Hypervisor-Based Efficient Proactive Recovery

Hans P. Reiser
Departamento de Informática,
University of Lisboa, Portugal
hans@di.fc.ul.pt

Rüdiger Kapitza
Department of Computer Sciences 4
University of Erlangen-Nürnberg, Germany
kapitza@informatik.uni-erlangen.de

Abstract

Proactive recovery is a promising approach for building fault and intrusion tolerant systems that tolerate an arbitrary number of faults during system lifetime. This paper investigates the benefits that a virtualization-based replication infrastructure can offer for implementing proactive recovery. Our approach uses the hypervisor to initialize a new replica in parallel to normal system execution and thus minimizes the time in which a proactive reboot interferes with system operation. As a consequence, the system maintains an equivalent degree of system availability without requiring more replicas than a traditional replication system. Furthermore, having the old replica available on the same physical host as the rejuvenated replica helps to optimize state transfer. The problem of remote transfer is reduced to remote validation of the state in the frequent case when the local replica has not been corrupted.

1 Introduction

The ability to tolerate malicious intrusions is becoming an increasingly important feature of distributed applications. Nowadays, complex large-scale computing systems are interconnected with heterogeneous, open communication networks, which potentially allow access from anywhere. Application complexity has reached a level in which it seems impossible to completely prevent intrusions. In spite of all efforts to increase system security in the past decade, computer system intrusions are commonplace events today.

Traditional intrusion-tolerant replication systems [6, 7, 17] are able to tolerate a limited amount of faulty nodes. A system with n replicas usually can tolerate up to $f < n/3$ replicas that fail in arbitrary, malicious ways. However, over a long (potentially unlimited) system lifetime, it is most likely that the number of successful attacks exceeds this limit. Proactive recovery [2, 5, 8, 18] periodically recovers replicas from potential penetrations by reinitializing them from a secure base. This way, the number of replica failures that can be tolerated is limited within a single recovery round, but is unlimited over the system lifetime.

Support for proactive recovery reduces the ability to tolerate genuine faults, or requires a higher number of replicas to maintain the same level of resilience [22]. In the first case, the recovery of a node is perceived as a node failure. Thus, the number of real failures that the system is able to tolerate in parallel to the recovery is reduced. For example, in a typical system with $n = 4$ replicas, which is able to tolerate a single intrusion, no additional failure can be tolerated at all during recovery. In the second case, additional replicas have to be supplied. In the previous example, using $n = 6$ replicas would allow tolerating one malicious fault in parallel to an intentional recovery [21].

Both variants have disadvantages in practice. Adding more replicas not only increases hardware costs and runtime costs, but also makes it more difficult to maintain diversity of the replica implementations. Intrusion-tolerant systems face the problem that if an intruder can compromise a particular replica, he might similarly be able to compromise others [10]. Diversity is a useful tool to mitigate this problem [9, 14]. On the other hand, not adding more replicas increases the risk of unavailability during proactive recoveries, which is also inconsistent with the target of reliable distributed applications.

This paper proposes a solution that does not require additional replicas, but still minimizes unavailability caused by proactive recoveries. The proposed architecture uses a hypervisor (such as Xen [3]) that provides isolation between applications subject to Byzantine failures and secure components that are not affected by intrusions. The hypervisor is able to shut down and reboot any operating system instance of a service replica, and thus can be used for proactive recoveries.

The novel contribution of this paper is a seamless proactive recovery system that uses the hypervisor to instantiate a new system image before shutting down the replica to be recovered. In a stateless replication system, the transition

from old to new replica version is almost instantaneous. This way, the recovery does not cause a significant period of unavailability. In a stateful system, a new instance of operating system, middleware, and replica can be created in parallel to system operation, but the initialization of the new replica requires a state transfer. Besides improving the start-up phase, our approach also enhances the state transfer to the new replica. 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.

The proactive recovery design presented in this paper applies to systems with replication across multiple physical replicas as well as to virtual replication systems on a single host, such as our RESH [19] system, which use locally redundant execution of heterogeneous service versions for tolerating random transient faults as well as malicious intrusions. A hybrid system model that assumes Byzantine failures in application replicas and a crash-stop behaviour of the replication logic allows the toleration of f Byzantine failures using only $2f + 1$ replicas.

This paper is structured as follows. The next section discusses related work. Section 3 describes the VM-FIT system. Section 4 presents in detail the proposed architecture for proactive recovery. Section 5 evaluates our prototype, and Section 6 concludes.

