Reliability Analysis of Resilient Applications

Kathleen McGill, *Student Member, IEEE,* and Stephen Taylor, *Member, IEEE*
Thayer School of Engineering, Dartmouth College, 8000 Cummings Hall, Hanover, NH 03755
Kathleen.N.McGill@dartmouth.edu, Stephen.Taylor@dartmouth.edu
tr11-008

Abstract

The notion of resiliency is concerned with constructing applications that are able to operate through a wide variety of computer failures and attacks. Several approaches have been proposed to provide fault tolerance through the replication of resources. In general, these approaches provide graceful degradation of performance to the point of failure but do not *guarantee* progress in the presence of multiple cascading and recurrent failures. Our approach is to dynamically replicate processes, detect inconsistencies in their behavior, and restore the level of resiliency as the computation proceeds. This paper presents an analytical framework to perform reliability analysis of the proposed approach to resilience. The framework includes an analytical model of reliability of resilient applications. In addition, three analytical models from the fault tolerant literature are included: an independent and identically distributed fault model, a correlated fault model, and a computer worm model. These models provide a basis for quantitative analysis of reliability. We show that the reliability of resilient applications is orders of magnitude greater than static fault tolerance in the presence of correlated faults. In addition, resilience allows applications to operate through computer worm attacks, provided that it is part of a comprehensive defense strategy.

1 Introduction

Commercial-off-the-shelf (COTS) computer systems have traditionally provided several measures to protect organizations from hardware failures, such as RAID file systems [1] and redundant power supplies [2]. Unfortunately, there has been relatively little effort to provide similar levels of fault tolerance to software errors and exceptions. In recent years, computer network attacks have added a new dimension that decreases overall system reliability. A broad variety of technologies have been explored for detecting these attacks using intrusion detection systems [3]-[5], file-system integrity checkers [6]-[7], rootkit detectors [8]-[11], and a host of other technologies. Unfortunately, creative attackers and trusted insiders have continued to undermine confidence in software. These robustness issues are magnified in distributed applications, which provide multiple points of failure and attack.

A variety of approaches have emerged to provide reliability for distributed applications that rely on the static replication of resources. These include checkpoint/restart [12]-[16], process migration [17]-[23], and process replication [24]-[28]. In contrast, computational resiliency provides reliability through *dynamic* replication of resources [28]-[30]. Figure 1 displays the relative impact of resiliency and static replication on the reliability of applications under attack [30]. Static replication alone provides graceful performance degradation to the point of failure because each attack reduces the number of replicas permanently. In contrast, resiliency dynamically reconstitutes the desired number of process replicas after each attack, allowing the application to continue operation with the same level of assurance. The malicious

*This material is based on research sponsored by the Defense Advanced Research Projects Agency (DARPA) under agreement number FA8750-09-1-0213.

actions have no long term effect.

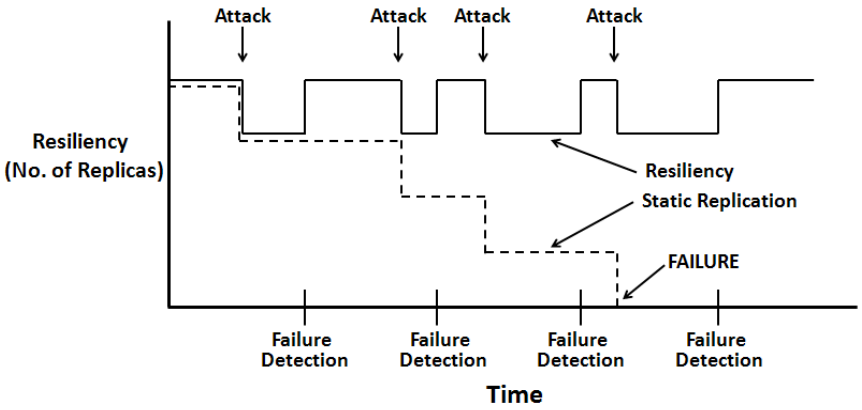


Figure 1. Reliability of applications using computational resiliency and static replication [30].

Our approach to resilience is to dynamically replicate processes, automatically detect inconsistencies in their behavior, and transparently restore the level of resiliency as the computation proceeds. Figure 2 illustrates how this strategy is achieved [30]. At the application level, three processes share information using message-passing. The underlying *operating system* directly implements a resilient view that replicates each process and organizes communication between the resulting *resilient process groups*. Individual processes within each group are mapped to different computers to ensure that a single failure or attack cannot impact an entire group.

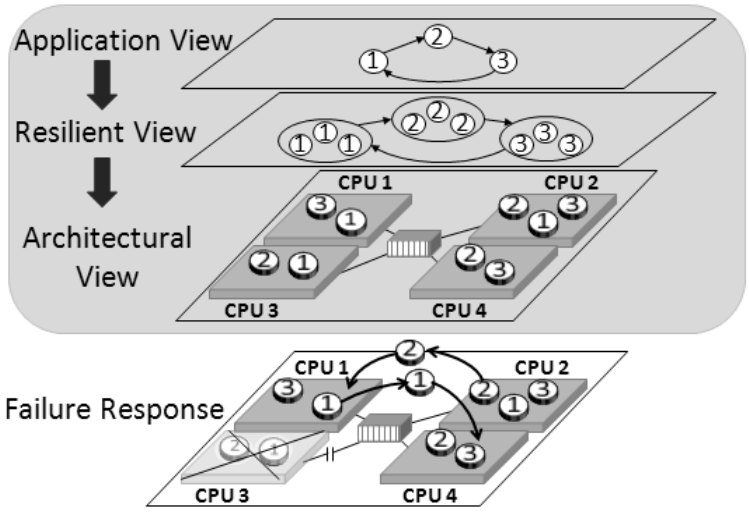


Figure 2. Dynamic process regeneration

The base of the figure shows how the process structure responds to failure or attack [30]. The figure assumes that an attack is perpetrated against processor 3, causing processes 1 and 2 to fail or to portray communication inconsistencies with other replicas within their group. Failures are detected by communication timeouts and message comparison. These failures trigger automatic process regeneration; the remaining consistent copies of processes 1 and 2 dynamically regenerate a new replica and migrate it to processors 4 and 1, respectively. As a

result, process resiliency is reconstituted, and the application continues operation with the same level of assurance.

This approach requires several features that are not directly available in modern operating systems. Process *replication* is needed to transform single processes into process groups. Point-to-point communication between application processes must be replaced by group communication between process groups. Mechanisms to detect process failures and inconsistencies must be available to initiate process *regeneration*, and process *migration* is required to move a process from one processor to another. Finally, as processes move around the architecture, it is necessary to provide control over where processes are *mapped*. In order to prevent prohibitive communication costs, process management policies are desired to *maintain locality* within process groups. This tactic enables *locality-based failure detection*, in which transit delays from replicated messages are used to predict an upper bound on the delay for communication timeouts.

We define resilience as the ability to provide and maintain an acceptable level of service in spite of a range of faults and attacks [31]. In order to maintain the application's level of service, this research is concerned with *process failures*. Two kinds of failures are addressed: complete failures and Byzantine failures. A complete failure is when a process halts, and the external state is constant [32]. The process ceases communication with other processes of the application. A Byzantine failure, on the other hand, is a failure in which the process behaves in an unpredictable manner [33]. Byzantine failures encompass a range of failures, such as the omission of some action, corruption of local state, or inconsistent communication [32], [33].

A collection of novel operating system technologies that provide resilience for distributed applications has been developed in prior work [34]-[39]. These technologies include resiliency *mechanisms* and *policies* associated with a resilient message-passing technology, *rMP*. The resiliency mechanisms achieve process replication, adaptive failure detection, and dynamic process regeneration automatically and transparently within the operating system. Resiliency policies are explored to manage of resilient process groups. This paper presents an analytical framework to perform reliability analysis of the proposed approach to resilience. An analytical model of the reliability of resilient applications is presented to predict the probability of failure for applications in a number of fault and threat scenarios. Three fault and threat models from the fault tolerant literature are included to provide a basis for quantitative analysis of reliability and to examine the strengths and weaknesses of our approach.

2 Background

There are a number of terms and metrics in the fault tolerant and computer security communities with unclear definitions and usage. Several fundamental definitions are provided to clarify the discussion of this paper.

