Tolerating Correlated Failures in Wide-Area Monitoring Services

Suman Nath, Haifeng Yu, Phillip B. Gibbons, and
Srinivasan Seshan

IRP-TR-04-09

May 2004

Research at Intel

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, life sustaining applications.
Intel may make changes to specifications and product descriptions at any time, without notice.

Copyright © Intel Corporation 2004
* Other names and brands may be claimed as the property of others.

Tolerating Correlated Failures in Wide-Area Monitoring Services

Suman Nath*, Haifeng Yu†, Phillip B. Gibbons†, Srinivasan Seshan*
†*Intel Research Pittsburgh* **Carnegie Mellon University*

Revised: June 9, 2004

Abstract

Recently, there has been increasing deployment of distributed infrastructure and applications such as Akamai, PlanetLab and DHTs. A highly-available monitoring service for these systems is vital to ensuring their smooth operation. In this setting, a key challenge to high availability is the presence of correlated failures, not addressed by previous monitoring services. This paper presents our approach to achieving high availability in IRISLOG, a customizable wide-area monitoring service. IRISLOG incorporates a replication design based on Signed Quorum Systems and a load-balancing design based on a novel XML database-fragmentation algorithm to achieve our availability goals. Using a combination of simulation under a tunable correlated failure model and results derived from a real world deployment on PlanetLab, we show that IRISLOG is able to achieve an availability level significantly higher than previous techniques.

1 Introduction

Over the past few years, there has been increasing interest in deploying new distributed infrastructures (e.g., Akamai [1], PlanetLab [3], RON [5]) and new distributed applications (e.g., Distributed Hash Tables and Content Delivery Networks). Monitoring such applications and infrastructures plays a key role in ensuring their availability, evaluating their performance, identifying bugs, etc. However, the development of such monitoring services is often left to the last minute despite the fact that it may rival the monitored system in both size and complexity. As a result, the monitoring service is often haphazardly and poorly built, with key robustness requirements being largely ignored. In fact, the robustness requirements of the monitoring service are often more stringent than that of the service being monitored, because monitoring results are most useful when problems arise. The focus of this paper is to identify key design principles for making monitoring services highly robust.

This paper presents our approach to achieving high availability in IRISLOG, a customizable wide-area monitoring service that we have developed. Like several other monitoring services (e.g., PIER [16] and Astrolabe [30]), IRISLOG uses a (distributed) database-centric approach to monitoring. IRISLOG provides a variety of mechanisms to efficiently query the hierarchical, distributed XML database that stores the monitoring data.

The greatest challenge in making IRISLOG robust is the presence of correlated failures – when one node of the monitored system fails, it is likely that others have failed as well. Such correlations may result from a variety of reasons, including: 1) a single request to the monitored system may trigger the same bug on many nodes; 2) the monitored system may be targeted by a denial of service attack or subject to localized phenomena such as power outages or network damage; 3) the similarities between the monitored nodes make them susceptible to particular failures (worms, viruses, etc.); and 4) human errors in the management of the monitored system may disable several nodes. Anecdotal evidence on PlanetLab[1] concurs with this observation. Because IRISLOG must co-exist on the same nodes as the system it monitors (to interact with the system and to minimize the total resources needed), it must continue to provide service despite these correlated failures. To the best of our knowledge, we are the first to comprehensively study the impact of these correlated failures on monitoring services.

Unfortunately, we find that commonly used fault-tolerant replication designs, such as the Majority quorum system (MAJORITY) [29], perform poorly when failures become correlated. While MAJORITY works well with uncorrelated failures, and is provably the best among all quorum systems [7], we show that correlated failures have a devastating effect on its availability. An important result from our study is that adding more replicas provides rapidly diminishing availability improvements for MAJORITY when failures are correlated. Under some levels of correlation, adding more replicas no longer helps beyond a certain point, providing an upper limit on the best possible availability MAJORITY can achieve, regardless of the number of replicas used.

To achieve high availability despite correlated failures, IRISLOG uses two complementary techniques to address

[1] Recent emails on the `planetlab-users` mailing list indicate many failures up to the OSDI'04 submission deadline, as well as many highly overloaded nodes. One email on 5/11/2004 indicated 30 simultaneous node failures.

two complementary types of correlated failures. First, IRISLOG applies a replication design based on Signed Quorum Systems (SQS) [35]. IRISLOG's design improves availability over MAJORITY by eliminating the requirement that a majority of a data item's replicas be available for consistency maintenance, at the cost of a small probability of reading stale data. Furthermore, because neither reads nor writes need to access a majority, IRISLOG enables the use of a large number of replicas without incurring a large cost for reads and writes. IRISLOG's replication design mainly serves to guard against *untargeted* correlated failures, i.e., failures not directly targeting the nodes in a particular replica group. For example, suppose a monitored application causes simultaneous failures of 10 of the 100 nodes on which IRISLOG is deployed. For untargeted failures, in any 20 nodes that IRISLOG may have chosen to use as a replica group, we will likely see 2 instead of 10 simultaneous failures. We expect most system software or hardware failures to be untargeted as long as we choose replicas randomly. Likewise, we expect most external attacks on IRISLOG to result in untargeted failures as long as we are able to conceal the specifics of replica groupings.

Second, IRISLOG may also suffer from *targeted* correlated failures, i.e., failures focused on the nodes in a particular replica group. A primary source of targeted failures is the simultaneous overload of the replicas in a replica group, by intentionally or accidentally directing bursts of monitoring queries to the same monitoring data,[2] thereby overloading all the nodes hosting replicas of that data. To this end, the second technique in IRISLOG for providing high availability is a decentralized load-balancing algorithm called POST. When a node is approaching overload, POST examines the workload directed to the monitoring data stored at the node. It identifies the portions of the monitoring data that should be offloaded to other (lightly loaded) nodes in order to both reduce the load below a target threshold and minimize the communication between nodes. POST exploits both the hierarchical structure of the monitoring database and the typical routing of monitoring queries, in order to make these complex fragmentation decisions efficiently.

We have implemented IRISLOG (the source code is available at `www.intel-iris.net`), and used it to build a variety of customized monitoring services. One service, called PLAB-MONITOR, monitors 310 PlanetLab nodes, and has been publicly available since November 2003 (`www.intel-iris.net/irislog`). Another IRISLOG service, called ESM-MONITOR, monitors deployments of End System Multicast [15], an operational Internet broadcast system based on overlay multicast.

Through both analytical study and extensive simulation using a tunable model for correlated failures, we show that with IRISLOG, the negative effects of correlation are significantly smaller than with MAJORITY. In one setting, for example, our design with just 15 replicas (of which only 4 need to be accessed on a read or write) achieves a level of unavailability plus inconsistency that is smaller than the level of unavailability achieved by MAJORITY even with 200 replicas. Moreover, IRISLOG's advantage over MAJORITY grows with increasing number of replicas and increasing correlation.

To evaluate our fragmentation algorithm POST, we emulate our PLAB-MONITOR service on Emulab [2]. We use the real IRISLOG code base and the trace of user queries collected from PLAB-MONITOR over 4 months. The results show that POST yields roughly as good a fragmentation as an optimal algorithm, in a fraction of the time (in one setting, POST takes a few seconds to compute the fragmentation while the optimal algorithm takes over an hour). Thus, POST enables the system to react swiftly in anticipation of overload, thereby avoiding correlated overload failures. Finally, we also report the performance on PlanetLab of the aforementioned PLAB-MONITOR and ESM-MONITOR.

In summary, the contributions of this paper are:

1. We are the first to study how to build a monitoring system that is highly available despite correlated failures. We show that traditional designs are unable to cope with such failures. We describe the design of IRISLOG, a customizable wide-area monitoring service highly resilient to correlated failures.

2. We describe how IRISLOG incorporates SQS into its replication design, and study the availability of IRISLOG under untargeted correlated failures, using both simulation and analysis. Our results show that even the sum of inconsistency and unavailability in IRISLOG is significantly smaller than the unavailability of MAJORITY under modest to large correlation.

3. We present POST, a new, efficient algorithm that balances load by dynamically fragmenting hierarchically-organized data (in our case XML data) across a collection of nodes. This mitigates the negative effects of overload-based targeted correlated failures.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 presents the IRISLOG architecture. Section 4 presents IRISLOG's replication design and its analysis under a tunable correlation model. Section 5 describes POST. Section 6 reports on our experimental study. Finally, Section 7 presents conclusions.

2 Related Work

A number of wide-area monitoring systems have been developed recently, including ACME [25], Astrolabe [30]

[2]Similarly, actions by the monitored system can trigger bursts of *updates* to the same monitoring data.

Ganglia [22], PIER [16], SDIMS [33] and Sophia [31]. Though these systems differ in many aspects from each other, none of them focuses on providing high-availability against correlated failures. Some systems (e.g., PIER) use distributed hash tables (DHTs) to improve availability, but they do not provide data consistency in the presence of ongoing data updates. Our replication techniques and evaluation results are applicable to DHT-based systems as well.

Traditional availability studies typically assume independent failures. Correlated failures have drawn more attention from researchers recently [6, 10, 12, 32]. Cui et al. [10] study how to take correlation into account when choosing backup paths in overlay routing, based on Internet measurement traces. Correlated failures are also studied [6] in the context of survivable storage systems based on availability traces of desktops [8] and different web servers. One major difference between these efforts and our research is that the availability in IRISLOG is influenced by consistency maintenance and replica regeneration. These issues are not present in the previous two efforts and is in fact our focus. In the context of [6, 10], it is also easier to derive the actual level of correlation from measurement data. It is more challenging to construct a representative correlation model for use in IRISLOG, because our goal is to guard against buggy application and DoS attacks, and also because of the need for overprovisioning in our context. Thus instead, we strive to make observations that are likely to extend beyond our specific model of correlation. We also study a larger number of replicas than [6], where only 10 replicas are considered.

From a theoretical perspective, researchers [12] have studied how to design fault-tolerant algorithms when the exact correlation pattern (i.e., which set of nodes always crash simultaneously) is known. In comparison, our study on IRISLOG is based on a tunable correlation model where the correlation pattern is not known beforehand. Weatherspoon et al. [32] focus on finding a set of replicas with small failure correlation in order to improve availability, which is complementary to our efforts of improving availability under a given level of correlation.