2 Related Work

Virtualization is an old technology that was introduced by IBM in the 1960s [15]. Systems such as Xen [3] and VMware [23] 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. Bressoud and Schneider [4] demonstrated the use of virtualization for lock-stepped replication of an application on multiple hosts. Our RESH architecture [19] proposes redundant execution of a service on a single physical host using virtualization. This approach allows the toleration of non-benign random faults such as undetected bit errors in memory, as well as the toleration of software faults by using N-version programming [1]. The VM-FIT architecture [20] extends the RESH architecture for virtualization-based replication control across 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 [16]. Furthermore, the separation simplifies formal verification of components [24]. 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 [12, 13]. 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 [2, 5, 8, 18]. It is a technique that periodically refreshes nodes in order to remove potential intrusions. The BFT protocol of Castro and Liskov [8] 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.

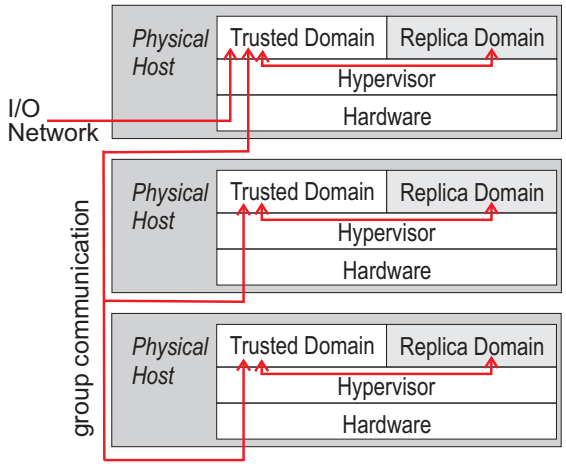
Sousa et al. [22] specifically discuss the problem of reduced system availability during proactive recovery of replicas. The authors define requirements on the number of replicas that avoid potential periods of unavailability given maximum numbers of simultaneously faulty and recovering replicas. Our 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.

3 The VM-FIT Architecture

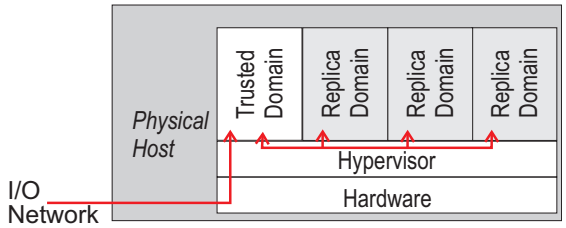
The VM-FIT architecture [20] is a generic infrastructure for the replication of network-based services on the basis of the Xen hypervisor. Throughout this text, we use the terminology of Xen: the *hypervisor* is a minimal layer running at the bare hardware; on top, service instances are executed in *guest domains*, and a privileged *Domain 0* controls the creation and execution of the guest domains. 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. The system transparently intercepts remote communication between clients and replicas below the guest domain.

We assume that the following properties hold:

- All client–service interaction is intercepted at the network level. Clients exclusively interact with a remote



(a) REMH — Replication on Multiple Hosts



(b) RESH — Replication on a Single Host

Figure 1. Basic VM-FIT replication architecture

service on the basis of request/reply network messages.

- The remote service can be modelled as a deterministic state machine.
- Service replicas, including their operating system and execution environment, may fail in arbitrary (Byzantine) ways. At most $f < \lfloor \frac{n-1}{2} \rfloor$ replicas may fail within a single recovery round.
- The other system parts (hypervisor and trusted domain) fail only by crashing.

We assume that diversity is used to avoid identical attacks to be successful in multiple replicas. It is not necessary that replicas execute the same internal sequence of machine-level instructions. The service must have logically deterministic behaviour: the service state and the replies sent to clients are uniquely defined by the initial state and the sequence of incoming requests. In order to transfer state to a recovering replica, each replica version must be able to convert its internal state into an external, version-independent state representation (see Section 3.3).

