Fault and Intrusion Tolerance on the Basis of Virtual Machines *

Hans P. Reiser
LaSIGE
Universidade de Lisboa
Portugal
hans@di.fc.ul.pt

Rüdiger Kapitza
Informatik 4
University of Erlangen-Nürnberg
Germany
rrkapitz@cs.fau.de

Abstract

Fault and intrusion tolerance is an important paradigm for building distributed systems that work in spite of accidental and malicious faults. This paper discusses how to harness virtualization technology for building such dependable systems. We show that virtualization promotes a hybrid fault model that allows tolerating malicious intrusions in application domains with little overhead. The proposed architecture features mechanisms for supporting heterogeneity of the replicas. A hypervisor-based replication controller achieves perpetual operation through periodic proactive recovery of the replicas. Re-mapping of state storage between virtual machines speeds up the state transfer of a stateful replicated service. Our VM-FIT prototype implements the core functionality of such a virtualization-based replication architecture. We present some performance measurements and close with a discussion of future research directions.

1 Introduction

The ability to operate correctly in spite of the occurrence of accidental and malicious faults is becoming an important requirement of distributed applications. Intrusion tolerance [24] has become popular as a paradigm for building systems that function correctly in spite of intrusions. This paper discusses the use of virtualization technology for the construction of fault and intrusion tolerant systems.

Virtualization is an old technology that was introduced by IBM in the 1960s [12]. Systems such as Xen [4] made this technology popular on standard PC hardware. Virtualization enables the execution of multiple operating system instances simultaneously in isolated environments on a single physical machine. While mostly being used for issues related to resource management, virtualization can also be used for constructing fault-tolerant systems. The aim of this paper is to investigate to what extent modern virtualization technologies, such as the Xen hypervisor, can be used for constructing dependable systems. We identify the following four main issues:

1. Virtualization technology provides isolation between application domains, the hypervisor, and a privileged system domain. This separation allows adopting a hybrid fault model that supports malicious intrusions within application domains (including the operating system and middleware infrastructure) and a crash-stop model in the isolated system domain.

2. Any virtualization technology provides core mechanisms for the creation and destruction of domains. These mechanisms can be harnessed for designing efficient proactive recovery mechanisms. A timely periodic recovery can be triggered by a recovery service in the isolated system domain. Creating a new domain in parallel to the execution of the other applications permits a reduction of the downtime during recovery to a minimum.

3. Virtualization technology simplifies the introduction of diversity, as the replication logic (in a isolated system domain) can have full control over what operating system and service variant to execute in the application domains.

4. Virtualization can also be applied to disk storage managed by the hypervisor. Virtualized persistent state storage can be used for efficiently re-mapping the state from one application domain to another, and thus allows the implementation of low-cost state transfer strategies.

In the following section, we individually discuss these four aspects of virtualization-based fault and intrusion tolerance. Several of them have been implemented as part of our VM-FIT prototype, which we describe and evaluate in Section 3. Section 4 gives a brief overview of related work, and finally Section 5 concludes.

2 Virtualization-based Fault and Intrusion Tolerance

2.1 Hybrid Fault Model

One key mechanism of virtualization infrastructures such as the Xen hypervisor is the provision of separation between *domains*. In the ideal case, the hypervisor and a protected control domain are fully isolated from one or more application domains that execute

*This work was partially supported by the EC through project IST-2004-27513 (CRUTIAL) and the Large-Scale Informatic Systems Laboratory (LaSIGE).

services. In a *hypervisor-based replication architecture*, application replicas are placed in isolated application domains, while the replication logic resides in a separated system domain.

Virtualization-based replication allows using a *hybrid fault model* that assumes Byzantine failures within application domains and crash-stop failures within the hypervisor and the system domain that contains the replication logic. In this model, the replication controller can use majority voting to detect invalid results from replicas. It is possible to tolerate up to f Byzantine faults using a total number of only $n = 2f + 1$ replicas.

