A Simulation Framework and Evaluation for Optimistically Replicated Filing Environments[✧]

An-I A. Wang Peter L. Reiher Rajive Bagrodia
Computer Science Department
University of California, Los Angeles Los Angeles, CA 90095-1596
{awang, reiher, rajive}@cs.ucla.edu

Abstract

Optimistic replication of data is becoming increasingly popular in distributed systems. However, the performance characteristics of such systems are not well understood. We have developed a simulation to evaluate optimistically replicated file systems in terms of their operating costs (e.g., computing resources) and service quality (e.g., consistency of data perceived by users).

The simulation built-in language and library components ease the specification of various system settings. The traffic generator captures the inverse relationship between sharing and write accesses, and the traffic is based on a three-month long trace collected at Locus Computing Corporation.

Our result shows that the widely used conflict rate metric is misleading: Both extremes of data propagation frequencies result in near zero conflict rates. Also, conflict resolutions can generate many intermediate versions of data that are in conflicts, or meta-conflicts. We have proposed alternative service quality metrics and ways to reduce meta-conflicts.

Our study further demonstrates that the effectiveness of a data propagation topology depends on the propagation distance between replicas as well as the number of propagation processes that can occur in parallel. Optimistically replicated filing scales well; further exploiting user sharing patterns can achieve further economies. Per user adjustment of data propagation frequencies based on the work cycle hurts the service quality of overall system. Finally, we have observed the tradeoffs between data propagation rate and service quality, and traffic loads on shared replicated data.

1 Introduction

Optimistic data replication has become an increasingly important technology. It allows parallelism in making airline reservations, use of an ATM in the face of network outages, and simultaneous cooperative accesses to shared data on laptops disconnected from networks. With its high data availability, resiliency to network outages, and cooperative data sharing, optimistic replication has become an important tool of mobile computing.

Early research on optimistic replication was largely performed in the context of file systems. Most studies emphasized proving the correctness and practicality of the method. The Locus Operating System [Popek et al., 1985] produced the key research necessary to make an optimistically replicated file system practical, including the

[✧] This work was sponsored by the Defense Advanced Research Project Agency under contract DATB63-94-C-0080. The authors can be reached at the Computer Science Department, UCLA, Los Angeles, CA 90095, or by email to {awang, reiher, rajive}@cs.ucla.edu.

concept of version vectors [Parker et al., 1983]. The first practical optimistically replicated file system that achieved general popularity was Coda [Satyanarayanan 1989]. This system utilized the very limited replication capabilities in the Andrew File System [Kazar 1989] to offer not only optimistic replication, but also support for operation with network disconnection. Coda, in addition, produced seminal research on client-server optimistic replication, one of the two major approaches to optimistic replication. Years of actual use by a large number of users have validated Coda's early results.

Later systems, such as Ficus [Guy et al., 1990] offered further evidence of the practicality of optimistic replication. Coda, Ficus, and other projects demonstrated that concurrent modifications could be correctly resolved without data loss [Reiher et al., 1994; Kumar et al., 1995]. By deploying systems among substantial numbers of real users, they showed also that optimistic replication lives in reality rather than just in theory [Kistler et al., 1992; Noble et al., 1994; Heidemann et al., 1995]. Oracle 7 [Daniels et al., 1994], Bayou [Terry et al., 1995], Ingres, Lotus Notes [Kawell et al., 1992], Microsoft Brief Case, Concurrent Version System [Berliner 1998], and LDAP [Weiser et al. 1999] are popular deployments of optimistic replication concept [Kung 1994].

Despite of its wide deployments and growing complexity, the performance and scaling properties of optimistic replication are not well understood. The lack of performance results, we believe, is attributable to high overheads and difficulties in obtaining those results.

Why is evaluation so hard? Optimistic replication works well because of certain assumptions of user behaviors (Section 2.1). Evaluating its performance involves capturing human-to-human interaction (how data are shared) as well as human-to-machine behaviors (how data are accessed by each user). The system dynamic and its large parameter space render pure analytical and empirical studies prohibitively expensive. Simulation is an alternative, but designing, implementing, and validating a simulation for arbitrary network interconnections and system attributes impose nontrivial overheads.

We chose to design and implement a simulation framework as the primary approach, to investigate various hypothetical settings in the optimistically replicated file system context. We also incorporated measurement and analytical approaches when appropriate. Our simulation reveals that the widely used conflict rate metric is ambivalent in expressing the service quality (e.g., consistency of data perceived by the end users): a conflict measurement number can simultaneously imply both high and low service qualities. We analyzed the underlying causes of this anomalous behavior and proposed alternative metrics to evaluate the file system service quality of optimistic replication. Also, we found that conflict resolution may generate intermediate versions of data (e.g., meta conflicts) that further the number of conflicting data versions. This finding leads to recommendations of delaying the conflict resolution until the object in conflict is referenced, or restricting the conflict resolution to update-contributing sites or other heuristically chosen sites.

Remaining results show that the effectiveness of a data propagation topology depends on the propagation distance between replicas as well as the number of propagation processes that can occur in parallel. Meta-conflicts can contribute significantly to performance metrics and vary significantly among replication topologies. Adjusting data propagation frequencies based on the work cycle of each end-user does not achieve the highest service quality at the aggregate system level. In addition, the operational costs predicted by the simulation study suggest that

optimistically replicated filing scales well, and further economies and service qualities could be realized by further exploiting user sharing patterns. Finally, we examined the tradeoffs of data propagation rate and service quality, and the effects of varying traffic loads on shared replicated data.

Section 2 explains how optimistic replication works. Section 3 details the simulation framework design. Section 4 describes the experiments we ran with the simulation. Section 5 presents the resulting knowledge gained about optimistically replicated file systems. Section 6 relates our work to other performance analyses of optimistically replicated file systems, and Section 7 concludes.

2 Background

This section describes the motivation of optimistic replication, its possible styles of implementation, configuration space of its algorithms, the choices we made, and the applicability of our work. We will also define the terminology used for the remaining paper.

2.1 Motivation for Optimistic Replication

In large-scale distributed systems, replication is a good technique for providing high availability to allow data sharing across machine boundaries, because each machine can store a local copy of data. Two replication approaches are possible—the conservative approach and the optimistic approach.

Conservative replication attempts to provide the same semantics as having a single copy of the data. This approach is particularly important for situations where inconsistent views of data can be little tolerated, and it assumes high availability of the network connection. Some examples are military command-and-control systems, air traffic control systems, and systems for conducting stock exchanges, where valid decisions crucially rely on the consistency and correctness of the data being presented to the users. Such replication schemes typically use tokens, voting, or primary site methods to ensure that no concurrent updates occur [Johnson et al., 1990].

Optimistic replication is based on the observation that many areas of computing do not demand immediate data consistency. For example, a slightly aged library database system can still deliver meaningful service in most cases. Therefore, immediate propagation of new updates to all replicas is often not vital. Modern computing trends of cooperative data sharing and mobile computing further identify optimistic replication as an alternative replication scheme, when availability of data is more important than the immediate consistent view of data. For banking applications, reservation systems, and mobile file accesses operation in the face of network failures is vital.

The practicality of optimistic replication also depends on certain user behaviors: Most files have a single writer, particularly over a short time period. Thus, allowing writes to any data replica rarely results in concurrent updates. For many applications, the majority of concurrent data modifications can be performed in parallel. With proper handling, the modifications can be later merged automatically or manually without data loss. Directories are an important example. Independent file creations can be applied to two replicas of a directory without causing problems, because the differing directory replicas can be easily merged into a single directory.

2.2 Optimistic Replication Algorithms

Optimistic replication permits copies of data to become temporarily inconsistent, so the algorithm must be able to bring differing replicas into synchronization. Sometimes the updates can be propagated to other replicas immediately, but the advantage of allowing updates to a replica isolated from networks carries with it the disadvantages of being unable to propagate those updates as they occur. Therefore, the replication system must also be able to propagate them later. Typically, a **reconciliation process** brings replicas into synchronization by comparing replicas and applying the updates at some convenient time (e.g., when portable computers are temporarily connected to the network).

Reconciliation involves finding recent updates for the sites involved in the process, typically using either logging [Satyanarayanan 1989] or scanning [Guy et al., 1990; Ratner et al., 1996]; exchanging the updates; and detecting updates that are in **conflicts**. Since it is difficult to define conflicts precisely and concisely, the exact algorithm used in this paper to detect conflicts will be described in Section 3.5. Typically, optimistic replication systems use application-specific libraries to resolve the majority of inconsistencies resulting from conflicts automatically. Conflicts that are not automatically resolved require user intervention [Popek et al., 1990; Reiher et al., 1994; Kumar et al., 1995].

Consider the airline reservation example. Reservation records are optimistically replicated at various geographical locations. Multiple people can simultaneously make reservations at various locations, and each location updates the local replica of reservation record. When reconciliation among these replicas occurs, the conflict will be detected and automatically resolved by first merging the non-duplicate seat reservations. For duplicate seat reservations and overbooking, the system can automatically adjust the seating using predefined rules or require manual adjustments. Either the system or users of the reservation system would provide a **resolver** program that could examine the conflicting updates or versions of the reservation record and produce the proper merged version.

As for ATM machines, although optimistic replication provides convenient account accesses that can survive network outages, it also exposes the concern for an account to be intentionally over-withdrawn by withdrawing across the network partition boundaries. At the typical rate of daily reconciliation, an account could be over-withdrawn many times before the true account balance is realized. Banks compromises accessibility of accounts and risks of fraudulent withdraws by imposing upper daily limits on cash dispense and by providing surveillance systems to catch the illegal acts.

2.3 Parameters of Optimistic Replication

Optimistic replication introduces many parameters into system configuration. The quality of service (**QoS**) of such a system might be sensitive to user access patterns. Optimistically replicated systems might have scaling constraints based on the number of data copies, or **replication factor**. The rate, the direction, and the path of data synchronization might affect the degree of data consistency. Other factors also contribute to final performance results, such as the granularity of data being replicated and imposed constraints by the optimistic algorithm.

As noted earlier, all optimistic systems must be able to apply updates to replicas after the update has occurred, sometimes long afterwards. The **reconciliation interval**, therefore, is an important factor in replication system performance. Depending on user behavior, frequent reconciliation could mean either closer synchronization of replicas or just extra overhead that does not improve the QoS. In most systems, reconciliation is performed periodically [Ratner et al., 1996], on demand [Satyanarayanan 1989], or immediately [Guy et al., 1993] to bring a local replica into synchronization with a remote one.

In the case of three or more replicas, the reconciliation process could be initiated with different choices of partners. The initiating replica selects the target replica based on a specified **reconciliation topology**. Some optimistic replication systems use a client-server model, in which one or several of the replicas are servers, and the rest are clients. Client sites may only propagate their updates to servers. Often, updates can only be propagated from a given client to a particular server. Servers may propagate their updates more freely. Alternatively, all replicas can be peers. In this kind of replication system, any replica can trade updates with any other replica. Conceptually, any client-server model can be emulated as a peer model with the appropriate reconciliation topology. For example, a system with a single server machine and multiple clients can be emulated by a peer system using a star reconciliation topology, with the server at the center and the clients attached to the server. From an implementation standpoint, client-server and peer-model replication systems have important differences, but a simulation needs to encapsulate only the ones that contribute to performance metrics.

Reconciliation can be a uni-directional process, in which one replica receives updates from its partner, or it can be a bi-directional process, in which both replicas trade updates. We refer to this choice of direction of data propagation as the **reconciliation protocol**. One-way reconciliation shortens the time spent in a single reconciliation, but two-way reconciliation may provide more globally efficient propagation of updates, in terms of both time spent by all parties and the speed with which updates propagate. A one-way reconciliation process can be performed either as a push (one site pushes its updates to the other), or a pull (one site pulls updates from the other). Pushing gives control to the side sending them; pulling gives the control over data transfer to the side receiving the updates.

Replication granularity, or the level of replication control, introduces many decision points—the granularity which one can select which data to be or not be held by a replica, the granularity of the data items addressed by an update, granularity of the data items for which conflicts are detected, and the granularity for which the data set is defined or administered. In the context of the file system, possible granularities include the individual file, the directory, or sub-trees of a file hierarchy (typically called a **volume**). Fine granularities offer flexible control at the cost of high meta-data storage overheads for each replicated granule. Coarse granularities lower these overheads, but offer less flexibility.

2.4 Applicability of Our Work

Although our simulation has the flexibility to accommodate many system settings, the full exploration of system implementations and configurations would require a number of experiments proportional to all combinations of various settings, and it is impractical for us to explore all possibilities due to the length of computation and available

computing resources. We chose to validate our simulation against an early version of the Rumor Replicated File System (RRFS) [Reiher et al., 1996], a user-level version of Ficus. RRFS is a state-based system, and the reconciliation algorithm used by RRFS will be described in Section 3.5. Because of its nonkernel-intrusive nature, RRFS uses scanning to find recent updates. We expanded our study to include experiments that remove scanning related overheads, and resulting QoS values are within 25 percent of the original values.

For both RRFS and the simulation, we configured the reconciliation processes to be periodic. However, if we decrease the reconciliation interval to an arbitrarily small interval, the effects on QoS should approximate immediate data propagation, and the QoS for demand-driven data propagation should be between these two aggressive and lazy options. Each reconciliation process, like most of replication systems, involves only two replicas although multi-way reconciliation strategies are possible. We examined both one-way and two-way reconciliation protocols. For the one-way reconciliation protocol, we arbitrarily chose the initiating replica to pull the updates.

To simplify the problem domain, we assume that all files are replicated under a single volume, and the volume is fully replicated across the machines. Each machine contains only one replica of the volume; only one local user accesses each replica. In addition, each file can be individually accessed, and conflicts are detected and resolved at the granularity of a single file, although a reconciliation process would process the entire volume. Because reconciliation at the volume granularity is a heavy-duty process, typical real installations of optimistic replication systems ensure that at most one reconciliation process is in progress at any point at a given machine. This **reconciliation locking** constraint prevents the reconciliation traffic from overloading the CPU. Practically, this constraint implies that if a site is participating in a reconciliation process, the site will deny reconciliation requests from other sites. All denied reconciliation requests are aborted. Remote accesses and failures are not modeled for this study, but will be an area of interest for future study.

