VM-FIT: Supporting Intrusion Tolerance with Virtualisation Technology

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

Rüdiger Kapitza
Department of Compter Science 4
University of Erlangen-Nürnberg, Germany
kapitza@cs.fau.de

ABSTRACT

The use of virtualisation technology on modern standard PC hardware has become popular in the recent years. This paper presents the VM-FIT architecture, which uses virtualisation for realising fault and intrusion tolerant network-based services. The VM-FIT infrastructure intercepts the client–service interaction at the hypervisor level, below the guest operating system that hosts a service implementation, and distributes requests to a replica group. The hypervisor is fully isolated from the guest operating system and provides a trusted component, which is not affected by malicious intrusions into guest operating system, middleware, or service. Furthermore, the hypervisor allows the implementation of more efficient strategies for proactive recovery in order to cope with the undetectability of malicious intrusions.

Categories and Subject Descriptors

D.4.5 [**Operating Systems**]: Reliability—*Fault-tolerance*; C.4 [**Performance of Systems**]: Fault tolerance

General Terms

Reliability

Keywords

Virtualisation, Byzantine fault tolerance, proactive recovery, innovative system architectures

1. INTRODUCTION

Services in distributed systems have become omnipresent, and the reliability of such services is becoming an increasingly significant issue for a growing class of applications. In the area of fault-tolerant systems, the toleration of malicious intrusions is one of the most difficult challenges. In order to enable the use of intrusion-tolerant concepts in a wide range of application domains, light-weight and efficient concepts are needed.

This paper investigates the use of virtualisation technology for the construction of fault and intrusion tolerant systems. Virtualisation provides a hypervisor component that is fully isolated from the guest operating system that hosts the actual service implementations. We identify two potential gains from virtualisation technology: First, the hypervisor is able to provide an isolated trusted component that does not have all the vulnerabilities of the guest systems hosting the actual service. Second, the hypervisor has full control over the guest systems, and thus can support an efficient proactive recovery of the guest system instances.

The paper is structured as follows. The next section discusses related work. Section 3 describes the core concepts of the VM-FIT (virtual machine – fault and intrusion tolerance) system architecture. Section 4 presents details of our current prototype implementation and discusses future work. Finally, Section 5 concludes.

2. BACKGROUND AND RELATED WORK

Frequently, software-based replication schemes are implemented in middleware systems such as fault-tolerant CORBA [12]. Most systems assume a crash-stop behaviour of nodes and cannot tolerate non-benign faults such as undetected random bit errors in memory, invalid states caused by software faults, or malicious intrusions of an attacker. Handling that kind of faults is the aim of Byzantine fault-tolerant mechanisms, such as the Castro-BFT algorithm [4] and the intrusion-tolerant SINTRA architecture [3]. One problem of intrusion-tolerant systems is that compromised components may remain undetected for extended periods of time. In the course of time, an attacker might obtain control of an increasing number of nodes, finally exceeding the maximum number of faulty nodes that the system is able to tolerate. Proactive recovery [13, 5, 16] is a technique that periodically refreshes nodes in order to remove potential intrusions; as a side effect, the refresh operation may also deploy new software versions that eliminate known vulnerabilities.

Virtualisation is an old technology that was introduced by IBM in the 1960s [8]. Systems such as Xen [1] and VMware [17] made this technology popular on standard PC hardware. Virtualisation 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, virtualisation can also be used for constructing fault-tolerant systems. Bressoud and Schneider [2] demonstrated the use of virtualisation for lock-stepped replication

of an application on multiple hosts. Besides such direct replication support, virtualisation 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 [11]. Furthermore, the separation simplifies formal verification of components [18]. Using virtualisation is also popular for intrusion detection and analysis. Several systems transparently inspect a guest operating system from the hypervisor level [6, 7].

The RESH architecture [15] proposes redundant execution of a service on a single physical host using virtualisation. This approach allows the toleration of non-benign random faults such as undetected bit errors in memory, as well as N-version programming in order to tolerate software faults.

The contributions of this paper towards virtualisation-based intrusion tolerance differ from previously published approaches. Unlike strictly-coupled replication systems, we aim at replicating network-based services across heterogeneous nodes, using asynchronous communication networks. We use virtualisation to supply all nodes with a trusted entity that is fully isolated from the potentially faulty guest domain, which hosts a service together with an operating system and a middleware environment. Active replication of a service is complemented with support for proactive recovery, in order to eliminate potentially undetected intrusions into the guest domains. This way, we obtain mechanisms for proactive recovery that are more efficient than previous solutions.

