USENIX Association

Proceedings of HotOS IX: The 9th Workshop on Hot Topics in Operating Systems

Lihue, Hawaii, USA
May 18–21, 2003

The *Phoenix* Recovery System: Rebuilding from the ashes of an Internet catastrophe

Flavio Junqueira Ranjita Bhagwan Keith Marzullo Stefan Savage Geoffrey M. Voelker
Department of Computer Science and Engineering
University of California, San Diego

1 Introduction

The Internet today is highly vulnerable to *Internet catastrophes*: events in which an exceptionally successful Internet pathogen, like a worm or email virus, causes data loss on a significant percentage of the computers connected to the Internet. Incidents of successful wide-scale pathogens are becoming increasingly common on the Internet today, as exemplified by the Code Red and related worms [6], and LoveBug and other recent email viruses [11]. Given the ease with which someone can augment such Internet pathogens to erase data on the hosts that they infect, it is only a matter of time before Internet catastrophes occur that result in large-scale data loss.

In this paper, we explore the feasibility of using data redundancy, a model of dependent host vulnerabilities, and distributed storage to tolerate such events. In particular, we motivate the design of a cooperative, distributed remote backup system called the *Phoenix* recovery system. The usage model of *Phoenix* is straightforward: a user specify an amount F of bytes from its disk space the system can use, and the goal of the system is to protect a proportional amount F/k of its data using storage provided by other hosts.

In general, to recover the lost data of a host that was a victim in an Internet catastrophe, there must be copies of that data stored on a host or set of hosts that survived the catastrophe. A typical replication approach [10] creates t additional replicas if up to t copies of the data can be lost in a failure. In our case, t would need to be as large as the largest Internet catastrophe. As an example, the Code Red worm infected over 359,000 computers, and so t would need to be larger than 359,000 for hosts to survive a similar kind of event. Using such a large degree of replication would make cooperative remote backup useless for at least two reasons. First, the amount of data each user can protect is inversely proportional to the degree of replication, and with such a vast degree of replication the system could only protect a minuscule amount of data per user. Second, ensuring that such a large number of replicas are written would take an impractical amount of time.

Our key observation that makes *Phoenix* both feasible and practical is that an Internet catastrophe, like any large-scale Internet attack, exploits shared vulnerabilities. Hence, users should replicate their data on hosts that do not have the same vulnerabilities. That is, the replication mechanism should take the dependencies of host failures—in this case, host diversity—into account [5]. Hence, we formally represent host attributes, such as its operating system, web browser, mail client, web server, etc. The system can then use the attributes of all hosts in the system to determine how many replicas are needed to ensure recoverability, and on which hosts those replicas should be placed, to survive an Internet catastrophe that exploits one of its attributes. For example, for hosts that run a Microsoft web server, the system will avoid placing replicas on other hosts that run similar servers so that the replicas will survive Internet worms that exploit bugs in the server. Such a system could naturally be extended to tolerate simultaneous catastrophes using multiple exploits, although at the cost of a reduced amount of recoverable data that can be stored. Using a simulation model we show that, by doing informed placement of replicas, a *Phoenix* recovery system can provide highly resilient and available cooperative backup with low overhead.

In the rest of this paper, we discuss various approaches for tolerating Internet catastrophes and motivate the use of a cooperative, distributed recovery system like *Phoenix* for surviving them. Section 3 then describes our model for dependent failures and how we apply it to tolerate catastrophes. In Section 4, we explore the design space of the amount of available storage in the system and the redundancy required to survive Internet catastrophes under various degrees of host diversity and shared vulnerabilities. We then discuss system design issues in Section 5. Finally, Section 6 concludes the paper.

2 Motivation

Backups are a common way to protect data from being lost as a result of a catastrophe. We know of three approaches to backup.

Local backup is the most common approach for recovering from data loss, and it has many advantages. Users and organizations have complete control over the amount and frequency with which data is backed up. Furthermore, tape and optical storage are inexpensive, high capacity devices. However, large organizations that have

large amounts of data have to employ personnel to provide the backup service. Individual home users often do not use it because of the time and hassle of doing so, causing home systems to be highly vulnerable to exploit and potential data loss.

Another approach is to use a *commercial remote backup* service, such as DataThought Consulting [4] or Protect-Data.com [9]. This approach is convenient, yet expensive. Currently, automatic backup via a modem or the Internet for 500MB of data costs around $30-$125 a month.