A **fault** is a flaw in a system whose presence may or may not lead to a failure [32]. In order to cause a failure, the fault condition must be executed by a system component. An **error** is a deviation between the expected behavior and the actual behavior of a component *internal* to the system [32]. Errors occur during execution, due to the activation of a fault. A **failure** is an event in which an observable system behavior deviates from its specification [32]. Failures are events that are detectable at the system boundary. For example, process failures are detected through the rMP mechanisms. Generally, there is a progression in which a fault is activated, causing an error, which may result in a failure.

A different set of terms is used in the computer security community. A **vulnerability** is defined as a flaw or weakness in system security procedures, design, implementation, or internal controls [40]. A vulnerability can be unintentionally triggered, in the case of an **accident**, or intentionally exploited, in the case of an **attack**. A **threat** is defined as an accident or attack that triggers a vulnerability, which may cause a security breach [40]. A **security breach** is a violation of the system security specification. Like faults, vulnerabilities do not always propagate into security breaches**.**

There are a number of metrics used to describe the desirable attributes of mission-critical systems. The minimal set of attributes is a question of ongoing debate. However the working definitions of terms that pertain to this paper are provided by Trivedi *et al.* [41]. In addition, each term is a discussed in the context of application resilience.

Reliability is the probability that a system performs a specified service throughout a specified interval of time [42]. This is a measure of the continuity of service, or the probability that the system has not failed once since it started. Generally, individual component failures are allowed during this interval, but not whole system failures. Reliability serves as the primary metric for this analysis. However, it is often more natural to refer to the probability of failure of a system over the specified interval. The reliability, *R(t)*, and probability of failure, *F(t)*, always sum to unity, as defined in (1).

$$R(t) + F(t) = 1 \qquad (1)$$

Availability is the ability of the system to perform its prescribed function at a specific instant of time or over a specified period of time. The metric is usually expressed as a ratio of the units of time when service was available to the specified service period. Availability is closely related to reliability. However, it depends on the service provided by the application. For instance, in a manager-worker application, the failure and regeneration of a worker process might not compromise availability. However, if a manager process fails, the application service might become temporarily unavailable. Because of these discrepancies, availability is not considered in this analysis.

Survivability is the capability of a system to fulfill its mission, in a timely manner, in the presence of attacks, failures, or accidents [43]. The mathematical definition of survivability is a separate research question [44], [45], [46]. According to the working group on Network Survivability Performance, survivability depicts the time-varying system behavior after a failure, attack, or accident occurs [47]. A thorough treatment of the survivability metric is beyond the scope of this paper. It emphasizes analysis on time scales of a failure event. Instead, reliability is used to describe the survivability on the scale of the service interval.

Finally, a few parameters used to define the fault models are presented for reference. The **Mean Time Between Failure (MTBF)** is an estimate of the elapsed time between failures in a component or system. A straightforward interpretation is the operational time divided by the number of failures. The **failure rate (λ)** is the inverse of the MTBF (1/MTBF), expressed in failures per unit time.

3 Model of Application Reliability

This analysis aims to characterize the reliability of our *approach* to resilience. An ideal implementation is assumed in which the rMP technology provides application resilience as proposed. All process failures are detected through communication timeouts and message comparison, processes are reliably regenerated, and the process management strategies ensure that process replicas are mapped to different hosts. Thus, reliability analysis is conducted with

respect to the application; the reliability of the underlying technology and the operating system is assumed.

The following application model is used. An application consists of p processes with a level of resiliency r, so the resilient implementation includes a total of $p*r$ processes. Each process is mapped to a different processor. Assume that the application will fail if all r replicas of a particular process group experience complete process failures at the same time *or* if greater than (r-1)/2 replicas of a particular process group experience Byzantine process failures at the same time. This number of replica failures for Byzantine process failures stems from the detection of compromised processes through majority voting protocols.

Although application failure requires the replicas be failed at the same time, it is not necessary that the *root causes* of process failures are simultaneous. In the rMP technology, process failures are detected during group communication. As a result, there is a failure time interval between communications during which replica failures can be considered simultaneous. This interval begins after a consistent message is sent and continues through subsequent computation, the next sent message, failure detection, and process regeneration. Portions of this interval can be estimated from the performance benchmarks of failure detection and regeneration mechanisms, but the time between subsequent communications is strictly application dependent. For this analysis, it is assumed that processes communicate relatively frequently, and the failure time interval is 30 seconds.

To compare the reliability of rMP applications to the fault tolerant community, analysis is also included for applications using static replication without process recovery. It is assumed that an application using static replication will fail with the same number of replica failures as rMP applications, but now those failures can occur over the duration of the computation. For this analysis, the failure time interval is the required service interval of the application. An interval of 20 minutes is selected as an initial pass. Otherwise, the same parameters are used for static replication as rMP applications.

To construct an analytical model of the probability of application failure, we assume the probability of a single process failure, P_f, over interval t, is provided by the fault or threat model under consideration. The probability of failure of m processes in the same interval, F_m , is defined by (2).

$$F_m = P_f^m \tag{2}$$

The number of process failures required for the failure of an entire process group depends on the process failure mode. For simplicity, it is assumed that applications experience either complete or Byzantine process failures, but not both. Using this assumption, the number of replica failures required for process group failure, m, is defined by (3) for complete failures, m_C, and by (4) for Byzantine failures, m_B.

$$m_C = r \tag{3}$$

$$m_B = floor(\frac{r-1}{2} + 1) \tag{4}$$

Thus, (2) provides the probability of failure of a single process group for the appropriate definition of m. Conversely, the reliability of a single process group, R_m is defined by (5),

$$R_m = 1 - F_m = 1 - P_f^m \tag{5}$$

However, for an application to be reliable, all process groups must be reliable. The probability that all p process groups are reliable, R_{app}, is defined by (6).

$$R_{app} = (1 - P_f^m)^p \quad (6)$$

Finally, the probability of application failure, F_{app}, is defined by (7).

$$F_{app} = 1 - R_{app} = 1 - (1 - P_f^m)^p \quad (7)$$

By substituting the appropriate definitions of m, the probability of application failure due to complete process failures, $F_{app,C}$, and the probability of application failure due to Byzantine process failures, $F_{app,B}$, are defined by (8) and (9), respectively.

$$F_{app,C} = 1 - (1 - P_f^r)^p \quad (8)$$

$$F_{app,B} = 1 - (1 - P_f^{(floor((r-1)/2+1))})^p \quad (9)$$

Thus, equations (8) and (9) provide the probabilities of application failure in terms of the number of process groups in the application, the level of resiliency, and the probability of a single process failure. The reliability of applications is considered as a function of the level of resiliency for three application sizes: 100, 1000, and 10,000 process groups. This number of processes may correspond to a single application or multiple critical applications running simultaneously. One process is mapped to each processor of the system, so the number of processors used increases with the level of resiliency.

An architecture of 10,000 distributed hosts is assumed. Each host contains at least 8 processors, so the system has the capacity to run up to 8 replicas of 10,000 processes simultaneously. Each host includes application software which may contain vulnerabilities. However, it is assumed that any software vulnerabilities are restricted to the *application* layer of the host. The operating system software is reliable. In order to conduct analysis in the context of mission-critical applications, we define a *maximum acceptable probability of failure for mission-critical applications as* 10^{-6} [48].

4 Quantitative Analysis

A number of fault and threat models are presented to provide the basis for quantitative analysis of reliability. These models are not meant to be exhaustive of the threat space. The space is too large for a thorough treatment to be practical. Rather, they lend a starting point to examine the strengths and weaknesses of the rMP approach. Each model is used to determine the probability of failure for a single process in the environment to input in the analytical model of application reliability.

4.1 Independent and Identically Distributed Fault Model

The most common fault model applied in the fault tolerant community assumes independent and identically distributed faults in hardware components [25], [49]. This model offers a simple reference case for analysis. Figure 3 provides a summary of the key assumptions, parameters, and equations of the independent fault model analysis.

Independently and Identically Distributed Fault Model

Goal: Determine the probability of application failure on a scalable multi-processor architecture with independently and identically distributed faults.

Assumptions:
Independent and identically distributed faults
Exponential processor failure distribution
Faults are transient or repaired manually.
A fault causes complete failure of the process that activates it when it occurs.

