this two-round protocol can add considerable overhead across the wide area, especially at stronger consistency levels (where the pushing replica has a fairly good notion of the writes seen by the target replica). Thus, we allow replicas to push writes using their local view as a hint, reducing two rounds of communication to one round at the cost of possibly propagating unnecessary writes. While the current implementation uses this one round protocol by default, dynamically switching between the variants based on the consistency level would be ideal.

TACT replicas also implement a consistency manager responsible for bounding numerical error, order error and staleness. The variables needed by the Split-Weight AE and Inductive RE algorithms are maintained in hash tables to reduce space overhead and enable the system to potentially scale to thousands of conits.

In bounding numerical error, a replica may need to push a write to other replicas before the write can return, e.g., if a write has a weight that is larger than another replica's absolute error bound. There are two possible approaches for addressing this. One approach is a one-round protocol where the local site applies the write, propagates it to the necessary remote replicas, awaits acknowledgments, and finally returns. This one-round protocol is appropriate for applications where writes are interchangeable such as resource accounting/load balancing. For other applications, such as the airline reservation example, a reservation itself observes a consistency level (the probability it conflicts with another reservation submitted elsewhere). In such a case, a stronger two-round protocol is required where the replica first acquires remote data locks, pushes the write to remote replicas, and then returns after receiving all acknowledgments. Such a two-round protocol ensures the numerical error observed by a write is within bound at the time the update is submitted. In our prototype, both protocols are implemented and the application is allowed to choose based on its own requirements.

5 Experience and Evaluation

Given the description of our system architecture, we now discuss our experience in building the three applications described in Section 2 using the TACT infrastructure. We define conits and weights in these applications according to the analysis in Section 3.3. The experiments below focus on TACT's ability to bound numerical error and order error. While implemented in our prototype, we do not present experiments addressing staleness for brevity and because bounding staleness is well-studied, e.g., in the context of Web proxy caching [10].

5.1 Bulletin Board

For our evaluation of the bulletin board application, we deployed replicas at three sites across the wide area: Duke University (733 Mhz Pentium III/Solaris 2.8), University of Utah (350 Mhz Pentium II/FreeBSD 3.4) and University of California, Berkeley (167 Mhz Ultra I/Solaris 2.7). All data is collected on otherwise unloaded systems. Each submitted message is assigned a numerical weight of one (all messages are considered equally important).

We conduct a number of experiments to explore the behavior of the system at different points in the con-

[1] We assume that server clocks are loosely synchronized.

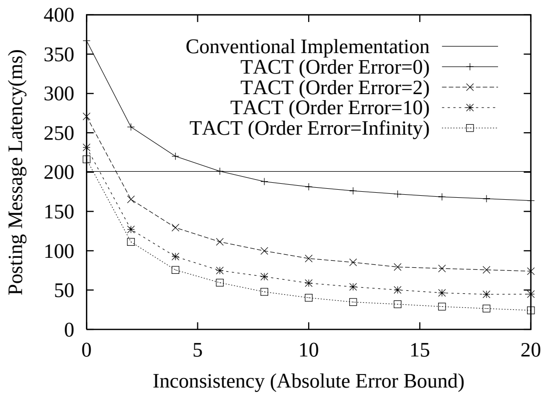

Figure 3: Average latency for posting messages to a replicated bulletin board as a function of consistency guarantees.