This number is less than the requirements of traditional intrusion-tolerant replication systems, which typically need $n = 3f+1$ replicas [8]. A reduction in the number of required replicas strongly simplifies the provision of independent, heterogeneous implementation versions.

Our VM-FIT architecture [18, 19] is a generic infrastructure for the replication of network-based services on the basis of the Xen hypervisor. VM-FIT uses the hypervisor technology to provide communication and replication logic in a privileged domain, while the actual service replicas are executed in isolated guest domains.

In the RESH (Redundant Execution on a Single Host) variant, VM-FIT allows the redundant execution of a service on a single physical host. In this variant, multiple application domains are used to deploy replicas on a single host. This approach allows the toleration of non-benign random faults in the replicas,such as undetected bit errors in memory, as well as the toleration of software faults by using N-version programming. The REMH (Redundant Execution on Multiple Hosts) supports replication on multiple machines using the same core architecture. The main difference to RESH is the integration of group communication facilities in the replication logic. This variant allows tolerating full crashes of some of the replica hosts, instead of tolerating faults only within a virtual domain.

2.2 Proactive Recovery

Traditional intrusion-tolerant systems [7,8,15] typically use Byzantine fault tolerant replication algorithms, which are able to tolerate a finite number of f faults in group of n replicas. However, these systems face the problem that, given sufficient time, an adversary might be able to compromise more than f replicas. Proactive recovery [9, 16] has been proposed as a solution to overcome this limitation of intrusion-tolerant systems. The core idea is to periodically recover all replicas by reinitializing them from a secure base. For example, a tamper-proof external hardware might be used for rebooting the node from a secure code image. This approach removes potential intrusions; as a result, the number of replica failures that the system can tolerate is limited only within a single recovery round, but is unlimited over the system lifetime.

Under the assumption of malicious faults, it is not possible to trigger the recovery within service replicas, as an intruder can inhibit the recovery of the replica. A classic approach is using dedicated hardware that resets and reinitialises a replica periodically. Using virtualization, the replication logic in a separated, intrusion-free domain can be used as a trusted entity that is able to completely re-initialise the replica domains, without requiring dedicated hardware. The recovery operation initializes the complete replica (including operating systems, middleware, and service instance) with at "clean" state, securely obtaining the service state from other replicas with a fault-tolerant state transfer protocol.

Proactive recovery, however, may reduce availability, as the recovery of a replica reduces the number of available replicas (see also Sousa et al. [21]). This disadvantage can be compensated by increasing the number of replicas, but this not only increases hardware costs and run-time costs, but also makes it more difficult to maintain diversity of the replica implementations. A different approach is to try to minimize the time needed for recovery.

Virtualization technology can help building efficient proactive recovery infrastructures [18]. The hypervisor can be used to shut down and restart a replica running in a virtual machine. In addition, the new replica instance can be started using an additional virtual machine in parallel to the execution of the old replica. This allows a substantial reduction of downtime during recovery, as the boot process does not affect replica availability. After initialisation of the new replica, the replication coordinator can shut down the old replica and trigger the activation of the new one. This approach has some impact on service performance, as the local resources (such as CPU and memory) have to be shared between the replica domains. But on the other hand it minimises the time of complete replica unavailability to the time necessary for the coordinated state transition form the old replica instance to the new one.

2.3 Diversity

Diversity of replicas is essential in intrusion tolerant systems in order to avoid that an adversary can exploit the same vulnerability multiple times within a short time interval. A virtualization-based recovery mechanisms provides an ideal basis for introducing diversity both in space and in time.

A hypervisor-based replication architecture allows a *transparent interception* of the client–service interaction, independent of the guest operating system, middleware, and service implementation. As long as the assumption of deterministic behaviour is not violated, the service replicas may be completely heterogeneous, with different operating systems, middleware, and service implementations.

On the one hand, this independence simplifies the use of heterogeneous versions in the application domains. There is no need to integrate the replication logic in each replica variant. Instead, a common replication mechanisms is implemented in the protected domain. The replica implementations still have to provide some prerequisites for replication, such as having deterministic behaviour and supporting transformation of version-specific state representation in a common abstract state format.