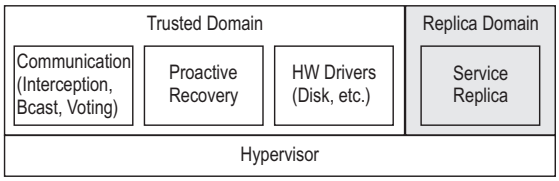


Figure 2. Internal composition of the VM-FIT architecture

The VM-FIT architecture relies on a *hybrid fault model*. While the replica domains may fail in an arbitrarily malicious way, the trusted components fail only by crashing. We justify this distinction primarily by the respective code size of the trusted entity and a complex services instance, which includes the service implementation, middleware, and operating system (see Section 3.2).

3.1 Replication Support

The basic system architecture of VM-FIT for replication on multiple hosts is shown in Figure 1(a). The service replicas are running in isolated virtual machines called *replica domains*. The network interaction from client to the service is handled by the replication manager in the *trusted domain*. The manager intercepts the client connection and distributes all requests to the replica group using a group communication system. Each replica processes client requests and sends replies to the node that accepted the client connection. At this point, the replication manager selects the correct reply for the client using majority voting.

This 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.

The VM-FIT system may also be used for replicating a service in multiple virtual domains on a single physical hosts, as shown in Figure 1(b). This configuration is unable to tolerate a complete failure of the physical host. It can, however, tolerate malicious intrusions and random faults in replica domains. Thus, it provides a platform for N-version-programming on a single physical machine.

3.2 Internal Structure of the VM-FIT Architecture

Our prototype implementation of VM-FIT places all internal components of the replication logic on Domain 0 of a standard Xen system. The replica domains are the only

components in the system that may be affected by non-benign failures. Figure 2 illustrates thes internal composition of VM-FIT.

For redundant execution on a single host, the components for replica communication, proactive recovery, and local hardware access are non-replicated. Thus, they cannot tolerate faults, and Byzantine faults must be restricted to the replica domains. For replication on multiple hosts, a fine-grained definition of the failure assumptions on the components of VM-FIT permits alternative implementations that allow to partially weaken the crash-stop assumption. We distinguish the following parts:

Hypervisor: The hypervisor has full control of the complete system and thus is able to influence and manipulate all other components. Intrusions into the hypervisor can compromise the whole node; thus, it must be a trusted entity that fails only by crashing.

Replica Communication: The replica communication comprises the device driver of the *network device*, a *communication stack* (TCP/IP in our prototype system), the replication logic that handles *group communication* to all replicas, and *voting* on the replies created by the replicas. A crash-stop model for this component allows the use of efficient group-communication protocols and the toleration of up to $f < n/2$ malicious faults in the service replicas. As an alternative, Byzantine fault tolerant mechanisms for group communication and voting can be used, which usually can tolerate up to $f < n/3$ Byzantine faults in a group of n nodes.

Proactive Recovery: The proactive recovery part handles the functionality of shutting down and restarting virtual replica domains. Intrusions into this component can inhibit correct proactive recovery. Thus, this component must fail only by crashing. In addition, in order to guarantee that recoveries are triggered faster then a defined maximum failure frequency, this component must guarantee *timely* behaviour.

Local hardware access: Access to hardware (such as disk drives) requires special privileges in a hypervisor-based system. In Xen, these are generally handled by Domain 0. Malicious intrusions in such drivers may invalidate the state of replica domains. In a REMH configuration, such faults can be tolerated.

In Xen, Domain 0 usually runs a complete standard Linux operating system. Thus, the complexity of this privileged domain might put the crash-stop assumption into question, as any software of that size is frequently vulnerable to malicious intrusions and implementation faults. In previous work [20], we have proposed the separation of replica communication from Domain 0 in a REMH environment, by creating a new trusted "Domain NV" (network and voting). The rational between that is the following:

- All external communication and the replication support is removed from Domain 0, thus eliminating the possibility of network-based attacks on Domain 0.
- Domain NV can be implemented as a minimalistic system, which is easier to verify than a full-blown Linux system, thus reducing the chances that exploitable bugs exist.
- Domain NV can be implemented for a crash-stop and for a Byzantine failure model. While the first variant allows cheaper protocols and requires less replicas, the second one can tolerate even malicious intrusions into the replica communication component.

In the evaluation of this paper, we assume the simple system model with intrusion only in the replica domains. Domain 0 of Xen is used as a trusted domain with crash-stop behaviour and hosts all other functional parts of VM-FIT.

3.3 Application State

We assume that the replica state is composed of the general *system state* (such as internal data of operating system and middleware) and the *application state*. The system state can be restored to a correct version by just restarting the replica instance from a secure code image. The system state is not synchronized across replicas, but it is assumed that potential inconsistencies in the system state have no impact on the observable request/reply behaviour of replicas. The application state is the only relevant state that needs to be kept consistent across replicas.