Parameters:
Processor Mean Time Between Failure (MTBF): 5 years
Processor Failure rate (λ): 0.2 failures/year
Failure time interval (t): 30 seconds for rMP applications
20 minutes for static replication

Equations:
Probability of application failure from complete process failures:

$$F_{app,C} = 1 - (1 - P_f^r)^p \quad (8)$$

Probability of processor failure with exponential failure distribution:

$$P_f = 1 - e^{-\lambda t} \quad (10)$$

Figure 3. Summary of assumptions, parameters, and equations for the analysis of the probability of application failure in the presence of independent and identically distributed faults.

The model considers faults caused by processor failures, in which all processors have an equal failure rate. An exponential failure distribution is assumed for the processors, such that the probability of a processor failure, P_f, over a time interval t is defined by (10).

$$P_f = 1 - e^{-\lambda t} \quad (10)$$

Unfortunately, there is no clear consensus of general processor MTBF in the literature, but a conservative estimate of 5 years is commonly used [49], [50], [51], [52]. The time interval, t, corresponds to the failure time interval of the application model: 30 seconds for rMP and 20 minutes for static replication applications. This model assumes that all faults are transient or may be repaired manually. However, the outcome of a fault is *complete* process failure for the process that activates it when it occurs.

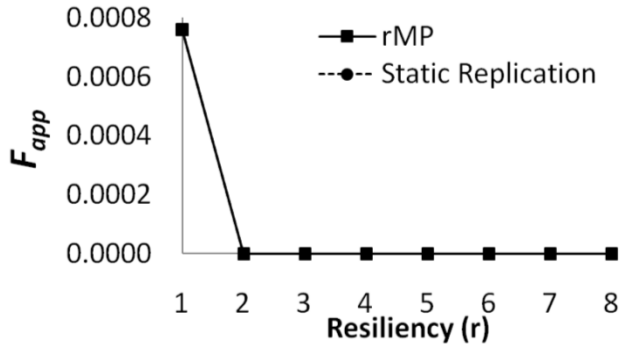
For scalable multi-processor architectures, it is often worthwhile to consider the system MTBF. For a system with n processors and independent and identically distributed faults, the system MTBF, $MTBF_{SYS}$, is defined by (11).

$$MTBF_{SYS} = \frac{MTBF}{n} \quad (11)$$

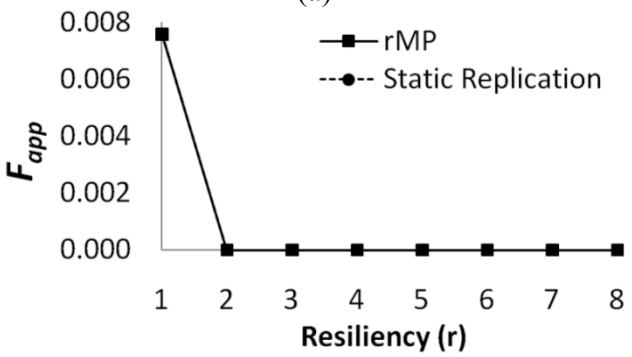
For systems with large numbers of processors, failures become more frequent. For example, the system MTBF for a platform with 100,000 processors, each with an individual 5 year MTBF, is approximately 26 minutes. At this frequency, failures are guaranteed to interrupt long running applications and represent an intolerable threat to any mission-critical application. This

phenomenon is a major driving force for fault tolerance to independent and identically distributed faults as systems increase in scale.

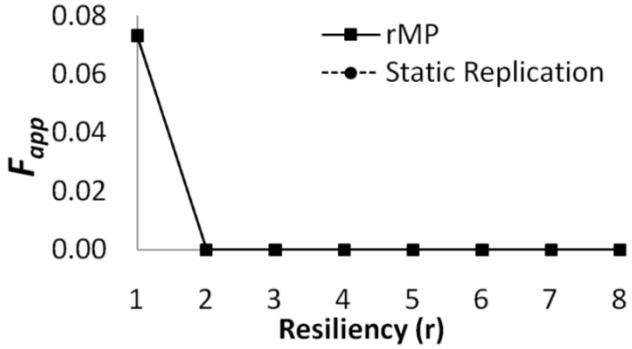
Figure 4 shows the probability of application failure due to complete process failures caused by independent faults for rMP and static replication applications. Not surprisingly, the probability of failure is very low for both replication strategies. For all applications, the probability of application failure is nearly indiscernible, below 10^{-6} with a single replication (r = 2). These results are consistent with the expectation that a properly functioning system would not be likely to cause application failure. However, note that a system with 10,000 processes and no fault tolerant strategy has a 0.073 probability of failure, which exceeds what is acceptable for mission-critical applications.



(a)



(b)



(c)

Figure 4. Probability of application failure due to complete process failures caused by independent faults for rMP and static replication with increasing levels of resiliency for applications with (a) 100, (b) 1000 , and (c) 10,000 processes.

4.2 Correlated Fault Model

In reality, faults are not usually independent and identically distributed. Hardware faults are often correlated, due to environmental conditions, temperature, manufacturing defects, incorrect configuration, etc. Other correlated failures are caused by software, storage systems, networks, human error, and a number of unknowns. A final cause for correlated failures may be a coordinated attack. In this worst case scenario, we may assume an adversary has installed a collection of implants in the system that may be activated through remote command and control to create multiple, cascading faults in the system.

The analysis of correlated faults on a variety of computing platforms is an active area of research [53]-[55]. These studies use existing failure logs to model the observed failure patterns of multi-processor systems. One pertinent finding is that failures tend to be periodic and strongly correlated in time. Often there are failure peaks or bursts during which multiple failures occur in relatively short time intervals [55]. To assess the reliability of the rMP technology in these real-life correlated failure scenarios, a model of correlated faults is developed. Figure 5 provides a summary of the key assumptions, parameters, and equations of the correlated fault analysis.

Correlated Fault Model

Goal: Determine the probability of application failure on a scalable multi-processor architecture with correlated faults.

Assumptions:
Bursts of multiple hardware faults occur in short intervals *during which* there are:

- Independently distributed faults
- Exponential processor failure distributions

Faults are transient or repaired manually.
A fault causes complete failure of the process that activates it when it occurs.

Parameters:
Processor Mean Times Between Failure (MTBF): 10.6 hours and 4.7 hours
Processor Failure rates (λ): 826 failures/year and 1883 failures/year
Failure time interval (t): 30 seconds for rMP applications
20 minutes for static replication

Equations:
Probability of application failure from complete process failures:

$$F_{app,C} = 1 - (1 - P_f^r)^p \quad (8)$$

Probability of processor failure with exponential failure distribution:

$$P_f = 1 - e^{-\lambda t} \quad (10)$$

Figure 5. Summary of assumptions, parameters, and equations for the analysis of the probability of application failure in the presence of correlated faults.

The data and failure analysis obtained from two platforms in the literature is leveraged for this analysis [54]-[55]. These platforms were selected because they represent distributed systems most likely to utilize the rMP technology: high performance computing (HPC) and grid computing systems. The HPC data is obtained from 22 HPC systems at the Los Alamos National Laboratory (LANL). The grid representative is Grid 5000, a scientific instrument of geographically distributed systems. Table 1 summarizes the key platform characteristics, including the type of distributed system, the number of processors, and years during which the data was collected [54]-[55].

