Achieving High Availability at the Application Level in the KVM Virtualization Environment

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.

In this paper we present an high-availability (HA) solution for the applications running in the virtual machines. In our solution, the key to achieving high availability at the application level efficiently is to leverage an State Machine Replication system in the network layer which replicates programs using RDMA. Evaluation on four widely used server programs (e.g., `MySQL` and Redis) shows that this solution 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.

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

State Machine Replication (SMR) is an attractive approach to achieving 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 [3].

A naive approach for realizing highly available services is to integrate the PAXOS algorithm into every application. But PAXOS is notoriously difficult to understand [4] and implement [2].

In this paper we present an efficient HA solution for the applications running in the virtual machines. Our solution keeps replicas in sync by leveraging the SMR system FALCON in the network layer. Each incoming network packet is considered an input request, and the RDMA-based PAXOS protocol provided by FALCON is invoked to ensure that a quorum of the replicas sees the same exact sequence of the incoming network packets. In the event of a hardware failure or application failure, another backup replica takes over providing the service.

The remainder of the paper is organized as follows. In §2 we present the background of our work. We conclude in §8.

2 Background

2.1 KVM

KVM is a more recent hypervisor which embeds virtualization capabilities in Linux kernel using x86 hardware virtualization extensions [1]. It is a full virtualization solution, where guests are run unmodified in VMs. It consists of two modules, namely, kvm.ko module and an ar-

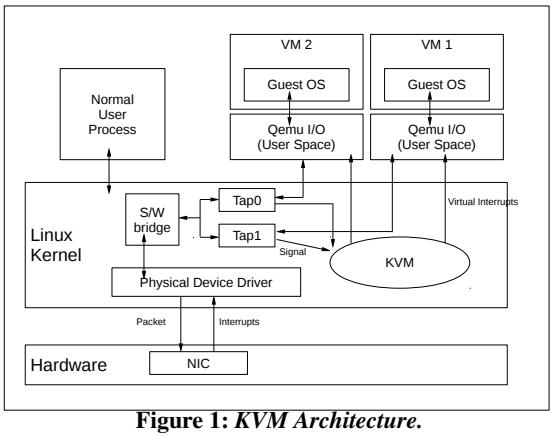

Figure 1: ***KVM Architecture.***

chitecture dependent kvm-amd.ko or kvm-intel.ko module. Under KVM, each VM is spawned as a regular linux process named KVM and scheduled by the default linux scheduler.

For using shared I/O hardware, these VMs interact with Qemu emulator in host user space which provides emulated I/O devices for virtual machines. For instance, in the case of network related applications, Qemu provides emulated Network Interface Card (NIC) to VMs and interacts with tun-tap device on the other side. The tap device is connected to physical NIC through a software bridge.

Figure 1 shows the typical KVM architecture, with reference to a network related application.

As depicted in the 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 FALCON

FALCON's deployment model is similar to a typical SMR's. In a FALCON-replicated system, a set of $2f+1$ machines (nodes) are set up in a InfiniBand cluster. Once the FALCON system starts, one node becomes the *primary* node which proposes the order of requests to execute, and the others become backup nodes which follow the primarys proposals. An arbitrary number of clients in LAN or WAN send network requests to the primary and get responses. If failures occur, the nodes run a leader election to elect a new leader and continue.

On receiving a client network request, it invokes a RDMA-based consensus process on this request to enforce that all replicas see the same sequence of input requests. This process has three steps. In the first step, the leader assigns a global, monotonically increasing viewstamp to this request, stores this request into an entry that is appended to the consensus log, and does a forced write to the local disk. The second step is to replicate the log entry on remote servers using a one-sided RDMA Write operation. Usually when the RDMA NIC (RNIC) completes the network steps associated with the RDMA operation, it pushes a completion event to the queue pair's associated completion queue (CQ) via a DMA write. Using completion events adds extra overhead. Since PAXOS could help handle the reliability issues, FALCON takes advantage of unsignaled RDMA write operations, i.e., a completion event will not be pushed for these operations, to reduce that overhead. In the last step, the leader thread waits for acknowledgments from a majority of nodes.

In addition to the distributed consensus protocol for coordinating the sequence of input requests, FALCON also runs an output checking protocol which compares each replica's network outputs occasionally to detect the divergence of execution states.

3 Design of Architecture

Figure 2 shows our proposed architecture in KVM virtualization environment. Only the primary VM advertises its presence on the network, so all network inputs come to the primary VM.

It contains three main components: the Serviceguard, a tailored version of QEMU, and the SMR system FALCON.

The Serviceguard is deployed on the virtual machine, and provides monitoring capabilities for applications running inside virtual machines in the KVM virtualization environment.

4 Serviceguard

The Serviceguard is installed on the virtual machine, and monitors the overall health of the configured applications. It conveys the application health status to FALCON in the form of a heartbeat via a private VLAN. If FALCON does not receive the heartbeat within a specified interval on the leader node, it will stop sending heartbeats to the other nodes and a leader election starts. The new leader will take over serving client requests. Different from the traditional PAXOS leader election process, our solution adds an additional restriction on which servers may be elected leader. We prevent a candidate from winning an election unless all the con-

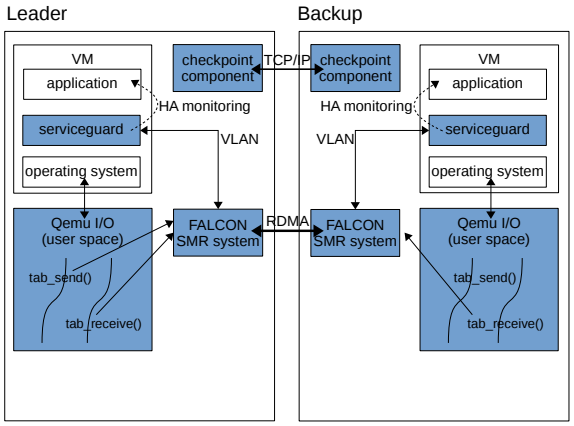

Figure 2: ***Proposed architecture in KVM virtualization environment.*** **Key components are shaded (and in blue).**

figured applications running thereon are healthy and its log is at least as recent as any other log in a majority. Here, which of two logs is more recent is determined by comparing the index and view of the last entries in the logs. If the logs have last entries with different views, then the log with a higher view is more recent. If the logs end with the same view, then the log with a higher index is more recent.

5 Tailored QEMU

In order to provide the server programs running inside the virtual machine the exact same inputs in the same order, we interpose on the `tap_send()` function in QEMU and invoke the PAXOS consensus component of FALCON on the received networking packets. Once the consensus process is finished, QEMU continues with its normal execution.

We modified the `tap_receive()` function in QEMU to maintain a packet queue for each application that captures the corresponding outgoing packets. The network outputs are pushed into the queue and whenever the queue is full, a new hash value is calculated by $h_i = H(h_{i-1}||H(queue))$ where $H()$ is a hash function and $||$ stands for concatenation. Such a computation links the hash value to all the previous network outpus. Then, after every T_{comp} hash values are generated, the latest one is passed to FALCON and the output checking protocol is invoked. The index of this hash value in the hash chain is consistent across replicas because each replica implements the same mechanism.

The checkpoint component is invoked every minute on a backup replica. It captures the entire execution state of the running VM. After each checkpoint, the compressed file is dispatched to the other replicas.

6 Evaluation

7 Related Work

8 Conclusion

We have presented the design of REPVM.

References

[1] 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.

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

[3] 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.

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