3 Simulation Architecture

The simulation framework is designed to scale and support heterogeneous configurations. Figure 3.1 illustrates the top-level view of the simulation framework. To avoid individual or redundant specifications of each simulated component and traffic generator, we designed a hierarchical input language and a dedicated front end to allow groupings and compositions of library components. To provide parallel execution of the simulation, we delegated the centralized front end and the simulation driver only the responsibility of initializing servers and their interconnections for network communications; no centralized coordination is required after the simulation is launched. This simulation supports multiple servers, one for each simulated machine.

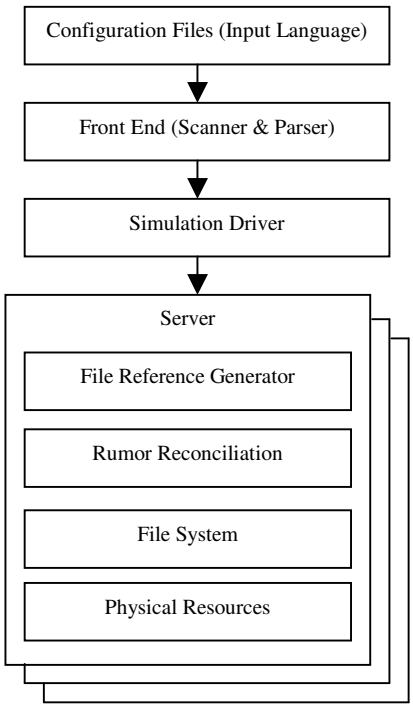

Figure 3.1: Simulation structural diagram. The front end parses the specification for each server. The simulation driver sets up the server interconnections and the replication parameters. Each server contains a file reference generator to model the user behavior on that server. The Rumor reconciliation module simulates the core algorithms of the replication system being studied. The file system module records the data modifications, and the physical resource module simulates the delay characteristics of the various hardware components and the network interconnections.

We chose Maisie [Bagrodia et al., 1994] as our primary implementation language for its strengths in several areas. First, Maisie is a C-based language; the learning overhead for us is relatively low compared to other non-C-based simulation languages. Second, the language-level abstraction of the underlying message-passing interface simplifies implementation. Third, Maisie supports both sequential and parallel simulation with high transparency; the simulation model in this paper has the future option of parallel execution. Although our model permits parallel execution, this paper does not address the design challenges that pertain to parallel simulation; rather, it concentrates on the application of the simulator in evaluating the performance of optimistically replicated file systems.

3.1 Simulation Front End

The simulation front end (scanner and parser) sets up the experimental scenario and initializes various system parameters such as network interconnections for servers, reconciliation topology, and user access patterns for the file reference generator. The chosen languages of the front-end implementation are Lex and Yacc, which grammar-encode the syntax and provide an extensible and flexible interface for future development.

```
# sim.rsim
# lines started with "#" are comments
RSim1.0

include "server.rsim" serverType
include "cluster.rsim" clusterType
include "channel.rsim" channelType

# a cluster of two pentium machines
cluster roomA {
    server inst serverType:pentium aldrin
    server inst serverType:pentium bean
}

# a cluster of two sparc machines with two sub-clusters
# connected by an Ethernet
cluster ficus channel inst channelType:ethernet10Mb ethernet {
    server inst serverType:sparc5 dover
    cluster inst roomA roomA
    server inst serverType:sparc5 thetford
    cluster inst clusterType:roomB roomB
}

# simulate the ficus cluster with a random seed number
# for a duration of 576 hours.
sim ficus {
    seed 19238483
    sim 576 hr
}
```

Figure 3.2: Simulation input specification. This piece of code shows the flavor of our input language. Starting from the top, lines starting with **#** denote comments. **RSim1.0** is the version number for our simulation, which helps the parser to identify the version of syntax for our input language. **cluster** is one of the predefined interchangeable components, or modules, which closely map to the physical world (e.g., disk and CPU). To include modules from other files, specify **include** followed by the file name and module name. The input language supports two levels of scopes to avoid name collisions of modules, one at the file level and the other one between curly braces. The **sim** statement indicates the module to be simulated, in this case, **cluster** *ficus*. Global configurations (e.g., seed number and simulation duration) are specified in the **sim** statement. The **cluster** *ficus* is connected by a channel instance (**channel inst**) of Ethernet (*ethernet10Mb*). The cluster *ficus* contains two server instances (**server inst**) of Sparc 5 (*sparc5*) from *"server.rsim"* specified at **include** statements, one cluster instance of *roomA*, and one cluster instance of *roomB* from *"cluster.rsim."* The cluster *roomA* contains two server instances of *pentium*. Since cluster *roomA* does not specify a channel, it uses the channel specified at cluster *ficus*.

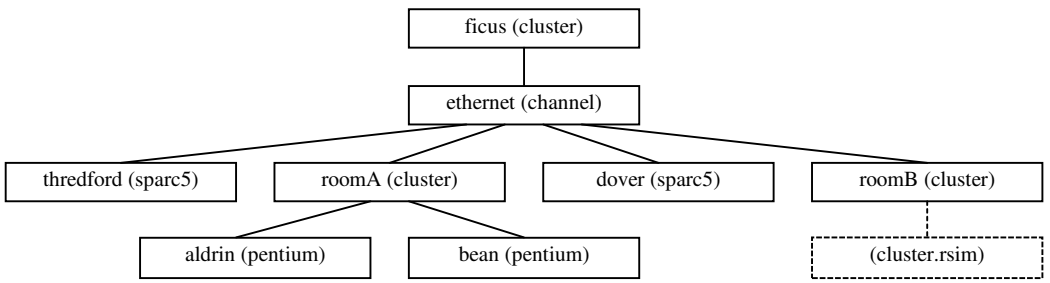

Figure 3.3: Hierarchical model construction. This diagram depicts the model constructed by the input specifications in Figure 3.2. The solid lines and boxes are specified in this input file. The box in the dotted line is specified from the included files.

The front end provides a number of features to facilitate the specification of a large, heterogeneous interconnection of servers: nested file inclusion, lexical scoping, and version numbering. Figure 3.2 illustrates the use of our built-in language to specify a two-level construction of an Ethernet network in which the construction is comprised of two different types of machines. Figure 3.3 pictorially shows the corresponding configuration.

A complete description of the input language is beyond the scope of this paper; we mention a few key features. For example, hierarchical abstraction of library components permits rapid composition of a complex model. Input file A contains a library of disk modules, and input file B contains a library of CPU modules. Input file C contains a variety of server modules, which are composed of various combinations of disk and CPU modules specified in A and B. Implementing this concept requires the support of nested file inclusion, allowing separate file storage for each group of specifications. To avoid name collisions of modules, we introduced various levels of scopes to detect naming collisions. Commenting enhances the readability and maintainability of the specifications. Inheritance of top-level parameters allows the specification of global parameters such as simulation duration.

3.2 Traffic Generator

Our simulation input traffic is based on a three-month data trace of file access patterns, collected at Locus Computing Corporation [Kuenning et al., 1994]. This trace consists of the actual activities of software developers, working on DOS-based software. The traced system did not use replication, but had large numbers of users and machines connected by a local area network, performing both local and remote file accesses. To a certain extent, the results presented are dependent on the characteristics of this workload. However, such a dependency is inescapable for any choice of workload, and the chosen workload is realistic and representative of a software development environment workload. Future work may examine other important file access workloads.

Extensive pilot runs show that the approaches adopted by common traffic generators—capturing the temporal and spatial localities—are insufficient to evaluate optimistically replicated file systems. In such systems, the consistency and freshness of data perceived by end-users highly depend on how data are shared, as well as how data are accessed. In addition, nonuniform loads of user activities can contribute to the overall QoS measures.

For example, our trace data reveal that as the degree of file sharing[1] increases, the percentage of write accesses decreases. Thus, our experiments adopt a fine-grain decomposition of file sharing and operation patterns as shown in Figure 3.4. Without this fine-grained characterization of file sharing, the percent of write access is significantly overestimated for files at a high sharing factor, and the growth rate of conflicts is theoretically exponential, with maximum values constrained by available physical resources [Heidemann et al., 1995].

<table>
<tr><td colspan="6">File access</td></tr>
<tr><td rowspan="3">Read-only access</td><td colspan="5">Read-write access</td></tr>
<tr><td colspan="2">Nonshared access</td><td colspan="3">Shared access</td></tr>
<tr><td colspan="2"></td><td colspan="2">2-way sharing</td><td>2+way sharing</td></tr>
<tr><td>Read access</td><td>Write access</td><td>Read access</td><td>Write access</td><td>(Further splitting)</td></tr>
</table>

Figure 3.4: Fine-grained decomposition of file accesses. File references fed to the model are divided into read-only accesses and read-write accesses. Read-write file accesses are partitioned into accesses to shared and nonshared files. Shared files are further classified as files shared between exactly two users and files shared among more than two users (e.g., two-way shared and three-plus-way shared).

File references fed to the model are divided into read-only accesses and read-write accesses. Read-only accesses do not affect most of the results in the simulation, except by contributing to the total number of read accesses. Read-write file accesses are partitioned into accesses to shared and nonshared files. Assuming each user always accesses

a particular replica, updates to nonshared files will never create conflicts and always contain the most up-to-date data, because only a single replica is used for access to such files. Shared files are further classified as files shared between exactly two users and files shared among more than two users (e.g., two-way shared and three-plus-way shared); maintaining only two writable versions of a data item is much simpler than maintaining three or more.

To capture daily work cycles, we model the trace traffic at hourly intervals with an exponential distribution, with mean inter-arrival time equal to the mean inter-arrival time of the corresponding daily hour in the trace. Average hourly traffic can be obtained by averaging the mean inter-arrival time of a certain hour of the day across the entire trace. For hourly boundary conditions, if the randomly chosen arrival time causes the next reference to arrive within the current hourly interval, we simply proceed. If the next reference arrives beyond the current hourly interval, we move to the next hour. (No more references will arrive for the remaining hour.) This boundary handling approach is not statistically accurate, but the inaccuracy mostly affects the hours with light traffic, and it introduces roughly two percent of error.

Our simulation also captures the nonuniform loads generated by users, by first finding the aggregate traffic volume, and then distributing the volume by the expected percentage contribution of each simulated user. If the number of simulated users is different from the number in the trace (in our case, always less than the actual number in the trace), we linearly scale down the aggregate traffic volume so the per-user volume remains the same. However, the percentage contribution for each user is adjusted to approximate the original skewing of user loads. Figure 3.5 compares both the original load characteristics and the simulated load characteristics.

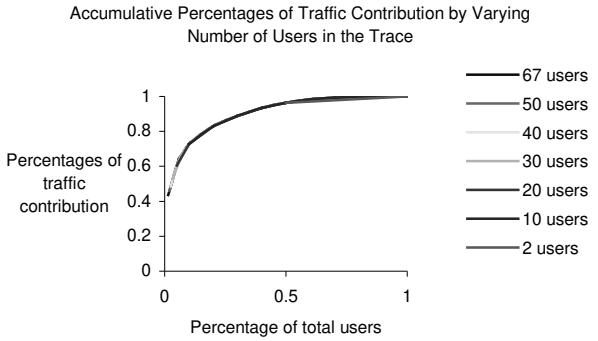

Figure 3.5: Approximating nonuniform traffic loads of 67 users with varying number of users. For each user population setting, the users are first sorted by the traffic contributions in decreasing order. Each data point represents the percentage traffic contribution by the top x percentage of traffic contribution users.

One concern of our traffic generator is the fidelity of simulating interleaving accesses. The following example illustrates the concern. In the first scenario, user *A* makes 50 updates followed by user *B*'s 50 updates. In the second one, user *A* and *B* update in an interleaving fashion. Both examples result in similar aggregate statistics, but the second scenario can potentially generate more conflicts than the first scenario.

However, we chose to overlook the case of interleaving accesses for two reasons. First, conflicts are detected at reconciliation times; the exact ordering of updates between reconciliation would not affect the QoS

[1] By sharing we assume inter-user sharing as opposed to intra-user sharing (same user, different machines.)

significantly. Second, the lack of locality and random updates tend to produce worse performance numbers (from the illustrated example of interleaving updates), giving a more conservative picture of system performance.

3.3 Physical Resource Module

The physical resource module models the physical delays in the system. Sub-modules simulate the primary machine components, including a CPU model, disk system model, network interface model, and network model. The level of detail for each sub-module is determined primarily by its significance to the reconciliation process. For instance, consider the network model. Our experiments assume that machines in a cluster are connected using an Ethernet, a popular network medium. Pilot runs indicate that the typical bandwidth of an Ethernet (10Mb/sec) far exceeds the requirements of the reconciliation process. Consequently, we used an analytical model to abstract the Ethernet delay characteristics. The validation results indicate that this model is sufficiently accurate when the network is lightly loaded. The disk module simulates the seek delay, rotational delay, and data transfer delay. Although the disk is not the bottleneck, this level of detail is required to capture the fixed overhead costs of the file system scans performed by the RRFS reconciliation process.

The CPU appears to be the current bottleneck in the reconciliation process. However, a detailed simulation of the CPU is very costly. For the purpose of the application-level investigation, we chose to compose an analytical model of CPU utilization via direct measurement on the operational RRFS. The measurement involves instrumentation by placing timing system calls in the source code. Certain results (e.g., running time of the system) are thus contingent on the versions of software and hardware we chose. For this paper, we measured RRFS running on Dell Latitude XP 486, 100 MHz laptops with 12 Mbytes of memory and a Linux 1.2.x operating system. The resulting analytical model is a function of system attributes—the number of files and directories, the volume size, the number of replicas, the number of updates, and the number of conflicts. The validation of the analytical model is described in Section 3.6.