Table 1. Platform Data Set Characteristics [54]-[55]

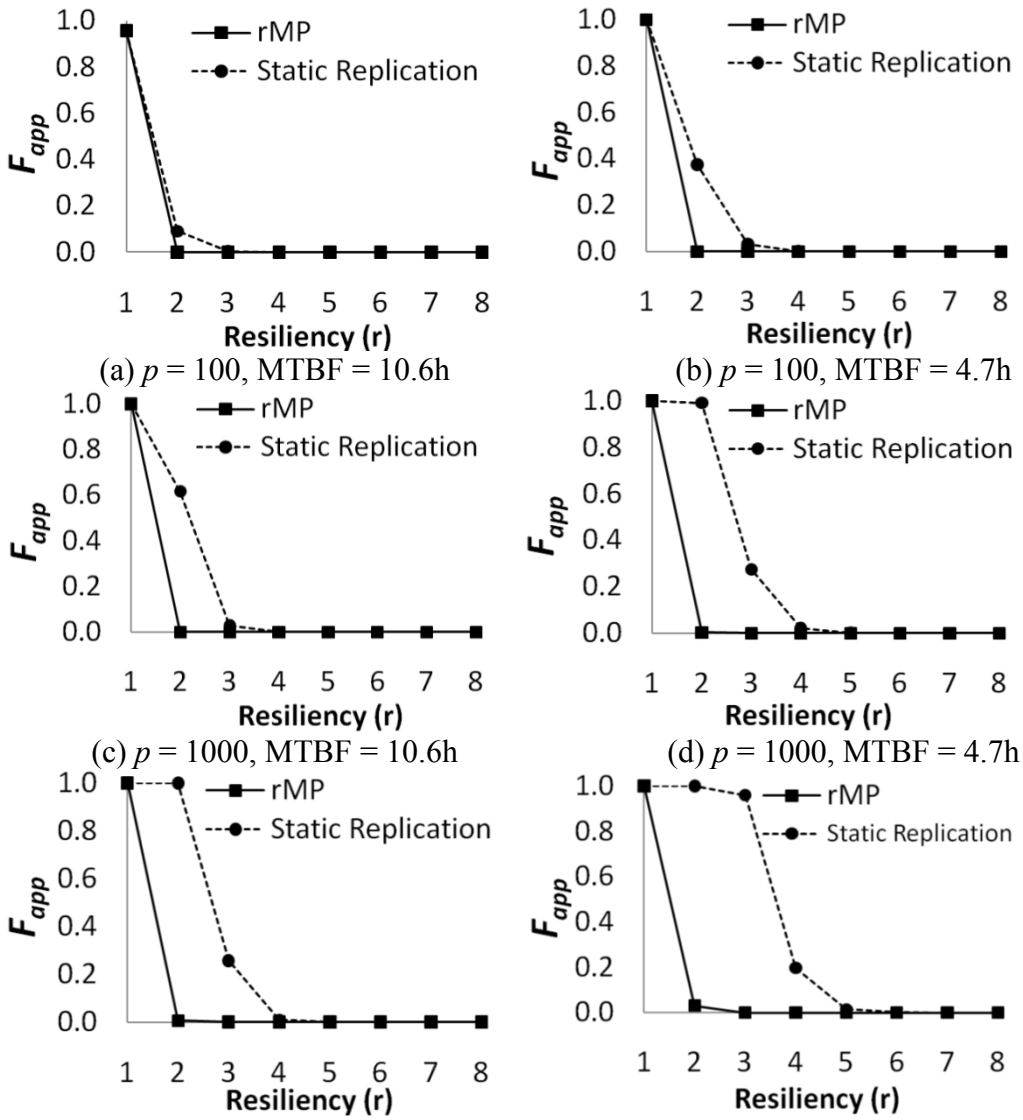
Platform	Type	Number of Processors	Years
LANL	HPC	4750	1996-2005
Grid5000	Grid	1288	2005-2006

A peak periods model of failures is used to characterize the failure history of these platforms [55]. The model parameters, presented in Table 2, describe the average duration of failure peaks and the system MTBF *during these peaks*. The average duration of failure peaks is greater than an hour for both systems, long enough to have a sustained impact on applications. In addition, the system MTBF is adjusted to an individual processor MTBF, using (11). The processor MTBF and corresponding failure rates are included in the table. This adjustment enables calculation of the probability of individual processor failures according to (10). This calculation assumes correlated faults *within* peak periods are independently distributed and obey an exponential failure distribution. While these assumptions are not necessarily valid in the case of correlated faults, they provide a first order approximation for this analysis. Like the independent fault model, this model assumes all faults are transient or may be repaired manually, and the outcome of a fault is *complete* process failure for the process that activates it when it occurs.

Table 2. Failure Parameters for Correlated Failure Analysis [55]

Platform	No. of Processor	Avg. Failure Peak Duration (hr)	System MTBF (hr)	Processor MTBF (hr)	Processor λ (failures/yr)
LANL	4750	1.1	0.0022	10.6	826
Grid5000	1288	1.4	0.0036	4.7	1883

Figure 6 shows the probability of application failure due to complete process failures caused by correlated faults for rMP and static replication. The left column of the figure shows the response to the LANL model with the MTBF = 10.6 hours, and the right column shows the Grid5000 fault model with the MTBF = 4.7 hours. With correlated faults, the probability of application failure is orders of magnitude greater than with independent faults. All applications without replication have at least 0.95 probability of failure within 20 minutes of execution. This result demonstrates the necessity for fault tolerance in mission-critical applications on large scale architectures.



(a) p = 100, MTBF = 10.6h (b) p = 100, MTBF = 4.7h

(c) p = 1000, MTBF = 10.6h (d) p = 1000, MTBF = 4.7h

(e) p = 10,000, MTBF = 10.6h (f) p = 10,000, MTBF = 4.7h

Figure 6. Probability of application failure due to complete process failures caused by correlated faults for rMP and static replication. The left collumn uses the LANL MTBF = 10.6h, and right column uses the Grid5000 MTBF = 4.7h.

The probability of failure for rMP applications is less than static replication for the same level of resiliency of all applications. This result exhibits the benefit of rMP process regeneration which narrows the failure time interval to cause application failure. Table 3 displays the required level of resiliency to maintain acceptable probability of failure for mission-critical applications. Static replication requires greater levels of resiliency than the rMP technology. In addition, the failure time interval and the required level of resiliency would increase with the service interval for applications using static replication but not for those using rMP.

Table 3. Level of Resiliency Required for Mission-critical Reliability ($F_{app} < 10^{-6}$)

Model	rMP	Static Replication
MTBF = 10.6 h, p = 100	3	6
MTBF = 10.6 h, p = 1000	3	6
MTBF = 10.6 h, p = 10,000	4	7
MTBF = 4.7 h, p = 100	3	7
MTBF = 4.7 h, p = 1000	4	8
MTBF = 4.7 h, p = 10,000	4	9

4.3 Computer Worm Model

A computer worm is a program that is able to self-replicate and propagate throughout a computer network without user intervention. In the last ten years, prolific worms have captured the attention of security researchers [56], [57], [58]. The Code Red worm, first released in 2001, targeted a Microsoft IIS web server vulnerability and propagated by scanning randomly generated IP addresses for the same vulnerability [57]. Subsequently, more efficient worms have emerged or have been theorized, such as Code Red II, Nimda, the Warhol worm, and Flash worms [58]. Two models are derived to determine the probability of application failure on a network penetrated by a computer worm. Figure 7 provides a summary of the key assumptions, parameters, and equations of the computer worm analysis.

Computer Worm Model

Goal: Determine the probability of application failure on a network penetrated by a computer worm. Two models are considered: an unmitigated worm and a worm in a network with countermeasures in place.

Assumptions for Both Models:
All hosts are connected.
All hosts are vulnerable.
Vulnerabilities are in application software only. The operating system is reliable.
Host infections cause Byzantine process failures.

Unmitigated Worm Assumptions:
Host infections are permanent.

Worm in a Network with Countermeasures Assumptions:
Host infections are temporary due to a repair mechanism.
The repair mechanism restores infected applications to a gold standard.
Repairs occur at a constant repair rate of D repairs/hour.

Parameters:
N = 10,000 hosts in the network.
K = 2.6 compromises/hour
D = 1, 2, 2.5, 2.7, and 120 repairs/hour
t = time since initial infection
a = proportion of hosts infected in the network

Equations:
Probability of application failure from Byzantine process failures:

$$F_{app,B} = 1 - (1 - P_f^{(floor((r-1)/2+1))})^p \quad (9)$$

Proportion of hosts infected by an unmitigated worm as a function of time:

$$a = \frac{e^{K(t-T)}}{1+e^{K(t-T)}} \quad (14)$$

Proportion of hosts infected by a worm with countermeasures as a function of time:

$$a = (\frac{K-D}{K}) \frac{e^{(K-D)(t-T)}}{1+e^{(K-D)(t-T)}} \quad (16)$$

Figure 7. Summary of assumptions, parameters, and equations for the analysis of the probability of application failure in the presence of two computer worm models.