While the internal representation of the application state may differ between heterogeneous replica implementations, it is assumed that the state is kept consistent from a logical point of view, and that all replicas are able to convert the internal into an external representation of the logical state, and vice versa. Such an approach requires explicit support in the replica implementations to externalize their state.

This way, a new replica can be created by first starting it from a generic code base, and then by transferring the application state from existing replicas. In a Byzantine model with up to f faulty nodes, at least $f + 1$ replicas must provide an identical state in order to guarantee the validity of the state. This can be assured either by transferring the state multiple times, or by validating the state of a single replica with checksums from f other replicas.

Xen is able to transfer whole domain images in a transparent way [11], and thus it might be considered for state transfer in a replication system as well. However, such an approach means that the memory images of "old" and "new" instances have to be completely identical. This contradicts our goal of using heterogeneous replica version in order to benefit from implementation diversity. Furthermore, our architecture is unable to ensure 100% deterministic execution of operating system instances, and thus replica

domains on different host are likely to have different internal state (for example, they may assign different internal timestamps and may assign different process IDs to processes). The separation of application state and irrelevant internal system state thus is a prerequisite for our replication architecture with proactive recovery.

Besides allowing for heterogeneity, the separation of system state and application state also helps to reduce the amount of state data that needs to be transferred. No (potentially large) system state needs to be transferred, the transfer is limited to the minimally necessary applications state.

4 Virtualization-based Proactive Recovery

In this section, we explain the design of the infrastructure for proactive recovery in the VM-FIT environment. All service replicas are rejuvenated periodically, and thus potential faults and intrusions are eliminated. As a result, an upper bound f on the number of tolerable of faults is no longer required for the whole system lifetime, but only for the duration of a rejuvenation cycle. The first advantage of the VM-FIT approach is the ability to create new replicas instances before shutting down those to be recovered. We assume the use of off-the-shelf operating systems and middleware infrastructures, which typically need tens of seconds for startup. In our approach, this startup can be done before the transition from old to new replica, thus minimizing the time of unavailability during proactive recoveries. The second benefit of our approach is the possibility to avoid costly remote state transfer in case that the local application state has not been corrupted.

4.1 Overview

A replication infrastructure that supports proactive recovery has to have a trusted and timely system component that controls the periodic recoveries. It is not feasible to trigger the recovery within a service replica, as a malicious intrusion can cause the replica component to skip the desired recovery. Thus, the component that controls the recoveries must be separated from the replica itself. For example, a tamper-proof external hardware might be used for rebooting the node from a secure code image. In the VM-FIT architecture, the logic for proactive recovery support is implemented in a trusted system component on the basis of virtualization technology.

The proactive recovery component in VM-FIT is able to initialize all elements of a replica instances (i.e., operating systems, middleware, and service instance) with a "clean" state. The internal system state (as defined in Section 3.3) is rejuvenated by a reboot from a secure code image, and the application state is refreshed by a coordinated state-transfer mechanisms from a majority of the replicas.

```
1  Upon periodic trigger of checkpoint:
2      create and boot the new virtual machine
3      wait for service start-up in new virtual machine
4      stop request processing
5      broadcast CHECKPOINT command to all replicas
6
7  Upon reception of CHECKPOINT command:
8      create local checkpoint
9      resume request processing on non-recovering replicas
10     transfer application state to new replica domain
11
12 Upon state reception by the new replica:
13     replace old replica with new replica in replication logic
14     start to process client requests by new replica
15     shut down old replica
```

Figure 3. Basic proactive recovery strategy

Unlike other approaches to proactive recovery, the hypervisor-based approach permits the initialization of the rejuvenated replica instance concurrent to the execution of the old instance. The hypervisor is able to instantiate an additional replica domain on the same hosts. After initialization, the replication coordinator can trigger the activation of the new replica and shut down the old one (see Figure 3). This way, the downtime of the service replica is minimized to the time necessary for the creation of a consistent checkpoint and the reconfiguration the replication logic for the new replica.