Majority quorum system [29] is an instance of traditional quorum systems [7]. Signed Quorum Systems (SQS) were initially proposed in [35], which focuses on the theoretical structure and optimality of SQS. This paper implements SQS in a real system and is the first to study its behavior under correlated failures. We also address the refresh issue in SQS that was not previously discussed in [35]. The regeneration design in IRISLOG is influenced by RAMBO [21] and Om [37], but neither RAMBO nor Om studies the system behavior under correlated failures.

The POST fragmentation algorithm is related to the classic graph partitioning problem, which has been addressed in many contexts such as VLSI circuit optimization [17, 28]. Although the problem is NP-Complete for general graphs, polynomial time algorithms are known for DAGs and Trees. The algorithm in [28] finds an optimal partitioning of a DAG in $O(n^3)$ time, where n is the number of nodes in the DAG. A simpler algorithm [19] addresses a constrained version of the problem (tree with integer weight functions) and finds the optimal partitioning in pseudo-linear time (effectively close to $O(n^3)$ in our scenario). None of these previous algorithms are intended to be used online and their computational overheads make them infeasible for the case of thousands of XML elements.

3 IrisLog Architecture

This section provides an overview of the architecture of IRISLOG. IRISLOG takes a database-centric and sensor/actuator-centric approach to address several key challenges of monitoring a distributed application (or infrastructure) running on a wide-area collection of physical nodes (called *hosts*). First, applications are modified to provide a sensor-like report at each host about the status and operation of the application component running on the host. Similarly, an actuator-like interface is provided at each host that enables modifying the component's behavior or configuration. Second, a distributed XML database is maintained to store the continuously changing data obtained from the sensors monitoring the distributed applications. Finally, users can use XPATH [4], a standard query language for XML, both to inspect the status of the collection of application hosts and to control their behavior.

Our database-centric approach enables us to support a powerful query language and arbitrary queries on the sensor data, even if the queries were not thought of *a priori*. The database is distributed for scalability and availability. The semi-structured nature of XML gives the flexibility of not being restricted to a predefined rigid database schema (hence part of the schema can be modified online without affecting the rest) and enables a rich and evolving set of data types, aggregate fields, etc., through the use of self-describing tags. Moreover, the hierarchical nature of XML provides the opportunity to perform efficient in-network aggregation and to efficiently scope queries to only the relevant parts of the database.

While the high-level advantages of a distributed XML database are clear, realizing such a design is challenging. IRISLOG leverages our earlier work on a wide-area sensor network, IRISNET [11, 23], to address some of these challenges. IRISLOG maintains a separate XML database for each application being monitored. The XML database can be viewed as a tree defining a hierarchy,[3] and each tree node is an *XML element*. The XML database can be fragmented and replicated among a set of hosts. IRISLOG supports a

[3]Note that multiple hierarchies can be defined over the same data.

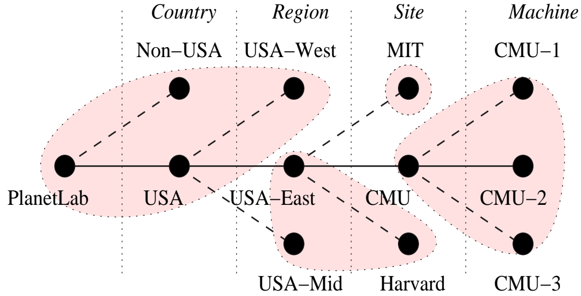


Figure 1: Part of the XML database used by PLAB-MONITOR, a PlanetLab monitoring service. The database is fragmented and placed in four hosts (shown as shaded).

flexible fragmentation of the XML database—any subset of XML elements can be placed in any set of hosts. Figure 1 shows the hierarchy used in our PLAB-MONITOR service and a hypothetical fragmentation of it among four hosts.

The XML semantic hierarchy allows users to scope their queries to individual subtrees in the hierarchy instead of routing all queries via the root. IRISLOG currently supports two types of queries: *pull queries* for collecting a snapshot of the sensor data, and *push queries*, which cause sensor data or aggregates (periodically or on change) to be pushed up the hierarchy for continuous data collection. While each pull query is routed independently, the data resulting from push queries are persistently stored on all relevant hosts. One nice property of our query routing, inherited from IRISNET, is that pull queries only reach (and push queries are only stored on) hosts containing relevant XML elements. IRISLOG also lets users install database-style *triggers* that interact with a set of actuators when a user-defined condition is satisfied. Such functionality is useful in gluing together sensing and actuation to automate system management tasks.

To customize IRISLOG in order to monitor a new application, application components residing on the host nodes need to export the appropriate sensor/actuator interface [26] and an *XML schema* needs to be created, describing the hierarchical organization of the sensors and actuators. If desired, custom aggregation functions (e.g., computing histograms) not supported by XPATH can also be defined and incorporated using a simple Java-based interface [9] provided by IRISLOG. Finally, IRISLOG's schema extension tool can be used to modify the schema online when, for example, new sensors/actuator types and new application hosts are added.

With the above architecture as a context, the rest of this paper will focus on two particular aspects of IRISLOG that help IRISLOG tolerate correlated failures and achieve high availability.

4 Replication Design in IRISLOG

This section provides a brief description of the replication design in IRISLOG and then studies its availability using an analytical approach. Our discussion on the design is not intended to be complete, but rather, to focus on the issues that are related to our availability study.

4.1 Design Overview

In IRISLOG, each host maintains a fragment of the XML database containing sensor data such as monitored CPU load over time, and user data such as triggers inserted by users. For robustness, the XML elements in a fragment are partitioned into a collection of replication units, called *data items*, and each such data item is replicated on multiple hosts.[4] The set of hosts for a given data item is called its *replica group*. Each such host is called a *replica*.

A replica is a *primary* of a replica group if it believes that it has the smallest IP[5] among all live replicas in the replica group. Such belief is obtained using inaccurate failure detection, and it is possible for one replica group to have multiple primaries. A primary of a replica group containing particular XML elements periodically (every 10 seconds in IRISLOG) pushes corresponding updates to the replica groups containing the parent XML elements. A primary does not necessarily push updates upon every update in order to control the traffic incurred. Otherwise if the XML elements on a replica group have many children XML elements, the replica group will constantly receive updates from below and try to push them upward. When the sensor has some critical data that needs to be propagated promptly, we also allow the sensor to flag the update which will incur immediate push.

A primary is also responsible to periodically evaluate user triggers installed at the replica group. We choose to use a primary to avoid unnecessary redundant push traffic or trigger evaluation. The correctness of our system is not affected by multiple primaries, since multiple push of the same data will be filtered by the upper level replica group. While it is possible that a trigger is triggered multiple times, such a scenario is provably unavoidable in a failure-prone environment (since it reduces to the **CoordinatedAttack** problem [20]).

4.2 Quorum Intersection for Consistency

In WAN environments, failures are the norm and a writer (e.g., a sensor updating sensor data, or a primary pushing

[4] As discussed in Section 5, the partitioning of the database into data items is dynamic.

[5] In our design, it is not necessary to use IPs for such ordering purpose. Any ordering among the replicas suffices as long as all replicas agree on such ordering.

data to the parent replica group) to a data item may not always be able to update all the replicas, and a reader (e.g, a user posing a query) may not be able to read from all the replicas. Worse, failures may not always be detected accurately. For example, it is possible for a writer to believe that a replica has failed while a reader is able to reach the very same replica. These issues lead to the possibility that a reader may not see the values written by previous writers, i.e., the data item it reads is *stale*, not *fresh*. In IRISLOG, our consistency model is to require any read that starts (in physical time) after a write W finishes to observe W. Note that we assume that all writes commute and hence we do not serialize writes. This is motivated by our monitoring context where writes from the same sensor have version numbers or timestamps to indicate which write to a data item is fresher. On the other hand, writes from different sensors update different data items and do not conflict.

Let a *quorum* denote the set of replicas updated by a writer or read by a reader. If quorums intersect, then the reader will observe at least one fresh copy, and can return the value with the largest version number. A standard way of ensuring quorum intersection is the MAJORITY approach: each reader and writer is required to access a majority of the replicas. If a reader or writer is indeed able to access a quorum, we say the replica group is *available*. Otherwise, the replica group is *unavailable* and the read or write fails. MAJORITY is known to provide the most availability among traditional quorum systems (i.e., those that guarantee intersection) [7], so it provides a good comparison point for our design. In IRISLOG we use a different approach for quorums that will be discussed in the next section.

4.3 Replica Regeneration

In large-scale systems such as IRISLOG, it is possible to *regenerate* failed replicas in a replica group. Namely, when replicas fail, instead of waiting for them to recover (in fact they may never recover), we can recruit other hosts in the system as new replicas. The hope is that regeneration takes much shorter time. To regenerate, we need to shift the replica group to exclude the failed replicas and to add new ones. The regeneration protocol in IRISLOG is a simplified version of RAMBO's [21] with some small modifications. Following we provide a brief description, focusing on the differences between IRISLOG and RAMBO.

The reads and writes in IRISLOG use a much simplified protocol than RAMBO's two-phase protocol. RAMBO's two-phase protocol serves to achieve *linearizability* [13] for reads and writes. As mentioned earlier, IRISLOG assumes that all writes are commutable and does not attempt to serialize writes. Thus in IRISLOG, a read simply reads from a read quorum and a write writes to a write quorum.

Regeneration is tricky in the face of false failure detection. For example, two replicas may simultaneously believe the failure of each other and then regenerate independently. This may result in two disjoint new replica groups for the same data item. To avoid such scenarios, regeneration in RAMBO involves the Paxos consensus protocol [18] that requires a majority of the replica group to be up and to coordinate with each other, in order to guarantee that a new replica group is unique.

```
public class ReplicaGroup {
    int sequenceNum;
    Host[] replicas;
    long TTL;
}
public class Token {
    String dataItemID;
    // List of replica groups sorted by sequenceNum
    ReplicaGroup[] repGroupList;
}
```

Figure 2: A token.