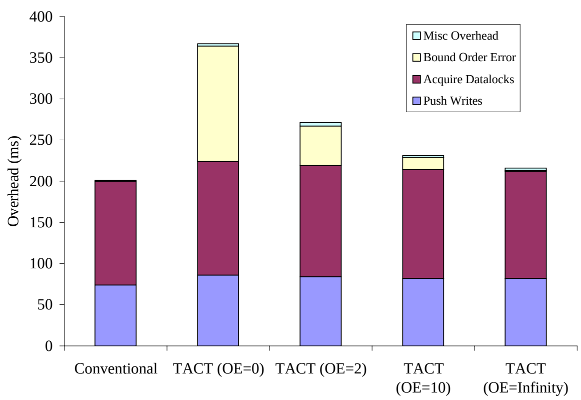

Figure 4: Breakdown of the overhead of posting a message under a number of scenarios.

sistency spectrum. Figure 3 plots the average latency for a client at Duke to post 200 messages as a function of the numerical error bound on the x-axis. For comparison, we also plot the average latency for a conventional implementation using a two-phase update protocol. For each write, this protocol first acquires necessary remote data locks, then propagates the update to all remote replicas. The figure shows how applications are able to continuously trade performance for consistency using TACT. As the numerical error bound increases, average latency decreases. Increasing allowable order error similarly produces a corresponding decrease in average latency. Relative to the conventional implementation, allowing each replica to have up to 20 unseen messages and leaving order error unbounded reduces average latency by a factor of 10.

One interesting aspect of Figure 3 is that TACT performs worse than the standard two-phase update protocol at the strong consistency end of the spectrum. To investigate this overhead, Figure 4 summarizes the performance overheads associated with message posts using TACT at four points in the consistency spectrum (varying order error with numerical error set to zero) in comparison to the conventional two-phase update protocol. All five configurations incur approximately 130ms to sequentially (required to avoid deadlock) acquire data locks from two remote replicas and 80ms to push writes to these replicas in parallel. Since the cost of remote processing is negligible, this overhead comes largely from wide-area latency. Compared to the conventional implementation, TACT with zero numerical error and zero order error (i.e., same consistency level) incurs about 83% more overhead. This additional overhead stems from the additional 140ms to bound order error. This is an interesting side effect associated with our implementation. Our design decomposes consistency into two orthogonal components (numerical error and order error) that are bounded using two separate operations, doubling the number of wide-area round trip times. When order error and numerical error are both zero, TACT should combine the push and pull of write operations into a single step as a performance optimization, as is logically done by the conventional implementation. This idea is especially applicable if we use the recently proposed quorum approach[16, 17] to commit writes. A preliminary implementation of this optimization shows that TACT's overhead (at strong consistency) drops from 367ms to about 217ms, within 8% of the conventional approach.

5.2 Airline Reservation System

We now evaluate our implementation of the simple airline reservation system using TACT. Once again, we deployed three reservation replicas at Duke, Utah and Berkeley. We considered reservation requests for a single flight with 400 seats. Each client reservation request is for a randomly chosen seat on the flight. If a tentative reservation conflicts with a request at another replica, a merge procedure attempts to reserve a second seat on the same flight. If no seats are available, the reservation is discarded. A conit is defined over all seats on the flight, with an initial value of 400. Each reservation carries a numerical weight of -1.

In Section 3.3, we derived a relationship between the reservation conflict rate R and the relative error bound γ: $R_{max} = 1 - 1/(1+\gamma)$ and $R_{avg} = (1 - 1/(1+\gamma))/2$. We conduct the following experiment to verify that an application can limit the reservation conflict rate by simply bounding the relative numerical error. Figure 5 plots the measured conflicting reservation rate R, the computed upper bound R_{max} and the computed average rate R_{avg} as a function of relative numerical error. Order error and staleness are not bounded in these experiments.

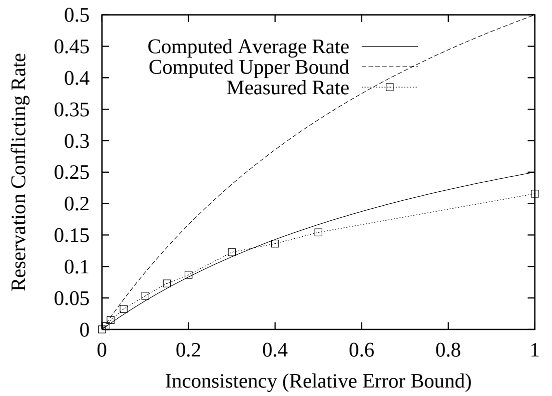

Figure 5: Percentage of conflicting reservations as a function of the bound on numerical error.