As discussed by Sousa et al. [22], the recovery of a node has an impact on either the ability to tolerate faults or on the system availability. The VM-FIT architecture avoids the costs of using additional spare replicas for maintaining availability during recovery. Instead, it accepts the temporary unavailability during recovery, and uses the advantages of virtualization in order to minimize the duration of the unavailability. Instead of tens of seconds that a complete reboot of a replica typically takes with a standard operating system, the unavailability due to a proactive recovery is reduced to fractions of a second.

The state of the rejuvenated replica needs to be initialized 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 confirmation of $f + 1$ replicas.

For transferring application state, the VM-FIT architecture is able to exploit the locality of the old replica version on the same host. The actual state is transferred locally, with a verification of its validity on the basis of checksums obtained from other replicas. That is, for example, in line 9 of Figure 3, only a recovering replica transfers its state to the trusted communication component. All other replicas com-

pute and disseminate a hash value that identifies the state. We discuss this state-transfer issue in more detail in the Section 4.2.

In summary, virtualization-based proactive recovery in VM-FIT allows restarting service replicas without additional hardware support in an efficient way.

4.2 State Transfer and Proactive Recovery

For proactive recovery, every recovery operation requires a consistent checkpoint of $f + 1$ replicas. This checkpoint has to be transferred to the recovering replica; the target should be able to verify the validity of the checkpoint. Finally, the recovering replica has to be reinitialized by the provided application state. In our prototype, we assume that a secure code basis for the replica is available locally, and only the application state is required to initialize the replica.

Creating checkpoints and and transferring state 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 at a time by doing all three steps necessary for proactive recovery in parallel. The trusted domain prepares a *shadow replica domain*. This domain will later be used replace the existing local replica instance. After startup of the new virtual replica, every replica receives a checkpoint message and determines its state. Thereby, the state is provided as a stream and checksums on the stream data are generate for a configurable block size. These checksums are distributed to all other nodes hosting replicas of the service. Before a certain block is used for the initialization of the shadow replica, it has to be verified by the majority of all state-providing replicas via the checksums. If a block turns out to be faulty, it is requested from one of the nodes of the majority. After the state transfer, every replica has a fully initialized shadow replica that is a member of the replication group. In a final step, the old replicas can be safely shut down as the shadow replicas already substitute them.

This approach reduces the downtime due to checkpointing to one checkpoint every recovery period. Furthermore, the amount of transferred data over the network is reduced as only faulty blocks have to be requested from other nodes. Finally, the state transfer is sped up in the average case as only checksums have to be transferred.

5 Experimental Evaluation

Our prototype of the VM-FIT architecture allows the replication of network-based services. It uses the Xen 3.0.3 hypervisor and Linux kernel 2.6.18 both for Domain 0 and for the replica domains.

5.1 VM-FIT Setup for Replication on a Single Physical Host

The following two experiments examine the behaviour of the VM-FIT proactive recovery architecture for replicating a service on a single machine. In the first experiment, we use a simple desktop machine with a single CPU, while in the second experiment we host the replicas on a modern server machine with two dual-core CPUs.

In both 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 com-

pletes 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.

5.2 VM-FIT Measurements on a Single-CPU Machine

In the first experiment, the VM-FIT-based replicas are placed on a desktop PC with a 2.66 GHz Pentium 4 CPU and 100MBit/s switched Ethernet. Figure 4 shows a typical run of the experiment. Without proactive recovery (A), the impact of the first replica fault at $t = 600s$ is clearly visible. All replicas run on the same CPU, which means that the CPU load caused by the faulty replica has an impact on the other replicas. The average performance drops from about 900 requests/s about 750 requests/s (-17%). After the second replica failure at $t = 1000s$, the service becomes unavailable.

The simple recovery scheme (B) works well in the absence of concurrent failures (i.e., for $t < 600s$). Two replicas continue to provide the service, while the third one recovers; the shut-down of a replica even causes a brief speed-up of the throughput of the remaining replicas to over 1200 requests/s. After the first replica failure, the system becomes unavailable during recovery periods (see markers on the X-axis at $t = 600s, 700s, 1000s, 1100s, \ldots$), for a duration of approximately 40...50 seconds. Only a single replica remains available besides the faulty one and the recovering one, which is insufficient for system operation. After the recovery of the faulty node, the system again behaves as in the beginning. For example, replica R_1 becomes faulty at $t = 600s$, and recovers at $t = 700s$.