After regeneration, care must be taken so that the old replica group is properly retired, otherwise it is possible that a reader uses the old replica group while a writer uses the new replica group. In RAMBO, old replica groups are explicitly garbage collected using the data access quorums. Any later reads or writes will see such information from their quorums, and thus realize that the replica group has expired.

In IRISLOG, because we will later use *Signed Quorum Systems* (SQS) as the data access quorums and SQS does not always guarantee intersection, directly adopting RAMBO's design would potentially result in inconsistency regarding whether a replica group has expired. Thus we use the following design that utilizes loosely synchronized clocks on replicas. Every replica group, once established, has a time-to-live (TTL) of one day in IRISLOG. We use a token (Figure 2) to denote the list of currently unexpired replica groups for a given data item. A new token is created as the result of the Paxos protocol by appending the new replica group at the end of the list. The token is then published into the IRISNET [11] naming layer. A token is *valid* if its TTL for the latest replica group has not expired. For a read/write to access the data, the reader or writer must obtain a valid token and then access a quorum from *every* unexpired replica group in the token. To ensure that there always exists valid tokens for a data item, Paxos is executed once every 12 hours even when there is no need for regeneration.

Requiring the reader or writer to access a quorum from all unexpired replica groups may appear suboptimal. However, from a practical perspective, since regeneration occurs only when there are failures, the list of unexpired replica groups

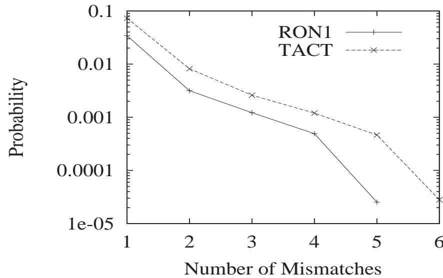


Figure 3: Probability of simultaneous mismatches. These are sample results from [34] based on RON [5] and TACT [36] traces.).

will likely to be short. Further, the different replica groups will have many replicas in common, so the cost of accessing one quorum from k replica groups is likely to be much smaller than the cost of accessing k disjoint quorums.

4.4 Improving Availability under Correlated Failures

We now describe a key feature of our replication design that enables IRISLOG to be far more robust against correlated failures than previous approaches: its use of *Signed Quorum Systems* (SQS) [35].[6]

The use of MAJORITY serves to guard against false failure detections. For example, consider a replica group of three replicas A, B, and C, where B has failed. Suppose a writer believes that B and C have failed, while a reader believes that A and B have failed. Under the MAJORITY approach, the replica group is unavailable and neither will succeed. On the other hand, if a majority is not required, the writer will only update A while the reader will only read C and return stale data. We can define the status of each replica to be one of the following four cases:

$(+,-)$ Reachable from the writer but not from the reader.
$(-,+)$ Reachable from the reader but not from the writer.
$(-,-)$ Not reachable from either the writer or the reader.
$(+,+)$ Reachable from both the writer and the reader.

The status of $(+,-)$ and $(-,+)$ are called *mismatches*. In the previous scenario of stale read, there must be mismatches on both hosts A and C.

The intuition behind SQS is that MAJORITY is overly pessimistic regarding the likelihood of $(-,+)$ and $(+,-)$ in the current Internet. Even though the probability of one replica being in such a status may not be small, the probability that *multiple* replicas are all in such a status is small. Figure 3 provides some sample results from [34] regarding such probabilities.[7]

Based on this intuition, it is now possible for us to construct quorums that are smaller than a majority. For example, we can simply probe all n replicas and permit the reader or writer to proceed even if only s replicas are reached. Clearly, if the reader and writer do not intersect (i.e., access some replica in common), there must be at least $2s$ mismatches.

The previous design, however, requires the reader and writer to contact all replicas. The SQS construction we will use in IRISLOG is adopted from the *optimal* construction in [35]: **A writer or reader probes the replicas according to the same fixed order[8] until either s replicas are successfully accessed or all replicas have been probed.** In the former case, it can easily be proved that if the writer and reader do not intersect, there are at least s mismatches. In the latter case, the replica group is unavailable. The value of s can be tuned, and may be much less than $n/2$. A larger s decreases the probability of inconsistency but increases the probability of unavailability.

It is obvious that the above SQS construction improves availability because we now need only s instead of $n/2$ available replicas. This becomes critically important in the presence of highly correlated failures, because once $n/2$ replicas have failed, we are unable to regenerate and must wait for hosts to recover. With MAJORITY the replica group is unavailable during this time, while with IRISLOG, reads and writes can still proceed as long as $s \ll n/2$ remain available. A less obvious, but equally important advantage is that each read and write now accesses only s replicas instead of $n/2$ replicas. In a large-scale distributed system, sometimes the limiting factor on the number of replicas is *not* whether there are sufficiently many hosts or sufficient disk space. For example, a popular web object can be replicated in a large number of proxy caches, and the same is true for a DNS entry. It is more likely that the overhead of reads and writes limits the number of replicas. Because SQS potentially decouples read and write overhead from the number of replicas, we are able to use a larger number of replicas than was previously feasible.

4.5 Data Refresh

With quorum systems, not all replicas have fresh data. For example, with MAJORITY on five replicas A, B, C, D and E, suppose A and B are temporarily unavailable, then only a majority of the replicas (C, D and E) have the fresh data. Later, when C crashes and A recovers, the reader is still guaranteed to see fresh data if it reads from a majority.

This becomes different in SQS. Suppose $s = 2$ and data were first written to C and D. Later when A recovers and D

[6]We avoid the SQS formalism used in [35] to study the theoretical structure of SQS, but rather focus on a specific, practical SQS design.

[7]For such results to hold, the replicas must be randomly distributed in the wide-area (instead of residing in the same LAN).

[8]Same as for choosing the primary, such ordering can be any ordering but we use IP ordering in IRISLOG.

crashes, we need to refresh the data item, so that A and C have the fresh data. The original design [35] of SQS targets scenarios where the read happens not long after the write, so that the probability of such a scenario is negligible. In IRISLOG however, a user may pose a query long after the data item is written by the sensor. As a result, we need to monitor the status of the replicas so that we always try to ensure the first (according to the same order as used in SQS) s live replicas have the fresh data. Note that this refresh is fundamentally different from replica regeneration in that we only need to copy the data item in order to create more fresh replicas. We do not need a majority of replicas to achieve consensus.

Specifically, refresh is performed in the following way in IRISLOG. Each live replica R with fresh data monitors the preceding (according to the same order as used in SQS) live replica and ensures that the previous live replica also has fresh data. The very first live replica monitors the status of all replicas in the replica group, and rewrites the data to the first s live replicas whenever failures or recoveries occur. As long as there are at least s live replicas and at least one of which has fresh data, the above *data refresh check* process ensures that the first s live replicas have fresh data. In IRISLOG, data refresh check is performed every 30 seconds.

4.6 An Analytical Study of IrisLog Availability under Correlated Failures

In this section, we analyze the effectiveness of the replication design described above to mitigate the negative effects of (untargeted) correlated failures. We will first develop a tunable synthetic model for untargeted correlated failures. Then we use this model to analytically study the unavailability and inconsistency of IRISLOG, and compare it to the unavailability of MAJORITY.[9]

4.6.1 A Tunable Model for Correlated Failures

With correlated failures, a *failure event* causes the failure of one or more hosts. If every failure event were to cause the failure of all the hosts, no system could survive. In practice, there is likely to be a distribution on the number of failures caused by each failure event. Obtaining a representative distribution for the failure correlation in today's Internet is an open research question by itself, and is beyond the scope of this paper. Furthermore, it is likely that the distribution may vary significantly from case to case. For example, imagine that IRISLOG is monitoring a buggy DHT implementation where certain requests cause the failures of all hosts in the peer's routing table. Clearly, failure correlation will then depend on the internal structure of the application. Given the lack of a real representative correlation distribution, we will instead evaluate (by analysis and simulation) the availability of IRISLOG under a tunable synthetic model, described next.

Consider a *universe* of u hosts, which are the hosts running IRISLOG. Failure events arrive at each host independently, according to a Poisson distribution with that host's MTTFE (mean time to failure event). To introduce correlation, each failure event causes the failures of $i \geq 1$ hosts, including the host where the event originates, with probability p_i (the choice of p_i's is discussed below), for $1 \leq i \leq u$. In our model, we assume that these $i-1$ additional hosts are randomly chosen from the universe. From a practical perspective, such an assumption is rarely true (e.g., in the buggy DHT example above). However, it is appropriate when the hosts that form each IRISLOG replica group are chosen at random from the universe (so that there is no correlation to the application structure – the failures are not targeted). In applications with potential targeted attacks on IRISLOG, making this assumption valid requires (at minimal) that IRISLOG does not reveal to the attacker which hosts comprise a replica group. Because of the multiple random failures for each failure event, each host's true MTTF exceeds its MTTFE.

Our model has a tunable parameter ρ between 0 and ∞ that intuitively controls how strong the correlations are. When $\rho = 0$, it means that failures are independent, while $\rho = \infty$ means that every failure event causes the failure of all hosts. Specifically, for $0 < \rho < \infty$, we use the following geometric sequence for the values of the p_i's: $p_i = c \cdot \rho^i$. The normalizing factor c serves to make $\sum_{i=1}^{u} p_i = 1$ and equals $\frac{1-\rho}{\rho(1-\rho^u)}$.[10] When $\rho < 1$, it is less and less likely for an event to cause more failures. If $\rho > 1$, then failures are so highly correlated that it is more likely for an event to cause a large number of failures than a small number of failures (e.g., failure of over half the hosts is more likely than failure of less than half). Each failed host recovers independently, according to a Poisson distribution with that host's MTTR (mean time to recovery).

An important property of our correlation model is that the "strength" of correlation does not change linearly with ρ. For a universe of size 200, Figure 4 plots the cumulative distribution for the number of failures per event. The figure shows that correlation is rather small when $\rho = 0.5$. Even when $\rho = 0.9$, half of the events fail 7 or fewer nodes, which is only 3.5% of the nodes. When $\rho = 1.0$, each event fails i nodes with equal probability for $1 \leq i \leq 200$.