On the other hand, hypervisor-based recovery can be used to provide diversity in time. For example, the operating system can be changed in the new version, or internal configurations can be modified on each recovery reboot. Address space randomization is another popular technique for obtaining diversity.

The provision of multiple deterministic versions of complex applications is not a trivial case. In the FOREVER project [1], we plan to further investigate the introduction of diversity with a virtualization-based recovery service.

2.4 State Transfer

In a stateless replication system, the transition from an old to a new replica version is almost instantaneous. In a stateful system, after creating a new instance of operating system, middleware, and service, the new replica needs to be initialized with the service state, which requires a state transfer. The state transfer increases the time that a replica is unavailable during proactive recovery, as request execution has to be stopped during the creation of a state checkpoint. Furthermore, the state transfer also affects service operation by consuming network and CPU resources.

Virtualization technology can be used to enhance the state transfer to the new replica. During a proactive recovery operation, having both old and new replica running in parallel on a single machine enables a simple and fast state copy in the case that the old replica is not faulty; this fact can be verified by taking a distributed checkpoint on the replicas and verifying the validity of the local state using checksums. Only in the case of an invalid state, a more expensive remote state transfer is necessary.

State transfer must also cope with heterogeneity between replicas. Heterogeneity is introduced by diversity. The transformation of state into local replica-specific representations needs some support from replica implementations. The generic state transfer mechanisms must provide means to adapt the state accordingly, by interacting with this application-level adaptation functionality. We are currently investigating how the disk virtualization mechanisms of the Xen hypervisor can be exploited in order to optimize the state transfer on the basis of disk snapshots and disk remapping.

3 VM-FIT Prototype

The VM-FIT prototype [18, 19] supports hypervisor-based replication of network-based services. A replication controller in a privileged domain intercepts client interaction with a replicated service, handles communication with replicas and voting on replies, and provides support for proactive recovery.

3.1 Architectural Overview

The VM-FIT prototype implements the basic system architecture shown in Figure 1. Service replicas are executed in isolated domains (Dom. Guest). The network interaction from client to the service is handled by the replication manager in the system domain (Dom. 0). The manager intercepts the client connection and distributes all requests to the replica group using the Spread group communication system [2]. Each replica processes the client requests and sends a reply to the node that accepted the client connection. At this point, the replication manager selects the correct reply for the client using majority voting.

The replication logic provides mechanisms for instantiating and initialising service replicas. For each replica variant, a disk image of a Xen virtual machine with a preconfigured operating system and middleware environment has to be provided. After domain initialisation, a state transfer from other replicas to the new domain is triggered. In our prototype, we assume that a secure code basis for

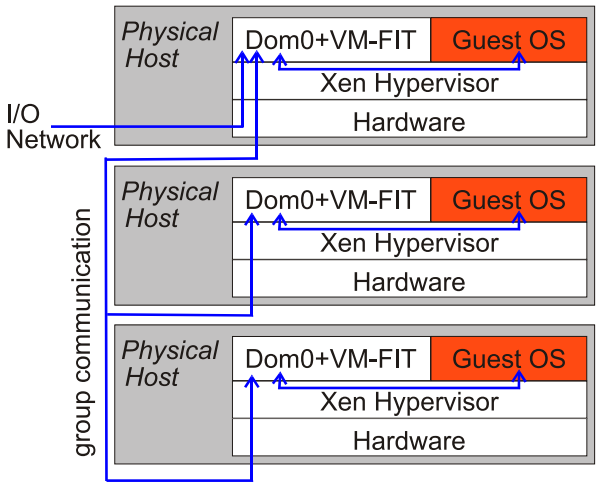

Figure 1. VM-FIT basic replication architecture

the replica is available locally, and only the data state is required to initialise the replica. We assume that the replication logic can request an application-controlled serialisation of the state in an abstract format.