The VM-FIT round-robin virtual recovery scheme (C) avoids such periods of unavailability. The creation of a new replica in parallel to the existing instances has some minor impact on the performance, which drops to 710 requests/s on average, but does not inhibit system operation. The parallel recovery of all nodes (D) creates a higher system load

variant \ time	100s.. 400s	650s.. 950s	1050s.. 1350s	600s.. 1800s	max. RTT
A	904	752	0	(-)	∞
B	859	667	687	638	48s
C	783	612	636	633	1s
D	798	701	554	624	<250ms

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

during the instantiation of the virtual machines, and the duration of the instantiation takes a longer time (e.g., throughput drops to 545 requests/s on average during the first recovery cycle and recovery duration is $115s$) However, only one distributed checkpoint is needed for rejuvenating three replicas.

Table 1 provides a more precise comparison of the system performance. The values show the average number of requests per second in an interval without failures ($t = 100s \ldots 400s$), after the first failure ($t = 650s \ldots 950s$), after the second failure ($t = 1050s \ldots 1350s$), and in a large interval with failures ($t = 600s \ldots 1800s$). An observation interval of $300s$ (or multiples thereof) ensures a fair comparison between all variants, as the same number of recoveries happen in the variants (B), (C), and (D).

In terms of throughput in a failure-free run, the version without proactive recovery is the most efficient. Variant (B) reduces this throughput by about 5%; (C) and (D) reduce throughput by 13% and 12%, respectively. The advantage of variant (B) over (C) and (D) vanishes in the presence of faulty nodes. A closer observation reveals that in case (B), client requests are delayed for up to $48s$ during recoveries, while in case (C) and (D), the maximum round-trip time does not exceed $1s$. The average throughput of (C) and (D) is almost identical. In the experiment, the application state consists only of a single number, and thus state serialization and transfer is cheap. We expect that in systems with large application state (and thus high costs of state serialization), variant (D) will be superior to (C) due to the reduced frequency of checkpoint creations.

5.3 VM-FIT Measurements on a Multi-CPU Machine

In the second experiment, the VM-FIT-based replicas are placed on a Sun X4200 server with two dual-core Opteron CPUs at 2.4 GHz and 1 GBit/s switched Ethernet.

Figure 5 shows the performance obtained with this setup. Unlike in the first experiment, the efficiency without proactive recovery shows no significant performance degradation after the first replica fault. Due to the availability of mul-