The Random Constant Spread (RCS) model is a worm model developed by Staniford *et al*. [58] through analysis of the Code Red worm. In the RCS model, N is the number of vulnerable hosts on a network. For simplicity, it is assumed that all host are vulnerable (at the application level only) and that the network is a complete graph in which all hosts are connected [57]. Let K be the constant average compromise rate, or the average number of vulnerable

machines that an infected host can compromise per hour. Define *a(t)* as the proportion of vulnerable hosts that have been compromised at time *t*, where *t* is measured from the initial infection. The number of infected hosts, *n(t)*, at time *t* is defined by (12).

$$n(t) = a(t) * N \tag{12}$$

Assuming that a host can only be infected once and stays infected thereafter, a simple differential equation for the system is defined by (13) with a solution defined by (14) [58].

$$\frac{da}{dt} = Ka(1-a) \tag{13}$$

$$a = \frac{e^{K(t-T)}}{1+e^{K(t-T)}} \tag{14}$$

In (14), *T* is a constant of integration that fixes the time of the initial incident. This solution has been fitted to the data measured in several worm outbreaks to estimate values for *K*. Staniford *et al.* [58] proposed a model for a Code Red-like worm, capable of scanning 10 hosts/second using $K = 2.6$ compromises/hour. The compromise rate observed in the network is significantly lower than the scan rate because the worm generates random addresses within in the IP address space of size 2^{32}, and the overwhelming majority of addresses scanned are misses in the network.

The Code Red-like model can be used to model the spread of a random scanning worm in a network after an infection. Figure 8 displays the number of infected hosts as a function of time in a network of 10,000 hosts with $K = 2.6$ compromises/hour. This curve is obtained through a numerical model of (13) with one infection at time zero or from a plot of (14) with $T = 3.54$ hours, as determined by the initial conditions. In general, the shape of the curve is not affected by the number of hosts in the network, but the number of hosts and the initial conditions can shift the curve in time. Note that, unmitigated, the worm can infect the entire network within 6 hours. In this analysis, it is assumed that worm infections on the host cause *Byzantine* process failures.

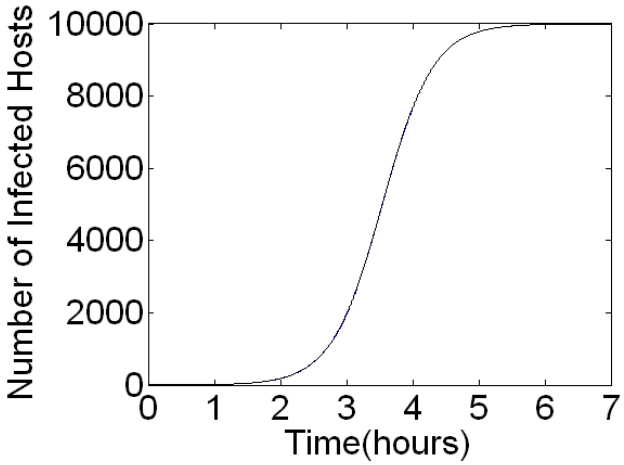


Figure 8. Number of infected hosts v. time of propagation of Code Red-like scanning worm in a network of 10,000 hosts.

The impact of a computer worm on a network is modeled under two different conditions: an unmitigated worm and a worm in a network with countermeasures in place to recover infected applications. In this analysis, there is no distinction made between rMP and static replication technologies. This is because the rMP technology has no knowledge of which hosts are infected at a given time and no mechanism to prevent regenerating a failed process on an infected host. As a result, the impact of process regeneration is reduced because processes are equally probable to fail on an infected host as they are to be regenerated on an infected host. At the same time, the

model of worm propagation in the presence of countermeasures is considered for rMP applications exclusively. It is assumed that systems hosting static replication technologies have no mechanism to recover applications infected by the computer worm.

4.3.1 Unmitigated Computer Worm

The first model, common in the literature, assumes that the worm propagates too quickly for response [56]. There are no countermeasures to mitigate the worm propagation, and it is assumed that hosts are permanently infected. In this case, the RCS model is used to determine the probability of infection of a given host in the network as defined by the fraction of infected hosts in the network, according to (14). In a network of homogeneous hosts, this is also the probability of failure of a given process in the network. Figure 9 displays the probability of host infection as a function of time after the initial infection of an unmitigated worm.

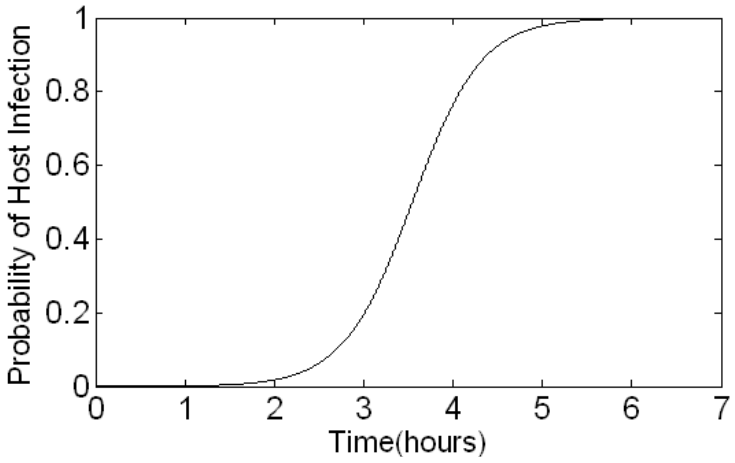


Figure 9. Probability of host infection v. time for the propagation of an unmitigated Code Red-like scanning worm in a network of 10,000 hosts.

Figure 10 displays the probability of application failure as a function of time due to Byzantine process failures from unmitigated worm propagation. Not surprising, the probability of application failure exceeds mission-critical levels within the first 1.8 hours of the attack for all applications. Applications utilizing only triple resiliency exceed mission-critical levels in the first *minute* of the attack. While these results are discouraging, they represent a worst case scenario in worm propagation. Mission-critical networks are likely to deploy countermeasures to mitigate the propagation rate and would not be represented by this model.

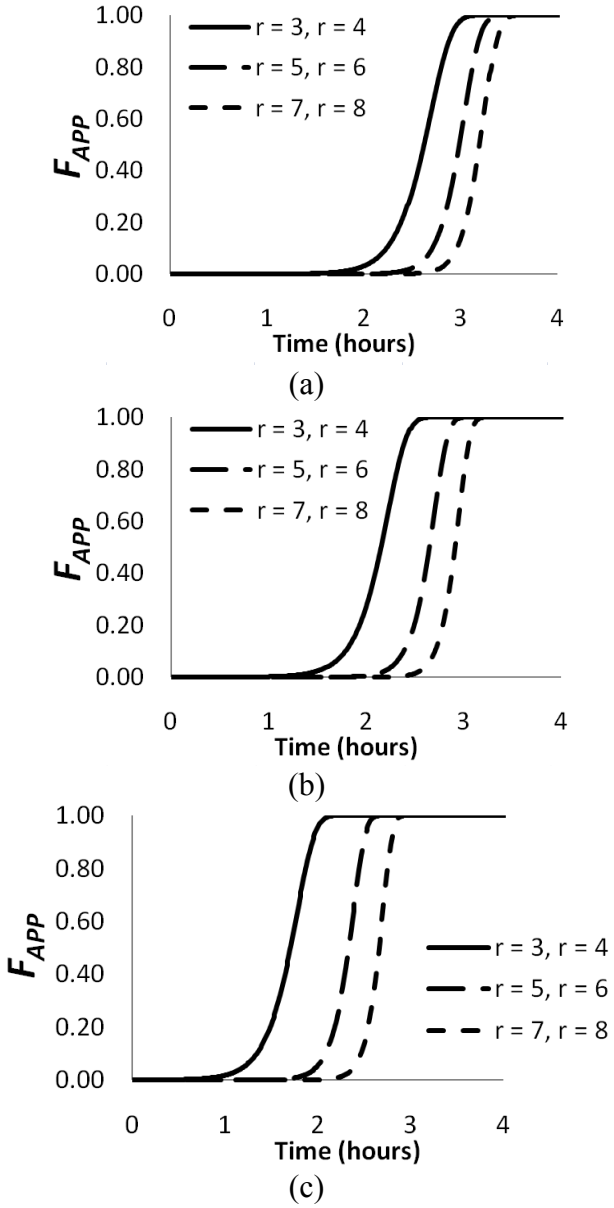


Figure 10. Probability of application failure as a function of time due to Byzantine process failures caused by an unmitigated worm attack for rMP replications with increasing levels of resiliency of (a) 100, (b) 1000, and (c) 10,000 processes.