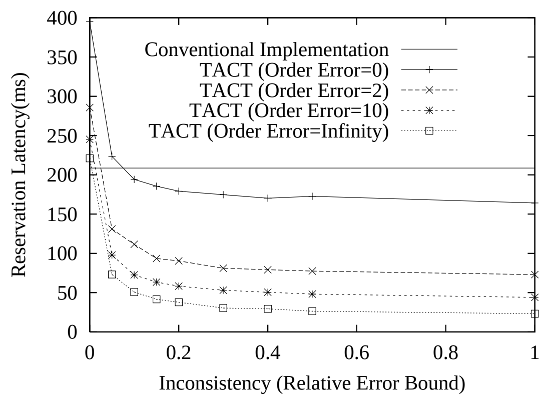

Figure 6: Average latency for making a reservation as a function of consistency guarantees.

The experiments are performed with two replicas on a LAN at Duke, each attempting to make 250 (random) reservations with the results averaged across four runs.

The measured conflict rate roughly matches the computed average rate and is always below the computed upper bound, demonstrating that numerical error can be used to bound conflicting accesses as shown by our analysis. Note that as the bound on relative error is relaxed, the discrepancy between the measured rate and the computed average rate gradually increases because of conservativeness inherent in the design of our Inductive RE algorithm (i.e., at relaxed consistency, our algorithm performs more write propagation than necessary). As described in Section 3, this conservative behavior greatly improves performance by allowing each replica to bound relative error using only local information.

The latency and throughput measurements, summarized in Figures 6 and 7 for airline reservations are similar to the bulletin board application described above.

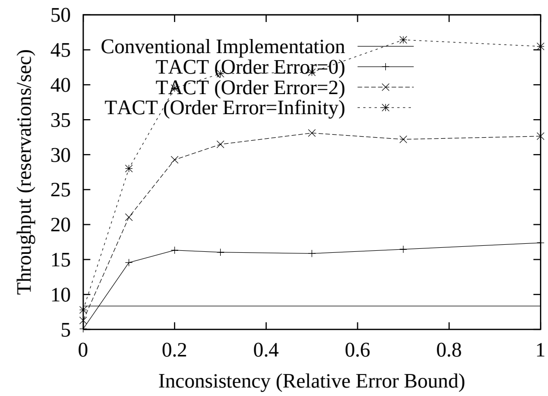

Figure 7: Update throughput for airline reservations as a function of consistency guarantees.

The latency experiments are run on the same wide-area configuration as the bulletin board. The plotted latency is the average observed by a single Duke client making 400 reservations. For throughput, we run two client threads at each of the replica sites, with each thread requesting $400/(2 \times 3) = 67$ (random) seats in a tight loop. We also plot the application's performance using a two-phase update protocol, showing the same trends as the results for the bulletin board application. As consistency is gradually relaxed, TACT achieves increasing performance by reducing the amount of required wide-area communication.

5.3 Quality of Service for Web Servers

For our final application, we demonstrate how TACT's numerical error bound can be used to accurately enforce quality of service (QoS) guarantees among Web servers distributed across the wide area. Recall that a number of front-end machines forward requests on behalf of both standard and preferred clients to back end servers. In our implementation, we use TACT to dynamically trade communication overhead in exchange for accuracy in measuring total resources consumed by standard clients. The front ends estimate the standard client resource consumption as the total number of outstanding standard requests on the back ends. If this resource consumption exceeds a pre-determined resource consumption limit, front ends will not forward new standard client requests until resource consumption drops below this limit. For simplicity, all our experiments are run on a local-area network at Duke on seven 733 Mhz Pentium III's running Solaris 2.8. Three front ends (each running on a separate machine) generate requests in a round robin fashion to three back end servers running Apache 1.3.12.