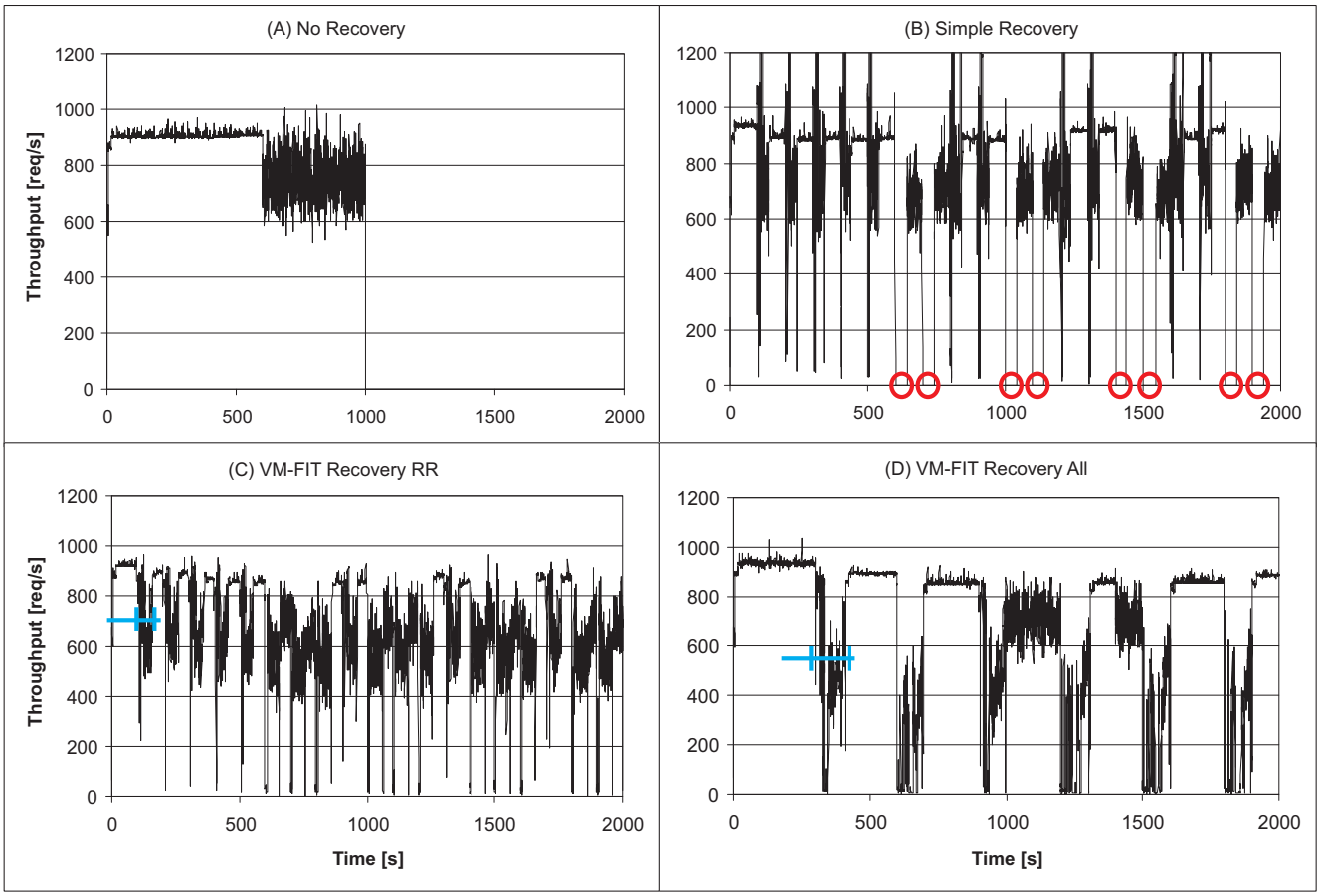


Figure 4. Throughput measurements on a single-CPU machine

tiple 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, similar to the measurements on a single CPU (see markers on X-axis).

In variant (C), the over-all behaviour seems to be better than on the single-CPU version, as 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).

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

Table 2 shows the average system performance in the same intervals as in the previous section. On the multi-CPU machine, 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).