Cooperative remote backup services provide the convenience of a commercial backup service but at a more attractive price. Instead of paying money, users relinquish a fraction of their computing resources (disk storage, CPU cycles for handling requests, and network bandwidth for propagating data). pStore [1] is an example of such a service. However, its primary goal is to tolerate local failures such as disk crashes, power failures, etc. Pastiche [2] also provides similar services, while trying to minimize storage overhead by finding similarities in data being backed up. Its aim is also to guard against localized catastrophes, by storing one replica of all data in a geographically remote location.

We believe that a cooperative, distributed system is a compelling architecture for providing a convenient and effective approach for tolerating Internet catastrophes. It would be an attractive system for individual Internet users, like home broadband users, who do not wish to pay for commercial backup service or do not want the hassle of making their own local backups frequently. Users of *Phoenix* would not need to exert any significant effort to backup their data. Specifying what data to protect can be made as easy as specifying what data to share on a file sharing peer-to-peer system. Further, a cooperative architecture has little cost in terms of time and money; instead, users relinquish a small fraction of their computer resources to gain access to a highly resilient backup service. A user specifies an amount F of bytes from its disk space to be used by the system, and the system would protect a proportional amount F/k of its data. We observe that the value k depends on the host diversity, and can differ among the hosts. In addition, the system would limit the network bandwidth and CPU utilization to minimize the impact of the service on normal operation.

To our knowledge, *Phoenix* is the first effort to build a cooperative backup system resilient to wide-scale Internet catastrophes.

3 Taking Advantage of Diversity

Traditionally, reliable distributed systems are designed using the threshold model: out of n components, no more than $t < n$ are faulty at any time. Although this model can always be applied when the probability of having a total failure is negligible, it is only capable of expressing the worst-case failure scenario, and it is best suited when failures are independent and identically distributed. The worst-case, however, can be one in which the failures of components are highly correlated.

Failures of hosts in a distributed system can be correlated for several reasons. Hosts may run the same code or be located in the same room, for example. In the former case, if there is a vulnerability in the code, then it can be exploited in all the hosts executing the target software. In the latter case, a power outage can crash all machines plugged into the same electrical circuit.

As a first step towards the design of a cooperative backup system for tolerating catastrophes, we need a concise way of representing failure correlation. We use the *core* abstraction to represent correlation among host failures [5]. A core is a reliable minimal subset of components: the probability of having all hosts in a core failing is negligible, for some definition of negligible. In a backup system, a core corresponds to the minimal replica set required for resilience.

Determining the cores of a system depends on the failure model used and the desired degree of resilience for the system. The failure model prescribes the possible types of failures for components. These types of failures determine how host failures can be correlated. In our case, hosts are the components of interest and software vulnerabilities are the causes of failures. Consequently, hosts executing the same piece of software present high failure correlation. This information on failure correlation is not sufficient, however, to determine the cores of a system. It also depends on the desired degree of resilience. As one increases the degree of resilience, more components are perhaps necessary to fulfill the core property stated above.

To reason about the correlation of host failures, we associate attributes to hosts. The attributes represent characteristics of the host that can make it prone to failures. For example, the operating system a host runs is a point of attack: an attack that targets Linux is less likely to be effective against hosts running Solaris, and is even less effective against hosts running Windows XP. We could represent this point of attack by having an n-ary attribute that indicates the operating system, where the value of the attribute is 0 for Linux, 1 for Windows XP, 2 for Solaris, and so on. Throughout this paper, we use A as the set of attributes that characterize a host.

To illustrate the concepts introduced in this section, consider the system described in Example 3.1. In this system, hosts are characterized by three attributes and each attribute has two possible values. We assume that hosts fail due to crashes caused by software vulnerabilities, and at most one vulnerability can be exploited at a time. Note

that the cores shown in the example have maximum resilience according to the given set of attributes.

Example 3.1 :

Attributes: *Operating System* = {*Unix, Windows*};
Web Server = {*Apache, IIS*};
Web Browser = {*IE, Netscape*}.

Hosts: H_1 = {*Unix, Apache, Netscape*};
H_2 = {*Windows, IIS, IE*};
H_3 = {*Windows, IIS, Netscape*};
H_4 = {*Windows, Apache, IE*}.

Cores = $\{H_1H_2, H_1H_3H_4\}$.