Regardless of the propagation model, the overlapping curves displayed in Figure 10 demonstrate an important effect in Byzantine failure detection. For the purposes of majority voting, odd levels of resiliency are optimal. Two healthy replicas are required to outvote a single compromised replica. Thus, the level of resiliency must increase by two to increase the number of Byzantine failures allowed in a single process group. As a result, there is no gain in reliability from increasing to an even level of resiliency, as seen in the figure. However, there is a reduction in performance.

4.3.2 Computer Worm in a Network with Countermeasures

The second model of worm propagation includes countermeasures to slow the propagation of the worm. The model incorporates a repair mechanism capable of removing the worm infection from the host application. This repair mechanism may be an application regeneration, similar to the rMP technology, or an alternative technique provided by the host. Regardless of the mechanism, it is assumed that the application is restored to a gold standard after infection. This system is modeled by modifying (13) to include the rate of repair for infected hosts, D repairs/hour. The new model assumes a constant repair rate is applicable to infected hosts of the network [59] and is defined by (15) with a solution given by (16).

$$\frac{da}{dt} = Ka(1-a) - Da \tag{15}$$

$$a = \left(\frac{K-D}{K}\right)\frac{e^{(K-D)(t-T)}}{1+e^{(K-D)(t-T)}} \tag{16}$$

Equation (16) provides the probability of infection of a given host in the network with a constant repair rate.

Figure 11 displays the probability of host infection as a function of time after the initial infection for a range of repair rates. These repair rates include values that are less than, comparable to, and greater than the compromise rate, K. The slower repair rates, D = 1 and D = 2 repairs/hour demonstrate that countermeasures slow the propagation of the worm until the proportion of infected machines reaches a steady state value of $(K-D)/K$. The repair rates that are comparable to the compromise rate, D = 2.5 and 2.7 repairs/hour, appear to cause virtually zero probability of host failure. For D = 2.5 repairs/hour, this value converges to 0.038 probability of failure. However, for all values of D > K, the probability of host failure converges to zero. The repair rate of D = 120 repairs/hour is included to represent a mechanism similar to rMP process regeneration, in which failures are regenerated in 30 seconds.

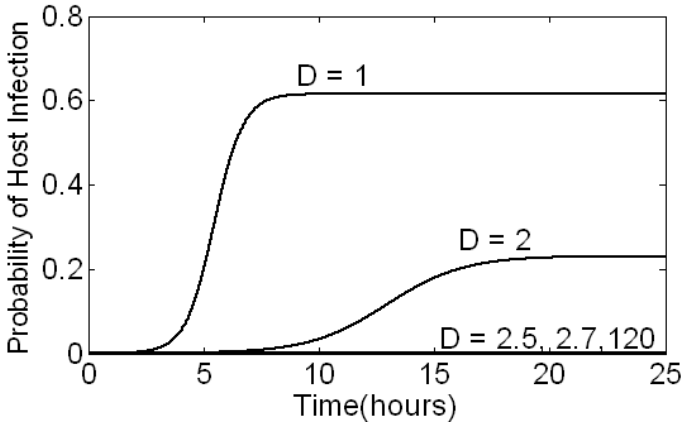
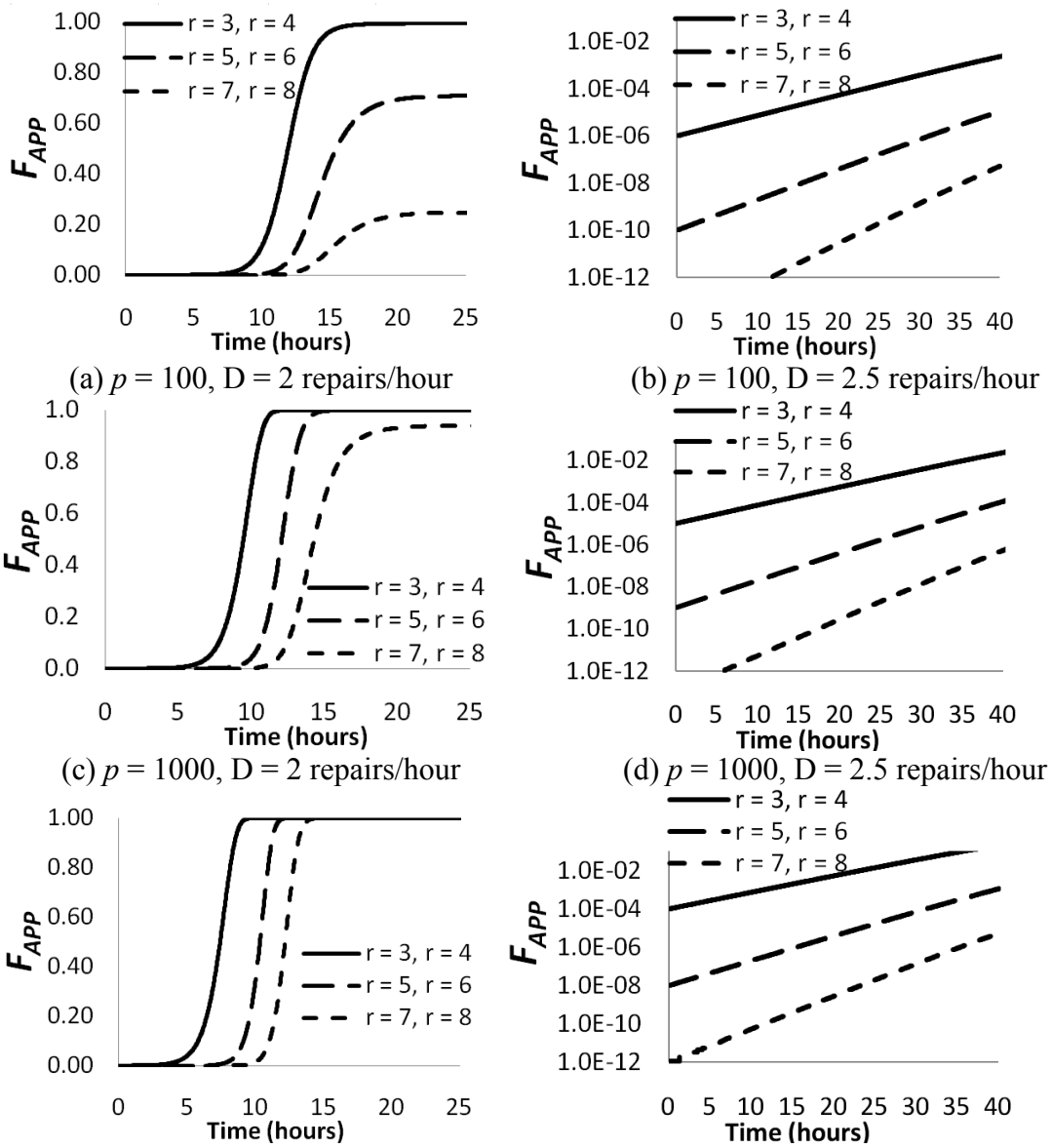


Figure 11. Probability of host failure v. time for the propagation of a Code Red-like scanning worm in a network of 10,000 hosts for rate of repair D = 1, 2, 2.5, 2.7, 120 repairs/hour.

Figure 12 displays the probability of application failure as a function of time due to Byzantine process failures from a worm attack in the presence of countermeasures. The left column of the figure includes a repair rate slower than the compromise rate, D = 2 repairs/hour. The right column includes a repair rate comparable to the compromise rate, D = 2.5 repairs/hour. Note that the time scales in Figure 12 are greater than the scales in Figure 10. With a slow repair rate, the probability of application failure still exceeds mission-critical levels for all applications, but it takes longer for this to occur. Applications utilizing triple resiliency exceed mission-

critical probabilities of failure within a minute, but applications with r > 4 maintain acceptable reliability for at least 2.6 hours. However, while the reliability of rMP applications increases with a slow repair rate, application failure is still imminent.



(a) p = 100, D = 2 repairs/hour (b) p = 100, D = 2.5 repairs/hour

(c) p = 1000, D = 2 repairs/hour (d) p = 1000, D = 2.5 repairs/hour

(e) p = 10,000, D = 2 repairs/hour (f) p = 10,000, D = 2.5 repairs/hour

Figure 12. Probability of rMP application failure due to Byzantine process failures caused by worm propagation with repair mechanims. The left collumn uses a rate D = 2 repairs/hour, and right column uses a rate D = 2.5 repairs/hour.