The replication logic also offers support for proactive recovery. Unlike other approaches, the hypervisor-based approach permits the initialisation of the rejuvenated replica instance concurrent to the execution of the old instance. The hypervisor is able to instantiate a second Domain Guest on the same hosts. After initialisation, the replication coordinator can shut down the old replica and trigger the activation of the new one.

The state of the rejuvenated replica needs to be initialised on the basis of a consistent checkpoint of all replicas. As replicas may be subject to Byzantine faults and thus have an invalid state, the state transfer has to be based on an majority agreement of all replicas. The VM-FIT architecture uses the local state of the old replica version on the same host. This state is transferred locally to the new replica, combined with a verification of its validity on the basis of checksums obtained from other replicas. Only if the local state is invalid, a remote state transfer becomes necessary.

The checkpointing and state transfer are time-consuming operations. Furthermore, their duration depends on the state size. During the checkpoint operation, a service is not accessible by clients; otherwise, concurrent state-modifying operations might cause an inconsistent checkpoint. Consequently, there is a trade-off between service availability and safety gained by proactive recovery given by the recovery frequency of replicas. To reduce the unavailability of a service, while still providing the benefits offered by proactive recovery, more than one replica could be recovered at a time. However, in previous systems with dedicated hardware for triggering recovery, the number of replicas recovering in parallel is limited by the fault assumption, as every recovering replica reduces the number of available nodes in a group and, consequently, the number of tolerable faults.

The VM-FIT architecture is able to offer a parallel recovery of all replicas simultaneously. If service replicas have to be recovered, every node receives a checkpoint message and determines the replica state. The replication logic receives this state and prepares a

shadow replica domain. This domain will later be used replace the existing local replica instance and is initialised by the state transfer operation. This approach reduces the downtime due to checkpointing to one checkpoint every recovery period.

3.2 Experimental Results

The current VM-FIT prototype uses the Xen 3.0.3 hypervisor and Linux kernel 2.6.18 both for Domain 0 and for the replica domains. In the following, we describe a few experiments that evaluate the basic proactive recovery scheme, which have first been published in [18]. The experiments examine the behaviour of the VM-FIT proactive recovery architecture for replicating a service on a modern server machine (Sun X4200 server with two dual-core Opteron CPUs at 2.4 GHz and 1 GBit/s switched Ethernet).

In the experiments, a single client on a separate machine sends requests via a LAN network to the service host, which runs 3 replicas of the same network-based service. The replicated service has a very simple functionality: on each client request, it returns a local request counter. It is a simple example of stateful service, which requires a state transfer upon recovery (i.e., the initialization of a new replica with the current counter value). As a performance metric, we measure the number of client requests per second, obtained by counting the number successful requests in 250ms intervals at the client side; in addition we analyse the maximum round-trip time as an indicator for the duration of temporary service unavailability.

We study four different configurations. The first configuration does not use proactive recovery at all. The second configuration implements a "traditional" recovery strategy; every 100s, a replica, selected via a round-robin strategy, is shut down and restarted. A distributed checkpoint of the application state is made before shutting down a replica. This checkpoint ensures that the system can initialize a replica with a correct state (validated by at least $f + 1$ replicas), even if a replica failure occurs concurrent to a recovery operation. The third configuration uses the virtual recovery scheme proposed in this paper: it first creates a new replica instance in a new virtual machine, and then replaces the old instance in the group with the new one. The last configuration uses the same basic idea, but restarts all replicas simultaneously.

The recovery frequency (one recovery each 100s) was selected empirically such that each recovery easily completes within this interval. Typically, a full restart of a replica virtual machine takes less than 50s on the slow machines, and less than 20s on the fast machines. In configuration 4, the recovery of all replicas is started every 300s. This way, the frequency of recoveries per replica remains the same (instead of recovering one out of three replicas every 100s, all replicas are recovered every 300s).