There are a few interesting facts to be observed about Example 3.1. First, H_1 and H_2 form what we call an *orthogonal core*, which is a core composed of hosts that have different values for every attribute. Note that in this case the size of the orthogonal core is two because of our assumption that at most one vulnerability can be exploited at a time. This implies that it is necessary and sufficient to have two hosts with different values for every attribute. Even though it is not orthogonal, $H_1H_2H_3$ is also a core since it covers all attributes. Second, when choosing a core for host H_1 to store replicas of its data, there are two possibilities: H_1H_2 and $H_1H_3H_4$. The second option for a core is larger than the first. Thus, choosing the second leads to unnecessary replication. The optimal choice in terms of storage overhead is therefore H_1H_2.

Choosing a smallest core may seem a good choice at first because it requires fewer replicas. We observe, however, that such a choice can adversely impact the system. In environments with highly skewed diversity, the total capacity of the system may be impacted by always choosing the smallest core[1]. Back in Example 3.1, H_1 is the only host which has some flavor of Unix as the operating system. Consequently, a core for every other host has to contain H_1. For a small system as the one in the example this should not be a problem, but it is a potential problem for large-scale deployments. This raises the question of how host diversity impacts on storage overhead, storage load, and resilience. We address this question in the next section.

4 Host Diversity

We now develop a metric for specifying attribute diversity among a set of hosts, and a system model for representing sets of hosts with various degrees of host diversity. We then use this model to quantify the core sizes, and hence the amount of replication, required to achieve high degrees of resilience to Internet catastrophes under a wide range of diversities of host vulnerabilities.

[1]By skewed diversity, we mean a distribution of attribute configurations that is not uniform.

4.1 Diversity and Core Sizes

If one knew the probability of attack for each vulnerability, then given a target system resilience one could enumerate cores with that target resilience. In our case, it is not clear how one would determine such probabilities. Instead, we define a core c for a host p to be a minimal set of hosts with the following additional properties: 1) $p \in c$; 2) for every attribute $a \in A$, either there is a host in c that differs from p in the value of a or there is no host in the system that differs from p in the value of a. Such a subset of hosts is a core for a host p if we assume that, in any Internet catastrophe, an attack targets a single attribute value. Although it is not hard to generalize this definition to allow for attacks targeted against multiple attribute values, in the rest of this paper we focus on attacks against a single attribute value.

Smaller cores means less replication, which is desirable for reducing storage overhead. A core will contain between 2 and $|A| + 1$ hosts. If the hosts' attributes are well distributed, then the cores will be small on average: for any host p, it is likely that there is a host q that has different values of each of the attributes, and so p and q constitute an orthogonal core. That is, a fair number of orthogonal cores are likely to exist. If there is less diversity, though, then the smallest cores may not be orthogonal for many hosts, thus increasing storage overhead.

A lack of diversity, especially when trying to keep core sizes small, can lead to a more severe problem. Suppose there are k hosts $\{p_1, p_2, \ldots p_k\}$ and an attribute a such that all have the same value for a. Moreover, there is only one host p that differs in the value of a. A core for each host p_i hence contains p, meaning that p will maintain copies for all of the p_i. Since the amount of disk space p donates for storing backup data is fixed, each p_i can only use $1/k$ of this space. In other words, if p donates F bytes for common storage to the system, then each p_i can back up only F/k bytes. Note that k can be as large as the number of hosts, and so F/k can be minuscule. In Example 3.1, host H_1 is the only one to have a different value for attribute "Operating System", and hence has to store copies for all the other hosts.

Characterizing the diversity of a set of hosts is a challenging task. In particular, considering all possible distributions for attribute configurations is not feasible. Instead, we define a measure f that condenses the diversity of a system into a single number. According to our definition, a system with diversity f is one in which a share f of the servers is characterized by a share $(1 - f)$ of the combinations of attributes. Although this metric is coarse and does not capture all possible scenarios, it is expressive enough to enable one to observe how the behavior of a backup system is affected by skewed diversity. Note that f is in the interval $[0.5, 1)$. The value $f = 0.5$ corre-

sponds to a uniform distribution, and a value of f close to 1 indicates a highly skewed diversity.

We use this metric to study how storage overhead, storage load, and resilience vary with skew in diversity. We define the storage overhead of a host p as the size of the core that p uses to backup its data. Host p maintains copies for k other hosts. We define the storage load of p to be such a value k. Note that storage load may vary among the hosts. Thus, we define the storage load of the system as the maximum value of k across all the hosts. In the remainder of this paper, we refer to the storage load of the system as just storage load. Finally, resilience depends on the number of attributes covered in a core, and it decreases as the number of non-covered attributes increases. We then define the resilience of the system for a host p as the percentage of attributes covered by the core c that p uses to backup its data.