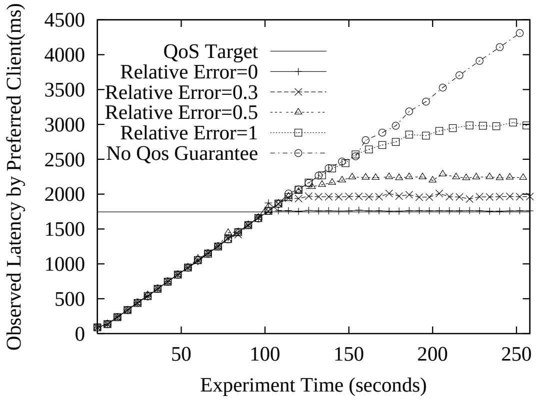

Figure 8: The average latency seen by a preferred client as a function of time.

Configuration	Consistency Messages
Relative Error=0	300
Relative Error=0.3	46
Relative Error=0.5	30
Relative Error=1	16
No QoS Guarantee	0

Table 2: The tradeoff between TACT-enforced numerical error and communication overhead.

For our experiments, the three front end machines generate an increasing number of requests from standard clients. As a whole, the system desires to bound the number of outstanding standard client requests to 150. A fourth machine, representing a preferred client, periodically polls a random back end to determine system latency. Each of the three front ends starts a new standard client every two seconds which then continuously requests the same dynamically generated Web page requiring 10ms of computation time. If all front ends had global knowledge of system state, each front end would start a total of 50 standard clients. However, depending on the bound placed on numerical error, front ends may in fact start more than this number (up to 130 in the experiment described below). For simplicity, no standard clients are torn down even if the system learns that too many (i.e., more than 150) are present in aggregate. Ideally, this aggregate number would oscillate around 150 with the amplitude of the oscillation being determined by the relative numerical bound.

Figure 8 depicts latency observed by the preferred client as a function of elapsed time (corresponding to the total number of standard clients making requests). At time 260, each front end has tried to spawn up to 130 standard clients. The curves show the average latency observed by the preferred client for different bounds on numerical error. For comparison purposes, we also show the latency (1745ms) of a preferred client when there are exactly 150 outstanding standard client requests. In the first curve, labeled "Relative Error=0," the system maintains strong consistency. Therefore, the front ends are able to enforce the resource limit strictly. The curve corresponding to a relative error of 0 flattens at 100 seconds (when three front ends have created a total of 150 standard clients) with latency very close to the ideal of 1745ms. As the bound on relative error is relaxed to 0.3, 0.5, and 1, the resource consumption limit for standard clients is more loosely enforced. The curve "No_QoS" plots the latency where no resource policy is enforced. Similar to the airline reservation application, the discrepancy between the relative error upper bound of 1 and the "No_Qos" curve stems from the conservativeness of the Inductive RE algorithm.

Table 2 quantifies the tradeoff between numerical error and communication overhead. Clearly, front ends can maintain near-perfect information about the load generated from other replicas at the cost of sending one message to all peers for each event that takes place. This is the case when zero numerical error is enforced by TACT: Each replica sends 50 messages to each of two remote replicas (for a total of 300) corresponding to the number of logical events that take place during the experiment. Once each front end starts 50 standard clients, strong consistency ensures that no further messages are necessary. Of course, such accuracy is typically not required by this application. Table 2 shows that communication overhead drops rapidly in exchange for some loss of accuracy. Note that this drop off will be more dramatic as the number of replicas is increased as a result of the all-to-all communication required to maintain strong consistency.

6 Related Work

The tradeoff between consistency and performance/availability is well understood [7, 8]. Many systems have been built at the two extremes of the consistency spectrum. Traditional replicated transactional databases use strong consistency (one-copy serializability [4]) as a correctness criterion. At the other end of the spectrum are optimistic systems such as Bayou [28, 34], Ficus [14], Rumor [15] and Coda [18]. In these systems, higher availability/performance is explicitly favored over strong consistency. Besides Bayou, none of the above systems provide support for different consistency levels. Bayou provides session guarantees [9, 33] to ensure that clients switching from one replica to another view a self-consistent version of the underlying database. However, session guarantees do not provide any guarantees regarding the consistency level of a particular replica.

