REPVM

Paper #10

Abstract

By allowing multiple servers to be consolidated on a smaller number of physical hosts, virtualization is widely used in computing environments of various kinds and scales. Nevertheless, previous high availability solutions in virtualized environments are still unable to guarantee service-continuity in case of application failure because they do not target the availability at the application level, but rather at the VM level.

This paper presents REPVM, an SMR system for KVM based virtualized machines, which efficiently replicates programs running in the virtual machines. REPVM achieves distributed consensus at the networking level. Evaluation on four widely used server programs (e.g., `MySQL` and Redis) shows that REPVM is easy to use and has low overhead.

1 Introduction

Server virtualization has emerged as a powerful technique for consolidating servers in data centers. The reduction of the number of physical hosts also contributes to cutting back the power consumptions in the data centers.

At the same time, the dependability issues such as service availability become a major concern in consolidated server systems using virtualization. There has been a tremendous progress in achieving high availability in virtualized environments, but existing approaches still fail to meet the high availability requirements of applications running in the VM because they are oriented toward protecting the VM. Protecting the VM alone does not guarantee uptime for applications and services. Detecting and remediating VM failure falls short of what is truly vital, detecting and remediating application and service failures.

State Machine Replication (SMR) is an attractive approach to providing high availability at the application level. SMR runs replicas of the program and uses a distributed consensus protocol (typically PAXOS) to ensure the same sequence of input requests for replicas, as long as a quorum (typically a majority) of the replicas agrees on the input request sequence.

However, PAXOS is slow because each decision takes at least three message delays between when a replica proposes a command and when some replica learns which command has been chosen.

Fortunately, Remote Direct Memory Access (RDMA)-capable networks have dropped in price and made substantial inroads into datacenters. RDMA operations allow a machine to read (or write) from a pre-registered memory region of another machine without involving the CPU on the remote side. Compared to traditional message passing, RDMA achieves the smallest round-trip latency ($\sim$3 μs), highest throughput, and lowest (zero) CPU overhead [5].

A naive approach for realizing highly available services is to integrate the PAXOS algorithm into every application. But PAXOS is notoriously difficult to understand [6] and implement [4].

To address nondeterminism, SMR systems either reply on deterministic multithreading and replay approach [1] or manually annotate all shared states to detect divergence of execution states [2]. These approaches suffer from poor automation or high overhead.

This paper presents REPVM, an SMR system that efficiently replicates programs running in the VM for high availability. Within each replica, REPVM interposes on the network packets to keep replicas in sync. Specifically, it considers each incoming network packet an input request, and runs a PAXOS consensus protocol [4] to ensure that a quorum of the replicas sees the same exact sequence of the incoming network packets. To deal with nondeterminism, it maintains a packet queue that captures outgoing packets and invokes the output checking protocol periodically.

The remainder of the paper is organized as follows. §2 provides details on the implementation.

2 Implementation

This section discusses technical issues regarding our implementation.

2.1 KVM

We chose the Linux Kernel Virtual Machine (KVM) [3] as the platform of this study.

Fig 1 shows the typical KVM architecture, with reference to a network related application. As depicted in the

Submitted to APSys '16, August 4-5, 2016, Hong Kong, China.

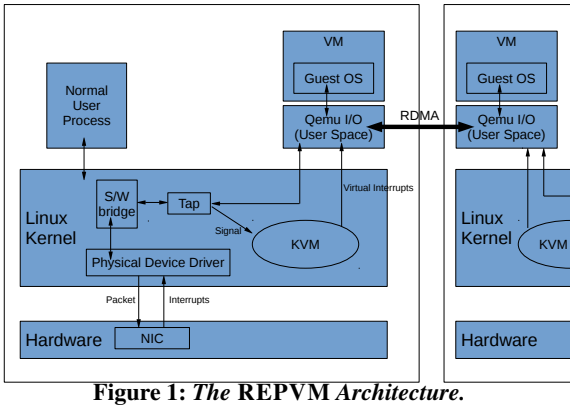

Figure 1: *The* REPVM *Architecture.*

picture, when a packet arrives at physical NIC, interrupts generated by NIC are handled by the physical device driver. The device driver forwards the packet to software bridge. The bridge, then pushes the packet to the tap device of the corresponding VM. The tap device is a virtual network device that sends a signal to KVM module. KVM module in turn, generates a virtual interrupt to the user space Qemu of the target VM. Qemu then copies the packet from tap device and generates the interrupt for the guest OS emulating the virtual NIC. Again, the physical device driver in the guest OS handles the packet transfer to the VMs address space. A major advantage of the KVM architecture is the full availability of user-space tools in the QEMU process, such as threading, libraries and so on.

2.2 Replication Logic

The replication logic is entirely implemented in qemu-kvm, a KVM tailored version of QEMU. We maintain a packet queue to capture outgoing packets.

3 Conclusion

We have presented the design of REPVM.

References

[1] Z. Guo, C. Hong, M. Yang, D. Zhou, L. Zhou, and L. Zhuang. Rex: Replication at the speed of multi-core. In *Proceedings of the 2014 ACM European Conference on Computer Systems (EUROSYS '14)*, page 11. ACM, 2014.

[2] M. Kapritsos, Y. Wang, V. Quema, A. Clement, L. Alvisi, M. Dahlin, et al. All about eve: Execute-verify replication for multi-core servers. In *Proceedings of the Tenth Symposium on Operating Systems Design and Implementation (OSDI '12)*, volume 12, pages 237–250, 2012.

[3] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. kvm: the linux virtual machine monitor. In *Proceedings of the Linux symposium*, volume 1, pages 225–230, 2007.

[4] D. Mazieres. Paxos made practical. Technical report, Technical report, 2007. http://www. scs. stanford. edu/dm/home/papers, 2007.

[5] C. Mitchell, Y. Geng, and J. Li. Using one-sided rdma reads to build a fast, cpu-efficient key-value store. In *Proceedings of the USENIX Annual Technical Conference (USENIX '14)*, June 2013.

[6] D. Ongaro and J. Ousterhout. In search of an understandable consensus algorithm. In *Proceedings of the USENIX Annual Technical Conference (USENIX '14)*, June 2014.