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

Flavio Junqueira Ranjita Bhagwan Keith Marzullo Stefan Savage
Geoff Voelker
University of California, San Diego Dept. of Computer Science & Eng.

January 10, 2003

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 [MS02] and LoveBug and other recent email viruses. Although they have caused severe denial of service costing billions of dollars [Com], we are fortunate that these kinds of pathogens have yet to cause significant data loss. However, 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, peer-to-peer remote backup system called the *Phoenix* recovery system, and we argue that *Phoenix* is a compelling architecture for providing a convenient and effective approach for tolerating Internet catastrophes for Internet users like home broadband users. The usage model of *Phoenix* is straightforward: users specify what percent F of their disk space the system can use, and the system protects a proportional amount F/k of their 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 [Sch90] 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 340,000 computers, and so t would need to be at least 340,000 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 vulnerabilites. That is, the replication mechanism should take the dependencies of host failures—in this case, host diversity—into account [JM03]. The system can formally represent host attributes, such as its operating system, web browser, mail client, web server, etc. For each user's data, 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. If necessary, the system can 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 reliable and available cooperative backup and recovery with low overhead: with as few as 2 replicas, the system can backup and recover at least the equivalent of 20% of storage contributed by each host in the system.

In the rest of this paper, we discuss various approaches for tolerating Internet catastrophes and motivate the use of a cooperative, peer-to-peer 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. Finally, Section 5 outlines our requirements for designing and implementing a prototype of *Phoenix.*

2 Motivation

There are two high-level approaches for tolerating Internet catastrophes, mitigating them and tolerating them.

1) Mitigating catastrophes. The first class of approaches focuses on mitigating the cause of the catastrophe itself. Prevention technologies [CPM$^+$98, WFBA00, Nec97] that increase system security and reduce vulnerabilities can limit the extent of a catastrophe by reducing the vulnerable population. Treatment technologies like virus detectors [Sym] and operating system update features [Mic] can also reduce the vulnerable population, as well as reduce the rate at which a catastrophe occurs. Finally, containment technologies like firewalls, content filters, and routing blacklists [MVS03] can significantly abate, or potentially halt, a catastrophe as it occurs.

Although these technologies are an important and valuable aspect of protecting the Internet against catastrophe, they are also unlikely to completely eliminate the Internet's vulnerability to catastrophe. Given current software engineering practices, widespread software vulnerabilities will persist for the foreseeable future and make prevention difficult. Treatment can be effective on a long-term basis, but is reactive in nature and primarily useful only after a catastrophe has occurred. Containment is a promising approach, but is challenging to implement with wide-spread effectiveness [MVS03].

2) Tolerating catastrophes. The second class of approaches uses redundancy to mask data loss caused by an Internet catastrophe. Rather than preventing hosts from becoming victims, they enable victims to survive the catastrophe. We know of three approaches in this class:

Local backups: 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, and tape and optical storage is both inexpensive and high capacity.

However, local backup does have its price in terms of the money and time required to perform it. Large organizations have large amounts of data and have to employ personnel to provide the backup service. Individual home users can afford the equipment and media for home backup, but often do not use it because of the time and hassle of doing so. If the degree to which hosts remained vulnerable to the Code Red worm months after the outbreak are any measure of the attention users pay to administrative tasks [MS02], then home backup systems both remain underutilized and home systems remain highly vulnerable to exploit and potential data loss.

Remote backup service: Another approach is use to a commercial remote backup service , such as DataThought Consulting [Datom] and Protect-Data.com [Proom]. The primary advantage of this approach is convenience, but of course this convenience comes at a cost. Currently, one can contract for automatic backup via a modem or the Internet of 500Mb of data for around $30-$125 a month.

Cooperative remote backups: Cooperative remote backup services provide the convenience of a commercial backup service but at a more attractive price. In a cooperative service, 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) for the common good. Pstore [BBST01] is an example of such a service. However, its primary goal is to tolerate local failures such as disk crashes, power failures, etc. Pastiche [CN02] 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.

The *Phoenix*recovery system.

We argue that a cooperative, peer-to-peer 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. Users of *Phoenix*would not need to exert any significant effort to backup their data, and they would not require local backup systems. 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 disk, CPU, and network resources to gain access to a highly resilient backup service. Users specify what percent F of their disk space is to be used by the system, and the system would protect a proportional amount F/k of their data. 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 are faulty at any time, where $t < n$. Although this model can always be applied when a total failure can not occur, it is only capable of expressing the worst-case failure scenario. The value of t bounds the largest subset of components that can fail, but there may sets of less than t components that are unlikely to fail together. This will be true if the failures of components are 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 exploited in both, and in the latter case, a power outage can crash both hosts.

As a first step towards the design of a backup system, we need a concise way of representing failure correlation. Junqueira and Marzullo proposed an abstraction called *core* for this purpose [JM03]. A core is a reliable minimal subset of components: the probability of having at least one component in a core not failing is high. Determining the cores of a system depends on the failure model assumed and desired degree of reliability for the system.