3.4 File-System Module

The file-system module models a file system that supports the common file operations (i.e., open, read, write, create, directory create, and directory delete) in the context of a specific file system (e.g., UNIX[2]). The time for each operation is estimated mainly by using the disk model. The CPU overheads for those operations are negligible compared to the disk access time. The caching effects are not modeled because our measurements suggest that caching does not work well with the measured system for two plausible reasons. First, because RRFS is nonkernel invasive, detecting data modifications requires linear scanning of the replicated files. The scanning process has low locality. Second, the replicated volume size in our experiment (~220 Mbytes) is much larger than the memory size (12 Mbytes), and the reconciliation process effectively flushes the cache during the file-system scanning phase of the algorithm.

One design decision of the simulated file system is simplification from a hierarchical name space to a flat name space. Because our file reference generation is essentially probabilistic, and we currently do not distinguish

the file reference patterns among files at different levels of the file hierarchy, if some high-level directory nodes are deleted, the results of the remaining simulation can be hard to interpret. Because our trace analyses indicate that sharing patterns dictate the performance metrics for optimistically replicated file systems, a flat name space is sufficient. Both the disk and the CPU modules are adjusted to reflect this simplification. The resulting file access time in a flat name space is the weighted average of file and directory access times in a hierarchical name space, and the weights are based on the relative numbers of references to directories and files in the trace data. The model can be extended to support a hierarchical name space, if necessary.

Both file and directory operations are mapped to read and write operations. This simplification implies that all conflicts are caused by concurrent updates. (In a real system, other causes of conflicts are possible—file deletion, file creation, and file renaming.) The replication-specific attributes of the file system are currently modeled after RRFS. For example, the replication storage overhead is empirically measured from the operational RRFS.

3.5 Reconciliation Simulation

Finally, the reconciliation module simulates the reconciliation process of RRFS. A reconciliation process is periodically initiated at each simulated server based on the interval specified for the experiment. For R replicas, each file of replica i carries a version vector $V_i = \{v_{i1} \ldots v_{iR}\}$, where each element is initialized to zero. The following steps are involved in simulating a one-way pulling reconciliation protocol by using the appropriate modules in the simulator:

1. Scan the local replica i using file system and disk modules. If the timestamp of a file is advanced from the timestamp kept for RRFS at replica i, record the new timestamp and increment v_{ii} by one.
2. Locate a remote replica. If the remote replica is busy, retry or abort the current request depending on the topology specification.
3. Scan the remote replica j using file system and disk modules. If the timestamp of a file is advanced from the timestamp kept for RRFS at replica j, record the new timestamp and increment v_{jj} by one.
4. Obtain file states from the remote site through the network module.
5. For each file, compare local states to remote states using the CPU module.

   For all k, if $v_{ik} \geq v_{jk}$, V_i dominates V_j.

   For all k, if $v_{ik} \leq v_{jk}$, V_j dominates V_i.

   If V_i dominates V_j, and V_j dominates V_i, two files are identical.

   If neither V_i dominates V_j, nor V_j dominates V_i, two files are in conflict.
6. Transfer the remote files that are dominant and in conflict with local files.
7. For each file dominated by the remote replica, use file system and disk modules to overwrite the content of the local file. V_i of the file is set to V_j.
8. For each file in conflict, resolve the conflicts using file system, disk, and CPU modules.

   If the local file is chosen to dominate the remote version, perform the following steps:

[2] UNIX is a registered trademark of AT&T.

For all k not equal to i, v_{ik} is set to the larger number between v_{ik} and v_{jk}.

Merge the files properly; record the new local timestamp; and increment v_{ii} by one.

If the remote file is chosen to dominate the local version, perform the following steps:

For all k not equal to j, v_{ik} is set to the larger number between v_{ik} and v_{jk}.

Merge the files properly; record the new local timestamp; and increment v_{ij} by one.

This protocol can be extended to the two-way reconciliation protocol with minor variations. The two-way reconciliation processes in this paper repeat steps 4 through 8 in the reverse direction.

3.6 Validation

More than 400 physical system runs were conducted to validate the reconciliation time as a function of the following system attributes, and interacting terms:

- Number of files and directories
- Volume size in bytes
- Number of replicas
- Number of updates
- Number of conflicts

Each of the preceding attributes was initially validated independently. For each attribute, the predictions made by the simulation model were compared with measurements on the operational RRFS, and the results were used to refine the model as necessary.

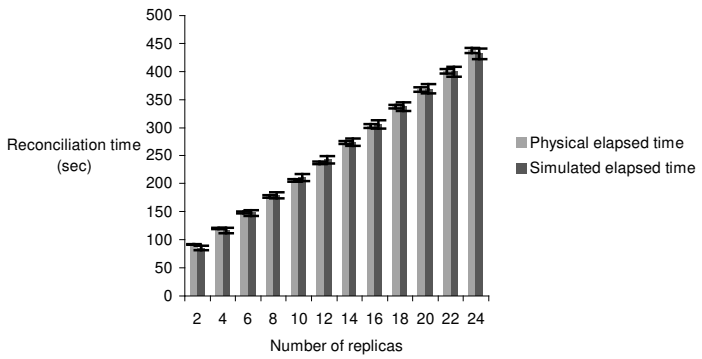

Figure 3.6: Physical vs. simulated reconciliation time. The reconciliation times of the physical system closely match the regression model.

	File	Directory	Replica	Volume size	Updates	Conflicts
R^2	0.9958	0.9991	0.9995	0.9727	0.9963	0.9809

Table 3.1: Coefficient of determination for various linear regressions (fraction of variance explained by the linear regression).

Figure 3.6 compares the physically measured reconciliation time against the simulated reconciliation time as a function of replication factor. Table 3.1 presents the quality of linear regressions in each of the preceding attributes. Subsequently, the experiments were extended to incorporate multiple model attributes simultaneously to determine

the degree of independence among the selected attributes. The model is refined further by again comparing the model predictions and experimental measurements. The model was deemed sufficiently accurate when the predicted reconciliation completion time was within five percent of the actual completion time on our selected hardware and operating system.

4 Experiments

Through our experiments, we seek to understand the effects of various system parameters and to identify opportunities for improving system performance, in terms of both the cost and the QoS metrics. Table 4.1 defines various cost and QoS metrics.

Metrics		Definitions
Cost metrics	Reconciliation time	Elapsed time from the moment at which a replica requests reconciliation with a remote replica to the moment at which all files in the local replica are reconciled, excluding aborted sessions[3]
	CPU overhead	Percent CPU utilization per replica due to reconciliation processes, including the aborted sessions
	Transmission volume per replica	Bandwidth required per replica for periodic reconciliation processes
	Aggregate transmission volume	Global bandwidth required for all replicas to execute periodic reconciliation processes
	Storage overhead	Storage required per replica to keep track of the replicated states
QoS metrics	Conflict rate	Number of conflicts per replica per hour
	Stale read rate	Number of stale reads per replica per hour
	Stale write rate	Number of stale writes per replica per hour

Table 4.1: Definitions of various metrics. The cost metrics include the anticipated running time of a successful reconciliation process, CPU consumption for both successful and failed reconciliation processes, the network bandwidth expenditures in transferring data and meta-data related to updates, and file system storage. The QoS metrics include conflicts detected during reconciliation processes and stale (out-of-date) accesses of replicated data.

The reconciliation time metric indicates the anticipated running time of a successful reconciliation process. The CPU overhead captures the CPU consumption for both successful and failed reconciliation processes. Both reconciliation time and CPU overhead depend heavily on the selected hardware, operating system environment, and the number of files being replicated. The transmission metrics characterize the network bandwidth expenditures in transferring data and meta-data related to updates. Because of our intimate knowledge of the RRFS design, we can deduce the disk storage overhead without simulation. We include the analytically deduced disk storage overhead in our results. Conflicts are generated only during reconciliation processes. A stale read or a stale write occurs whenever a user reads or writes an out-of-date replica of a file.

Our investigated variables include user access pattern, scaling, reconciliation frequency, reconciliation protocol, and the logical update propagation topology used by the reconciliation process. More than 2000 pilot runs were conducted over a large parameter space (1- to 168-hour reconciliation interval, 2 to 67 replicas[4]) and various granularities of parameter space partitionings to locate the regions of high variability for further investigations. The

[3] The time spent for aborted sessions is the elapsed time from the moment a replica requests reconciliation with a remote replica to the moment the replica receives the rejection message from the remote replica. This metric excludes the aborted sessions because it measures the average execution time for a successful reconciliation process.

[4] "Sixty-seven" is the number of users in the trace. We discovered that 50 users are sufficient to characterize the scaling properties for this study.

parameter space and resolutions were reduced when necessary to accelerate the data collection process. Table 4.2 summarizes the simulation parameters.

Parameters		Specifications
Environment configuration	Simulation duration	576 hours
	Calibrated hardware platform	Dell Latitude XP 486 100 Mhz (12 Mbytes of memory)
	Calibrated OS platform	Linux 2.0.x
	Network channel	10Mb Ethernet
	Physical topology	Single-level Ethernet-connected servers
System configuration	File-sharing pattern	Trace data based
	User access skewing function	Distribution mapped from the trace data
	Number of files	10150 files with ~220 MB of data (from the trace)
	File size distribution	Trace data based
Variables for investigation	Reconciliation interval	4 to 44 hours with an 8-hour step for reconciliation processes calibrated for RRFS
		0.5 to 4 hours with a 0.5-hour step and 4 to 44 hours with an 8-hour step, for zero reconciliation time processes
	Reconciliation topology	Ring, star, tree, and fully connected topologies
	Reconciliation protocol	One-way pulling, two-way
	Number of replicas	2, 10, 20, 30, 40, 50 replicas
	Percents of file accesses to shared files	13% (from the trace), 27% , 40% , 54%

Table 4.2: Simulation parameters. Environment parameters describes our chosen hardware and software platform. The system parameters are extracted the trace. The variables investigated include user access pattern, scaling, reconciliation frequency, reconciliation protocol, and the logical update propagation topology used by the reconciliation process.

For the actual experiments, the frequency of reconciliation was varied from 4 to 44 hours. The minimum reconciliation interval is based on the maximum reconciliation time (2.6 hours) in the selected range of parameters. It is not practical to set a reconciliation interval shorter than the time required to complete a reconciliation process, because reconciliation requests made before the completion of the current reconciliation are aborted. However, we did conduct experiments with zero reconciliation time to see the effects of removing implementation-specific overheads. In addition to the 4- to 44-hour reconciliation intervals, we also included 0.5- to 4-hour reconciliation intervals in these experiments.

At the beginning of a simulation run, each replica randomly initiates the first reconciliation process between time zero and the specified reconciliation interval. The second reconciliation process will be scheduled according to the specified reconciliation interval, plus or minus 50 percent. For example, a specified reconciliation interval of 4 hours is really the average reconciliation interval varying from 2 to 6 hours. This large random noise was inserted mainly to prevent potential live locks, where multiple replicas request and abort reconciliation processes repeatedly. However, it also seems more realistic than purely lock-stepped intervals.

The four selected topologies are ring, star, tree, and fully connected topologies. The ring topology can be viewed as a linear tree with connected ends, and the star topology as a tree with minimal depth. The chosen tree topology is a 4-ary tree, which balances the cost-QoS characteristics of the distributed ring and centralized star, assuming all metrics are equally important to the end users. In a fully connected topology, each user can reconcile with any other user. For all topologies, a user reconciles with other users permitted by the topology in a round-robin fashion.

The reconciliation protocols used were one-way pulling and two-way protocols. The number of replicas was varied from 2 to 50. A replication factor of 50 is sufficient to characterize the scaling properties for this study.

The percentage of file accesses to the shared files varies from approximately 13% (the observed percentage in the trace) to approximately 54% (four times the original percentages). The ratio of shared file accesses to the number of shared files is unchanged; the total number of file references in the system also remains unchanged.

5 Results

The performance characteristics of the optimistically replicated file system are complex due to the interdependencies of numerous parameters and metrics. Section 5.1 presents our findings on the conflict rate metric and proposes alternative QoS metrics. During the course of conflict rate discussion, we will introduce reconciliation interval and topology parameters; however, we will defer detailed discussions of these factors to Sections 5.2 and 5.3. In Section 5.4, we then interpret RRFS specific numbers in the context of scaling. Finally, Section 5.5 and 5.6 present secondary results on user sharing pattern and reconciliation protocol. The error bars show the 90 percent confidence interval, computed over five sets of experimental data.

5.1 Fragility of Current Standard Metric—Conflict Rate

One of our most surprising discoveries was the fragility of the conflict rate metric, which is a widely used metric to quantify the QoS of optimistic replication systems. Figure 5.1 shows two disturbing observations about conflict rates.

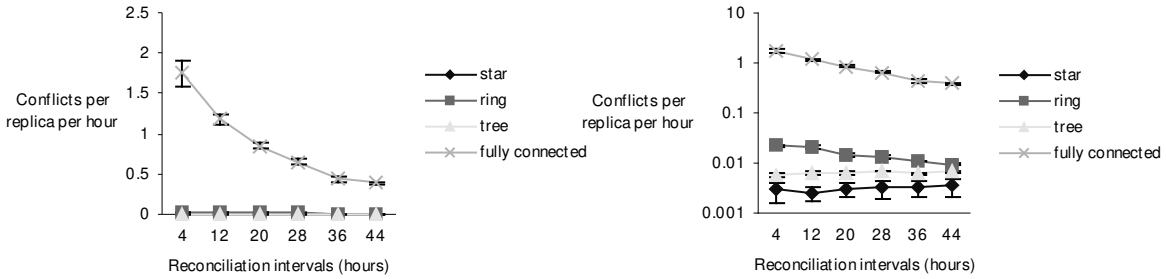

Figure 5.1: Average conflict rate with varying reconciliation intervals and topologies. The values correspond to a simulation environment with 50 replicas with two-way reconciliation protocol. The right graph presents the same data with conflict rate in log scale.