As mentioned earlier, we believe that the exact model of correlation can vary significantly depending on the infras-

[9] We assume throughout this paper that MAJORITY also employs regeneration to improve availability.

[10] For the corner case of $\rho = 1$, we set $c = 1/u$. Also note that the ρ parameter controls the correlation in a specific failure model, and hence is not to be confused with the standard correlation coefficient (which ranges from -1 to 1).

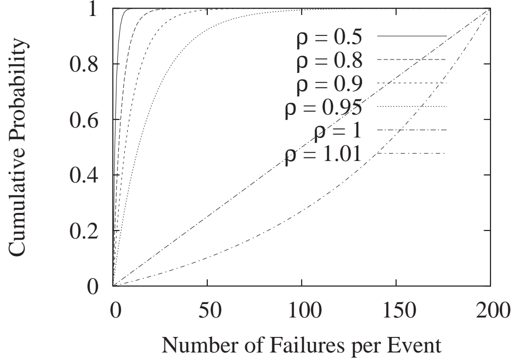


Figure 4: Effects of ρ.

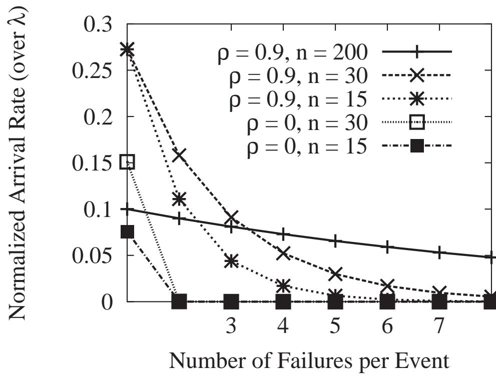


Figure 5: The arrival rate of events that cause a certain number of failures within 15, 30, or 200 replicas.

tructures and applications being monitored, and a full exploration is beyond the scope of this paper. Our evaluation will also try to draw observations that are likely to generalize beyond our particular correlation model. However, to provide a starting point of understanding, we collected a 50-day long failure trace of around 200 PlanetLab nodes by trying to ssh onto each node around every 30 minutes. We break the 200 nodes into four groups of 50, based on the alphabetical ordering of domain names. The universe size we use is 384, which is the total number of nodes in PlanetLab. We found that different windows roughly correspond to ρ values from 0.4 to 0.8, with the correlation increasing as the OSDI'04 deadline approached. This also confirms our belief that applications running on top of the infrastructure can cause correlated failures.

4.6.2 Analyzing IrisLog Availability

To analyze IRISLOG's availability under our tunable correlated failure model, we examine the impact of correlated host failures on an arbitrary replica group of $n \leq u$ hosts. For simplicity in the analytical formulas, we will assume that each host has the same MTTFE and the same MTTR.

Let $P(j,n)$ be the probability that a failure event causes exactly j failures in a replica group of n hosts ($0 \leq j \leq n \leq u$), for a given ρ ($0 < \rho < 1$). Then a careful analysis (see Appendix A) shows

$$P(j,n) = \binom{n}{j} \frac{(1-\rho)\rho^{j-1}}{1-\rho^u} \sum_{k=0}^{u-n} \rho^k \frac{\binom{u-n}{k}}{\binom{u}{k+j}} \qquad (1)$$

A closer examination of this formula reveals a subtle, yet fundamental effect of correlated failures on replication systems. To understand such effects, we can view the replica group as forming a "window" for us to observe failures from the whole universe of u hosts, and we only care about failures that happen within the window. Failure events arrive over the universe altogether at a rate of $\lambda = u/\text{MTTFE}$. Based on equation 1, Figure 5 plots the arrival rate of events that cause a certain number of failures (normalized by λ). This figure demonstrates that **increasing the window size effectively increases the "strength" of the correlation observed in the window**. To see this, note that in the absence of correlation ($\rho = 0$), doubling the window size n simply doubles the arrival rate of those events causing a single failure within the window. When $\rho = 0.9$, however, doubling n does not double the single failure arrival rate, nor any multi-failure arrival rates. This is apparent from the complex interaction of n on $P(j,n)$. In contrast to when the entire universe is considered (the $n = 200$ curve), the small window reduces the number of failures per failure event. Doubling n makes some of the single-failure events in the window become multiple-failure events in the window, while most remain single-failure events. As will be shown both analytically and with simulation, IRISLOG's SQS-based system is far superior to MAJORITY at dealing with strong correlations.

We now consider the availability of IRISLOG and MAJORITY. A replica group in MAJORITY is unavailable whenever over half its hosts simultaneously fail. Let U_{maj} be the unavailability of a replica group of n hosts in a universe of u hosts under MAJORITY. We can approximate the unavailability by (1) considering that a single failure event that fails at least half the replica group is needed to initiate unavailability, and (2) ignoring that additional failure events may arrive while waiting to completely recovery from this initiating failure event. That is, U_{maj} can be approximated by the arrival rate of failure events that fail j hosts (for $j \geq n/2$) multiplied by the time to recover $j - n/2 + 1$ of those hosts (so that we have fewer than $n/2$ failed):

$$U_{maj} \approx \frac{u}{\text{MTTFE}} \sum_{j=n/2}^{n} P(j,n) \sum_{i=n/2}^{j} \frac{\text{MTTR}}{i}, \qquad (2)$$

where $P(j,n)$ is defined in equation 1. For example, when $\rho = .95$ and the MTTFE is 14 times the MTTR, then even if the replica group uses all 200 hosts ($u = n = 200$), the

analysis works out to 0.0149 unavailability for MAJORITY. Thus, MAJORITY suffers greatly from correlated failures.

In contrast, an analysis of the unavailability of IRISLOG yields much more promising results. In IRISLOG, a replica group of size n and quorum size s becomes unavailable when over $n-s$ of its hosts are currently failed, and it remains unavailable until s or more of the hosts have recovered. Let $U_{irislog}$ be the unavailability of a replica group of n hosts in a universe of u hosts under IRISLOG with quorum size s. Under the same two assumptions as were used for equation 2 (with $n-s+1$ replacing "at least half"), and using a similar argument, we obtain:

$$U_{irislog} \approx \frac{u}{\text{MTTFE}} \sum_{j=n-s+1}^{n} P(j,n) \sum_{i=n-s+1}^{j} \frac{\text{MTTR}}{i} \qquad (3)$$

As an example of this result, note that when $\rho = .95$, the MTTFE is 14 times the MTTR, and the replica group uses all 200 hosts ($u = n = 200$), the result works out to 1.4e-06 unavailability for $s = 4$, 3.1e-06 unavailability for $s = 6$, and 5.5e-06 unavailability for $s = 8$.

Next we analyze the probability of inconsistency in IRISLOG, assuming availability. Inconsistency occurs in a replica group when the first s hosts reached by a reader all have stale data. Note that a writer updates the first s hosts it can reach (because we are assuming availability there will be s such hosts). Thus, in the absence of subsequent host failures there would always be s hosts with fresh data, and the only source of inconsistency would be simultaneous mismatches. The probability of mismatch is determined by the characteristics of network failures in today's Internet. Based on extensive results in [34] (also see Figure 3 for sample results) and also for simplicity, we use the closed-form formula of 0.1^i as the probability of having i simultaneous mismatches. As for subsequent host failures, we observe that because of the frequent data refresh checks in IRISLOG, the first s live hosts at any point in time will quickly get fresh data, as long as there is at least one live host with fresh data. Thus, a good analytical approximation for the inconsistency $I_{irislog}$ in IRISLOG for a replica group of n hosts with quorum size $s < n/2$ is given by: the probability of s mismatches plus the product of (a) the probability that all s in the quorum fail simultaneously and (b) the duration until one of the s has recovered. That is,

$$I_{irislog} \approx 0.1^s + \frac{u}{\text{MTTFE}} \cdot P(s,s) \cdot \frac{\text{MTTR}}{s}, \qquad (4)$$

where $P(s,s)$ is defined in equation 1. As an example of this result, when $\rho = .95$ and the MTTFE is 14 times the MTTR, then the analysis gives 0.0075 inconsistency for $s = 4$, 0.0013 inconsistency for $s = 6$, and 0.0004 inconsistency for $s = 8$. Thus low inconsistency can be obtained with very small quorum sizes. Also, for these settings, the first term in equation 4 is dominated by the second term by a factor of 75 or more, indicating that failures are a far greater source of inconsistency than mismatches and the results are likely to be insensitive to the formula of 0.1^i.

Summary. The analytical results using our correlation model appear to identify the dominant factors in unavailability and inconsistency. They show that MAJORITY suffers greatly from correlated failures, while IRISLOG better tolerates such failures, achieving orders of magnitude smaller unavailability and a small probability of data inconsistency. These conclusions are confirmed by our simulation results in Section 6. In addition, note from equations 2–4 that doubling the ratio of MTTFE to MTTR will halve the unavailability in both MAJORITY and IRISLOG and halve the dominant term in the inconsistency in IRISLOG.

5 Load Shedding to Avoid Correlated Host Overload

Signed quorum systems are most effective when failures are untargeted. In IRISLOG, there can be several sources for targeted correlated failures. A primary source of targeted failures is the simultaneous overload of the replicas in a replica group, by intentionally or accidentally directing bursts of monitoring queries to the same (replicated) data item. In this section, we describe the mechanism IRISLOG uses to guard against such host overload. The general idea is that IRISLOG hosts, when approaching overload, shed load by automatically partitioning their XML fragments and transferring them to other lightly loaded hosts. We focus on CPU overload and network overload in our discussion, though the algorithm can be applied to other resources.

5.1 Overview

Each host in IRISLOG maintains the exponentially weighted moving averages (EWMA) of the rate of queries and updates to each XML element in its database. To provide stability of the data placement process, each host sheds load when the load is above a *high-watermark* threshold until the load goes below a *low-watermark* threshold. A host sheds load by transferring to other hosts some of its XML elements, determined by a *fragmentation* algorithm explained later. By using its service discovery component, IRISLOG tries to choose a receiver host such that adding the new XML elements will (still) result in a connected fragment on that receiver (i.e., a connected subtree). If no such host is found, a random host is chosen. Our experience has shown that this simple heuristic performs significantly better than a purely random host selection strategy. More advanced selection strategies, e.g., based on geographic location, are beyond the scope of this paper.