The problem of finding a smallest core given a set of hosts and an attribute configuration for each host, however, is NP-hard (reduction from SET-COVER). For this reason, we used a randomized heuristic to find cores. This heuristic finds a core for a host p as follows:

1. It tries to find other hosts that have a fully disjoint set of attributes. If there is more than one host, then it picks one randomly;

2. If it finds no host in the previous step, then it randomly chooses hosts that have at least one different attribute until a core is constructed or there are no hosts left to choose.

This heuristic may not be the best; We have not yet done a thorough study of heuristics for finding cores. The results we present below, however, indicates that it is efficient in terms of storage overhead and resilience.

4.2 Modeling Diversity

To better understand the impact of diversity skew, we simulate a system of hosts with various attributes . On the Internet, most hosts run some version of Windows with Internet Explorer as the web browser [8], so we biased the attribute distribution towards having some fixed subset of attributes. The size of this subset depends on the value f chosen for the diversity of the system. To see this, consider a subset of size α. Assuming that each attribute has y possible values, for such a subset the total number of distinct configurations is $y^{|A|-\alpha}$. Thus, there is some integer α, $\alpha \leq |A|$, that satisfies the following equation:

$$\frac{y^{|A|-\alpha}}{y^{|A|}} \geq (1-f) > \frac{y^{|A|-(\alpha+1)}}{y^{|A|}}$$

$$\rightarrow \quad \frac{1}{y^{\alpha}} \geq (1-f) > \frac{1}{y^{\alpha+1}}. \qquad (1)$$

In our simulations, we compute the value of α using Equation 1, and then pick a subset $A' \subseteq A$ of attributes such that $|A'| = \alpha$. For every attribute a in A', we fix the value of a for a fraction f of the hosts. We then randomly choose values for the remaining attributes for this fraction of hosts. For the remaining hosts, we pick attribute configurations at random, but we make sure that each configuration is not a configuration of any host in the first fraction.

4.3 Simulation Results

Figures 1, 2, and 3 show the results of our simulations. We simulate a system of 1,000 hosts and present results for two scenarios: 8 attributes with 2 values each (8/2) and 8 attributes with 4 values each (8/4). The choice of 8 attributes is based upon an examination of the most targeted categories of software from public vulnerability databases, such as [7, 11]. From these databases, we observed 8 significant software categories and chose this value as a reasonable parameter for our simulations.

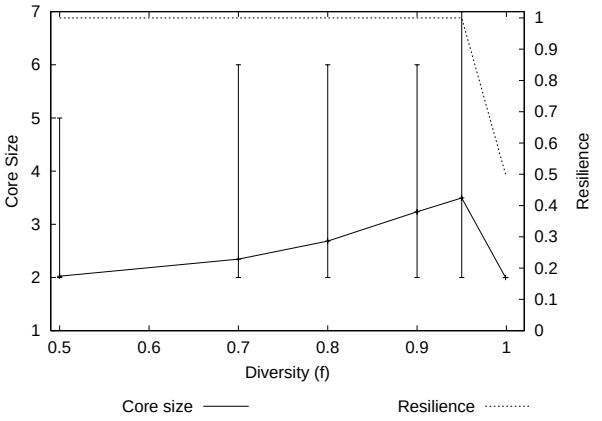

Figure 1: Core sizes as a function of diversity for 8 attributes, 2 values each.

We chose two different numbers of values per attribute to 1) explore a worst case and 2) show how core size and storage load benefit as a result of more fine-grained attribute configurations. The choice of 2 values per attribute corresponds to the coarsest division of hosts possible (e.g., Windows vs. Linux), and represents the worst case in terms of core size and storage load. We note that no vulnerabilities exploited by Internet pathogens have been so extreme, and that pathogens tend to exploit vulnerabilities at finer attribute granularities (e.g., Code Red and its variants exploited a vulnerability in Microsoft IIS running on Windows NT). The choice of 4 values per attribute represents a more fine-grained attribute configurations, and we use it to demonstrate how such configurations significantly improve core sizes and reduce storage load.