First, as the reconciliation interval lengthens, the conflict rate decreases for ring and fully connected topologies. How can longer data synchronization intervals improve QoS, since they would tend to slow propagation of the data? A bottleneck is not the answer because the star topology should be most susceptible to bottlenecks. Second, the magnitude of differences in conflict rate among the reconciliation topologies, at first glance, is incomprehensible. Take the star topology as the base case with a minimum number of conflicts, in which all the conflicts are resolved centrally. How can topologies account for the vast deviation of QoS? We will answer each question in turn.

5.1.1 Implicit Multiplicands of Conflict Rate

The answer for the first question lies in the implicit multiplication in the conflict rate metric. Conflict rate is a product of two sub-metrics—number of conflicts per reconciliation and number of reconciliations during the entire simulation. If one of sub-metrics approaches zero, and the other sub-metric does not have a compensating growth, conflict rate approaches zero. For example, because detecting and recording conflicts occur only during a reconciliation process, at one extreme we can reduce the number of conflicts to zero by never reconciling, resulting in no updates ever being propagated anywhere. Although it is a real phenomenon that requires occasional user intervention, the conflict rate alone cannot fully capture the QoS of optimistic replication.

To formulate a conceptual model, we hypothesized a simple model of two users updating a single object in a rigid interleaved fashion, each at a time interval of 16 steps (Figure 5.2). The simplified model lasts for 128 time steps. We varied the reconciliation interval from 1 to 128 steps to plot the conflict rate and its two implicit multiplicands.

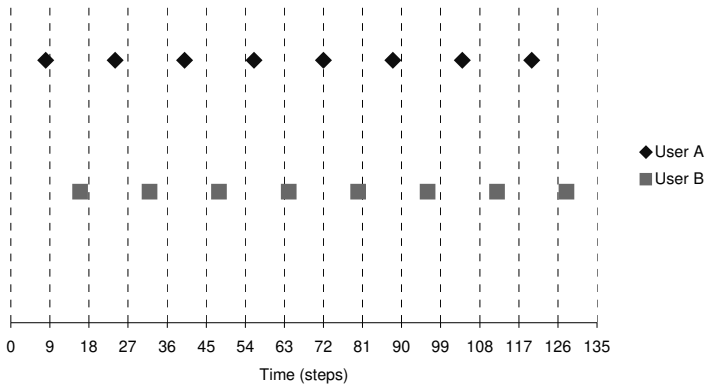

Figure 5.2: Simplified conceptual model. Users A and B update a single object in an interleaved fashion. The update interval is 16 time steps, and the model duration is 128 time steps. In this particular example, the dotted lines indicate the reconciliation interval of 9 time steps. A conflict is detected between time 72 and 81.

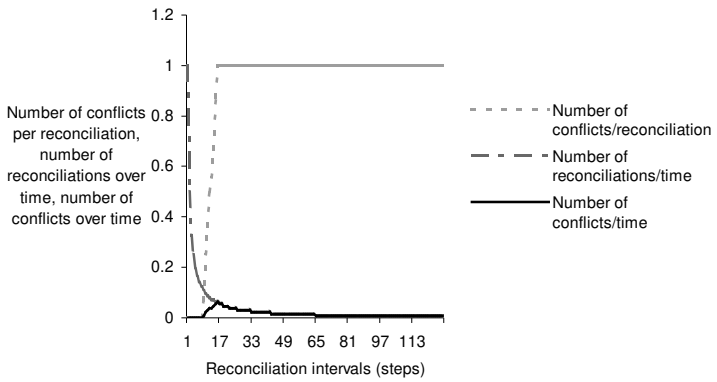

Figure 5.3: Conflict rate and its two implicit multiplicands. The values correspond to the simplified model described in Figure 5.2. As the reconciliation interval increases, the conflict rate first increases due to the increasing number of conflicts per reconciliation, and then decreases due to the reduced number of reconciliation processes.

Figure 5.3 shows the resulting conflict rate and its multiplicands. The number of conflicts per reconciliation starts to increase at a reconciliation interval of 9 time steps and reaches the maximum at an interval of 16. The maximum number of conflicts in a given reconciliation process is the total number of shared objects, in this case, one. The number of reconciliation processes behaves similarly to the inverse function of *1/X*. The product of both curves results in the popular conflict rate metric. As the reconciliation interval increases, the conflict rate first increases due to the increasing number of conflicts per reconciliation, and then decreases due to the reduced number of reconciliation processes. The peak of the conflict rate can be located by solving the equations formed by initial conditions and equating zero to the sum of the partial derivatives of its multiplicands. The increasing conflict rate trends of the star and tree topologies and the decreasing trends of ring and fully connected topologies are really two sides of the peak.

To verify this theory extracted from this simple two-user model, we extended our experiments to reduce reconciliation time to zero, so that we can explore the reconciliation intervals below 4-hour region.

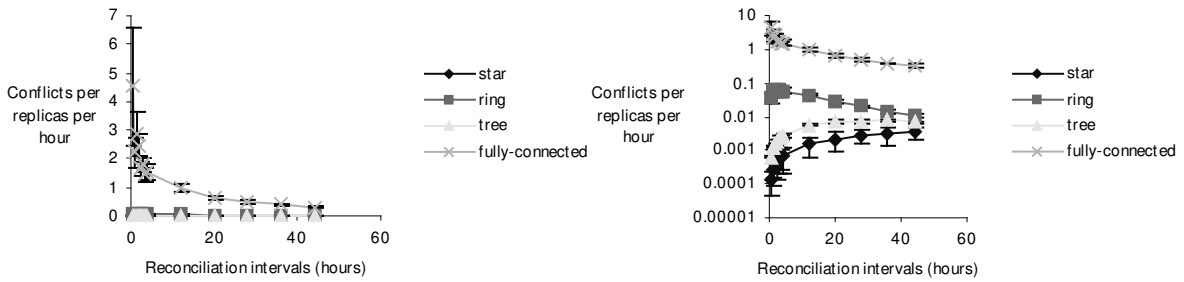

Figure 5.4: Average conflict rate with varying reconciliation intervals and topologies with zero-time-delay reconciliation processes. The values correspond to simulation environment with 50 replicas with two-way reconciliation protocol. Below the 4-hour reconciliation interval, a reconciliation interval varies at 0.5-hour interval; above the 4-hour reconciliation interval, 8-hour interval. The minimum reconciliation interval is half of an hour.

Figure 5.4 shows the conflict rate with 0.5- to 44-hour reconciliation intervals and zero-time-delay reconciliation processes. From the linear scale graph on the left, we fail to capture the rising edge of the conflict rate curve for the fully connected topology; shorter reconciliation intervals are needed to capture the rising edge. However, from the log-scale graph on the right, the data curve for the ring topology verifies our hypothesis.

Note that the relative positions of curves under various topologies remain unchanged, compared to Figure 5.1. The values obtained from reconciliation with and without time delay are generally within 25 percent of each other. In other words, the QoS results obtained from our experiments for the specific RRFS implementation, to a large extent, are applicable to other state-based replicated file systems as well.

5.1.2 Proliferation of Meta-Conflicts

One explanation of the vast conflict rate differences among topologies would be that it is possible for the same number of updates or versions to create a high variance of conflict rates. Resolving a conflict may involve the possible creation of a new version of data that combines the old versions, and this new version can cause subsequent conflicts, or **meta-conflicts,** as it propagates. Depending on the order in which the reconciliation events are generated, the total number of conflicts can vary significantly. Figure 5.5 and 5.6 present two sequences of reconciliation events to merge three different versions of data, located on three different servers. These two figures show that the total number of conflicts does not depend entirely on the number of data versions [Heidemann et al., 1995]. This observation suggests that the variability of conflict rate is potentially high.

Current versions			Events	Next versions			Total conflicts
Server 1	Server 2	Server 3		Server 1	Server 2	Server 3	
A	B	C	Server 1 pulls data from 2	AB	B	C	1
AB	B	C	Server 3 pulls data from 1	AB	B	ABC	2
AB	B	ABC	Server 2 pulls data from 3	AB	ABC	ABC	2
AB	ABC	ABC	Server 1 pulls data from 2	ABC	ABC	ABC	2

Figure 5.5: A reconciliation sequence to merge three different versions of data.

Current versions			Events	Next versions			Total conflicts
Server 1	Server 2	Server 3		Server 1	Server 2	Server 3	
A	B	C	Server 1 pulls data from 2	AB	B	C	1
AB	B	C	Server 2 pulls data from 3	AB	BC	C	2
AB	BC	C	Server 3 pulls data from 1	AB	BC	ABC	3
AB	BC	ABC	Server 1 pulls data from 2	ABC	BC	ABC	4
ABC	BC	ABC	Server 2 pulls data from 3	ABC	ABC	ABC	4

Figure 5.6: Another reconciliation sequence to merge three versions of data. Note that the number of conflicts does not entirely depend on the initial number of data versions.

Although possible, it seems unlikely that meta-conflicts alone can account for three orders of magnitude differences in conflict rate between fully connected topology and star topology. Worse, if it is indeed the case, optimistic replication has serious operational problems in noncentralized environments.

Since the accurate definition and distinction of a conflict and a meta-conflict become extremely unclear and controversial when the system experiences continuous incoming traffic [Heidemann et al., 1995], we examined only

one very common and well-defined type of meta-conflict called **identical conflicts**. An identical conflict occurs when updates to separate replicas are identical, and they typically occur as meta-conflicts. For example, users *A* and *B* create updates to separate replicas independently. *A* propagates her update to user *C*; *B*, to *D*. When *A* and *B* reconcile, they detect a conflict and subsequently generate an update to create the version *AB*. Similarly, *C* and *D* detect a conflict and generate an update to create the version *CD*. Now, when the version *AB* and version *CD* reconcile, we have an identical conflict. Identical conflicts are legitimate conflicts in the sense that concurrent updates (caused by reconciliation) have truly occurred; however, identical conflicts can be trivially resolved by comparing checksum information.

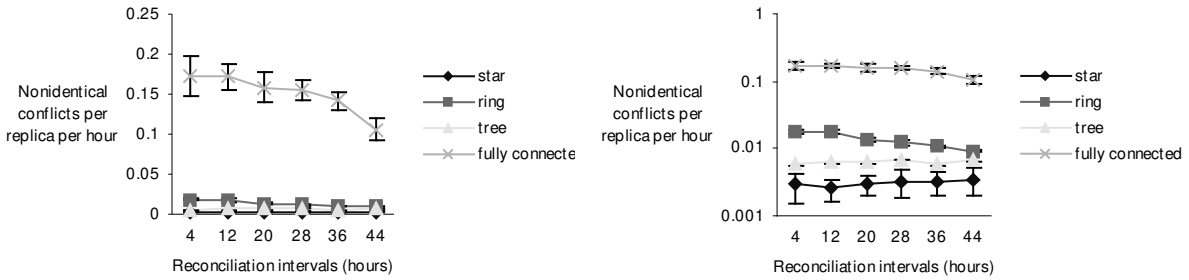

Figure 5.7: Average nonidentical conflict rate, with varying reconciliation intervals and topologies. The values correspond to a simulation environment with 50 replicas with two-way reconciliation protocol. The right graph presents the same data with conflict rate in log scale. By removing the identical conflicts, the conflict rate for fully connected topology improves by an order of magnitude.

Figure 5.7 shows the conflict rate curves after filtering out the identical conflicts. We see that identical conflicts, alone, account for 90 percent of conflicts in the fully connected topology. However, identical conflicts rarely occur in star and tree topologies because nearly all the conflicts are resolved in the centralized nodes. After this experiment, it is not as difficult for us to picture that other types of meta-conflicts can account for this wide range of QoS. For example, if 50 replicas initially have 4 conflicting versions of a file, during the next round of reconciliation, we can potentially have $\binom{4}{2}$ ways of generating meta-conflicts, and 4 conflicting versions can evolve into 6 conflicting "meta-versions" of a file. This process continues until eventually a version will emerge, which contains all 4 original conflicting versions.

Clearly, this explosion of meta-conflicts is not desirable. Aggressive conflict resolution, or resolving conflicts on detection, is one plausible explanation. To be explicit, in many existing optimistically replicated file systems, conflicts are resolved and new-versions of data are generated at reconciliation time. However, the site involved in resolving conflicts may not access the data at all for many rounds of the reconciliation processes. The conflict resolution process contributes no value to the local site and speeds up the proliferation of new versions due to the aggressiveness of conflict resolutions at each reconciliation process.

One remedy is to delay the conflict resolution, resolving conflicts only on reference. Another alternative is to restrict the conflict resolution to the update-contributing or other heuristically chosen subset of replicas. Of course, these lazy conflict resolution schemes either carry multiple conflicting versions of data at the replication

sites, or reduce the system's ability to perform indirect propagation of intermediately combined updates. However, the benefit of reducing the level of user-interventions to achieve scaling justifies the tradeoffs. Lazy conflict resolution schemes are perhaps an interesting area of future study, but are beyond the scope of this paper.

5.1.3 Alternative QoS Metrics—Stale Access Rates

After discovering the anomalies of conflict rate, we proposed an alternative pair of metrics to measure the QoS of optimistic replication—**stale read rate** and **stale write rate**. A stale read or a stale write occurs whenever a user reads or writes an out-of-date replica. Optimistic file replication can cause a user to access a file version older than the most current version in the overall system, because reconciliation has not yet propagated the newest update to the accessed replica. Measuring stale access metrics is difficult because it requires either global knowledge or post-processing. However, a simulation can accurately capture stale access metrics. We present both the conflict rate and the stale access rate to show the different service qualities defined by different metrics.