In [37], we present a position paper describing an earlier iteration of our consistency model, using different consistency metrics and concentrating on consistency/availability tradeoffs. A number of other efforts also attempt to numerically capture applications' consistency requirements. These techniques can be vaguely categorized into two classes: Relaxing consistency among replicas to reduce required communication (replica control) [3, 6, 19, 26, 32, 35] and relaxing consistency for transactions on a single site to allow increased concurrency on that site (concurrency control) [2, 20, 21, 29, 30, 36]. TACT is more closely related to replica control techniques. However, previous consistency models for replica control typically exploit the consistency semantics of a particular application class, abstracting its consistency requirements along a single dimension. Most of the proposed consistency metrics can be expressed within our model by constraining a subset of numerical error, order error, and staleness. Krishnakumar and Bernstein [19] propose the concept of an "N-ignorant" system, where a transaction runs in parallel with at most N conflicting transactions. By setting absolute numerical error bound to N and by assigning unit weights to writes, TACT demonstrates behavior similar to an "N-ignorant" system. Timed consistency [35] and delta consistency [32] address the lack of timing in traditional consistency models such as sequential consistency. These timed models can be readily expressed using our staleness metric. Quasi-copy caching [3] proposes four "coherency conditions," delay condition, frequency condition, arithmetic condition and version condition appropriate for read-only caching. TACT, on the other hand, is designed for more general read/write replication. Two recent efforts [6, 26] use metrics related to numerical error and staleness to measure database freshness. However, these systems do not provide mechanisms to bound data consistency using the proposed metrics. Relative to these efforts, our conit-based three-dimensional consistency model allows a wide range of services to dynamically express their consistency semantics based on application, network, and client-specific characteristics.

Concurrency control techniques using relaxed consistency models [2, 20, 21, 29, 30, 36] are related to replica control and TACT, in that consistency also needs to be quantified there. However, enforcing user-defined consistency levels is inherently easier in concurrency control than in replica control because in the former case most information needed to compute the amount of inconsistency is available on a single site. In other words, the consistency models do not need to consider "final image," which might be unknown to all replicas. Since all our three metrics are related to "final image," none of them can be expressed using relaxed consistency models for concurrency control.

In fluid replication [24], clients are allowed to dynamically create service replicas to improve performance. Their study on when and where to create a service replica is complementary to our study on tunable consistency issues among replicas. Similar to Ladin's system [22], fluid replication supports three consistency levels: last-writer, optimistic and pessimistic. Our work focuses on capturing the spectrum between optimistic and pessimistic consistency models. Varying the frequency of reconciliation in fluid replication allows applications to adjust the "strength" of the last-writer and optimistic models. Bounding staleness in TACT has similar effects. However, as motivated earlier, staleness alone does not fully capture application-specific consistency requirements.

Fox and Brewer [11] argue that strong consistency and one-copy availability cannot be achieved simultaneously in the presence of network partitions. In the context of the Inktomi search engine, they show how to trade harvest for yield. Harvest measures the fraction of the data reflected in the response, while yield is the probability of completing a request. In TACT, we concentrate on consistency among service replicas, but a similar "harvest" concept can also be defined using our consistency metrics. For example, bounding numerical error has similar effects as guaranteeing a particular harvest. Finally, Olston and Widom [25] address tunable performance/precision tradeoffs in the context of aggregation queries over numerical database records.