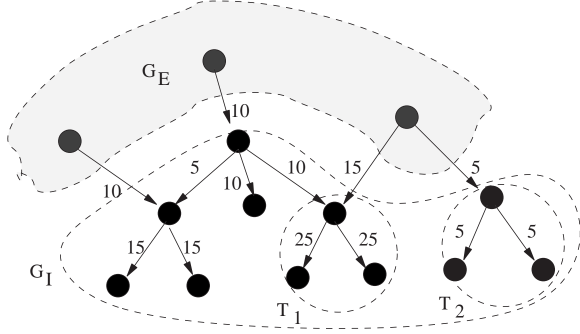


Figure 6: The workload graph of a host. The circles labeled T_1 and T_2 represent two partitions of size 3.

To minimize CPU and network overhead for subsequent queries, the XML elements transferred must be carefully chosen. An XML query requires accessing a set of XML elements in a given (partial) order, in order to properly evaluate the predicates and wildcards in the query. For most queries, this is a top-down order in which parent elements are accessed before their child elements. For example, in Figure 1, a query on all the CMU data requires first accessing the `CMU` element and then accessing its children elements (`CMU-1`, `CMU-2`, and `CMU-3`). Now, if the host containing all the CMU data (i.e., the rightmost shaded host) transfers its `CMU` element, the same query will generate three new subqueries sent on the network and three additional database accesses.[11] On the other hand, if the `CMU-1` element is transferred instead, that will result in only one new subquery and one more database access.

5.2 Fragmentation Problem

We formalize the previous fragmentation problem using the following terminology. For a given host holding a particular XML fragment, let G_I denote the set of local (*I*nternal) XML elements, and G_E denote the set of non-local (*E*xternal) XML elements (in fragments of other hosts) and the set of query sources. Define the *workload graph* (Figure 6) to be the DAG where nodes are the union of G_I and G_E, and edges are pointers connecting elements in the XML database and sources to elements. Under a given workload, an edge in the workload graph has a weight corresponding to the rate of queries along that edge. The weight of a node in G_I is defined as the sum of the weights of all its incoming edges (corresponding to its query load) and the weights of all its outgoing edges to nodes in G_E (corresponding to its message load). For example, the weight of the root node in T_1 is $10+15=25$.

For any set of nodes T within G_I, we define T's *cost* to be the sum of the weights of nodes in T. The $cut_{internal}$ of T is the total weight of the edges coming from some node in G_I to some node in T, and it corresponds to the additional communication overhead incurred if T were transferred. The $cut_{external}$ is the total weight of the edges coming from some node in G_E to some node in T, and it corresponds to the reduction of load on the host if T were transferred. In Figure 6, the $cut_{internal}$ of T_1 is 10, while the $cut_{external}$ is 15.

The fragmentation algorithm, at its core, requires solving a graph partitioning problem that has been well studied. However, previous algorithms [19, 28] tend to suffer in terms of computational cost. With a 3 GHz machine with 1 GB RAM, the algorithms in [19, 28] each takes over an hour to partition an XML fragment with 1000 XML elements. Such excessive computational overhead would prevent IrisLog from shedding load in a prompt fashion. On the other hand, trivial algorithms (e.g., the greedy algorithm in Section 6.2) do not yield "good" fragmentations.

5.3 Our Solution: POST

In this section we describe our algorithm POST (*Partitioning into Optimal SubTrees*), for quickly producing a good fragmentation.

Upon signs of overload (i.e., the high-watermark is reached), a host first constructs the workload graph and then invokes POST to determine which set T of XML elements to evict. The host passes a constant C to POST to indicate the *maximal* cost T may have. The value of C is determined by the extra load that other hosts may take. POST tries to find a "good" T under such constraints. Intuitively, we may want to minimize $cut_{internal}$ (achieved by T_2 in Figure 6) or maximize $cut_{external}$ (achieved by T_1 in Figure 6). To be comprehensive, we define our objective to be maximizing $(1-\gamma)*cut_{external}-\gamma*cut_{internal}$, and then we tune the value of γ between 0 and 1 to study its effects.

To design an efficient fragmentation algorithm in IRISLOG, we exploit the following important characteristics in the workload: *A typical monitoring query in a hierarchical database accesses all elements in a complete subtree of the tree represented by the monitoring database, and* IRISLOG *routes the query directly to the root of the subtree.* This observation is well supported by a trace of real user queries on PLAB-MONITOR (details are in Table 1), which shows that more than 99% of the user requests select a complete subtree from the XML database.[12] Under such access patterns, the optimal T is typically a subtree. The reason is that transferring only part of a subtree T from a host H_1 to another host H_2 may imply that a top-down query accesses elements in H_1 (the top of T) then in H_2 (the middle of T) and then back in H_1 (the bottom of T), resulting in suboptimal solution.

The above observation enables POST to restrict the search

[11]Each subquery results in one independent database access. With replication, these numbers are multiplied by the quorum size.

[12]More specifically, the query selects all the PLAB-MONITOR `machine` elements in some subtree.

space, and run in linear time. POST sequentially scans through all the nodes of the workload graph, and for each node, it considers the whole subtree rooted at it. For all the subtrees with size smaller than the given capacity C, it outputs the one with the optimal objective. The search space is further decreased by scanning the nodes in the workload-graph in bottom up fashion, and thus considering the subtrees in increasing order of their sizes. With the same experimental setup used for the previously mentioned algorithms, POST computes the result in less than two seconds. Yet, as we will show in Section 6.2, the quality of the fragmentation results is very close to that of the $O(n^3)$ optimal algorithms in practice.

Finally, we note that each time POST decides to transfer a partition T from the overloaded host to a lightly loaded host, it must coordinate with all the relevant replicas. Specifically, for each data item in T,[13] a consensus protocol must be run among the replicas for that data item, in order to drop the old host from the replica group and add the new host. If T contains only part of some data item, that data item is split into two, and a consensus protocol is used to update the old replica group and also to create a new one.

6 Experimental Evaluation

This section evaluates (1) the availability and consistency of IRISLOG under correlated failures, (2) the effectiveness and overheads of our fragmentation algorithm POST, and (3) the performance of IRISLOG in two wide-area monitoring services.

6.1 Availability Evaluation under Correlated Failures

Availability evaluation is always challenging because for highly-available systems, unavailability is a rare event, and a long experiment duration is necessary to observe stable results. For example, assuming a system with five nines availability and one-hour MTTR, we need on average over 11 years to observe a single system down event and even much longer to gain confidence in the results. Given the infeasibility of measuring IRISLOG availability through experimental deployment, we use a discrete event-driven simulator. For each experiment, we simulate roughly 300 years of physical time. Even with simulation, the length of the experiments still limits the universe size we can simulate to 200 hosts.

We simulate the complete IRISLOG replication design, as well as MAJORITY, under our tunable correlation model. We use the following default settings for the system parameters: the simulation quantum is 1 second, the host MTTFE is 14 days (recall that each failure event fails 1 or more hosts), the host MTTR is 1 day, the regeneration time is 1 minute,[14] refresh checks for a replica group are every 30 seconds, the universe size u is 200, and the probability of i mismatches is 0.1^i.

6.1.1 Availability of MAJORITY

This section studies the effects of correlated failures on MAJORITY. Figure 7(a) shows the unavailability of MAJORITY as a function of the number of hosts in a replica group, for four settings of ρ. When $\rho = 0.5$, because of the regeneration functionality, MAJORITY is effective in improving availability. In terms of absolute results, just seven replicas are sufficient to deliver five nines availability. On our log-scale graph, adding more replicas roughly decreases unavailability linearly. However, such effects quickly dampen as ρ increases. When ρ is 0.9 and 0.95, adding more replicas yields diminishing returns in availability improvement. After a certain point, the curves flatten and even small availability improvement requires a large increase in the replica number. For example, improving from three to four nines availability under $\rho = 0.9$ requires increasing the replica number from 15 to 43. Further, our simulation shows that the system can never achieve five nines availability (even if we use all 200 hosts as replicas) under $\rho = 0.9$. As analyzed in Section 4.6.2, the problem for MAJORITY is that increasing the replica number effectively increases the observed strength of the correlation and MAJORITY becomes unavailable as soon as half the replica group fails.

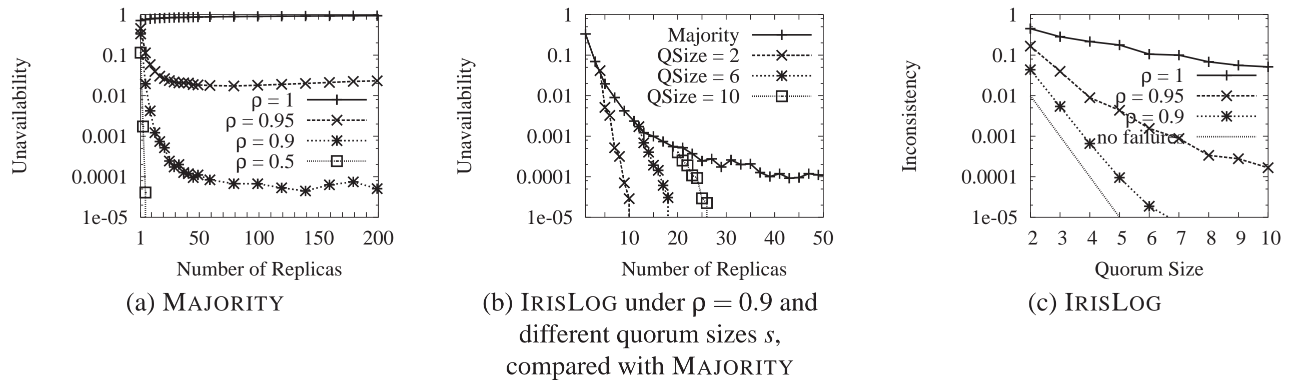
6.1.2 Availability and Consistency of IRISLOG

We now turn to the properties of the replication design in IRISLOG. Figure 7(b) plots the unavailability of IRISLOG as a function of the number of hosts in the replica group. Because the quorum size in IRISLOG is tunable, we plot the results for three different quorum sizes under the same ρ of 0.9. Since in SQS, s is always smaller than a majority, each curve thus starts from $n = 2s$. For comparison, we also replot the MAJORITY curve from Figure 7(a) for the same ρ. Clearly, a larger quorum size decreases availability, because more hosts must be available in order to have a quorum. However, in all cases IRISLOG shows dramatically improved availability over MAJORITY because (as highlighted by the analysis in equations 2 and 3) far more failures are needed before IRISLOG becomes unavailable.