Furthermore, the measurements include the simulation of malicious replicas. Malicious replicas stop sending replies to the VM-FIT replication manager (but continue accepting them on the network), and furthermore perform mathematical computations that cause high CPU load, in order to maximize the potential negative impact on other virtual machines on the same host. Malicious failures occur at time $t_i = 600s + i * 400s, i = 0, 1, 2, \ldots$ at node i mod 3. This implies that the frequency of failures (1/400s) is lower than that of complete recovery cycles (1/300s), consistent with the assumptions we make.

variant \ time	100s.. 400s	650s.. 950s	1050s.. 1350s	600s.. 1800s	max. RTT
A	4547	4479	0	(-)	∞
B	4502	3879	3726	3702	45s
C	4086	4046	4112	4067	1s
D	4169	3992	3960	3992	<250ms

Table 1. Average performance (requests/s) and worst-case RTT observed at the client on a multi-CPU machine

The measurement in Figure 2 shows that, without proactive recovery, there is no significant performance degradation after the first replica fault. Due to the availability of multiple CPUs, each replica can use a different CPU, and thus the faulty replica has (almost) no negative impact on the other replicas. After the second replica failure, the service becomes unavailable. In variant (B), periodic recovery works well in the absence of failures ($t < 600s$). The recovering replica disconnects from the replica group, and thus the replica manager has to forward requests only to the remaining nodes, resulting again in a speed-up during recovery. A faulty node in parallel to a replica recovery, however, causes periods of unavailability (see markers on X-axis). In variant (C), there is no noticeable service degradation during replica recovery. The only visible impact are two short service interruptions, which occur at the beginning of the creation of a new virtual machine and at the moment of state transfer and transition from old to new replica. These interruptions typically show system unavailability during a single $250ms$ measurement interval only. Similar observations also hold for variant (D).

Table 1 shows the average system performance in an interval without failures ($t = 100s \ldots 400s$), after the first failure ($t = 650s \ldots 400s$), after the second failure ($t = 1050s \ldots 1350s$) and in a large interval with failures ($t = 600s \ldots 1800s$). It can be observed that the first recovery strategy (B) has almost no influence on system throughput; variants (C) and (D) reduce the performance of the service by 10% and 8%, respectively, during the period without faults. With faulty replicas, the average throughput drops significantly in variant (B) due to the temporary service unavailability, while it remains almost constant in the case of (C) and (D).

3.3 Discussion

The measurements demonstrate that the VM-FIT proactive recovery schemes (C and D) are superior to the simple one (B). While there is not much difference in the average throughput, the simple scheme causes long periods of unavailability, which is undesirable in practice. The unavailability could be compensated by increasing the number of replicas. In practice, this would make implementation diversity more difficult (more different versions are needed). Furthermore, in a virtual replication scenario on a single physical host, adding another replica on that host would reduce the system performance.

The experiments only considered replication a single physical machine. The same proactive recovery mechanisms can also be used in VM-FIT for replication on multiple physical hosts. In this case,