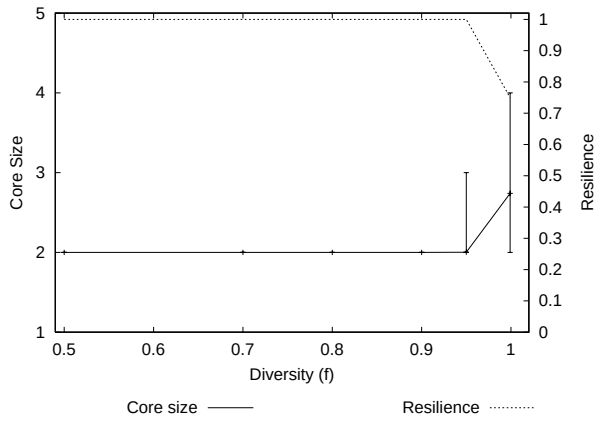

Figure 2: Core sizes as a function of diversity for 8 attributes, 4 values each.

We only show the results for one sample generated for each value of f, as we did not see significant variation across samples. Figures 1 and 2 show the core size averaged over cores for all of the hosts for different values of the diversity parameter f. We also include a measure of resilience that shows whether our algorithm was able to cover all attributes or not. A point in the resilience curve is hence the number of covered attributes, averaged over all hosts, divided by the total number of attributes. Note that resilience 1.0 means that all attributes are covered.

To show the variability in core size, we include error bars in the graphs showing the maximum and the minimum core sizes for values of f. The variability in core size is noticeably high in the 8/2 scenario, whereas it is lower in the 8/4 scenario. Because there are more configurations available in the 8/4 setting, it is likely that a host p finds a host q which has different values for every attribute even when the diversity is highly skewed.

Regarding the average core size, in the 8/2 scenario, it remains around 2 for values of f under 0.7, and goes up to average sizes around 3 for higher values of f. In either case, storage overhead is low, although it is overall higher than the average core size for the 8/4 scenario. The result of adding more attribute values to each attribute is therefore a reduction in storage overhead. In this scenario, the average core size remains around 2 for most of the values of f. It only increases for $f = 0.999$.

It is important to note that there is a drop in resilience for $f = 0.999$ in both scenarios. Observe that, for such a value of f, there are 999 hosts sharing some subset A' of attributes with a fixed value for each attribute and a single host p not sharing this subset. As a consequence, host p has to be in the core of a host q sharing A'. Host p, however, may not cover all the attributes of q. This being the case, there are possibly other hosts that cover the remaining attributes of q that p does not cover. If there are no such hosts, then there is no core for q which covers all attributes. The resilience for this host is therefore lower.

An important question that remains to be addressed is how much backup data a host will need to store. We address this question with the help of Figure 3. In this figure, the y-axis plots storage load. Thus, if $y = 10$, for example, there is a host p that must be in 10 cores given the core compositions that we computed, and every other host must be in 10 or fewer cores. As expected, storage load increases as f approaches 1, and reaches 1,000 for $f = 0.999$ in both scenarios. This is due to our previous observation that, for this value of f, there is a single host which has to be in the core of every other host. We conclude that the storage overhead for such a highly skewed diversity is small, but the total load incurred in a small percentage of the hosts can be very high.

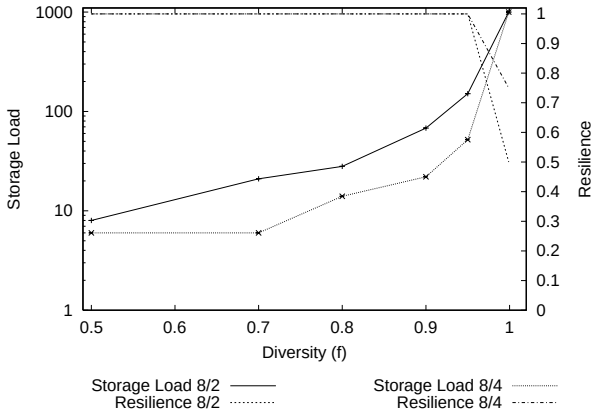

Figure 3: Storage load as a function of diversity for the 8/2 and the 8/4 scenarios.

Although we have presented results only for 1,000 hosts, we have also looked into other scenarios with a larger number of hosts. For 10,000 hosts and the same attribute scenarios, there is no reduction in resilience, and the average core size remains in the same order of magnitude. As we add more hosts to the system, we increase the probability of a host having some particular configuration, thus creating more possibilities for cores. The trend for storage load is the same as before: the more skewed the distribution of attribute configurations, the higher the storage load. For highly skewed distributions and large number of hosts, storage load can be extremely high. One important observation, however, is that as the population of hosts in the system increases, the number of different attribute configurations and the number of hosts with some particular configuration are likely to increase. Thus, for some scenario and fixed value of f, storage load does not increase linearly with the number of hosts. In our diversity model, it actually remains in the same order of magnitude.