7 Conclusions and Future Work

Traditionally, designers of replicated systems have been forced to choose between strong consistency, with its associated performance overheads, and optimistic consistency, with no guarantees regarding the probability of conflicting writes or stale reads. In this paper, we explore the space in between these two extremes. We present a continuous consistency model where application designers can bound the maximum distance between the local data image and some final consistent state. This space is parameterized by three metrics, *Numerical Error*, *Order Error*, and *Staleness*. We show how TACT, a middleware layer that enforces consistency bounds among replicas, allows applications to dynamically trade consistency for performance based on current service, network, and request characteristics. A performance evaluation of three replicated applications, an airline reservation system, a bulletin board, and a QoS Web service, implemented using TACT demonstrates significant semantic and performance benefits relative to traditional approaches.

We are investigating a number of interesting questions posed by the TACT consistency model. We are currently working on both theoretical and practical issues associated with trading system consistency for availability. Theoretically, is there an upper bound on availability given a consistency level with particular numerical error, order error and staleness? Practically, how close to this upper bound can the TACT prototype provide dynamically tunable consistency and availability? Similarly, can TACT adaptively set application consistency levels in response to changing wide-area network performance characteristics using application-specified targets for minimum performance, availability, and consistency?

Acknowledgments

We would like to thank David Becker for maintaining the experimental infrastructure at Duke. Jay Lepreau and Stephen Clawson were very helpful in providing access to the Utah computing resources available at the University of Utah Network Testbed (supported by NSF grant ANI-00-82493, DARPA/AFRL grant F30602-99-1-0503, and Cisco). Jeff Chase and Doug Terry provided a number of insightful comments on the TACT design. Finally, the comments of David Culler, Robert Grimm, Brian Noble, and the anonymous referees greatly improved the presentation of this paper.

References

[1] Atul Adya, Robert Gruber, Barbara Liskov, and Umesh Maheshwari. Efficient Optimistic Concurrency Control Using Loosely Synchronized Clocks. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, May 1995.

[2] D. Agrawal, A. E. Abbadi, and A. K. Singh. Consistency and Orderability: Semantics-Based Correctness Criteria for Databases. *ACM Transactions on Database Systems*, September 1993.

[3] Rafael Alonso, Daniel Barbara, and Hector Garcia-Molina. Data Caching Issues in an Information Retrieval System. *ACM Transactions on Database Systems*, September 1990.

[4] Phil Bernstein and Nathan Goodman. The Failure and Recovery Problem for Replicated Distributed Databases. *ACM Transactions on Database Systems*, December 1984.

[5] Ken P. Birman. The Proecss Group Appraoch to Reliable Distributed Computing. *Communications of the ACM*, 36(12):36–53, 1993.

[6] Junghoo Cho and Hector Garcia-Molina. Synchronizing a Database to Improve Freshness. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, May 2000.

[7] Brian Coan, Brian Oki, and Elliot 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.

[8] Susan Davidson, Hector Garcia-Molina, and Dale Skeen. Consistency in Partitioned Networks. *Computing Survey*, 17(3), 1985.

[9] W. Keith Edwards, Elizabeth Mynatt, Karin Petersen, Mike Spreitzer, Douglas Terry, and Marvin Theimer. Designing and implementing asynchronous collaborative applications with bayou. In *Proceedings of 10th ACM Symposium on User Interface Software and Technology*, October 1997.

[10] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2068, January 1997.

[11] Armando Fox and Eric Brewer. Harvest, Yield, and Scalable Tolerant Systems. In *Proceedings of HOTOS-VII*, March 1999.

[12] D. K. Gifford. Information Storage in a Decentralized Computer System. Technical Report CSL-81-8, Xerox PARC, 1983.

[13] Richard Golding. *Weak-Consistency Group Communication and Membership*. PhD thesis, University of California, Santa Cruz, December 1992.

[14] R. Guy, J. Heidemann, W. Mak, T. Page Jr., G. Popek, and D. Rothmeier. Implementation of the Ficus Replicated File System. In *Proceedings Summer USENIX Conference*, June 1990.