The right column of Figure 12 displays the probability of application failure with comparable repair and compromise rates. Note that the probabilities of failure are displayed on a logarithmic scale. These probabilities are orders of magnitude less than the probabilities with a slower repair rate of D = 2 repairs/hour. While some applications quickly exceed mission-critical probabilities of failure, the applications with r > 4 levels of resilience have acceptable

probabilities for the first 15 hours of the attack. This result represents significant improvement in application reliability, even with repair rates that are slightly slower than the compromise rate.

When the repair rate exceeds the compromise rate, the probability of application failure is zero; all rMP applications with $r > 2$ achieve mission-critical reliability. Because the rate of repair is greater than the rate of compromise, the proportion of infected machines always decreases after the initial infection. In this model, the initial infection penetrates a single host. Because all rMP applications with $r > 2$ are resilient to a single Byzantine process failure, they are able to operate through the attack.

5 Summary and Conclusions

We are interested in constructing resilient applications that are able to operate through a wide variety of computer failures and attacks. Our approach is to dynamically replicate processes, detect inconsistencies in their behavior, and restore the level of resiliency as the computation proceeds. In previous work, we have developed *rMP*—a proof-of-concept resilient message-passing technology for distributed applications. This paper presents an analytical framework to perform reliability analysis of our approach to resilience. The framework includes an analytical model of reliability for resilient applications. In addition, three analytical models from fault tolerant literature are included to provide a basis for quantitative analysis: an independent and identically distributed fault model, a correlated fault model, and a computer worm model. The models considered do not span the threat space of distributed applications, but they include a sample of likely failure scenarios as a starting point for reliability analysis.

In general, the rMP approach achieves mission-critical levels of reliability for applications in environments with independent and identically distributed faults and environments experiencing correlated faults. Relative to static fault tolerant technologies, the probability of application failure for resilient applications is several orders of magnitude lower in the presence of correlated faults. This benefit is the result of the rMP technology's ability to dynamically regenerate processes on short time scales during failure bursts.

The reliability of rMP applications in the context of worm propagation is considered with and without worm mitigation measures. Without mitigation, the reliability of rMP applications is unsatisfactory—the technology cannot overcome the complete infection of a network. This result highlights a critical aspect of resilience: no one technology provides absolute reliability. This notion is well recognized in the mission-critical communities in which a collection of security technologies are often applied as a comprehensive defense strategy. The value of this approach is demonstrated by the increase in reliability of rMP applications under computer worm attack when countermeasures are employed. When the countermeasures are able to repair host infections faster than the threat can propagate, rMP applications operate through the attack to complete their mission. This result makes a strong case for measures that slow the propagation of an attack, such as diversification techniques, or that quickly repair failed and infected components, such as regeneration. Overall, we believe that, provided rMP is used as a part of a comprehensive strategy, it achieves acceptable reliability for mission-critical applications.

On a final note, there is a surprising lack of concrete reliability analysis of resilient technologies in the literature. In most cases, analysis is limited to the assessment of scalable multi-processor systems with independent and identically distributed faults. Unfortunately, the analysis of this paper reveals that the independent fault model is relatively insignificant for mission-critical applications. On the other hand, the correlated fault and computer worm models require a number of assumptions about the models themselves, the underlying technology, and

the service requirements of the applications. Perhaps the uncertainty in these assumptions prevents investigators from exploring quantitative metrics. However, even with limited confidence in the underlying assumptions, this analysis provides worthwhile insights into the relative reliability of alternative replication strategies.

Notice

References

[1] D.A. Patterson, G. Gibson, and R.H. Katz, "A case for redundant arrays of inexpensive disks (RAID)," in *Proceedings of the 1988 ACM SIGMOD International Conference on Management Data*, 1988, pp. 109-116.

[2] Lefurgy, X. Wang, and M. Ware, "Server-level power control," in *Proceedings of the International Conference on Autonomic Computing*, 2007, pp. 4-13.

[3] H. Debar, M. Dacier, and A. Wespi. "A revised taxonomy for intrusion-detection systems," *Annals of Telecommunications*, vol. 55, no. 7, pp. 361-378, 2000.

[4] R. Di Pietro and L.V. Mancini, *Intrusion Detection Systems*, New York, Springer-Verlag, 2008.

[5] Singhal, "Intrusion Detection Systems," in *Data Warehousing and Data Mining for Cyber Security*, vol. 31, pp. 43-47, 2007.

[6] G.H. Kim and E.H. Spafford, "The design and implementation of tripwire: A file system integrity checker," in *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, 1994, pp. 18-29.

[7] J. Kaczmarek and M. Wrobel, "Modern approaches to file system integrity checking," in *Proceedings of the 1st International Conference on Information Technology*, 2008.

[8] T. Garfinkel and M. Rosenblum, "A virtual machine introspection based architecture for intrusion detection," in *Proceedings of the Network and Distributed Systems Security Symposium*, 2003, pp. 191-206.

[9] N.L. Petroni, T. Fraser, J. Molina, and W.A. Arbaugh, "Copilot- a Coprocessor-based Kernel Runtime Integrity Monitor," in *Proceedings of the 13th USENIX Security Symposium*, 2004, pp. 179-194.

[10] N.A.Quynh and Y. Takefuji, "Towards a tamper-resistant kernel rootkit detector," in *Proceedings of the 2007 ACM Symposium on Applied Computing*, 2007, pp. 276-283.

[11] R. Riley, X. Jiang, and D. Xu, "Guest-transparent prevention of kernel rootkits with VMM-based memory shadowing," in *Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection*, 2008, pp. 1-20.

[12] H. Zhong and J. Nieh, "CRAK: Linux Checkpoint / Restart As a Kernel Module", Technical Report CUCS-014-01, Department of Computer Science, Columbia University, November 2001.

[13] P. Hargrove and J. Duell, "Berkeley Lab Checkpoint/Restart (BLCR) for Linux Clusters," in *Proceedings of SciDAC 2006*, June 2006.

[14] G. Zheng, C. Huang, L.V. Kale, "Performance Evaluation of Automatic Checkpoint-based Fault Tolerance for AMPI and Charm++," *ACM SIGOPS Operating Systems Review: Operating and Runtime Systems for High-end Computing Systems*, 2006.

[15] S. Sankaran, J. M. Squyres, B. Barrett, A. Lumsdaine, J. Duell, P. Hargrove, and E. Roman, "The LAM/MPI checkpoint/restart framework: System-initiated checkpointing," in *LACSI*, Oct. 2003.

[16] J. Hursey, J.M. Squyres, T.I. Mattox, and A. Lumsdain, "The design and implementation of checkpoint/restart process fault tolerance for OpenMPI," in *Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium*, March 2007.

[17] D.S. Milojicic, F. Douglis, Y. Paindaveine, R. Wheeler and S. Zhou. "Process migration," *ACM Comput. Surv, vol. 32, no. 3*, pages 241-299, 2000.

[18] Chaudhary and H. Jiang, "Techniques for migrating computations on the grid," in Engineering the Grid: Status and Perspective, B. Di Martino et al., Eds. American Scientific Publishers, 2006, pp. 399– 415.

[19] F. Douglis and J. Ousterhout, "Transparent Process Migration: Design Alternatives and the Sprite Implementation," *Software- Practice and Experience,* vol. 2, no. 8, pages 757-785, 1991.

[20] G. Valle, C. Morin, J. Berthou, I. Dutka Malen, and R. Lottiaux, "Process migration based on gobelins distributed shared memory," in *Proceedings of the workshop on Distributed Shared Memory*, pages 325-330, May 2002.

[21] C. Wang, F. Mueller, C. Engelmann, and S. Scott, "Proactive Process-Level Live Migration in HPC Environments," in *Proceedings of the 2008 ACM/IEE Conference on Supercomputing*, 2008.

[22] G. Janakiraman, J. Santos, D. Subhraveti, and Y. Turner, "Cruz: Application-Transparent Distributed Checkpoint-Restart on Standard Operating Systems," in *Proceedings of the International Conference on Dependable Systems and Networks*, pages 260-269, 2005.