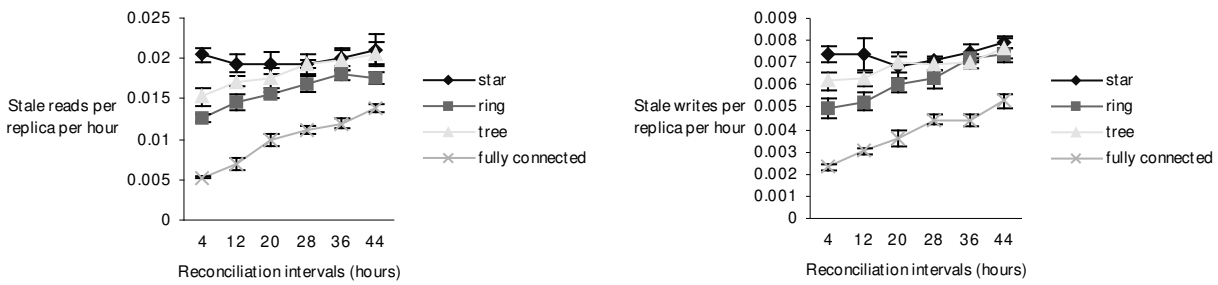

Figure 5.8: Average stale read rate (on the left) and average stale write rate (on the right), with varying reconciliation intervals and topologies. The values correspond to a simulation environment with 50 replicas with two-way reconciliation protocol. Note that as the reconciliation interval lengthens, stale access rates degrade for most topologies. For star topology, the initial improvement of stale access rates is caused by the effects of relaxing the centralized bottleneck.

The stale access curves (Figure 5.8) do not share the anomalies of the conflict rate curve. In addition, stale access rates agree with the intuition that QoS should degrade as reconciliation intervals increase. Most important, they measure quantities that are important to clients rather than an internal detail of the operation of the system. Note that stale write rate does not equal the conflict rate. It is possible for the stale write rate to be greater than the conflict rate—multiple stale writes can occur before a reconciliation process, and they will result in a single conflict. It is also possible for the conflict rate to be greater than the stale write rate. Conflicts subsequently generate meta-conflicts.

For the remaining sections, we do not abandon the conflict rate metric because it is a visible artifact of optimistic replication, and it requires user intervention at times. However, conflict rate provides a poor indication of the quality of data perceived by users.

5.2 Different User and System Perceptions of Reconciliation Intervals

So far, we have seen the effects of reconciliation intervals on the optimistic replication QoS metrics. To recap, as the reconciliation interval lengthens, conflict rate might decrease because conflicts are detected at reconciliation

time. Longer reconciliation intervals, in general, worsen stale access metrics (unless the intervals are short enough to overload the system.)

One curious detail we did not address is the change in QoS between reconciliation intervals of 12 and 20. From a user perspective of an 8- to 12-hour work schedule, the change of reconciliation interval from 12 to 20 hours should not contribute to QoS metrics—either interval should provide the user the perception of performing one reconciliation every day. However, users and machine perceive different pictures of QoS.

The key to understanding this counter-intuitive result is the rate of data flow in the entire system. Consider a ring topology with one-way propagation protocol. An 8-hour reconciliation interval assures an update propagates to a minimum of three replicas downstream in one day. In the same period, a 24-hour reconciliation interval assures that an update propagates to a minimum of only one replica. Every day, the user works on one reconciliation update, but that update can encapsulate the changes from one to three replicas upstream, depending on an 8- or 24-hour reconciliation period. The same argument can easily be extended to the fully connected topology, in which each update propagation also encapsulates the updates from other replicas involved in recent reconciliations. For the tree topology, a leaf node with an 8-hour reconciliation interval can propagate its update to its parent and, with high probability, to one sibling or grandparent in one day. With a 24-hour reconciliation interval, the leaf node can only propagate its update to its parent in the same period. For star topology, due to the centralized bottleneck, a shorter reconciliation interval does not necessarily guarantee that the update is propagated on schedule. However, since any replica is only two hops away from the most current updates, the small fraction of on-time reconciliation processes can make enough differences in QoS metrics. To extend this one-way reconciliation protocol example to a two-way protocol, we can decompose the two-way protocol into two one-way protocols in opposite directions.

In terms of cost metrics, Figure 5.9 shows the effects of reconciliation intervals on reconciliation time, percentage CPU utilization during reconciliation processes, transmission volume per replica, and aggregate transmission volume. Storage overhead (not shown) is unaffected by reconciliation intervals.

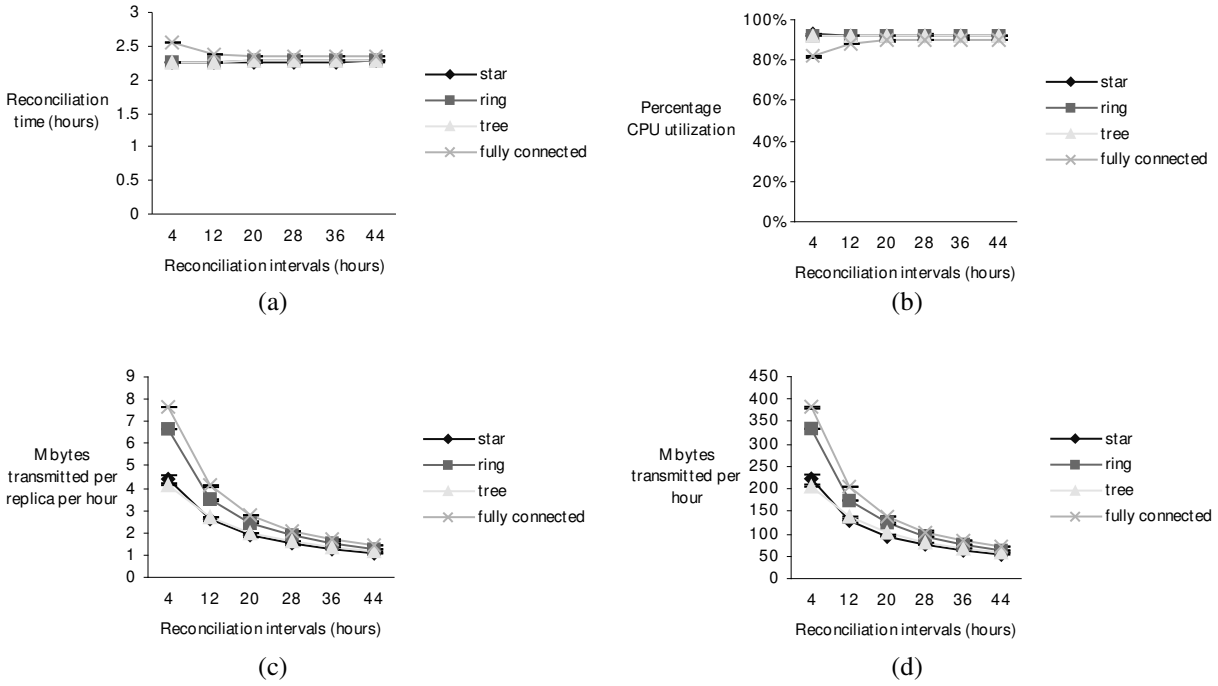

Figure 5.9: Effects of reconciliation intervals on optimistic file replication cost metrics: (a) reconciliation time, (b) percentage CPU utilization during reconciliation processes, (c) transmission volume per replica, and (d) aggregate transmission volume. The values correspond to a simulation environment with 50 replicas with two-way reconciliation protocol. Reconciliation time and CPU utilization are not as sensitive to reconciliation intervals as transmission metrics.

Figure 5.9 (a) shows that reconciliation time, with the exception of fully connected topology, increases slightly as the reconciliation interval lengthens (not visually noticeable from the graph). For the fully connected topology, the reconciliation time is high at lower reconciliation intervals because a replica needs to spend time to locate a nonbusy target replica for reconciliation. As the reconciliation interval decreases, the chance of finding a nonbusy target replica also decreases. Unlike the fully connected topology, other topologies have few and well-defined target replicas (with the exception of the single central server in the star topology). After attempting all possible target replicas specified by the topology, reconciliation stops trying and aborts. Figure 5.9 (b) tells the same story in terms of percentage CPU utilization. For fully connected topology, the utilization rate is lower at lower reconciliation intervals due to longer delays caused by seeking nonbusy replicas for reconciliation.

Transmission volume curves Figure 5.9 (c) and (d) share the same characteristic curves, except that the aggregate transmission volume is scaled up by the replication factor. As the reconciliation interval lengthens, the required transmission volume lessens. Because majority of data transmitted is metadata related overheads, lower transmission volume reflects the decrease in total number of reconciliation processes. Star topology has a slightly higher transmission volume than tree topology at a 4-hour reconciliation interval because of higher volume of request and abort messages caused by centralized bottleneck.

Overall, most metrics display monotonically increasing or decreasing trends, with the exception of the conflict rate metric. Reconciliation time and CPU utilization are not as sensitive to reconciliation intervals as other

metrics. Performance tradeoffs under reconciliation intervals are largely between the stale access rates and transmission volume, which reflect the total number of reconciliations.

5.3 Characteristics of Reconciliation Topologies

After understanding the monotonically increasing and decreasing trends of most metrics under various reconciliation intervals and topologies, and knowing that topology curves rarely cross, we feel comfortable taking a single reconciliation interval (12-hour interval) for topology comparisons. Figure 5.10 summarizes the effects of reconciliation topologies on various metrics. Storage overhead (not shown) does not vary across topologies. Figure 5.10 (a) and (b) show that the reconciliation time and percentage CPU utilization do not vary significantly across various topologies. However, in Figure 5.10 (c) and (d), fully connected and ring topology consume more transmission volume than star and tree topologies. Since the network expenditures are consumed mostly by fixed reconciliation overhead, transmission volume metrics indicate that more reconciliation processes have completed for ring and fully connected topologies within the same simulation time frame. As a result, ring and fully connected topologies have lower stale access rates compared to star and tree topologies, as shown in Figure 5.10 (g) and (h).

Topologies	Minimum number of reconciliation processes to reach all N replicas	Total number of reconciliation processes (N) allowed to run simultaneously
Fully connected	$\lceil \lg(N) \rceil$	$\left\lfloor \frac{N}{2} \right\rfloor$
Ring	$1+\left\lceil \frac{N-2}{2} \right\rceil$	$\left\lfloor \frac{N}{2} \right\rfloor$
Tree (assume complete K-ary trees)	KH to $(K+1)H$ depending on the placement of the update, where H is the height of the tree, and $H = \left\lfloor \log_K \left(\frac{KN-N+1}{K} \right) \right\rfloor$.	$K^{H-3}(K^2+1)$, where H is the height of the complete K-ary tree, and $H > 2$.
Star	$N-1$	1

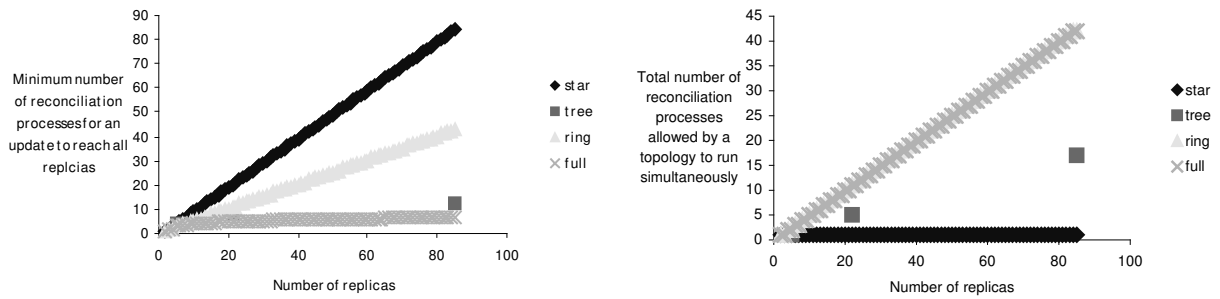

Figure 5.X: Characterization of various reconciliation topologies along two dimensions—minimum number of reconciliation processes to reach all replicas, and the total number of reconciliation processes allowed by a topology to run in parallel. The equations for tree topology are only applicable for complete 4-ary trees.

To understand the stale access results better, Figure 5.X characterizes topologies along two dimensions—the minimum number of reconciliation processes to reach all replicas, and the total number of reconciliation processes allowed by the topology to run simultaneously. For the former dimension, fully connected and tree

topologies should spread the updates faster due the smaller number of steps to disseminate updates to all sites. However, for the latter dimension fully connected and ring topologies can better propagate multiple updates in parallel. The combination of those two dimensions explains the relative rankings of topologies under stale access metrics.

For the conflict rate measurements in Figure 5.10 (e) and (f), they exhibit the opposite rankings of QoS as the stale access metrics. Notably, the fully connected topology has the highest conflict rate, but the lowest stale access rates. Fully connected topology offers high parallelism in disseminating updates, but at the same time offers vast opportunities for the proliferation of meta-conflicts. Without the meta-conflict remedies proposed at the end of Section 5.1.2, the alternatives are topologies with fast dissemination but limited parallelism. Tree and ring topologies are two intermediate examples.