Figure 7(c) plots inconsistency in IRISLOG under different ρ values and quorum sizes s. Clearly, larger ρ has a negative effect on inconsistency, because correlated failures result in fewer hosts with fresh data (until the data is refreshed) and, hence, make it more likely for all the hosts with fresh data to fail simultaneously. Once this happens,

[13] Recall that the unit of replication in IRISLOG is called a data item.

[14] Because the relative benefits of our design increase with regeneration time, this relatively small value only makes our results pessimistic.

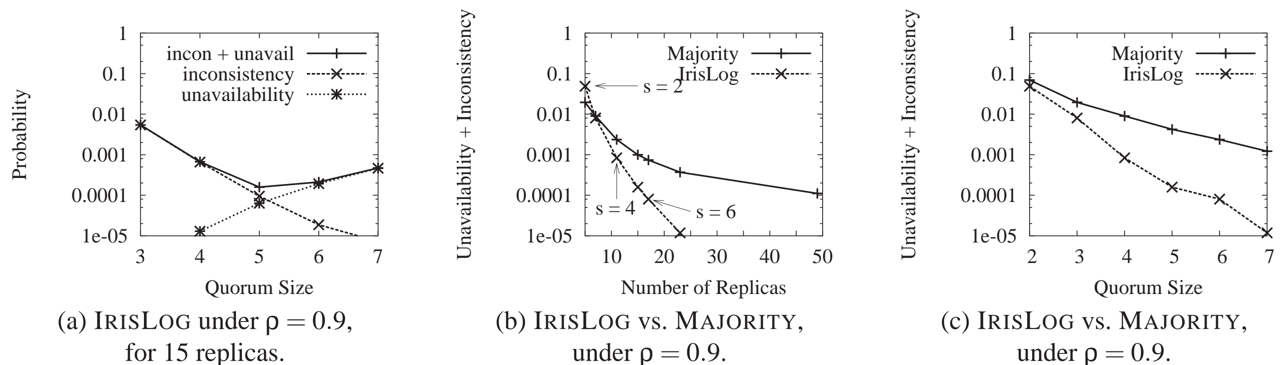


(a) MAJORITY

(b) IRISLOG under $\rho = 0.9$ and different quorum sizes s, compared with MAJORITY

(c) IRISLOG

Figure 7: Inconsistency and unavailability of IRISLOG and MAJORITY.



(a) IRISLOG under $\rho = 0.9$, for 15 replicas.

(b) IRISLOG vs. MAJORITY, under $\rho = 0.9$.

(c) IRISLOG vs. MAJORITY, under $\rho = 0.9$.

Figure 8: Inconsistency and unavailability of IRISLOG and comparison with MAJORITY.

IRISLOG must wait a relatively long time (i.e., MTTR$/s$ time on average) for one of these hosts to recover. An important observation, however, is that while larger ρ's have a devastating effect on the availability of MAJORITY, they have only a modest effect on the consistency (and availability) of IRISLOG. The curve for $\rho = 0.9$ is still close to the (idealized) curve where there are no host failures (and hence no dependence on ρ) and the only source of inconsistency is reachability mismatches. Even when $\rho = 1.0$, the curve is still decreasing roughly linear in our log-scale graph. We also find that these results are fairly robust to different parameter settings. For example, with small s ($s \leq 12$), increasing the time between data refresh checks from 30 seconds to 10 minutes increase inconsistency only slightly, because MTTR$/s$ is still an order of magnitude larger than the refresh time.

6.1.3 Comparison

In this section, we focus on further comparing the IRISLOG replication design to MAJORITY. To make the benefits of IRISLOG pessimistic, we add up the unavailability and inconsistency in our system, and then compare the total with the unavailability of MAJORITY. IRISLOG has a tunable quorum size, which controls the amount of inconsistency. As studied above, increasing the quorum size decreases inconsistency but increases unavailability for a fixed-sized replica group. Figure 8(a) shows that the sum of inconsistency and unavailability varies with the quorum size s, and the global minimum may occur at some intermediate value for s.

To compare our design against MAJORITY for the same replica group sizes, we choose the quorum size that minimizes the sum of inconsistency and unavailability. Figure 8(b) shows that when the replica group size n is small (less than 10), MAJORITY does better than IRISLOG. This is because the inconsistency in IRISLOG is relatively high with such small quorum sizes, and, hence, the sum of inconsistency and unavailability of IRISLOG exceeds the unavailability of MAJORITY. However, the benefit of our approach increases with n, and significantly outperforms MAJORITY for larger n. For example, with 23 replicas and a quorum size of 7, IRISLOG can achieve inconsistency plus unavailability of 1.2e-5, while MAJORITY is unable to achieve 1.2e-5 unavailability regardless of n. Under $\rho = 0.95$, we have similar observation that with $n = 15$ and $s = 4$, IRISLOG can achieve inconsistency plus unavailability of 1.1%, which is not achievable by MAJORITY under any n. We can

Table 1: Trace of user queries from 10Nov2003–15Dec2003 and 5Mar2004–20May2004 for the PLAB-MONITOR service deployed on 310 PlanetLab nodes.

Total queries	1576 (100%)
Queries selecting a complete subtree	1563 (99.2%)
Queries selecting all nodes	114 (7%)
Queries selecting a country	191 (12%)
Queries selecting a region	839 (53%)
Queries selecting a site	381 (24%)
Queries selecting a machine	38 (3%)
Queries not selecting a complete subtree	13 (0.8%)

also interpret the results in Figure 8(b) from another perspective. Namely, as long as our availability target is higher than 99% (99% is roughly where the curves cross), our approach has an advantage over the MAJORITY approach.

The above comparison clearly depends on the value of ρ. To understand the effects of ρ, we fix a target of 0.0001 for the sum of unavailability and inconsistency, and compare the replica group size needed in the two different designs to achieve the target. We find that approximately as long as $\rho \geq 0.8$, our design needs a small number of replicas. Furthermore, the larger the ρ value, the larger the advantage our approach has over MAJORITY. On the other hand, when ρ is smaller than 0.8, the availability improvement in our design can no longer offset the amount of inconsistency. One way to interpret these results is that as long as we want to overprovision against $\rho \geq 0.8$, our system has a benefit over MAJORITY. To make this correlation threshold more concrete, we note that when $\rho = 0.8$ roughly half of the failure events cause at most 3 failures (out of the 200 hosts).

Figure 8(b) does not yet fully demonstrate the benefit of our approach. MAJORITY with a replica group of size n always has a quorum size of $n/2$. In IRISLOG, the quorum size can be smaller than $n/2$. The overhead of reads and writes is purely determined by the quorum size. Thus, even under the same number of replicas, IRISLOG will incur less overhead for reads and writes. To quantify such benefits, Figure 8(c) plots the unavailability achieved by the two designs under the same quorum size s, where IRISLOG has even larger advantages over MAJORITY.

6.2 Fragmentation Algorithm Evaluation

6.2.1 Evaluation under Real Workload

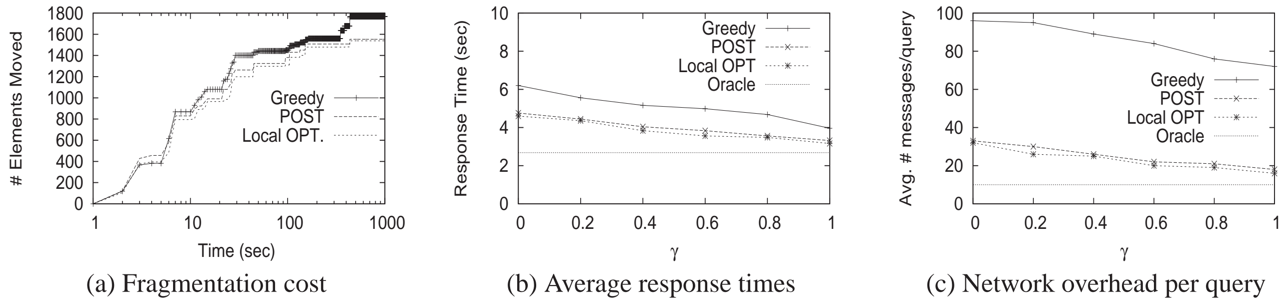
We evaluate the effectiveness of POST by running the PLAB-MONITOR service on Emulab [2]. We choose to use emulation for our evaluation so that we can compare different fragmentation algorithms under the same settings. Our emulation uses the same XML database and the same number (310) of hosts used by the PLAB-MONITOR service running on PlanetLab. The hosts are emulated using 60 Emulab nodes. For simplicity, we assign a 50ms latency between any two hosts. We use a real user query trace (Table 1) collected from our PLAB-MONITOR deployment to drive the emulation.

Each emulation experiment starts with a single host holding the entire XML database. The first half of the query trace is then injected to warm up the system and fragment the database. We next inject the second half of the trace and measure the end to end response times and network overheads of user queries. The timescale in the trace is compressed so that the average gap between two queries is 10 seconds.

We compare POST with three other algorithms. (1) In LOCAL OPT, each host partitions its XML fragment using an optimal tree partitioning algorithm [19] with $O(n^3)$ complexity. (2) In GREEDY, an overloaded host evicts individual XML elements in decreasing order of their loads. As a result, it makes finer granularity decisions, but does not try to keep the XML elements clustered. (3) ORACLEis an offline approach that takes the whole XML database and the query workload, and computes the optimal fragmentation (again using the algorithm in [19]). ORACLE cannot be used in a real system and only serves as a lower bound for us to compared against.