Suppose now that we want to determine a bound on f

for a real system given our preliminary results. According to [8], over 93% of the hosts that access a popular web site run some version of Internet Explorer. This is the most skewed distribution of software they report (the second most skewed distribution is the percent of hosts running some version of Windows, which is 90%). There are vulnerabilities that attack all versions of Internet Explorer [11], and so f for such a collection of hosts can be no larger than 0.93. Note that as one adds attributes that are less skewed, they will contribute to the diversity of the system and reduce f.

In the lists provided by [8], there are 14 web browsers and 11 operating systems. For an idea of how a scenario like this would behave, consider a system of 1,000 hosts with 2 attributes and 14 values per attribute. For a value of $f = 0.93$ we have an average core size of 2, a maximum core size of 2, and storage load of 24. We did not see significant changes in these values when changing the number of values per attribute from 14 to 11.

A storage load of 24 means that there is some host that has to store backup data from 24 other hosts, or 4% of its storage to each host. We observe that this value is high because our heuristic optimizes for storage overhead. In an environment with such a skewed diversity, a good heuristic will have to take into account not only storage overhead, but the storage load of available hosts as well.

5 System Design Issues

The previous section gives us an idea of how much replication and how much storage is required in *Phoenix*. We end by briefly mentioning a number of design issues that an implementation of *Phoenix* needs to address as well.

The heuristics used for core identification need to use an index that maps hosts to the different attributes they possess. *Phoenix* therefore needs to maintain this index, which we intend to implement using a distributed hash table (DHT). Once *Phoenix* has identified a core, it stores copies of data on the hosts in the core. To ensure the integrity of the data, we plan on using some encryption mechanism. Thus, data is encrypted before releasing it to the hosts of a core. As observed in the previous section, it is also necessary to ensure fairness of storage allocation across users. For this, our heuristic to find cores will have to be modified to take storage load into account. Finally, we need to more carefully model the set of vulnerabilities and allow for dynamically adding and removing attributes/values.

In the wake of an Internet catastrophe, *Phoenix* itself has to continue functioning satisfactorily. Since we intend to use a DHT as a platform, it will need to survive a scenario where a large number of hosts suddenly leave the system [3]. Moreover, once there is a catastrophe, many users may try to recover files at the same time, potentially overloading the system; since recovery time is not critical, a distributed scheduler using randomized exponential wait times can ease recovery demand.

We are currently working on addressing these issues in a prototype design and implementation of *Phoenix*.

6 Conclusions

In this paper, we have explored the feasibility of using a cooperative remote backup system called *Phoenix* as an effective approach for surviving Internet catastrophes. *Phoenix* uses data redundancy, a model of dependent host failures, and distributed storage in a cooperative system. Using a simulation model we have shown that, by performing informed placement of replicas, *Phoenix* can provide highly reliable and available cooperative backup and recovery with low overhead.

References

[1] C. Batten, K. Barr, A. Saraf, and S. Treptin. pStore: A secure peer-to-peer backup system. Unpublished report, Dec. 2001.

[2] L. P. Cox and B. D. Noble. Pastiche: Making backup cheap and easy. In *Proceedings of Fifth USENIX Symposium on Operating Systems Design and Implementation*, Boston, MA, Dec. 2002.

[3] F. Dabek, M. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *proceedings of the 18th ACM Symposium on Operating System Principles (SOSP)* , Oct. 2001.

[4] Datathought website, http://www.datathought.com.

[5] F. Junqueira and K. Marzullo. Synchronous Consensus for dependent process failures. In *Proceedings of the ICDCS 2003*, pages 274–283, May 2003.

[6] D. Moore, C. Shannon, and J. Brown. Code-Red: A case study on the spread and victims of an Internet worm. In *Proceedings of the 2002 ACM SICGOMM Internet Measurement Workshop*, pages 273–284, Marseille, France, Nov. 2002.

[7] National Institute of Standards and Technology (NIST). ICAT vulnerability database. `http://icat.nist.gov/icat.cfm`.

[8] OneStat.com. Provider of web analytics. `http://www.onestat.com`.

[9] Protect-data website, http://www.protect-data.com.

[10] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. ACM Computing Surveys, Dec. 1990.

[11] SecurityFocus. Vulnerability database. `http://securityfocus.com`.