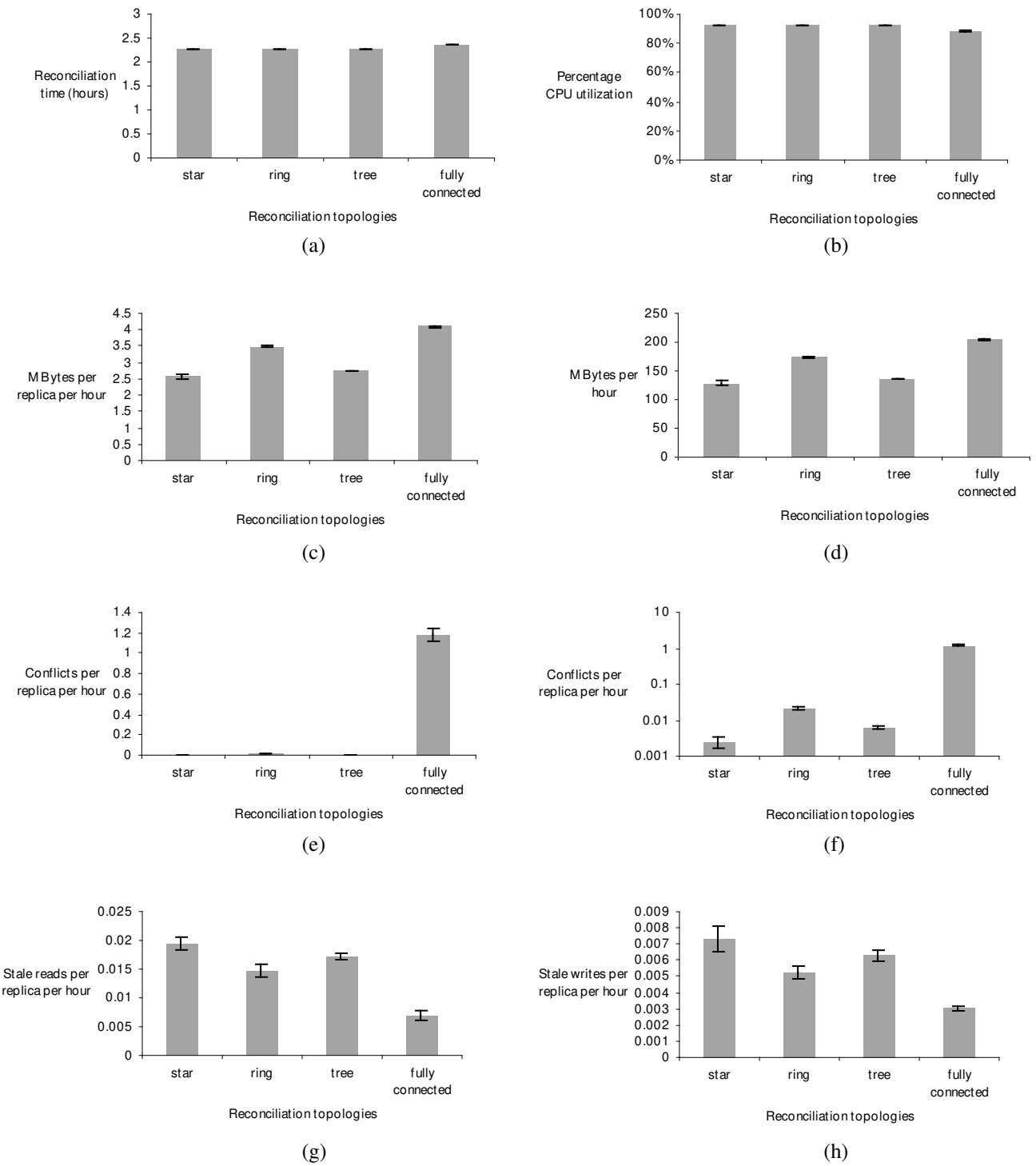

Figure 5.10: Effects of reconciliation topologies on various metrics: (a) reconciliation time, (b) percentage CPU utilization, (c) transmission volume per replica, (d) aggregate transmission volume, (e) conflict rate, (f) conflict rate in log scale, (g) stale read rate, and (h) stale write rate. The values correspond to a simulation environment with 50 replicas at the 12-hour reconciliation interval, with two-way reconciliation protocol. Note that conflict rate and stale access rates show contradicting rankings on topologies.

5.4 Potentials for Scaling

Scaling of optimistic replication is a controversial subject because of the following complications: (1) The conflict rate metric is poorly understood, and the prior applications of this metric to predict the scalability of such systems can be misleading; (2) Depending on the implementation, the size of the replicated states is often linearly proportional to the replication factor. The proliferation of replicated data structures can potentially pose storage, processing, and synchronization constraints. Our scaling experiment varied the number of replicas of a volume, holding the number of files within the replicated volume constant. The reconciliation interval was fixed at 12 hours, for the same reasons outlined in Section 5.3. Our results demonstrated that optimistic replication has the potential to scale well under most metrics.

Figure 5.11 illustrates the effects of scaling on various cost metrics. Figure 5.11 (a) shows that the storage overhead grows proportionally as the replication factor increases. The storage overhead varies from 16 Mbytes (7.1%) to 32 Mbytes (14%), at the growing rate of 0.33 Mbytes (0.15%) per replica. The overhead is high, but reducible. In the early RRFS implementation, each replica keeps states for all replicas; however, each replica does not share data with all replicas. One optimization is to minimize the states of each file by bookkeeping only the replicas involved in updating the file [Ratner 1998]. Since most files have a low degree of sharing (e.g., not greater than four), this optimization can actually bound the storage overhead to FS, where F is the per file storage overhead and S is the maximum degree of sharing for all files.

Figure 5.11 (b) shows the same proportional growth trend for reconciliation time. Reconciliation time varies from 19 minutes to 2.3 hours, at the growth rate of 2.6 minutes per replica. The reconciliation time appears to be high and suggests opportunities for optimization as well. By multiplying the reconciliation times with the percentage CPU utilization in Figure 5.11 (c), we obtain the CPU and I/O time components of reconciliation time. Figure 5.11 (d) reveals that the proportional growth of CPU time with replication factor explains the major fraction of reconciliation time. The I/O time for scanning the volume, network transmission, and waiting accounts for relatively fixed and minor fractions of reconciliation time. Perhaps the easiest optimization for reconciliation time is to use a faster processor. Moving from a 486 Pentium 100 MHz processor to a Pentium II 266 MHz processor results in 6x reconciliation time improvement. Other code optimizations have been applied to RRFS subsequent to validation and improved reconciliation time by another 40 percent. As for optimizing the I/O time component, the validated RRFS scans the source and target sites serially, and potential parallelism can be exploited for performance gain.

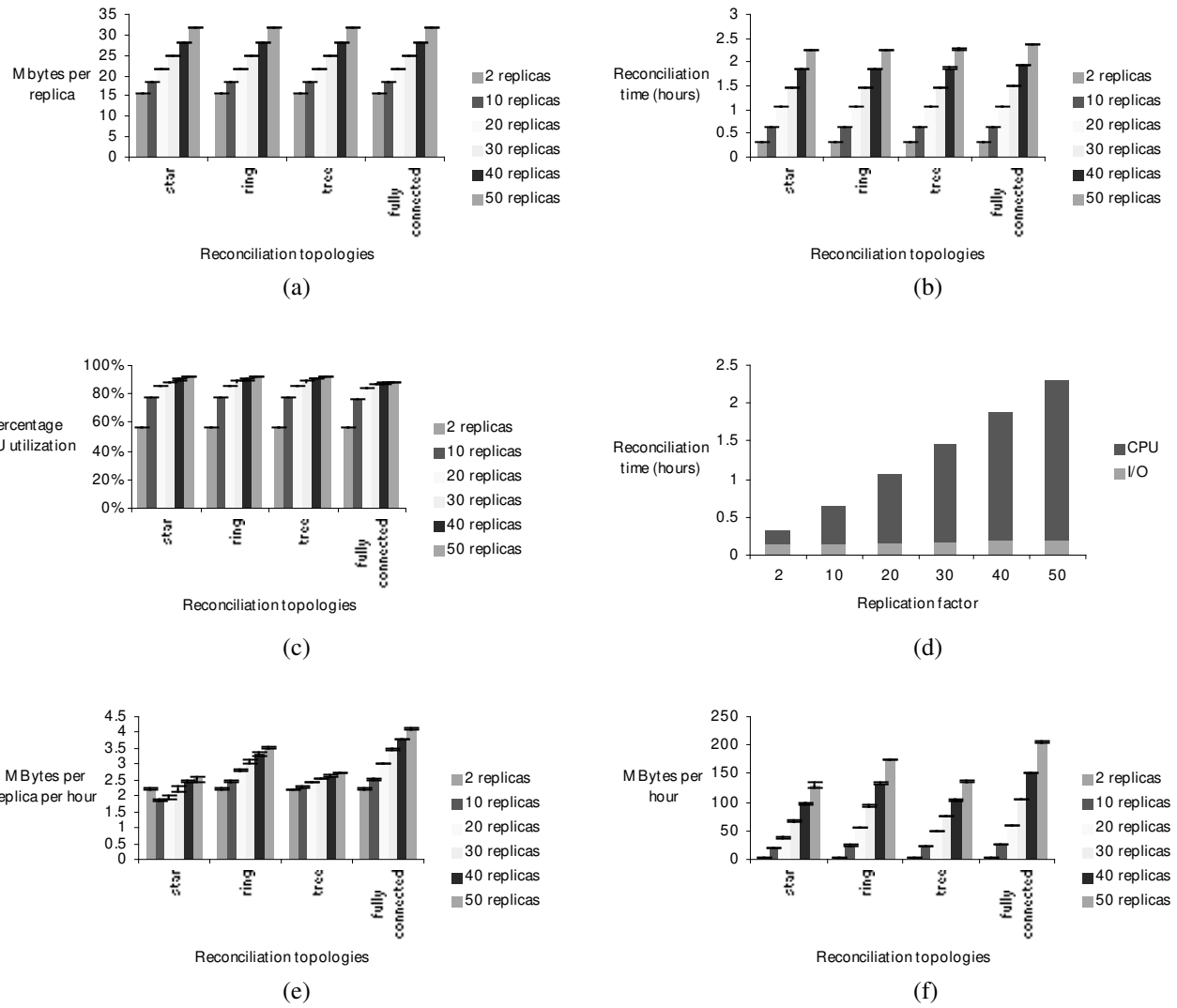

Figure 5.11: Effects of scaling on various topologies and cost metrics: (a) storage overhead, (b) reconciliation time, (c) percentage CPU utilization, (d) time components of reconciliation time, (e) transmission rate per replica, and (f) aggregate transmission volume. The values correspond to a simulation environment with the 12-hour reconciliation interval and two-way reconciliation protocol.

Figure 5.11 (e) shows the transmission volume per replica. The maximum network bandwidth required is only 4.1 Mbytes/hour, which is low for local area network, but may be relevant for modem transmission (6.5 Mbytes/hour for 14.4 Kbytes modems). For the star topology, the transmission volume drops from 2 to 10 replicas due to aborted colliding requests, but volume increases for higher replication factors because of the increasing amount of update information needed to be transmitted for each reconciliation. We also see dampening growth for all topologies with increasing replication factor due to lengthening reconciliation time and increasing probability of reconciliation locking.

For the aggregate network expenditures of the entire system, the maximum bandwidth required, as shown in Figure 5.11 (f), is 200 Mbytes/hr, which is also low for local area networks. Although we encounter a near-

quadratic growth trend of network traffic, we can still scale to hundreds of replicas without saturating the 10Mb Ethernet bandwidth. The proportional increases in both storage overhead and the number of replicas are the primary causes of this quadratic growth trend. Exploiting user sharing patterns to reduce storage and subsequent processing overheads seems to be the one common remedy for all cost reduction of optimistic replication.

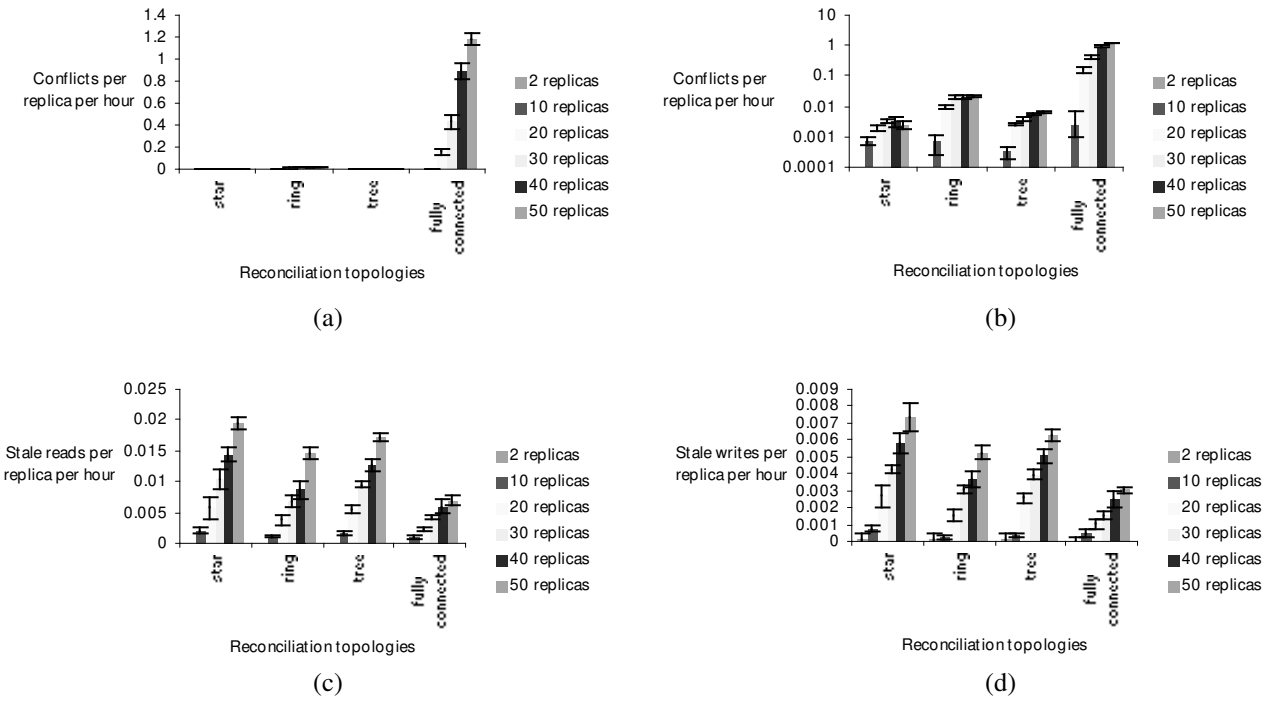

Figure 5.12: Effects of scaling on various QoS metrics: (a) conflict rate, (b) conflict rate in log scale, (c) stale read rate, and (d) stale write rate. The values correspond to a simulation environment with the 12-hour reconciliation interval and two-way reconciliation protocol.