3. VM-FIT ARCHITECTURE

The VM-FIT architecture provides generic support for the replication of network-based services. It transparently intercepts remote communication at the hypervisor level, and provides support for proactive recovery. We assume that the following properties hold:

- Clients use a request–reply interaction to access a remote network-based service.
- The service has deterministic behaviour, i.e., the service state and the replies sent to clients are uniquely defined by the initial state and the sequence of incoming requests.
- Byzantine faults may occur in a limited number of service replicas.

The first assumption allows the interception of client–service interaction at the network level. Together with the second assumption, the usual model for deterministic state machine replication is used. The failure model of the third assumption will be specified in more detail later on.

In the following, we adopt a terminology that is inspired by the Xen hypervisor [1]. 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. In the following, we introduce the terminology of a *Domain NV* (network/voting), which handles the communication with clients and among replicas, and the voting over replica replies. The Domain NV is fully isolated from all guest domains. It may or may not be integrated into the usual Domain 0.

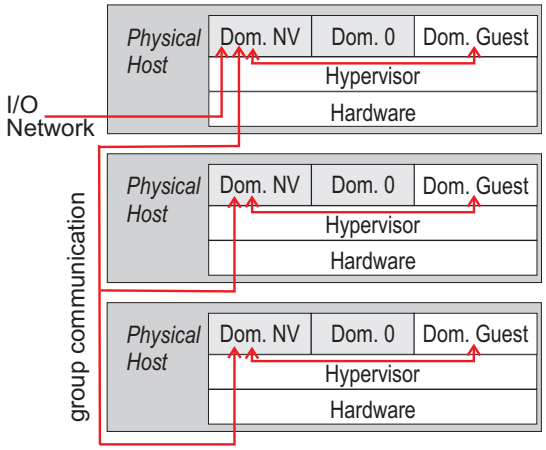

Figure 1: VM-FIT basic replication architecture

3.1 Replication Support

The basic system architecture is shown in Figure 1. The service replicas are running in isolated virtual machines in Domain Guest. The network interaction from client to the service is handled by the replication manager in the separate Domain NV. This manager intercepts the client connection and distributes all requests to the replica group using a group communication system. 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 first benefit from this architecture is 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 second benefit from this architecture is the support for a *composite fault model*. The architecture is composed of multiple isolated elements (Domain NV, Domain 0, Domain Guest), and each element may be subject to failures. For VM-FIT, we consider two possible variants:

- All system elements are potentially affected by Byzantine faults.
- The Domain NV (which contains the basic replication logic and network functionality, including network driver, communication stack, and group communication protocol) fails only by crash-stop of the complete host. The Domain Guest uses a Byzantine fault model, and thus can be subject to malicious intrusion. The same Byzantine model can be applied to the Domain 0, as long as the system makes sure that Domain NV is an protected, isolated entity that cannot be influenced from Domain 0.

The first variant is more powerful in terms of tolerable faults. It is able to tolerate intrusions even in the Domain NV. The drawback of this approach is that it requires more complex, Byzantine fault tolerant group communication protocols, compared to the second variant.

The second variant can be justified if the guest domain is a complex system with a legacy OS and a full-blown middleware, whereas the Domain NV executes in a fully isolated, lean domain. In the extreme case, the Domain NV could be an entity that can be subjected to full formal verification. In this case, simple, crash-stop group communication protocols can be used for distributing client requests. Thus, implementing consistency mechanisms in the privileged domain is expected to allow more efficient protocols than in systems that aim at tolerating Byzantine faults in all components of the system. Because of this aspect, we expect that our approach is best used in situations in which the first variant can be used. As a result, we obtain a hybrid system architecture that is able to use simple protocols for replication control, while still being able to tolerate malicious intrusions in the Domain Guest of a limited number of nodes.

In the case of a crash-stop Domain NV, the architecture can use majority voting for verifying client replies. In this case, the number of replicas must be $n \geq 2f + 1$ in order to tolerate up to f Byzantine faulty replica instances. In Section 4.4, we consider the execution of the group communication protocol in the non-crash-stop domains. In this case, Byzantine fault tolerant protocols are needed, which require $n \geq 3f + 1$ nodes.