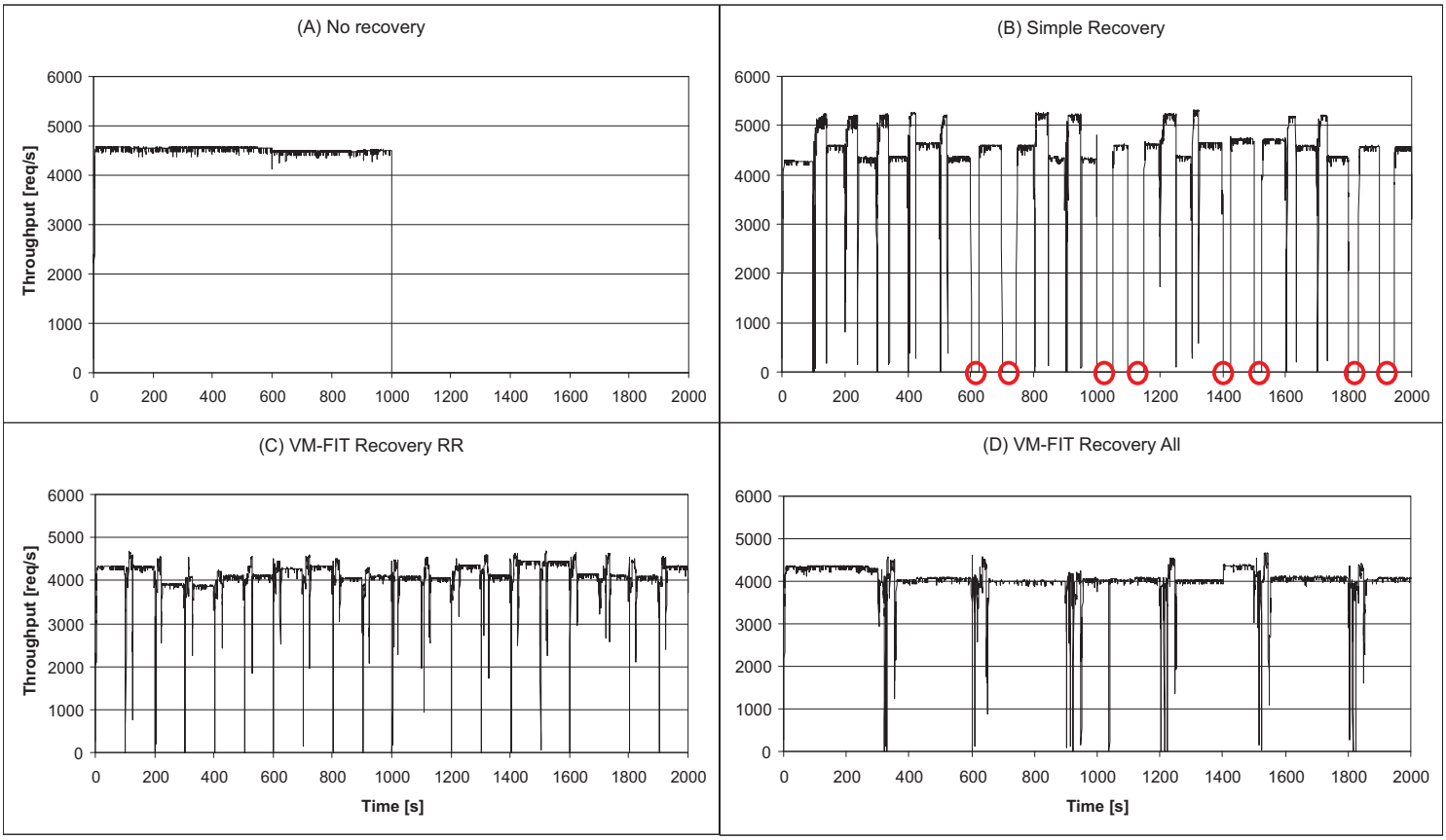

Figure 2. Throughput measurements on multi-CPU machine

client requests are distributed to all nodes using totally ordered group communication. The request distribution is the same for all variants of proactive recovery and thus will not have much impact on the relative performance. The main difference will be that there is no impact of a recovering node on the other replicas.

In the prototype, no attempts towards formal verification of the trusted component have been made. Indeed, the same operating system, an off-the-shelf Linux distribution, is used for Domain 0 and Domain Guest. As a consequence, vulnerabilities at the operating system level are currently present in both domains. We expect, however, that this is only a limitation of the early prototype. In future work we plan to use a hardened Linux system as Domain 0, or even use a minimalistic operating system that might permit formal verification.

4 Related Work

Virtualization has become popular on standard PC hardware by systems such as Xen [4] and VMware [22]. Virtualization enables the execution of multiple operating system instances simultaneously in isolated environments on a single physical machine. The L4Ka microkernel also offers virtualization functionality [13]. In addition, significant efforts towards a formal verification of the L4 kernel have been made by other researchers [23]. These results provide an excellent basis for justifying a hybrid fault model that assumes an intrusion-free hypervisor and system domain.

While mostly being used for issues related to resource management, virtualization has previously been applied for constructing fault-tolerant systems. Bressoud and Schneider [5] demonstrated the use of virtualization for lock-stepped replication of an application on multiple hosts.