[23] S. Osman, D. Subhraveti, G. Su, and J. Nieh, "The Design and Implementation of Zap: A System for Migrating Computing Environments," in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 361-376, 2002.

[24] Shye, T. Moseley, V. J. Reddi, J. Blomstedt and D.A. Connors, "Using Process-Level Redundancy to Exploit Multiple Cores for Transient Fault Tolerance," in *Proceedings of the 37th IEEE/IFIP International Conference on Dependable Systems and Networks*, Edinburgh, UK. June 25-28, 2007.

[25] S. Genaud and C. Rattanapoka, "P2P-MPI: A peer-to-peer framework for robust execution of message passing parallel programs on grids," *Journal of Grid Computing*, vol. 5, pp 27-42, 2007.

[26] R. Brightwell, K. Ferreira, and R. Riesen, "Transparent Redundant Computing with MPI," in *Proceedings of EuroPVM/MPI*, 2010.

[27] T. LeBlanc, R. Anand, E. Gabriel, and J. Subhlok. "VolpexMPI: An MPI Library for Execution of Parallel Applications on Volatile Nodes." in *Proceedings of the 16th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Helsinki, Finland, September, 2009.

[28] R. Batchu, J. Neelamegam, Z. Cui, M. Beddhua, A. Skjellum, Y. Dandass, and M. Apte, "MPI/FTTM: Architecture and taxonomies for fault-tolerant, message-passing middleware for performance-portable parallel," in *Proceedings of the 1st IEEE International Symposium of Cluster Computing and the Grid*. Melbourne, Australia (2001).

[29] J. Lee., S.J. Chapin, and S. Taylor. "Computational Resiliency", *Journal of Quality and Reliability Engineering International*, vol. 18, no. 3, pp 185-199, 2002.
[30] J. Lee and S. Taylor, "Advances in Computational Resiliency," in *Proceedings of the IEEE Aerospace Conference*, Big Sky, Montana, March 2001.
[31] J. Sterbenz, D. Hutchinson, and E. K. Cetinkaya, A. Jabbar, J.P. Rohrer, M. Scholler, and P. Smith "Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines," *Computer Network: Special Issue on Resilient and Survivable Networsk*, vol. 54, no. 9, pp. 1245-1265, June 2010.
[32] A. Avizienis, J.C. Laprie, B. Randell, and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, March 2004.
[33] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem" *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp.382–401, July 1982.
[34] K. McGill and S. Taylor, "Application Resilience with Process Failures," *The 2011 International Conference on Security and Management*, Las Vegas, NV, July 2011.
[35] K. McGill and S. Taylor, "Computational Resiliency for Distributed Applications," *MILCOM 2011*, submitted for publication.
[36] K. McGill and S. Taylor, "Operating System Support for Resilience," *IEEE Transactions on Reliability*, submitted for publication.
[37] K. McGill and S. Taylor, "Comparing swarm algorithms for multi-source localization," In *Proceedings of the 2009 IEEE International Workshop on Safety, Security, and Rescue Robotics*, Denver, Colorado, November 3-6, 2009.
[38] K. McGill and S. Taylor, "Comparing swarm algorithms for large scale multi-source localization," In *Proceedings of the 2009 IEEE International Conference on Technologies for Practical Robot Applications*, Woburn, Massachusetts, November 9-10, 2009.
[39] K. McGill and S. Taylor. "DIFFUSE algorithm for robotic multi-source localization", in *Proceedings of IEEE International Conference on Technologies for Practical Robot Applications*, April 2011.
[40] B. Bhargava and L. Lilien, "Vulnerabilities and Threats in Distributed Systems," *Intl. Conf. on Distributed Computing & Internet Technology (ICDCIT 2004)*, Bhubaneswar, India, Dec. 2004, pp. 146-157.
[41] K.S. Trivedi, D. Kim, A. Roy, and D. Medhi, "Dependability and Security Models," *Proceedings of 7th International Workshop on the Design of Reliable Communication Networks (DRCN 2009)*, Washington, DC, October 2009.
[42] D. M. Nicol, W. H. Sanders, K. S. Trivedi, "Model-Based Evaluation: From Dependability to Security," *IEEE Trans. Dependable and Secure Computing*, vol. 1, no. 1, 2004.
[43] R. J. Ellison, D. A. Fisher, R. C. Linger, H. F. Lipson, T. Longstaff, N. R. Mead, "Survivable Network Systems: An Emerging Discipline," Technical Report CMU/SEI-97-TR-013, 1997.
[44] P. E. Heegard, K. S. Trivedi, "Network survivability modeling," Computer Networks, vol. 53, no. 8, 2009.
[45] H. F. Lipson, D. A. Fisher, "Survivability—a new technical and business perspective on security," *Proc. of NSPW*, 1999.

[46] Y. Liu and K. S. Trivedi, "Survivability Quantification: The Analytical Modeling Approach," *Int. J. Performability Engineering*, vol. 2, no. 1, 2006.
[47] ANSI T1A1.2 Working Group on Network Survivability Performance, Technical Report on Enhanced Network Survivability Performance, ANSI, Tech. Rep. TR No. 68, 2001.
[48] E. Tran and P. Koopman, "Mission Failure Probability Calculations for Critical Function Mechanizations in the Automated Highway System," Carnegie Mellon University Robotics Institute Technical Report CMU-RI-TR-97-44, 1997.
[49] R. Reisen, K. Ferreira, and J. Stearley, "See Applications Run and Throughput Jump: The Case for Redundant Computing in HPC," In *Proceedings of the International Conference on Dependable Systems and Networks*, Chicago, IL, July 2010.
[50] E. N. Elnozahy, J. S. Plank, W. K. Fuchs, "Checkpointing for Peta-Scale Systems: A Look into the Future of Practical Roll-back-Recovery," *IEEE Trans. on Dependable and Secure Computing*, vol. 1, no. 2, 2004.
[51] D. P. Siewiorek, R. S. Swarz, Reliable Computer Systems: Design and Evaluation, 2nd ed., Digital Press, 1992.
[52] T. Courtney et al., "The Möbius Modeling Environment," *Tools of the 2003 Illinois Int'l Multiconference on Measurement, Modelling, and Evaluation of Computer Communication Systems*, Universität Dortmund Fachbereich Informatik research report no. 781, 2003.
[53] D. Tang, R. K. Iyer, "Analysis and Modeling of Correlated Failures in Multicomputer Systems," *IEEE Trans. on Computers*, vol. 41, no. 5, 1992.
[54] B. Schroeder, G. Gibson, "A Large-scale Study of Failures in High-performance-computing Systems," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN2006)*, Philadelphia, PA, USA, June 25-28, 2006.
[55] N. Yigitbasi, M. Gallet, D. Kondo, A. Iosup, and D. Epema "Analysis and Modeling of Time-Correlated Failures in Large-Scale Distributed Systems," Delft University of Technology Parallel and Distributed Systems Report Series, no PDS-2010-004, 2010.
[56] G. Serazzi and S. Zanero, "Computer Virus Propagation Models," In *Tutorials of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS'03*), 2003.
[57] D. Moore, C. Shannon, and J. Brown, "Code-Red: a case study on the spread and victims of an Internet worm," *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, New York, New York, 2002.
[58] S. Staniford, V. Paxson, and N. Weaver, "How to own the internet in your spare time," in *Proceedings of the 11th USENIX Security Symposium*, 2002.
[59] Zou, C.C., Gong, W., Towsley, D. "Code red worm propagation modeling and analysis," in *Proceedings of the 9th ACM conference on Computer and communications security*, 2002.
[60] top500.org, "Operating System Family Share for 6/2011," http://top500.org/stats/list/37/osfam, June 2011 [July, 8 2011].
[61] National Institute of Standards, "The NIST Definition of Cloud Computing," NIST Special Publication 800-145, January 2011.
[62] S. Taylor, M. Henson, M. Kanter, S. Kuhn, K. McGill, and C. Nichols, "Bear—A Resilient Operating System for Scalable Multi-processors," *IEEE Security and Privacy*, Nov/Dec 2011, submitted for publication.

[63] D. Kurzyniec and V. Sunderam, “Failure resilient heterogeneous parallel computing across multidomain clusters,” *International Journal of High Performance Computing Applications*, vol. 19, no. 2, pp. 143-155, 2005.