3.2 Proactive Recovery

Proactive recovery increases the resilience of the replicated service, as the faulty nodes become rejuvenated periodically, and thus the upper bound of f tolerable faults is no longer required for the whole system lifetime, but only for the duration of a rejuvenation cycle. Proactive recovery requires a system component that controls the re-initialisation of the local service instance (including the operating system and middleware). For example, a tamper-proof external hardware might be used for rebooting the node from a secure code image. It is not feasible to trigger the recovery within a service replica, as a malicious intrusion might cause the replica component to ignore the required recovery.

In the VM-FIT architecture, the Domain NV can be used as a trusted entity that is able to completely re-initialise the target domain that hosts the service. For this purpose, all elements (i.e., operating systems, middleware, and service instance) need to be initialised with a "clean" state, by securely obtaining the service state from other replicas. As discussed by Sousa et al. [16], 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 virtualisation in order to minimise this unavailability.

Unlike other approaches to proactive recovery, 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. This way, the downtime of the service replica is minimised to the time necessary for the coordinated transition with a consistent state.

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. For this purpose, 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. Only if the local state is invalid, a remote state transfer becomes necessary. We discuss this state-transfer issue in more detail in the Section 4.3.

In summary, virtualisation allows a secure proactive recovery of service instances without additional hardware support. At the same time, it minimises downtime during recovery by creating a new replica instance in parallel to the running instance, and it enhances the state-transfer efficiency by transferring local state.

3.3 Application Areas

The proposed VM-FIT architecture can be applied to various kinds of applications, ranging from simple web applications to critical infrastructure. Applications using VM-FIT, however, have to meet the constraints defined at the start of this section.

The implementation of *new intrusion-tolerant applications* using VM-FIT is the easier variant. In this case, the developer is aware of the constraints of the service replication. Specifically, the application can make the guarantee of being deterministic, can provide interfaces for state transfer, and strictly adhere to a state machine principle.

It might, however, also be desirable to apply the VM-FIT architecture to *existing services* without or with only minimal modifications. This approach requires several steps. First, it must be verified if the application behaviour is strictly deterministic. One source of nondeterminism that is found in many applications is the support for multithreading. Handling multiple client requests with independent threads easily violates the state-machine principle by applying state changes to replicas in an inconsistent order. A further problem is the use of local address data in requests and replies. Each replica uses a different local communication address that is only known to the replication controllers in Domain NV, and in all communication with the external clients, the "outer" address of Domain NV has to be used. Because of these problems, we anticipate that the VM-FIT architecture cannot be used for transparently making existing applications intrusion tolerant. This does not mean that the architecture is useless for such existing services. As an example, in the next section we illustrate how a prototype implementation of VM-FIT can be used to replicate CORBA-based services.

4. PROTOTYPE IMPLEMENTATION

We have implemented a simple prototype of VM-FIT, which enables the transparent replication of a CORBA-based service. In the prototype, the Xen 3.0 hypervisor was used with Linux as operating system for Domain 0 and for Domain Guest. This virtualisation software has reached a high level of maturity, is available as open source, and supports a wide range of guest operating systems. This broad support is essential for the envisioned goal of providing a generic interception and replication architecture that is independent from specific operating systems.

4.1 Implementation Details

In the first 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, and Domain NV is integrated into Domain 0. 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. For future work, we envision two potential options. The first variant uses Xen as basic hypervisor, but—instead of Linux—uses a minimalist operating system in Domain NV. The second variant is based on the L4ka microkernel [10]. This microkernel offers virtualisation functionality. In addition, significant efforts towards a formal verification of the L4 kernel have been made by other researchers [18]. These results provide an excellent basis for a trusted entity.

A core task of the Domain NV is the provision of mechanisms for the instantiation and initialisation of service replicas. A simple description language enables the developer to describe how the privileged domain has to instantiate a local replica in a new guest virtual machine. Currently, a disk image of a Xen virtual machine with a preconfigured operating system and middleware environment is provided.

After initialisation, as well as after recovery, a state transfer from the set of replicas is required. The actual transfer is described below; a prerequisite for the transfer is the serialisation of the replica state into a byte stream. We use an application-controlled serialisation mechanism for this purpose. This means that the Domain NV can request the service in the Domain Guest to serialise its state.

4.2 Consistency Management

The replication of a CORBA services requires custom mechanisms within the Domain NV. The first reason for this is the addressing mechanisms of CORBA. In CORBA, the remote address of a service is specified as an *interoperable object reference (IOR)*. Internally, the IOR contains the TCP/IP address of the object request broker (ORB). In a VM-FIT environment, each replica will create its own IOR, using the local network address. These addresses are private IP addresses only used for communication between the virtual machines of a single physical host, and cannot be used by clients. Therefore, the Domain NV of our prototype is able to construct a new service IOR for the replicated service, and publishes this address to a public naming service.