Fragmentation Overhead. Figure 9(a) plots the cumulative overhead of the fragmentation algorithms over time during the warm-up phase of the experiment. The overhead is measured as the number of XML elements transferred over the network due to load shedding. The graph shows several points. First, the amount of fragmentation decreases over time, which means that our load-based invocations of all the algorithms do converge under this workload. Second, POST incurs slightly higher overhead than LOCAL OPT. This is expected since POST selects complete subtree whose size may be slightly larger necessary. GREEDY incurs higher overhead than both POST and LOCAL OPT, which is surprising given that GREEDY chooses the fragments in a smaller granularity. This is explained by the fact that GREEDY's non-clustering fragmentation increases the overall system load which makes the hosts fragment more often than in the other algorithms.

Fragmentation Quality. Figure 9(b) compares different fragmentation algorithms in terms of the average query response time during the second phase of our experiment. As mentioned in Section 5, we try the algorithms with the objective of maximizing $(1-\gamma) * cut_{external} - \gamma * cut_{internal}$ and vary the value of γ from 0 to 1. Figure 9(a) shows a number of interesting points. First, our $O(n)$ time POST algorithm provides an average response time very close to the $O(n^3)$ time LOCAL OPT algorithm and also within twice the lower bound. GREEDY performs badly since it does not aim to keep the XML elements clustered, which results in more hops between the hosts. Second, the best response



(a) Fragmentation cost (b) Average response times (c) Network overhead per query

Figure 9: Plots comparing different fragmentation algorithms

times for GREEDY, POST and LOCAL OPT are found when $\gamma = 1$. Intuitively, this is because moving a fragment from one host to another translates its $cut_{internal}$ to additional subqueries. Since $\gamma = 0$ ignores $cut_{internal}$, it may increase the global load significantly higher than using $\gamma = 1$ that tries to minimize this additional load.

Figure 9(c) shows that POST and LOCAL OPT also have similar per query network overhead. GREEDY performs the worst, because it generates small fragments and user queries thus need to incur more network hops to access all the relevant XML elements.

6.2.2 Evaluation under Additional Synthetic Workload

To understand the sensitivity of POST to different properties of the workload, we simulate IRISLOG in a customized simulator under synthetic workload. We use 850 simulated hosts where each host has a GNP [24] network coordinate computed from a set of real Internet hosts [24]. The simulated network latency between two hosts is determined by the Cartesian distance of their GNP coordinates. The CPU processing time at each host, which depends on the size of the XML fragment on the host, is set as our micro-benchmark results obtained when running real IRISLOG code on a 2.7GHz Pentium IV machine having 1 GB RAM and running Redhat Linux.

The global XML databases in the workloads have hierarchies similar to that used by the PLAB-MONITOR service. Each set of synthetic workload contains 10,000 queries with the same probability distribution as those in PLAB-MONITOR of starting at specific levels of the hierarchy. The queries are injected one second apart from each other. Our simulation varies three aspects of the workload: database size, read-write ratio, and read burstiness. The *database size* of a workload is the number of XML elements in the XML database, with a default value of 1000. We vary the *read-write ratio* (default value is 0.002) of the workload by changing the write rates of the sensors. We define a *read burst* as a group of 100 queries issued 0.1 second apart from each other. *Read-burstiness* of the workload is the probability (default 0) that a read burst starts at the beginning of every minute.

Figure 10 plots query response time under the four fragmentation algorithms as the previous three parameters vary. All algorithms here use a γ value of 1. As expected, average query response time increases with the database size (Figure 10(a)), since larger databases mean that more hosts are involved in a query. Query response time also increases when the workload is more write-intensive (Figure 10(b)), because a higher rate of writes results in more fragmentation. The same is true when there are more read bursts (Figure 10(c)). Even though the absolute differences among the four algorithm vary across the parameter values, the the general trend between the algorithms remains consistent with our previous results under the real PLAB-MONITOR workload. Namely, GREEDY always performs much worse than the other algorithms; while POST performs very close to LOCAL OPT.

6.3 Performance Evaluation in a WAN

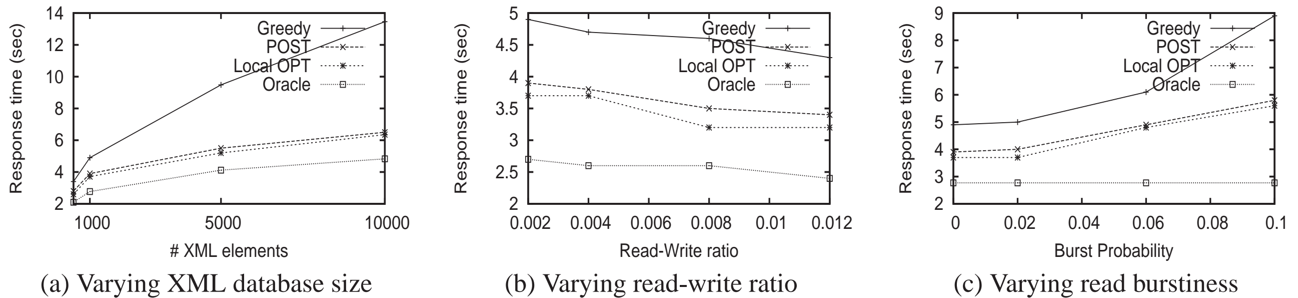
This section reports the performance of two monitoring and actuation services we have written on IRISLOG and deployed on PlanetLab.

6.3.1 Infrastructure Resource Usage Monitor

PLAB-MONITOR is a publicly available service (see http://www.intel-iris.net/irislog) built on IRISLOG to monitor the current resource usage (on a per node and per slice [15] basis) of PlanetLab. It is deployed on 310 PlanetLab hosts, organized in a hierarchy shown in Figure 1. Only around 240 nodes were available at the time of our experiments and the response times reported here include the timeout values for the dead nodes and their replicas.

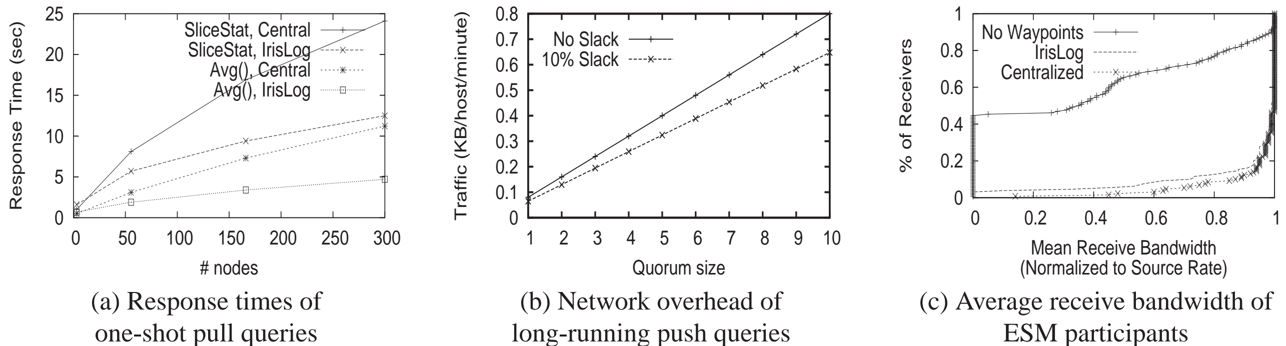
Figure 11(a) plots the end to end response times of two pull queries in IRISLOG and compares the results with a centralized scheme that pulls all data to a centralized server for processing. The first query `Avg()` computes the average available disk space of the nodes selected by the query,

[15] A slice in PlanetLab comprises a network of virtual machines spanning some set of physical nodes.



(a) Varying XML database size (b) Varying read-write ratio (c) Varying read burstiness

Figure 10: Plots showing the effects of different workload parameters on fragmentation algorithms



(a) Response times of one-shot pull queries (b) Network overhead of long-running push queries (c) Average receive bandwidth of ESM participants

Figure 11: Plots showing the performance of two IRISLOG monitoring services on a WAN.

and, thus, requires small intermediate results during the in-network aggregation. The second query `SliceStat` computes the average bandwidth usage of each active slice in PlanetLab. It maintains one aggregate for each slice and requires large intermediate results. The different number of nodes on the *x* axis represents the average number of nodes in a subtree rooted at different levels. The response times reported here are conservative because many PlanetLab nodes were extremely overloaded during these experiments. Figure 11(a) demonstrate the benefits of in-network aggregation in IRISLOG by spreading the network overhead and computational load across many hosts.

Figure11(b) plots the overhead of using replication in a long-running push query `Avg()`. As expected, the network overhead increases roughly linearly with the quorum size. One way to reduce this overhead is to use slack in the aggregation (e.g., pushing the aggregated value up only when it is changed by at least 10% from the previously pushed up value) as shown in the figure.

6.3.2 Application Self Monitoring and Actuation

ESM-MONITOR is a service built on IRISLOG to monitor and actuate an End System Multicast (ESM) [15] deployment, an operational Internet broadcast system based on overlay multicast. The ESM developers have noted that in many real broadcasts, participants do not have enough resources to construct a good multicast tree [14]. In such situations, ESM uses well-provisioned infrastructure hosts called *waypoints* to form a better tree. Currently, ESM uses a central server to monitor the resource usage and demand of the participants and to trigger the creation of waypoints. However, this solution does not scale (an event reported in [27] had 74,000 concurrent participants) and form a single point of failure.

To use IRISLOG for distributed monitoring and actuation in ESM, we modify ESM clients so that they report usage statistics to IRISLOG. Triggers are then installed in the intermediate nodes of the IRISLOG hierarchy which contact actuators to deploy waypoint when the need arises. The total modification to ESM required less than 300 lines of code.

