

Hypervisor-Based Redundant Execution on a Single Physical Host

Hans P. Reiser, Franz J. Hauck
Distributed Systems Lab
Faculty of Computer Science
University of Ulm, Germany
Email: {reiserh,fjh}@acm.org

Rüdiger Kapitza, Wolfgang Schröder-Preikschat
Department of Distributed Systems
and Operating Systems
University of Erlangen-Nürnberg, Germany
Email: {rrkapitz,wosch}@cs.fau.de

1 Introduction

Fault tolerance is becoming an increasingly important requirement in distributed systems. Many existing fault-tolerant infrastructures assume a fail-stop behaviour, which, however, is too idealistic to be always valid in practice. While techniques such as checksums and error correcting codes on storage entities can compensate or detect some hardware faults, they cannot fully eliminate all failures. Even more severe, faulty software can cause arbitrarily wrong behaviour. Current software usually has a complexity that makes it hardly possible to ensure it being fault-free.

Redundant provision of some functionality on multiple hosts is an established approach to build reliable applications. For detecting random value faults of the hardware on individual hosts, redundant execution of the same software, combined with output voting, is sufficient. To reduce the probability of software faults, N-version programming [1] is an established approach, in which each replica uses a different realisation of the same functionality, implemented by independent developers. However, using such redundancy is expensive, as it requires multiple hosts that provide the same functionality and an additional voter component, which often is implemented by dedicated hardware.

The use of virtualisation technology on modern standard hardware has again become popular in the recent years. Systems such as User Mode Linux [4] provide virtual execution environments on top of a base operating system. Systems such as Xen [2] and VMware ESX Server [7] create a small virtualisation layer below any operating system.

In this paper, we propose the use of virtualisation technology to provide local redundant execution and to support N-version programming on a single host. Our solution is efficient, as all interaction between replicas and a voter component takes place within a single host, and it is inexpensive in terms of hardware cost, as only a single physical machine is used. The fault-tolerance properties are similar to that of a traditional multi-version system.

Our architecture reduces the performance of CPU-

intensive applications, as the available CPU resources are divided between multiple virtual hosts. However, the increasing popularity of multi-CPU and multi-core CPU hardware provides an ideal basis for executing multiple virtual machines on a single host.

2 The RESH Architecture

Our RESH architecture (*Redundant Execution on Single Host*) provides a hypervisor-based infrastructure for the redundant execution of a networked service on a single physical host. RESH assumes a hypervisor that provides virtual machines on top of a small hypervisor layer that runs directly on the hardware; our prototype implementation will be based on Xen [2]. RESH assumes that the hypervisor is a trusted entity that does not fail. Having a minimal hypervisor kernel, without a full-featured operating system beneath, makes this a realistic assumption. A minimal hypervisor simplifies verification and test coverage, minimising the chance of failures in the hypervisor.

In Xen, multiple operating systems run in virtual machines above the hypervisor and may have direct access to the hardware. Usually, a single virtual machine, designated as *Domain 0*, provides real hardware drivers, and the remaining set of guest operating systems use virtual device drivers that communicate with *Domain 0*. For RESH, we split the *Domain 0* into two isolated virtual machines (Fig. 1). The *Domain NV* (network/voting) provides low-level network drivers, a network protocol stack, and the voting component. The remaining *Domain 0* provides all other privileged tasks, such as other hardware drivers, monitoring, and management of the guest virtual machines. This approach minimises the complexity of the *Domain NV*, which provides the voter, and fully isolates it from the other virtual machines.

We assume a system model in which the application interacts with the environment exclusively by input/output via a network. RESH uses interception at the level of TCP network sockets, but could easily be extended to support other

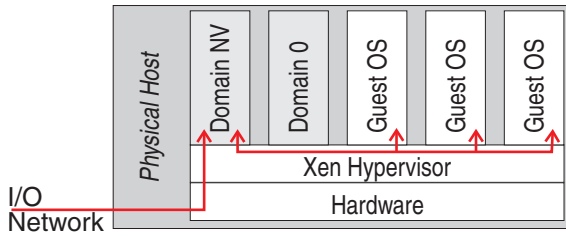


Figure 1. RESH System Architecture

network protocols. All input first arrives at *Domain NV* and then is transparently distributed to all local replicas. The voter in *Domain NV* collects the output of all replicas and forwards the result to the output.

The RESH architecture can be used for N-version programming. Multiple operating systems, middleware infrastructures, and service implementations can be deployed on the virtual machines of the guest systems. This enables the toleration of software implementation faults. In a simpler setting, the same architecture can be used to execute the same service in multiple instances. In this case, a systematic software fault can cause all local replicas to fail; that is, this approach provides weaker fault tolerance. It still has its virtues, as it enables the toleration of random faults that can cause one of the replicas to produce erroneous values.

Obviously, RESH is not able to tolerate the complete crash of the host. For this purpose, RESH can be combined with replication across multiple physical hosts. In such a setting, the benefit from RESH is that the physical host exhibits fail-stop behaviour on the external interface even if value faults caused by hardware or faulty software occur in one of the local replicas.

3 Related Work

To the best of our knowledge, no previous work has considered using virtualisation for local redundant execution on a single host. Several previous works, however, discuss other approaches for using virtualisation to increase dependability. Bressoud et al. [3] propose an architecture that offers step-lock execution of a primary and a backup host controlled by a hypervisor. This architecture addresses replication across multiple physical hosts, whereas our approach enables redundant execution on a single host.

Recent research on dependability and virtualisation focuses mainly on the areas of resource management, encapsulation, and intrusion detection. Hypervisor-based management can be used to simplify the deployment and migration of replicas, which minimises maintenance downtime. The encapsulation of individual system components such as device drivers in separate virtual machines can be used to increase system stability [6]. Furthermore, virtualisation enables intrusion detection by monitoring execu-

tion patterns of a guest operating system from an intrusion detection component within a separate virtual machine [5]. These approaches also benefit from the isolation property of virtual machines, but, besides that, have completely different objectives than RESH has.

4 Conclusions and Future Work

We have presented the RESH architecture, which uses virtualisation for the redundant execution of a network-based service on a single host. This architecture is able to tolerate non-fail-stop hardware failures using a trusted voting component encapsulated within a virtual host. The approach can be combined with N-version programming, which enables the toleration of software faults. The result is an inexpensive and efficient system, as only a single standard host is required and as all interactions between the redundant instances happen locally.

The design of the RESH architecture will be implemented in a Xen-based prototype, which we will use for a detailed efficiency evaluation. In addition, fault-tolerance properties will be analysed using fault injection. Beyond the core RESH architecture, we will focus on the combination of local redundant execution on a single host with replication on multiple physical hosts.

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] 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: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM Press.
- [3] T. C. Bressoud and F. B. Schneider. Hypervisor-based fault tolerance. *ACM Trans. Comput. Syst.*, 14(1):80–107, 1996.
- [4] J. Dike. *User Mode Linux*. Bruce Perens Open Source series. Prentice-Hall, 2006.
- [5] T. Garfinkel and M. Rosenblum. A virtual machine introspection based architecture for intrusion detection. In *Proc. Network and Distributed Systems Security Symposium*, February 2003.
- [6] J. LeVasseur, V. Uhlig, J. Stoess, and S. Götz. Unmodified device driver reuse and improved system dependability via virtual machines. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*, San Francisco, CA, Dec. 2004.
- [7] C. A. Waldspurger. Memory resource management in VMware ESX server. *SIGOPS Oper. Syst. Rev.*, 36(SI):181–194, 2002.