To determine the cores of a system, it is necessary to characterize the correlation of the failure of components; in our case, of hosts. We do this by assigning attributes to hosts. The attributes represent characteristics of the host that can make it fail prone. For example, the operating system a host runs is a point of attack: an attack that targets Linux will probably not be effective against hosts running Windows XP or Solaris. We could represent this point of attack by having an 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. Equivalently, we could represent the operating system with a set of binary-valued attributes, one for each operating system. If two hosts have an attribute a with the same value, then we assume that they share a vulnearability that can cause both to fail. Similarly, if two systems do not have any attribute with the same value, then the two systems fail independently. We assume that the same set of attributes A is associated with each host.

4 Skewed Diversity

If one knew the probability of attack for each vulnerability, then given a target system reliability one could enumerate minimal cores with that target reliability. In our case, it isn't clear how one would determine such probabilities. Instead, we define a core *Core*(C) for a host C to be a minimal set of processes that contains C and, for each attribute $a \in A$, there are two hosts in C that have different values of a. *Core*(C) is a core if we assume that in any Internet catastrophe, the attack targets only one attribute value. It is not hard to generalize this definition to allow for attacks targeted against multiple attribute values.

Smaller cores means less replication, which is desirable for better performance. A core will contain between 2 and $|A|$ hosts. If the hosts' attributes are well distributed, then the smallest cores will be small: for any host C, it is likely that there is a host C' that has different values of each of the attributes, and so C and C' constitute a core. If there is less diversity, though, then the size of the smallest cores may grow, hurting performance.

A lack of diversity, especially when trying to keep core sizes small, can lead to a more severe problem. Suppose there are n hosts $\{C_1, C_2, \ldots C_k\}$ that all have the same attribute values, and there is ony one host C that differs in all of it attributes' values. The smallest core for each host C_i contains C, meaning that C will maintain copies for all of the C_i. Since the fraction C donates for storing replicas is fixed, each C_i has only $1/k$ of this space. In other words, if each host donates F for common storage then each C_i can back up only F/k data. Note that k need only be less than the number of hosts, and so F/k can be minuscule.

Characterizing the diversity of a set of hosts is a challenging task. There is a large number of possibilities, and considering all of them is not feasible. So, 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 $f\%$ of the servers are characterized by $(1 - f)\%$ 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 $f = 0.5$ corresponds to a uniform distribution, and a value of f close to 1 indicates a highly skewed diversity.

We used this metric to determine the degree of replication and resilience for systems with varying diversity. The degree of replication is determined by the size of the core a host uses to backup data. The problem of finding an optimal core, 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 C as follows:

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

2. If there is no host found in the previous step, then it randomly chooses hosts that have at least one differ-

ent attribute until a core is constructed or it can not find any more hosts to choose.

This is a very simple heuristic, but it may not be the best; we have not yet done a careful study of heuristics for finding cores. The results we present below, however, indicates that it is not a bad heuristic in terms of the sizes of the cores it computes.

We created a set of attribute values, each representing a fictitious host. On the Internet, most of the hosts run some version of Windows and use Internet Explorer as their web browsers [One], and so we biased the attribute distribution towards having some fixed prefix. The length of prefix depends on the value f chosen for the diversity of the system. Assuming that all of the attributes have values from domains with y values, we have:

$$\frac{1}{y^i} \geq (1-f) \geq \frac{1}{y^{i+1}} \qquad (1)$$

for some integer i, $i \leq n$, where n is the number of attributes. The length of the prefix is i.

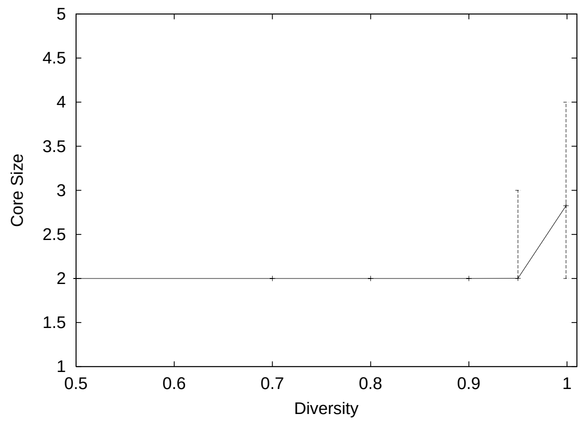

Figure 1: Core sizes as a function of diversity.

Using this scenario construction, we obtained the results shown in Figures 1 and 2. The system we assumed has 1,000 hosts, and eight attributes with four possible values each. We only show the results for one sample generated for each value of f, but we did not see significant variation across samples. Figure 1 shows the core size averaged over cores for all of the hosts for different values of the diversity parameter f. We also show in this graph the maximum and the minimum core sizes for every value of f. Note that the average core size remains around 2 for all the values of f. From this observation, we conclude even when the system presents low diversity, the degree of replication necessary to cope with vulnerabilities is small. Low diversity, however, may impact reliability. For $f = 0.999$, many of the cores do not cover all of the attributes because the attribute space is so highly skewed (there is only one host that need not share the common prefix). For all the other values of f, including $f = 0.95$, all of the cores obtained covered all the attributes.