The envisioned hypervisor-based replication architecture can be applied to various kinds of network-based distributed services. For validating the architecture, it is assumed that the service is implemented as a deterministic CORBA object. Thus, the initialisation of a service object includes the three levels operating system, middleware, and replica implementation. Using CORBA objects allow a comparison between replication support at the middleware level and at the hypervisor level.

As a basis for realising group communication on Domain NV, we currently use the AGC group communication system [14], which internally uses the Paxos algorithm for message ordering.

4.3 State Transfer and Proactive Recovery

For proactive recovery, every recovery operation requires a consistent checkpoint of a majority of replicas. This checkpoint has to be transferred to the recovering replica and verified by the majority of nodes (e.g., by checksums). Finally, the recovering replica has to be reinitialised by the provided state. In our prototype, we assume that a secure code basis for the replica is available locally, and only the data state is required to initialise the replica.

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 at a time by doing all three steps necessary for proactive recovery in parallel. If service replicas have to be recovered, every node receives a checkpoint message and determines the replica state. The Domain NV 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. 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 initialisation 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 initialised shadow replica that is a member of the replication group. In a final step, the old replicas can be safely shutdown 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.

4.4 Future Work

The current prototype enables an early evaluation of the core VM-FIT architecture. In future work, we plan to provide a detailed qualitative and quantitative evaluation of the system properties. For an experimental evaluation of the performance of the system, fault injection allows to simulate the effect of non-benign faults in the system to some extent. We plan to use [9] as a platform for executing such experiments. The results will not only provide an assessment of VM-FIT, but also will also permit a comparison with other replication infrastructures, such as a fault-tolerant CORBA implementation.

Another issue of future work is the applicability of VM-FIT to other existing distributed services. The CORBA replication prototype has shown that, for existing services,

dedicated support for these infrastructures has to be integrated into the Domain NV. We will discuss what extensions will be required for other, non-CORBA applications.

An even more important issue is the trust into the Domain NV. Currently, no real mechanisms are used to substantiate this trust. In the future, it is essential to replace the complex domain NV (replication controller application on top of a full Linux instance) with a leaner alternative. In the extreme case, a complete formal verification of Domain NV might be desirable.

Our prototype current implements group communication in the privileged Domain NV. In order to minimise the amount of code in the privileged domain, the advantages and disadvantages of handling group communication here or at the application layer need to be discussed. The existing AGC architecture could be used for a hybrid approach, which provides some basic services (such as distributed consensus) in the privileged part, which acts as a trusted base, and the remaining part of the group communication protocol is handled in the guest operating systems. Such a hybrid architecture is highly promising and feasible to implement within a short time on the basis of the existing system.

5. CONCLUSIONS

In this paper, we have investigated the use of virtualisation technology for the construction of fault and intrusion tolerant systems. The VM-FIT architecture provides basic support for transparent intrusion-tolerant replication of services and for proactive recovery of replicas. In this architecture, virtualisation is used to provide a trusted component on each machine. This enables the use of more efficient protocols. Furthermore, the hypervisor can be used for supporting proactive recovery of service instances. Our current prototype enables a detailed investigation of some core issues of the VM-FIT architecture. Future work will provide a more detailed quantitative study of dependability and efficiency.

6. ACKNOWLEDGEMENTS

This work has been supported by the DAAD.

7. REFERENCES

[1] 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.

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

[8] 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.

[9] H. Höxer, M. Waitz, and V. Sieh. Advanced virtualization techniques for FAUmachine. In R. Spenneberg, editor, *11th International Linux System Technology Conference, Erlangen, Germany, September 7-10, 2004*, pages 1–12, 2004.

[10] 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.

[11] 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.

[12] O. M. G. (OMG). Common object request broker architecture: Core specification, version 3.0.2. OMG document formal/02-12-02, 2002.

[13] 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.

[14] H. P. Reiser, U. Bartlang, and F. J. Hauck. A reconfigurable system architecture for consensus-based group communication. In *Proc. of the 17th IASTED Int. Conf. on Parallel and Distributed Computing and Systems*, pages 680–686. IASTED, 2005.

[15] H. P. Reiser, F. J. Hauck, R. Kapitza, and W. Schrder-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.

[16] 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.

[17] 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.

[18] 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.