Our experiment uses 140 PlanetLab hosts to emulate the behavior of 250 ESM participants (i.e., maximum 140 concurrent participants), each of which is given the properties (resource usage, dynamics etc.) of a participant in one hour snapshot of the real ESM broadcast trace (`Slashdot trace` [14]). We use 20 PlanetLab hosts as waypoints. Figure 11(c) plots the CDF of the participants sorted in increasing order of their receive bandwidth. It shows that ESM-MONITOR performs very close to the centralized solution, which means that the delay caused by the multi-level aggregation tree in ESM-MONITOR is minimal. On the other hand, ESM-MONITOR will not suffer from the availability and scalability problems of a centralized solution. As the

graph shows, server failure in the centralized solution (`No Waypoints`) can result in 40% of the participants receiving no data and 80% of the participants receiving less than 75% of the required bandwidth.

7 Conclusion

Motivated by the need for highly robust wide-area monitoring services, this paper discusses how we tolerate correlated failures in the context of IRISLOG, a customizable wide-area monitoring service we developed. Specifically, IRISLOG uses a replication design based on Signed Quorum Systems to tolerate untargeted correlated failures, and a novel efficient XML database-fragmentation algorithm to avoid targeted correlated failures caused by replica overload. Through analytical study, extensive simulation and results derived from a real world deployment, we demonstrate that IRISLOG is able to achieve significantly higher availability than the best previous designs, and that its advantage increases with increasing levels of correlation.

References

[1] Akamai web site. http://www.akamai.com/.

[2] Emulab - network emulation testbed home. http://www.emulab.net.

[3] Planetlab. http://www.planet-lab.net/.

[4] XML path language (XPATH). http://www.w3.org/TR/xpath.

[5] ANDERSEN, D., BALAKRISHNAN, H., KAASHOEK, F., AND MORRIS, R. Resilient Overlay Networks. In *Proc. SOSP* (2001).

[6] BAKKALOGLU, M., WYLIE, J., WANG, C., AND GANGER, G. On Correlated Failures in Survivable Storage Systems. Tech. rep., CMU, May 2002. SCS Technical Report CMU-CS-02-129.

[7] BARBARA, D., AND GARCIA-MOLINA, H. The Reliability of Voting Mechanisms. *IEEE Trans. Comput.* (October 1987), 1197–1208.

[8] BOLOSKY, W. J., DOUCEUR, J. R., ELY, D., AND THEIMER, M. Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs. In *SIGMETRICS* (2000).

[9] CHEN, S., GIBBONS, P. B., AND NATH, S. Stored Procedures for Distributed XML Databases. Tech. rep., Intel Research Pittsburgh, 2004.

[10] CUI, W., STOICA, I., AND KATZ, R. H. Backup Path Allocation Based on a Correlated Link Failure Probability Model in Overlay Networks. In *ICNP* (2002).

[11] DESHPANDE, A., NATH, S., GIBBONS, P. B., AND SESHAN, S. Cache-and-query for wide area sensor databases. In *ACM SIGMOD* (2003).

[12] F. P. JUNQUEIRA, K. M. Designing Algorithms for Dependent Process Failures. In *Proceedings of International Workshop on Future Directions in Distributed Computing* (2002).

[13] HERLIHY, M., AND WING, J. Linearizability: A Correctness Condition for Concurrent Objects. *ACM Transactions on Programming Languages and Systems 12*, 3 (July 1990).

[14] HUA CHU, Y., GANJAM, A., NG, T. S. E., RAO, S. G., SRIPANIDKULCHAI, K., ZHAN, J., AND ZHANG, H. Early experience with an internet broadcast system based on overlay multicast. In *Proceedings of Usenix Annual Technical Conference* (2004).

[15] HUA CHU, Y., RAO, S. G., SESHAN, S., AND ZHANG, H. A case for end system multicast. *IEEE Journal on Selected Areas in Communication (JSAC), Special Issue on Networking Support for Multicast 20*, 8 (October 2002).

[16] HUEBSCH, R., HELLERSTEIN, J. M., LANHAM, N., LOO, B. T., SHENKER, S., AND STOICA, I. Querying the internet with PIER. In *Proceedings of VLDB* (Germany, 2003).

[17] KARYPIS, G., AND KUMAR, V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing 20*, 1 (1998), 359–392.

[18] LAMPORT, L. The Part-Time Parliament. *ACM Transactions on Computer Systems 16* (May 1998), 133–169.

[19] LUKES, J. A. Efficient algorithm for the partitioning of trees. *IBM Journal of Research and Development 18*, 3 (1974), 217–224.

[20] LYNCH, N. *Distributed Algorithms*. Morgan Kaufmann, 1997.

[21] LYNCH, N., AND SHVARTSMAN, A. RAMBO: a reconfigurable atomic memory service for dynamic networks. In *DISC* (2002).

[22] MASSIE, M. L., CHUN, B. N., AND CULLER, D. E. The ganglia distributed monitoring system: Design, implementation, and experience. To appear in Parallel Computing.

[23] NATH, S., KE, Y., GIBBONS, P. B., KARP, B., AND SESHAN, S. Irisnet: An architecture for enabling sensor-enriched internet service. Intel Research Pittsburgh Technical Report IRP-TR-03-04, 2003.

[24] NG, T. S. E., AND ZHANG, H. Predicting internet network distance with coordinates-based approaches. In *INFOCOM* (2002).

[25] OPPENHEIMER, D., VATKOVSKIY, V., WEATHERSPOON, H., LEE, J., PATTERSON, D. A., , AND KUBIATOWICZ, J. Monitoring, analyzing, and controlling internet-scale system with ACME. UC Berkeley Technical Report UCB//CSD-03-1276, October 2003.

[26] ROSCOE, T., PETERSON, L., KARLIN, S., AND WAWRZONIAK, M. A simple common sensor interface for planetlab. PlanetLab Design Notes PDN-03-010.

[27] SRIPANIDKULCHAI, K., GANJAM, A., MAGGS, B., AND ZHANG, H. The feasibility of supporting large-scale live streaming applications with dynamic application end-points. In *ACM SIGCOMM* (2004).

[28] SZE, C. N., AND WANG, T.-C. Optimal circuit clustering for delay minimization under a more general delay model. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 22*, 5 (2003), 646–652.

[29] THOMAS, R. H. A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases. *ACM Transactions on Database Systems 4* (1979), 180–209.

[30] VAN RENESSE, R., BIRMAN, K. P., AND VOGELS, W. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems 21*, 2 (May 2003), 164–206.

[31] WAWRZONIAK, M., PETERSON, L., AND ROSCOE, T. Sophia : An information plane for networked system. In *Hotnets-II* (2003).

[32] WEATHERSPOON, H., MOSCOVITZ, T., AND KUBIATOWICZ, J. Introspective Failure Analysis: Avoiding Correlated Failures in Peer-to-Peer Systems. In *Proceedings of International Workshop on Reliable Peer-to-Peer Distributed Systems* (October 2002).

[33] YALAGANDULA, P., AND DAHLIN, M. A Scalable Distributed Information Management System. In *ACM SIGCOMM* (August 2004).

[34] YU, H. Overcoming the Majority Barrier in Large-Scale Systems. In *DISC* (2003).

[35] YU, H. Signed Quorum Systems. In *PODC* (2004).

[36] YU, H., AND VAHDAT, A. The Costs and Limits of Availability for Replicated Services. In *SOSP* (2001).

[37] YU, H., AND VAHDAT, A. Consistent and Automatic Replica Regeneration. In *NSDI* (2004).

A Derivation of Equation 1

In this appendix, we will derive equation 1 in Section 4.6.2. Recall that $P(j,n)$ is the probability that a failure event causes exactly j failures in a replica group of n hosts ($0 \leq j \leq n \leq u$), for a given ρ ($0 < \rho < 1$).

Let p_i be the probability that a failure event causes i failures in a universe of u hosts. In our tunable correlation model (Section 4.6.1), $p_i = \frac{1-\rho}{\rho(1-\rho^u)}\rho^i$. Let $P(i,j,n)$, where $j \leq i \leq u$, be the probability that a failure event causes exactly i failures in the universe and causes exactly j failures in the replica group of n hosts. Then $P(j,n) = \sum_{i=j}^{u} P(i,j,n)$. We observe that if there are j failures out of a replica group of size n, then there are $n-j$ non-failures in the replica group, and hence at most $u-(n-j)$ failures in the universe. Thus $P(i,j,n) = 0$ whenever $i > u-(n-j)$. This gives $P(j,n) = \sum_{i=j}^{u-n+j} P(i,j,n)$.

We will now derive an equation for $P(i,j,n)$. Each $P(i,j,n)$ is the product of (1) the probability p_i that a failure event causes i failures and (2) the probability, call it $Q(i,j,n)$, that such a failure event causes j failures in the replica group. We can analyze $Q(i,j,n)$ by considering all the $\binom{u}{i}$ possible ways of selecting i out of u hosts for failure, and how many of these select exactly j out of the n replicas. There are $\binom{n}{j}$ ways of selecting j out of n and for each of these, there are $\binom{u-n}{i-j}$ ways of selecting the remaining $i-j$ failures out of the $u-n$ hosts not in the replica group. Because all $\binom{u}{i}$ ways are equally likely, we have:

$$Q(i,j,n) = \frac{\binom{n}{j} \cdot \binom{u-n}{i-j}}{\binom{u}{i}}$$

Thus:

$$P(i,j,n) = p_i \cdot Q(i,j,n) = \frac{1-\rho}{\rho(1-\rho^u)}\rho^i \cdot \frac{\binom{n}{j} \cdot \binom{u-n}{i-j}}{\binom{u}{i}}$$

It follows that:

$$P(j,n) \quad = \quad \binom{n}{j} \frac{1-\rho}{\rho(1-\rho^u)} \sum_{i=j}^{u-n+j} \rho^i \cdot \frac{\binom{u-n}{i-j}}{\binom{u}{i}}$$

We can simplify this by pulling the common ρ^j term out from inside the sum and letting $k = i-j$ to get:

$$P(j,n) = \binom{n}{j} \frac{(1-\rho)\rho^{j-1}}{1-\rho^u} \sum_{k=0}^{u-n} \rho^k \frac{\binom{u-n}{k}}{\binom{u}{k+j}}$$

This completes the derivation of equation 1.