Figure 5.12 illustrates the effects of scaling on QoS metrics. The maximum of 1.3 conflicts per hour (28 conflicts per day) at 50 replicas poses serious concerns for scaling optimistically replicated filing. Fortunately, the conflict rate is not experiencing an exponential growth rate with an increasing replication factor, as shown in Figure 5.12 (a) and (b). For star topology, conflict rate actually decreases for large replication factors. The main reason for this phenomenon is the lengthening of reconciliation time for increasing replication factors. Lengthening reconciliation time increases the probability of triggering reconciliation locking and dampens growth of the total number of reconciliations in the system. Available system resources prevent conflict rates from growing without bound. However, as we optimize the reconciliation time, fewer reconciliation processes will be aborted, and the envelope of conflict rate will expand further. Developing approaches to reduce the meta-conflicts via choosing reconciliation topology or lazy conflict resolution is essential to combat the explosive growth of conflict rate. For example, the use of star topology can reduce conflict rate down to one conflict every 16 days.

Stale access rates appears to be unreceptive to the lengthening reconciliation time and demonstrate near proportional growth to the increasing replication factor, as shown in Figure 5.12 (c) and (d). The star topology yields the maximum stale access rates, which translate to one stale read every 2.1 days and one stale write every 5.7

days at 50 replicas. By changing to fully connected topology, we can have one stale read every 6 days and one stale write every 14 days. Note that star and fully connected topologies are opposite extremes in terms of conflict rate and stale access metrics. Practical recommendations would lie somewhere in the spectrum of topologies, depending on the emphasis of minimizing user-intervention or update propagation delay.

To summarize, the trends for the majority of metrics do not pose serious limitations for scaling. Most of the existing costs can be reduced by exploiting user sharing patterns. For each replica, we can restrict the storage, processing, and transmission overheads to only replicas involved in data sharing, and not all replicas in the system. The growth trend of conflict rate is a serious concern, but it is currently dampened by the available resources in the system. Developing ways to reduce the proliferation of meta-conflicts is the long-term solution. For example, manipulating the reconciliation topology alone can reduce the conflict rate by orders of magnitude. From various analyses, we believe that a large-scale, high-quality optimistic replication service can be economically achieved.

5.5 Data Propagation Rate and QoS Tradeoffs of Reconciliation Protocols

So far, we have presented our simulation data on two-way reconciliation protocols. This subsection examines the effects of the one-way reconciliation protocol on various metrics (Figure 5.13). Again, we fix the replication factor to 50 and reconciliation interval to 12 hours. Storage overhead does not vary across the protocols and thus is not shown.

In terms of costs, Figure 5.13 (a) shows that use of the one-way reconciliation protocol only saves minutes of reconciliation time. Recall that the two-way reconciliation protocol is implemented by concatenating two one-way reconciliation processes in opposite directions, without the second round of scanning process. This result implies that nearly 99 percent of reconciliation time is attributed to scanning related overheads, both in terms of CPU processing (e.g., creating and processing the data structures and constructing checksum information) and I/O (e.g., accessing meta-data from the disk). In Figure 5.13 (b), the marginal increase of the percentage CPU utilization of the one-way protocol reflects the marginal decrease of reconciliation time. As expected, the one-way reconciliation protocol saves about half of network expenditures due to savings on high transmission overheads of state information, as shown in Figure 5.13 (c) and (d).

With the given implementation and reconciliation time characteristics, the one-way reconciliation protocol requires twice as much reconciliation time to achieve the same rate of data propagation as the two-way reconciliation protocol. In other words, the one-way protocol in this paper propagates updates at half the rate of the two-way protocol. Figure 5.13 (e) and (f), however, indicate that conflict rate actually decreases as we switch from two-way to one-way protocol. Delaying propagation of concurrent updates also slows down the occurrence rate of conflicts. Also, this decoupling of a two-way reconciliation process into two one-way reconciliation processes introduces the probabilistic aspect of establishing two-way exchanges of updates. This effect is most prominent in the star topology with centralized update dissemination. The tree topology appears to be least susceptible to reconciliation protocol under this parameter setting because the inner nodes are not the bottlenecks of the system, and conflicts are resolved in regulated fashion (mostly at inner nodes imposed by the topology). For the remaining

metrics in Figure 5.13 (g) and (h), switching from two-way to one-way protocol, or slowing down the update propagation rate, induces higher stale access rates as anticipated.

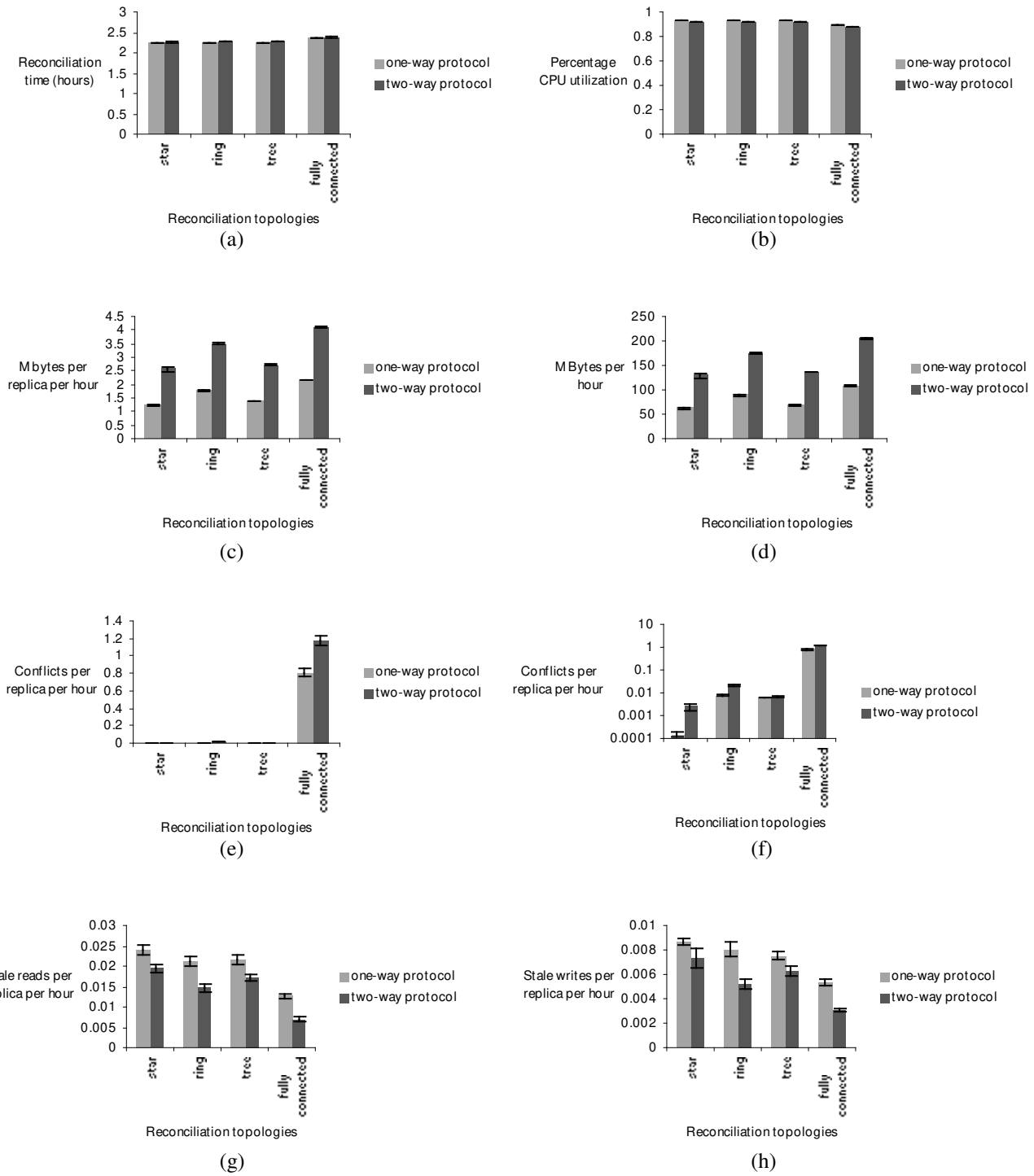

Figure 5.13: Effects of protocols on various metrics: (a) reconciliation time, (b) percentage CPU utilization, (c) transmission volume per replica, (d) aggregate transmission volume, (e) conflict rate, (f) conflict rate in log scale, (g) stale read rate, and (h) stale write rate. The values correspond to a simulation environment with 50 replicas at the 12-hour reconciliation interval. One-way reconciliation protocol decreases network expenditures but increases the stale access rates.

5.6 Effects of Percentage of Accesses to Shared Files

To extend the applicability of our results, we also investigated the effects of varying the percentage of file accesses that are directed at shared files (recall that the shared file accesses dictate the performance of optimistically replicated file system). Our trace indicated that the accesses to the shared files were restricted to about 13 percent of total references. We artificially increased the percentage of shared file accesses by factors of 2, 3, and 4. As reminders, we kept the ratio of shared file accesses to the number of shared files constant; the total number of file references in the system also remained unchanged. Figure 5.14 shows the effects on various metrics. Storage overhead does not vary with the percentage of access to shared files and thus is not shown.

From Figure 5.14 (a), (b) (c), and (d), growing percentages of accesses to shared files only pose trivial increases in most cost metrics (less than 5 percent). For star topology, with a growing percentage of shared file accesses, the trivial increase in reconciliation time and sharp decrease of network expenditures (which strongly reflect the number of successful reconciliation processes) suggest that star topology has reached its system limit. Figure 5.14 (e), (f), (g), and (h) show that all QoS metrics exhibit proportional degradation with increasing percentage of shared accesses.

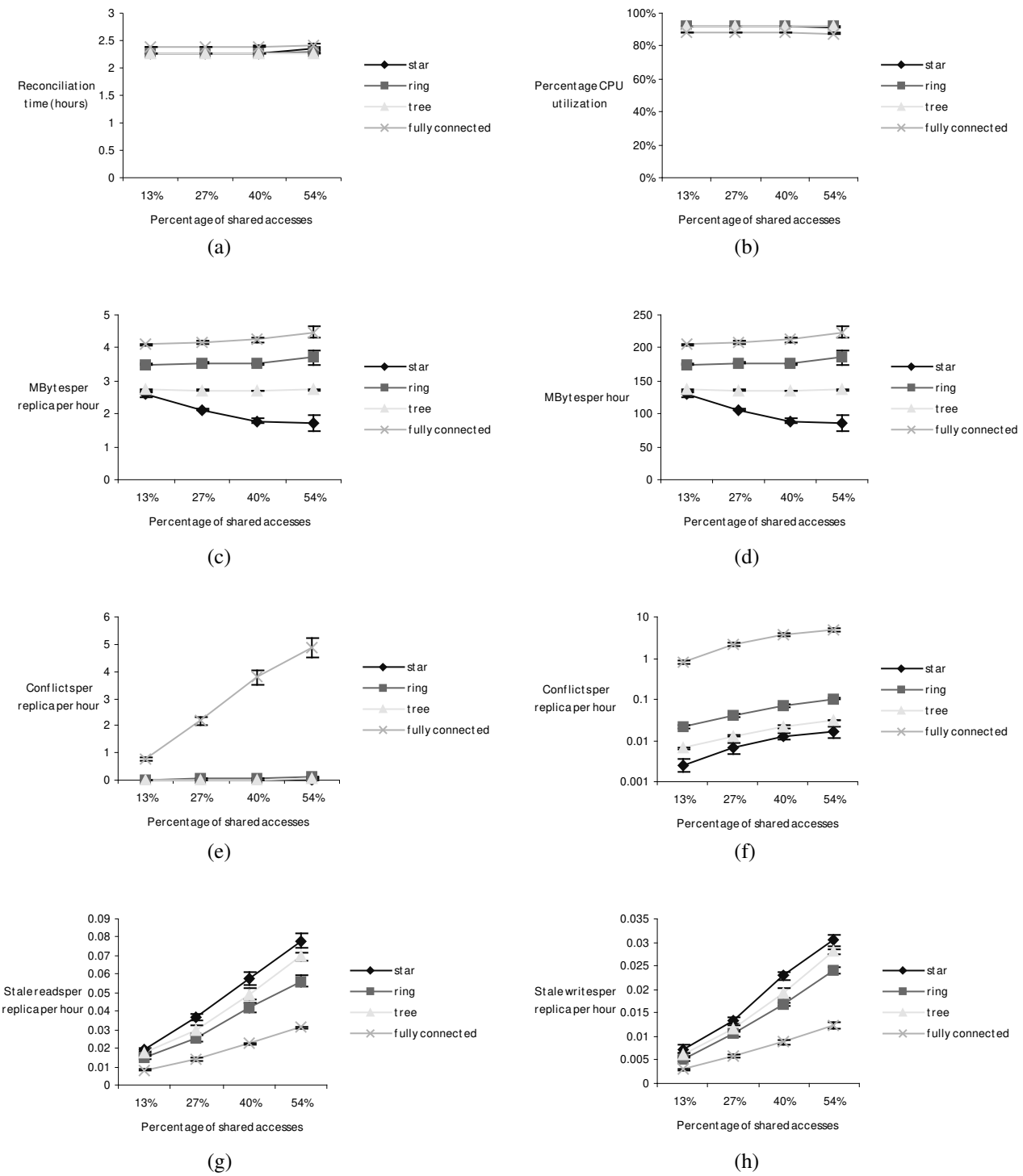

Figure 5.14: Effects of percentage accesses to shared files on various metrics: (a) reconciliation time, (b) percentage CPU utilization, (c) transmission volume per replica, (d) aggregate transmission volume, (e) conflict rate, (f) conflict rate in log scale, (g) stale read rate, and (h) stale write rate. The values correspond to a simulation environment with 50 replicas at the 12-hour reconciliation interval and two-way reconciliation protocol.