Besides such direct replication support, virtualization can also help to encapsulate and avoid faults. The separation of system components in isolated virtual machines reduces the impact of faulty components on the remaining system [14]. Furthermore, the separation simplifies formal verification of components [23]. In this paper, we do not focus on these matters in detail. However, such solutions provide important mechanisms that help to further justify the assumptions that we make on the isolation and correctness of a trusted entity.

Using virtualization is also popular for intrusion detection and analysis. Several systems transparently inspect a guest operating system from the hypervisor level [10, 11]. Such approaches are not within the scope of this paper, but they are ideally suited to complement our approach. Intrusion detection and analysis can be used to detect and analyse potential intrusions, and thus help to pinpoint and eliminate flaws in systems that could be exploited by attackers.

Several authors have previously used proactive recovery in Byzantine fault tolerant systems [3, 6, 9, 16]. It is a technique that periodically refreshes nodes in order to remove potential intrusions. The BFT protocol of Castro and Liskov [9] periodically creates stable checkpoints. The authors recognize that the efficiency of state

transfer is essential in proactive recovery systems; they propose a solution that creates a hierarchical partition of the state in order to minimize the amount of data to transfer.

Sousa et al. [21] specifically discuss the problem of reduced system availability during proactive recovery of replicas. The authors define requirements on the number of replicas that avoid potential periods of unavailability given maximum numbers of simultaneously faulty and recovering replicas. Our approach instead reduces the unavailability problem during recovery by performing most of the initialization of a recovering replica in parallel to normal system operation using an additional virtual machine.

In a recent publication, Silva et al. [20] use an approach similar to ours for software rejuvenation. The main difference is that these authors focus on recovering from error situations caused by "software ageing".

Ramasamy and Schunter [17] use combinatorial modelling to analyse how the use of virtualization can affect system dependability. Such a careful analysis allows a better judgement on the conditions that are necessary to make a system such as VM-FIT more reliable than non-replicated one.

5 Summary

This paper discussed the benefits that virtualization offers for constructing fault-tolerant systems. The most important benefits of such an approach are the use of a hybrid fault model that allows tolerating malicious intrusions with a minimum number of replicas; the support for heterogeneous replicated applications, middleware and operating systems on top of a hypervisor-based replication infrastructure; the support for efficient proactive recovery operations; and the potential for optimized checkpointing and state transfer using virtualization mechanism.

References

[1] FOREVER: Fault/intrusiOn REmoVal through Evolution & Recovery; http://forever.di.fc.ul.pt/.

[2] Y. Amir, C. Nita-Rotaru, J. Stanton, and G. Tsudik. Secure spread: An integrated architecture for secure group communication. *IEEE Trans. on Dependable and Secure Computing*, 02(3):248–261, 2005.

[3] B. Barak, A. Herzberg, D. Naor, and E. Shai. The proactive security toolkit and applications. In *CCS '99: Proc. of the 6th ACM conference on Computer and communications security*, pages 18–27, New York, NY, USA, 1999. ACM Press.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03: Proc. of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM Press.

[5] T. C. Bressoud and F. B. Schneider. Hypervisor-based fault tolerance. *ACM Trans. Comput. Syst.*, 14(1):80–107, 1996.

[6] C. Cachin, K. Kursawe, A. Lysyanskaya, and R. Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *CCS '02: Proc. of the 9th ACM conference on Computer and communications security*, pages 88–97, New York, NY, USA, 2002. ACM Press.

[7] C. Cachin and J. A. Poritz. Secure intrusion-tolerant replication on the internet. In *Intl. Conf. on Dependable Systems and Networks*, pages 167–176, 2002.

[8] M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *OSDI '99: Proc. of the 3rd Symp. on Operating Systems Design and Implementation*, pages 173–186. USENIX Association, 1999.

[9] M. Castro and B. Liskov. Proactive recovery in a byzantine-fault-tolerant system. In *4th Symp. on Operating Systems Design and Implementation (OSDI)*, San Diego, USA, Oct. 2000.