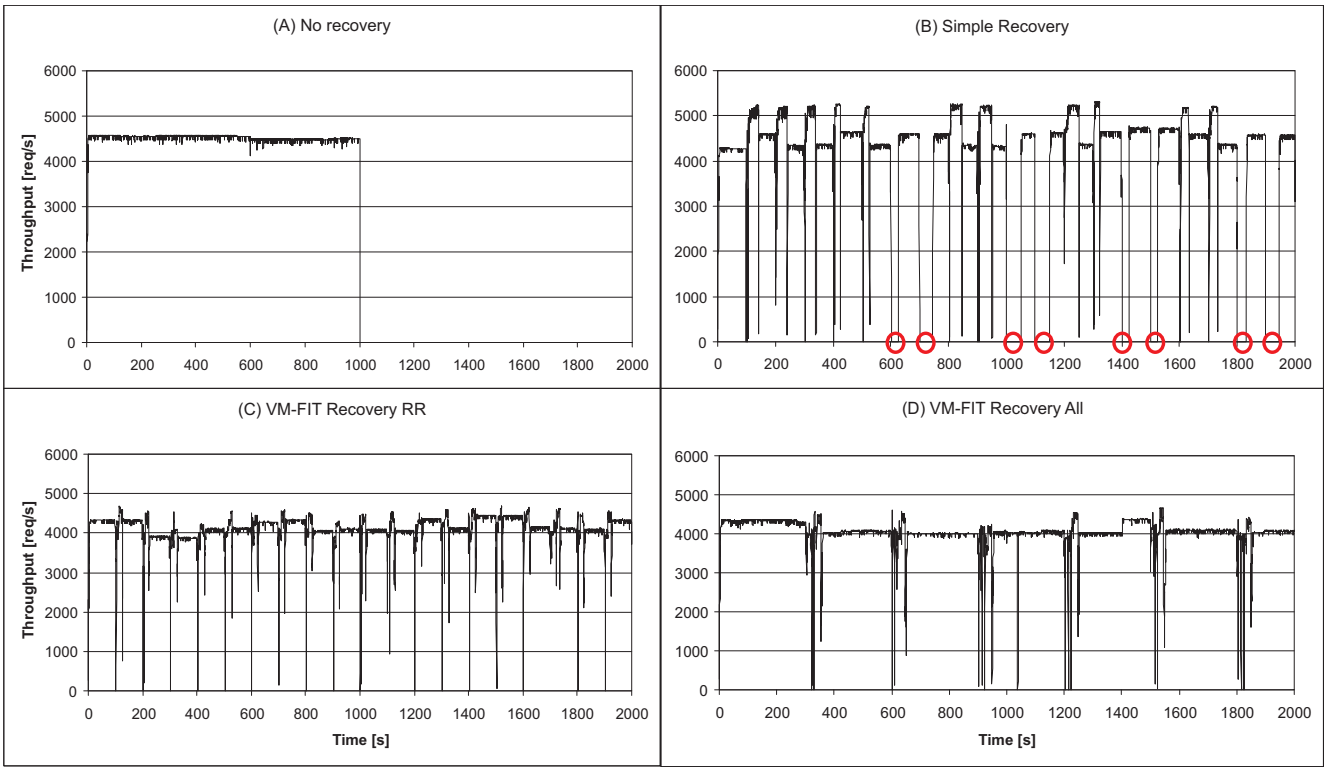


Figure 5. Throughput measurements on multi-CPU machine

5.4 Discussion

The measurements demonstrate that in both usage scenarios, 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.

All in all, it can be observed that the VM-FIT proactive recovery system performs superior on a multi-CPU system. On a single CPU, the parallel creation of a replica in a new virtual machine consumes local resources and thus reduces the throughout of other replicas. With multiple CPUs (with the number of replicas not exceeding the number of CPUs), the only visible degradation is a short (fractions of a second) unavailability at the moment of virtual machine creation and at the transition point between old and new replica instance.

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, 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.

6 Conclusions

In this paper, we have presented a novel approach for efficient proactive recovery in distributed systems. Our VM-FIT prototype uses the Xen hypervisor for providing an isolated trusted component in parallel to the virtual service node. The service node runs service-specific instances of operating system, middleware, and service; these components may fail in arbitrary, Byzantine ways. Our approach avoids the danger of system unavailability during recovery, as the recovery does not reduce the number of simultaneously tolerable faults. Our measurements indicate that periodic proactive recovery has only a modest impact on overall system performance.

In future work, we will further investigate the impact of transfer state size on the efficiency. We expect that recovering all replicas simultaneously will be the superior variant

in case of a large state size. Further experiments will aim at confirming this claim and analyse the break-even point between the two variants.

Acknowledgements

The authors would like to thank Franz J. Hauck and Paulo Sousa, as well as the anonymous reviewers for their valuable comments on improving this paper. This work was supported by the EU through project IST-4-027513-STP (CRUTIAL), by the Large-Scale Informatic Systems Laboratory (LaSIGE), and by the DAAD.

References

[1] A. Avižienis and L. Chen. On the implementation of N-version programming for software fault tolerance during execution. In *Proc. IEEE COMPSAC 77 Conf.*, pages 149–155, 1977.

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

[3] 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.

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

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

[6] 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.

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

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

[9] M. Castro, R. Rodrigues, and B. Liskov. Base: Using abstraction to improve fault tolerance. *ACM Trans. Comput. Syst.*, 21(3):236–269, 2003.

[10] R. Chinchani, S. J. Upadhyaya, and K. A. Kwiat. A tamper-resistant framework for unambiguous detection of attacks in user space using process monitors. In *Proc. of the IEEE Int. Workshop on Information Assurance*, pages 25–36, 2003.

[11] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proc. of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 273–286, Boston, MA, May 2005.

[12] 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.

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

[14] I. Gashi, P. Popov, and L. Strigini. Fault diversity among off-the-shelf sql database servers. In *DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN'04)*, page 389, Washington, DC, USA, 2004. IEEE Computer Society.

[15] 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.

[16] 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.

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

[18] 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.

[19] H. P. Reiser, F. J. Hauck, R. Kapitza, and W. Schröder-Preikschat. Hypervisor-based redundant execution on a single physical host. In *Proc. of the 6th European Dependable Computing Conf., Supplemental Volume - EDCC'06 (Oct 18-20, 2006, Coimbra, Portugal)*, pages 67–68, 2006.

[20] 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.

[21] P. Sousa. Proactive resilience. In *Sixth European Dependable Computing Conference (EDCC-6) Supplemental Volume*, pages 27–32, Oct. 2006.

[22] 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.

[23] 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.

[24] 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.