6 Related Work

Many analytical modeling and simulation studies have been done to compare the performance of various replication control protocols, but optimistic consistency strategies are often not addressed [Goldweber et al., 1989; Johnson et al., 1990; Liu et al., 1994].

Golding's thesis [1993] describes a *timestamped anti-entropy* protocol that has one-to-one mapping to our reconciliation process. The major difference between our works is that he does not address the situations of conflicts and stale accesses. Also, one QoS metric he used to measure various topologies is propagation latency, which measures the time required for an update message entered at time zero, and its acknowledgments to propagate to all available sites. As shown in our reconciliation topology section, the effectiveness of topologies also depends on the total number of data propagation processes that can occur in parallel.

Gray and associates [1996] have used the analytical approach to deduce the performance of optimistic replication. They, too, have predicted the explosive growth rate of conflict rate and questioned the scaling of optimistic replication. However, our study indicates that the explosive growth of conflict rate occurs under only certain conditions, and is bounded by available resources. Manipulating the reconciliation topology alone can affect conflict rate by orders of magnitude. In addition, conflict rate is misleading and inappropriate to measure the practicality of optimistic replication.

The design decisions involved in SynRGen, a synthetic file-reference generator developed at CMU, have motivated us in shaping our file-reference model. SynRGen captures the temporal and spatial localities of file references, and it models file sharing by having synthetic users perform activities in shared volumes. SynRGen characterizes file system activities by how each user behaves. Our file reference generator has an orthogonal view of a file system—how each file is accessed. For example, 1.7 percent of the files are two-way shared, and they receive 6.9 percent of file references. We designed our own file reference generator to handle the high sensitivity of our problem domain regarding the degree of file sharing. Other early works on file system characteristics and reference patterns also shed light on refining our file reference model [Satyanarayanan, 1981; Satyanarayanan, 1984; Ebling et al., 1994].

We have found that relatively few attempts have been made to explore the area of conflicts. The definition and counting of conflicts are well represented in the paper by Heidemann et al. [1995]. Conflict measurement studies are represented in the Ficus and Coda works. One major difference between the RRFS and these systems is the data propagation policy—Ficus and Coda respectively attempt to propagate data immediately and on demand. Ficus demonstrates that a conflict occurs approximately every four days [Popek et al., 1990; Reiher et al., 1994; Reiher et al., 1996]. The Coda results indicate that the chances of two different users modifying the same file within a week are less than 0.4 percent [Kistler et al., 1992; Noble at al., 1994].

Page [1996] has applied analytical methods to understand conflicts. However, his study assumes uniform file reference patterns, and the characterization of conflicts under wide parameter spaces can be explored only via a simulation framework. Carey's simulation study of various DBMS conflict detection algorithms emphasizes

differentiating various consistency protocols in terms of system response time. The area of conflict is not deeply explored [Carey et al., 1989].

7 Conclusions

We have developed a simulation model to analyze the cost, configuration, and QoS of an optimistically replicated filing environment. Unlike other file-system performance studies, our traffic generator captures the inverse relationship between sharing and write accesses, which plays the dominant role in obtaining meaningful results for optimistically replicated filing. Another design decision is to provide both language and library abstractions for complex modeling and future extensions. The model is validated against various dimensions of parameters, applying standard statistical validation techniques.

The study also identifies and analyzes the flaws of conflict rate metric and leads to the formulation of alternative metrics for evaluating optimistic replication. Conflict rate exhibits undesirable dependencies on total number of reconciliation processed allowed by the topology, reconciliation interval, and reconciliation protocol involved in resolving conflicts. Controlling the proliferation of meta-conflicts is an important direction of future research (e.g., delayed conflict resolution and limiting resolution to update contributing sites). Our alternative metrics, stale read and stale write rates, are more intuitive to understand and interpret. They measure quantities that are important to end-users rather than an internal detail of the operation of the system. They further demonstrate relatively dependency-free interactions and higher stability across various parameters.

Remaining results show that the effectiveness of a reconciliation topology depends on the propagation distance between replicas as well as the number of reconciliation processes that can occur in parallel. Meta-conflicts can contribute significantly to performance metrics and vary significantly among replication topologies. Adjusting reconciliation intervals based on the work cycle of each end-user does not achieve the highest service quality at the aggregate system level. In addition, the operational costs predicted by the simulation study suggest that optimistically replicated filing scales well, and further economies and service qualities could be realized by further exploiting user sharing patterns. Finally, we examined the tradeoffs of reconciliation interval and service quality, and the effects of varying traffic loads on shared replicated data.

8 Acknowledgment

Members of File Mobility Group and Parallel Simulation Group at UCLA have provided generous supports to this work. In particular, Dr. Vikas Jha provided technical support for the development of the simulation framework. Dr. Geoffrey Kuenning provided the file system traces for this study. Dr. Geoffrey Kuenning, Dr. David Ratner, Dr. Richard Guy, Mark Yarvis, Dr. Ted Kim, and Dr. John Saldanha contributed numerous invaluable discussions on the results of our experiments. In addition, I would like to thank Janice Martin, Alex Bui and George Lin for giving me helpful comments on this paper.

9 References

[Bagrodia et al., 1994] Bagrodia R, Laio WT. MAISIE: A Language for the Design of Efficient Discrete-Event Simulations. *IEEE Transactions on Software Engineering*, 20(4): 225-238, April 1994.

[Bagrodia 1995] Bagrodia R. *Maisie User Manual Release 2.2*. University of California, Los Angeles, 1995.

[Berliner 1998] Berliner B. *CVS II Parallelizing Software Development*. Prisma, Inc, 1998.

[Bertsekas et al., 1992] Bertsekas D, Gallanger R. *Data Networks, Second Edition*. Englewood Cliffs, NJ, Prentice-Hall, 1992.

[Carey et al., 1989] Carey MJ, Livny M. Conflict Detection Tradeoffs for Replicated Data. Technical report #826, University of Wisconsin, March 1989.

[Daniel et al., 1994] Daniels D, Doo LB, Downing A, Elsbernd C, Hallmark G, Jain S, Jenkins B, Lim P, Smith G, Souder B, Stamos J. Oracle's Symmetric Replication Technology and Implications for Application Design. *Proceedings of SIGMOD Conference*, p. 467, 1994.

[Ebling et al., 1994] Ebling MR, Satyanarayanan M. SynRGen: An Extensible File Reference Generator. *Proceedings of the 1994 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, Nashville, TN, May 1994.

[Golding et al., 1991] Golding R, Long DDE. Accessing Replicated Data in a Large-Scale Distributed System. *International Journal in Computer Simulation*, 1(2), 1991.

[Golding 1993] Golding RA. Modeling Replica Divergence in a Weak-Consistency Protocol for Global Scale Distribution Data Bases. Technical report UCSC-CRL-93-09, University of California, Santa Cruz, 1993.

[Goldweber et al., 1989] Goldweber M, Johnson DB, Raab L. A Comparison of Consistency Control Protocols. Technical report PCS-TR89-142, Department of Mathematics and Computer Science, Dartmouth College, 1989.

[Gray et al., 1996] Gray J, Helland P, O'Neil P, Shasha D. The Dangers of Replication and a Solution. *Proceedings of the 1996 ACM SIGMOD Conference*, pp.173-182, 1996.

[Guy et al., 1990] Guy R, Heidemann J, Mak W, Page T, Popek G, Rothmeier D. Implementation of the Ficus Replicated File System. *Proceedings of the Usenix Summer Conference*, pp. 63-71, June 1990.

[Guy et al., 1993] Guy R, Popek G, Page TW. Consistency Algorithms for Optimistic Replication. *Proceedings of the First International Conference on Network Protocols*, *IEEE*, October 1993.

[Heidemann et al., 1995] Heidemann J, Goel A, Popek G. Defining and Measuring Conflicts in Optimistic Replication. Technical report CSD-950033, University of California, Los Angeles, 1995.

[Irlam 1993] Irlam G. UNIX File Size Survey—1993, http://www.base.com/gordon/ufs.html, 1993.

[Johnson et al., 1990] Johnson DB, Raab L. A Tight Upper Bound on the Benefits of Replication and Consistency Control Protocols. Technical report PCS-TR90-157, Department of Mathematics and Computer Science, Dartmouth College, 1990.

[Kawell et al., 1992] Kawell LJ, Beckhardt S, Halvorsen T, Ozzie R, Greif I. Replicated Document Management in a Group Communication System. Groupware: *Software for Computer-Supported Cooperative Work*, IEEE Computer Society Press, 1992, pp. 226-235.

[Kazar 1989] Kazar M. Synchronization and Caching Issues in the Andrew File System. *Proceeding of the Winter Usenix Conference*, pp. 31-43, February 1998.

[Kistler et al., 1992] Kistler JJ, Satyanarayanan M. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems*, 10(1), February 1992.

[Kuenning et al., 1994] Kuenning GH, Popek GJ, Reiher PL. An Analysis of Trace Data for Predictive File Caching in Mobile Computing. *Proceedings of the 1994 Summer Usenix Conference, 1994.*

[Kumar et al., 1995] Kumar P, Satyanarayanan M. Flexible and Safe Resolution of File Conflicts. *Proceedings of the 1995 Usenix Technical Conference*, pp. 95-106, January 1995.

[Kung et al., 1981] Kung HT, Robinson J. On Optimistic Methods for Concurrency Control. *ACM Transactions on Database Systems*, 6(2), June 1981.

[Kung 1994] Kung HT. On Optimistic Methods for Concurrency Control. *Readings in Database Systems*. Morgan Kaufmann Publishers, San Francisco, California, 1994.

[Law et al., 1991] Law AM, Kelton WD. *Simulation Modeling & Analysis*. McGraw Hill, 1991.

[Liu et al., 1994] Liu ML, Agrawal D, Abbadi AE. What Price Replication? Technical report TRCS94-14, Computer Science Department, University of California, Santa Barbara, July 1994.

[Nelson et al., 1988] Nelson MN, Welch BB, Ousterhout JK. Caching in the Sprite Network File System. *ACM Transactions on Computer System*, 6(1), February 1988.

[Noble et al., 1994] Noble BD, Satyanarayanan M. An Empirical Study of a Highly Available File System. *Proceedings of the 1994 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, Nashville, TN, May 1994.

[Ousterhout et al., 1985] Ousterhout JK, DaCosta H, Harrison D, Kunze JA, Kupfer M, Thompson JG. A Trace-Driven Analysis of the UNIX 4.2 BSD File System. *Proceedings of the 10th ACM Symposium on Operating System Principles*, Orcas Island, WA, December 1985.

[Page 1996] Page TW. Models of Weakly Consistent Replicated Data. Unpublished manuscript. Department of Computer Science, Ohio State University, 1996.

[Parker et al., 1983] Parker DS, Popek G, Rudison G, Stoughton A, Walker B, Walton E, Chow J, Edwards D, Kiser S, Kline C. Detection of Mutual Inconsistency in Distributed Systems. *IEEE Transactions on Software Engineering*, pp. 240-247, May 1983.

[Popek et al., 1990] Popek GJ, Guy RG, Page TW, Heidemann JS. Replication in Ficus Distributed File Systems. *Proceedings of the Workshop on Management of Replicated Data*, Houston, Texas, November 1990.

[Popek et al., 1985] Popek G, Walker B. *The Locus Distributed Operating System*, MIT Press, 1985.

[Ratner 1995] Ratner D. Selective Replication: Fine-Grain Control of Replicated Files. Master's Thesis. University of California, Los Angeles, 1995.

[Ratner et al., 1996] Ratner D, Popek GJ, Reiher P. The Ward Model: A Scalable Replication Architecture for Mobility. *Proceedings of the OOPSLA'96 Workshop on Object Replication and Mobile Computing (ORMC'96)*, San Jose, California, October 7, 1996.

[Ratner 1998] Ratner DH. Roam: A Scalable Replication System for Mobile and Distributed Computing. Ph. D. Dissertation, University of California, Los Angeles, 1998.

[Reiher et al., 1994] Reiher P, Heidemann J, Ratner D, Skinner G, Popek G. Resolving File Conflicts in the Ficus File System. *Proceedings of USENIX Conference*, pp. 183-195, June 1994.

[Reiher et al., 1996] Reiher P, Popek J, Gunter M, Salomone J, Ratner D. Peer-to-Peer Reconciliation-Based Replication for Mobile Computers. *ECCOP 1996 Second Workshop on Mobility and Replication*, July 1996.

[Rosenblum et al., 1994] Rosenblum M, Herrod SA, Witchel E, Gupta A. SimOS: A Fast Operating System Simulation Environment. CSL-TR-94-631, Stanford University, 1994.

[Satyanarayanan 1981] Satyanarayanan M. A Study of File Sizes and Functional Life-Times. *Proceedings of the 8th ACM Symposium on Operating System Principles*, Pacific Grove, CA, December 1981.

[Satyanarayanan 1984] Satyanarayanan M. A Synthetic Driver for File System Simulations. *Proceedings of the International Conference on Modeling Techniques and Tools for Performance Analysis*, Paris, May 1984.

[Satyanarayanan 1989] Satyanarayanan M. Coda: A Highly Available File System for a Disconnected Workstation Environment. *Proceedings of the Second Workshop on Workstation Operating Systems*, September 1989.

[Tanenbaum 1996] Tanenbaum AS. *Computer Networks, Third Edition*. Prentice Hall PTR, 1996.

[Terry et al., 1995] Terry DB, Theimer MM, Petersen K, Demers AJ, Spreitzer MJ, Hauser CH. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. *Proceedings of the 15th ACM Symposium on Operating Systems Principle*, December 1995.

[Weiser et al. 1999] Weiser RF, Stokes E. *LDAP V3 Replication Requirements*, February 1999.