Practical Intrusion-tolerance in the Cloud

Rüdiger Kapitza, Tobias Distler
University of Erlangen-Nürnberg

Hans P. Reiser
Universidade de Lisboa

Abstract

Byzantine fault tolerant (BFT) replication is commonly associated with the overhead of $3f + 1$ replicas to handle f faults. We believe this large resource demand is one of the key reasons why BFT replication is not commonly applied. We present Spare, an approach that harnesses virtualization support as typically found in cloud-computing environments to reduce the resource demand of BFT replication. This is achieved by restricting replication and request execution to only $f+1$ nodes in the fault-free case, while rapidly activating up to f replicas using virtualization in case of replicas being faulty or slow. To maxize system availability, we keep spare replicas that are periodically updated in a suspended state. In the fault-free case, these passive replicas assist a resource-efficient proactive recovery.

1 Introduction

Despite numerous improvements in developing and managing software, bugs and security holes exist in today's products, and malicious intrusions happen frequently. Byzantine fault tolerant (BFT) replication is a key technology for enabling services to tolerate intrusions. However, traditional BFT replication involves high overhead, as it requires $3f + 1$ replicas to tolerate f faults [2]. Assuming that each replica has an intrusion-free trusted subsystem at its disposal, resource requirements can be reduced to as few as $2+1$ replicas [3,4]. This is still too much in many scenarios and therefore the use of BFT replication is often viewed as applicable for highly critical infrastructures and services. However, all kinds of web-based services so far considered of minor importance are more and more taking an increasing role in our everyday life. Typically, these services are confronted with much stronger economical constraints than mission-critical infrastructures. As a consequence, on the one hand, resource-intensive techniques such as BFT replication are not used in practice. On the other, this has serious effects on software quality and provision management, which turns these services into easy targets for intrusions.

Our general goal is to reduce the resource costs of BFT replication to make it widely applicable, e.g., for web-based services. There is an ongoing trend to run these services on top of some kind of virtualization platform that enables basic management of virtual machines. Such environments can be found in cloud-computing infrastructures, with Amazon EC2 [1] as one of its protagonists. We assume that the base virtualization infrastructure is secure. This assumption is backed by numerous efforts to make hypervisors small and trustworthy. However, software executed inside of virtual machines can fail maliciously.

Based on these assumptions, we present *Spare*, a system which utilizes only $f + 1$ *active replicas* during fault-free execution. This number is the minimum amount of replicas that allows the detection of up to f replica faults. If our system suspects faulty behavior of some of the replicas, it needs to activate up to f additional *passive replicas*. Technically, a passive replica is provided by suspending a virtual machine that has been initialized from a secure code and service state. An active replica is suspected to be faulty if it either produces replies that are inconsistent with other replicas, or if its response time exceeds a threshold. The latter is needed as attackers might delay responses to prevent further execution. The virtualization support of today's cloud-computing infrastructures offers an ideal basis for dynamically activating additional replicas, as it provides means for rapid activation and deactivation of virtual machines. Introducing passive replicas saves resources, as typically providing and applying state updates is less resource demanding than request execution. Albeit this already can result in substantial resource savings, passive replicas builds another great benefit: Replica managers ensure the correctness of state updates by agreement. This way, passive replicas have a correct state that can be used as the basis for resource-efficient proactive recovery (PR).

2 Architecture of Spare

Our approach is based on a virtual-machine monitor (VMM) that enforces isolation between different virtual machines on one physical host. In Spare, a replica manager is running within the privileged *Dom0*, while service replicas are being executed in completely separated application domains (*DomU* with guest operating system, middleware infrastructure, and service implementation) (see Figure 1). The replica manager is composed of basic support for receiving client requests, communication support for consistently distributing requests to all replicas, and a proactive