[10] G. W. Dunlap, S. T. King, S. Cinar, M. A. Basrai, and P. M. Chen. ReVirt: enabling intrusion analysis through virtual-machine logging and replay. *SIGOPS Oper. Syst. Rev.*, 36(SI):211–224, 2002.

[11] T. Garfinkel and M. Rosenblum. A virtual machine introspection based architecture for intrusion detection. In *Proc. Network and Distributed Systems Security Symposium*, February 2003.

[12] R. P. Goldberg. Architecture of virtual machines. In *Proc. of the workshop on virtual computer systems*, pages 74–112, New York, NY, USA, 1973. ACM Press.

[13] J. LeVasseur, V. Uhlig, M. Chapman, P. Chubb, B. Leslie, and G. Heiser. Pre-virtualization: soft layering for virtual machines. Technical Report 2006-15, Fakultät für Informatik, Universität Karlsruhe (TH), July 2006.

[14] J. LeVasseur, V. Uhlig, J. Stoess, and S. Götz. Unmodified device driver reuse and improved system dependability via virtual machines. In *Proc. of the 6th Symposium on Operating Systems Design and Implementation*, San Francisco, CA, Dec. 2004.

[15] D. Malkhi and M. Reiter. Byzantine quorum systems. In *STOC '97: Proc. of the twenty-ninth annual ACM symposium on Theory of computing*, pages 569–578, New York, NY, USA, 1997. ACM Press.

[16] R. Ostrovsky and M. Yung. How to withstand mobile virus attacks (extended abstract). In *PODC '91: Proc. of the tenth annual ACM symposium on Principles of distributed computing*, pages 51–59, New York, NY, USA, 1991. ACM Press.

[17] H. V. Ramasamy and M. Schunter. Architecting dependable systems using virtualization. In *Workshop on Architecting Dependable Systems: Supplemental Volume of the 2007 International Conference on Dependable Systems and Networks (DSN-2007)*.

[18] H. P. Reiser and R. Kapitza. Hypervisor-based efficient proactive recovery. In *Proc. of the 26th IEEE Symposium on Reliable Distributed Systems - SRDS'07 (Oct 10-12, 2007, Beijing, China)*, pages 83–92, 2007.

[19] H. P. Reiser and R. Kapitza. VM-FIT: supporting intrusion tolerance with virtualisation technology. In *Proceedings of the 1st Workshop on Recent Advances on Intrusion-Tolerant Systems (in conjunction with Eurosys 2007, Lisbon, Portugal, March 23, 2007)*, pages 18–22, 2007.

[20] L. M. Silva, J. Alonso, P. Silva, J. Torres, and A. Andrzejak. Using virtualization to improve software rejuvenation. In *Proc. of the 6th IEEE Int. Symp. on Network Computing and Applications (NCA 2007)*, volume 00, pages 33–44. IEEE Computer Society, 2007.

[21] P. Sousa, N. F. Neves, P. Verissimo, and W. H. Sanders. Proactive resilience revisited: The delicate balance between resisting intrusions and remaining available. In *SRDS '06: Proc. of the 25th IEEE Symposium on Reliable Distributed Systems (SRDS'06)*, pages 71–82, Washington, DC, USA, 2006. IEEE Computer Society.

[22] J. Sugerman, G. Venkitachalam, and B.-H. Lim. Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor. In *Proc. of the General Track: 2002 USENIX Annual Technical Conference*, pages 1–14, Berkeley, CA, USA, 2001.

[23] H. Tuch, G. Klein, and G. Heiser. Os verification — now! In M. Seltzer, editor, *Proc. 10th Workshop on Hot Topics in Operating Systems (HotOS X)*, 2005.

[24] P. E. Veríssimo, N. F. Neves, and M. P. Correia. Intrusion-tolerant architectures: Concepts and design. In *Architecting Dependable Systems*, volume Volume 2677/2003, pages 3–36. Springer Berlin / Heidelberg, 2003.