[15] R. G. Guy, P. Reiher, D. Ratner, M. Gunter, W. Ma, and G. J. Popek. Rumor: Mobile Data Access Through Optimistic Peer-to-Peer Replication. In *Proceedings of the 17th International Conference on Conceptual Modeling (ER'98)*, November 1998.

[16] J. Holliday, R. Steinke, D. Agrawal, and A. El Abbadi. Epidemic Quorums for Managing Replicated Data. In *Proceedings of the 19th IEEE International Performance, Computing, and Communications Conference*, February 2000.

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

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

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

[20] Tei-Wei Kuo and Aloysius K. Mok. Application Semantics and Concurrency Control of Real-Time Data-Intensive Applications. In *Proceedings of the IEEE Real-Time Systems Symposium*, 1992.

[21] Tei-Wei Kuo and Aloysius K. Mok. SSP: A Semantics-Based Protocol for Real-Time Data Access. In *Proceedings of the IEEE Real-Time Systems Symposium*, 1993.

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

[23] Leslie Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, July 1978.

[24] Brian Noble, Ben Fleis, and Minkyong Kim. A Case for Fluid Replication. In *Proceedings of the 1999 Network Storage Symposium (Netstore)*, October 1999.

[25] Chris Olston and Jennifer Widom. Bounded Aggregation: Offering a Precision-Performance Tradeoff in Replicated Systems. Technical report, Computer Science Department, Stanford University, 1999. `http://www-db.stanford.edu/pub/papers/trapp-ag.ps`.

[26] Esther Pacitti, Eric Simon, and Rubens Melo. Improving Data Freshness in Lazy Master Schemes. In *Proceedings of the 18th IEEE International Conference on Distributed Computing Systems*, May 1998.

[27] Vivek S. Pai, Mohit Aron, Gaurav Banga, Michael Svendsen, Peter Druschel, Willy Zwaenepoel, and Erich Nahum. Locality-Aware Request Distribution in Cluster-based Network Servers. In *Eighth International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1998.

[28] Karin Petersen, Mike Spreitzer, Douglas Terry, Marvin Theimer, and Alan Demers. Flexible Update Propagation for Weakly Consistent Replication. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP-16)*, pages 288–301, October 1997.

[29] Calton Pu, Wenwey Hseush, Gail E. Kaiser, Kun-Lung Wu, and Philip S. Yu. Distributed Divergence Control for Epsilon Serializability. In *Proceedings of the International Conference on Distributed Computing Systems*, 1993.

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

[31] Yasushi Saito, Brian Bershad, and Hank 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.

[32] Aman Singla, Umakishore Ramachandran, and Jessica Hodgins. Temporal Notions of Synchronization and Consistency in Beehive. In *Proceedings of the 9th ACM Symposium on Parallel Algorithms and Architectures*, June 1997.

[33] D. Terry, A. Demers, K. Petersen, M. Spreitzer, M. Theimer, and B. Welch. Session Guarantees for Weekly Consistent Replicated Data. In *Proceedings 3rd International Conference on Parallel and Distributed Information System*, September 1994.

[34] Douglas B. Terry, Marvin M. Theimer, Karin Petersen, Alan J. Demers, Mike J. Spreitzer, and Carl H. Hauser. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 172–183, December 1995.

[35] Francisco Torres-Rojas, Mustaque Ahamad, and Michel Raynal. Timed Consistency for Shared Distributed Objects. In *Proceedings of the 18th ACM Symposium on Principle of Distributed Computing*, May 1999.

[36] M. H. Wong and D. Agrawal. Tolerating Bounded Inconsistency for Increasing Concurrency in Database Systems. In *Proceedings of the 11th Symposium on Principles of Database Systems*, June 1992.

[37] Haifeng Yu and Amin Vahdat. Building Replicated Internet Services Using TACT: A Toolkit for Tunable Availability and Consistency Tradeoffs. In *Proceedings of the Second International Workshop on Advanced Issues of E-Commerce and Web-based Information Systems*, June 2000.

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