Figure 2 show the amount of storage available for different values of f. The y axis plots the largest value of k such that there is a host that must be in k cores given the core compositions that we computed. This means that that there will be hosts that will be able to backup only F/k data. As expected, k increases as f approaches 1, but for $f = 0.995$ the load drops because not all attributes could be covered, and so the reliability is lower.

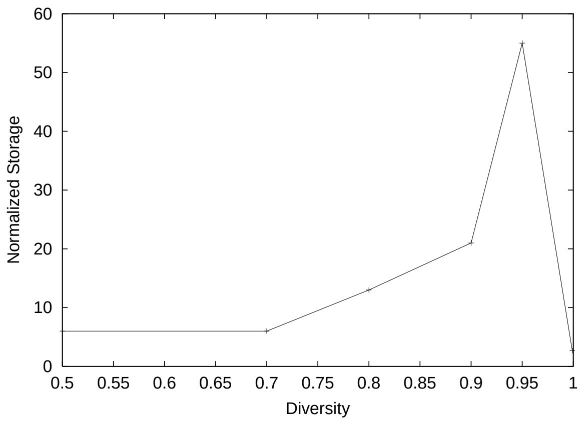

Figure 2: Storage load as a function of diversity.

To determine a bound on f for a real system, [One] claims that over 93% of the hosts that access a popular web site run some version InternetExplorer. 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 InternetExplorer [Sec], 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. If we consider a system with only two attributes: web browser (with 14 values, from the list at [One]) and operating system (with 11 values, again from [One]) then for this value of f we have an average core size of 2.03, a maximum core size of 3, and $k = 5$.

5 Further Work

In this paper, we have explored the feasibility of using a cooperative remote backup system called *Phoenix* as effective approach for surviving Internet catastrophes. *Phoenix* uses data redundancy, a model of dependent host vulnerabilities, and distributed storage in a cooperative, peer-to-peer system. Using a simulation model we have shown that, by performing informed placement of replicas, *Phoenix* provides highly reliable and available cooperative backup and recovery with low overhead: with as few as 2 replicas, the system can backup and recover at least the equivalent of 20% of storage contributed by each host in the system for a rich distribution of potential host exploits.

To implement *Phoenix* in practice, however, we must address a number of design issues. *Phoenix* needs to secure replicated data to ensure data privacy using encryption; in contrast to file sharing systems and other peer-to-peer backup systems, *Phoenix* does not share data across users. It needs to ensure fairness of storage allocation across users. We need to more carefully model the set of vulnerabilities and allow for dynamically adding and removing attributes. Finally, *Phoenix* itself needs to be protected from large-scale attack. We are currently working on addressing these issues in a prototype design and implementation of *Phoenix.*

References

[BBST01] C. Batten, K. Barr, A. Saraf, and S. Treptin. pstore: A secure peer-to-peer backup system. Unpublished report, December 2001.

[CN02] 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, December 2002.

[Com] Computer Economics. 2001 Economic Impact of Malicious Code Attacks. `http://www.computereconomics.com/cei/press/pr92101.html`.

[CPM+98] Crispan Cowan, Calton Pu, Dave Maier, Jonathan Walpole, Peat Bakke, Steve Beattie, Aaron Grier, Perry Wagle, Qian Zhang, and Heather Hinton. StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks. In *Proceedings of the 7th USENIX Security Conference*, pages 63–78, San Antonio, Texas, January 1998.

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

[JM03] F. Junqueira and K. Marzullo. Synchronous consensus for dependent process failures. In *Proceedings of the ICDCS 2003, to appear*, 2003.

[Mic] Microsoft Corporation. Microsoft windows update. `http://windowsupdate.microsoft.com`.

[MS02] David Moore and Colleen Shannon. 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, November 2002.

[MVS03] David Moore, Geoffrey M. Voelker, and Stefan Savage. Internet Quarantine: Requirements for Containing Self-Propagating Code. In *Proceedings of IEEE Infocom 2003*, April 2003.

[Nec97] George C. Necula. Proof-Carrying Code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '97)*, pages 106–119, Paris, France, January 1997.

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

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

[Sch90] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Computing Surveys, December 1990.

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

[Sym] Symantec. W32.nimda.a@mm. `http://www.symantec.com/avcenter/venc/data/w32.nimda.a@mm.html`.

[WFBA00] David Wagner, Jeffrey S. Foster, Eric A. Brewer, and Alexander Aiken. A First Step towards Automated Detection of Buffer Overrun Vulnerabilities. In *Proceedings of the Network and Distributed System Security Symposium*, pages 3–17, San Diego, CA, February 2000.