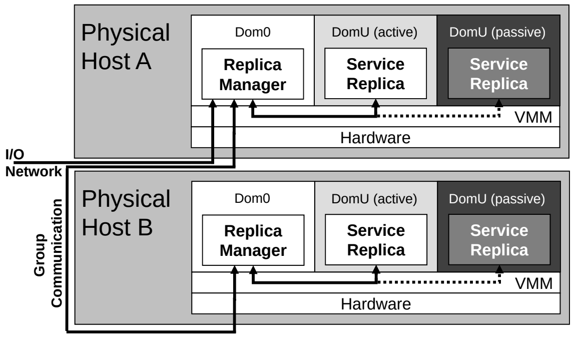

Figure 1. Minimal Spare architecture

recovery logic. In addition, the replica manager includes a custom voting component that handles the on-demand activation of passive replicas, mechanisms for handling state updates, and support for replica cloning. In combination with hypervisor and Dom0, the replica manager forms the trusted computing base (TCB) of Spare. Service replicas placed in isolated application domains are not trusted. This *hybrid fault model* allows coping with any kind of failure in application domains, including random non-crash faults and intentional malicious faults.

Passive Replicas Common BFT replication approaches execute all requests on every replica. In order to reduce resource consumption, we add the notion of *passive* replicas to BFT replication. In the context of the crash-stop failure model, this is a well-known concept to reduce replication overhead. If a master node crashes, a secondary passive replica takes over. For BFT replication, we cannot rely on a single master. Instead, we need at least $f+1$ active replicas to detect faults. All remaining replicas are put in passive mode. Figure 1 outlines a minimal setting comprising two physical machines each hosting one active and one passive replica. This configuration can tolerate one faulty replica. In the general case, Spare only needs $2f+2$ replicas to tolerate f malicious faults within application domains and to offer proactive recovery.

Activation If at least one of the $f+1$ active replicas provides a different result (or none at all), additional (i.e. passive) replicas have to execute the pending request. However, passive replicas first have to be activated. To make such an approach practical, the infrastructure must allow rapid activation of passive replicas, as the service is unavailable for the clients as long as it takes to decide on the outcome of pending requests. In Spare, virtual machines hosting passive replicas are put into suspended mode. This way, they only allocate memory and disk but no other resources, such as CPU or bandwidth. On the activation of a passive replica, the replica manager wakes up the corresponding virtual machine, which only takes hundreds of milliseconds.

State Updates In order to process a pending request after being activated, passive replicas must have the latest application state available. Spare uses regular state updates to bring passive replicas up to date. In particular, after executing a modifying request, replica managers retrieve state updates, representing the state changes caused by the request, from all active replicas. Next, replica managers agree on state updates and each stores the verified version in a local log. When the log size reaches a certain limit, replica managers temporarily activate local passive replicas to execute the accumulated state updates. Having updated and re-suspended passive replicas, replica managers truncate their logs. Note that these periodic updates of passive replicas reduce the overhead for updating on the occurrence of faults. When a passive replica is activated to tolerate a fault, only state updates since the last periodic update have to be executed to prepare the replica.

Supporting Proactive Recovery With faults accumulating over time, the number of faults may eventually exceed the fault-tolerance threshold of a BFT system. Therefore, we consider proactive recovery an important technique to provide long-running services. PR periodically initializes replicas with correct code and a correct application state that all non-faulty replicas agree on. This way, a replica is cleaned from corruptions and intrusions. As a consequence, PR basically allows to tolerate an unlimited number of faults as long as at most f faults occur during a single recovery period. In Spare, passive replicas are used as the basis for proactive recovery as they already have the latest correct application state available. The replica manager hands over request execution to the backup (i.e. former passive) replica, and shuts down the old active replica. This way, a potentially faulty replica is efficiently replaced with a clean one.

3 Conclusions

Spare is a system that enables BFT replication with PR at minimal resource cost. This is achieved by restricting request execution to $f+1$ replicas under graceful conditions and rapidly activating passive replicas on demand.

References

[1] Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2, 2009.

[2] M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *Proc. of the third Symp. on Operating Systems Design and Implementation*, pages 173–186, 1999.

[3] B.-G. Chun, P. Maniatis, S. Shenker, and J. Kubiatowicz. Attested append-only memory: making adversaries stick to their word. In *Proc. of twenty-first ACM SIGOPS Symp. on Operating Systems Principles*, pages 189–204, 2007.

[4] M. Correia, N. F. Neves, and P. Verissimo. How to tolerate half less one byzantine nodes in practical distributed systems. In *Proc. of the 23rd IEEE Int. Symp. on Reliable Distributed Systems*, pages 174–183, 2004.