Dynamic and Secure Application Consolidation with Nested Virtualization and Library OS in Clouds

Kouta Sannomiya
Kyushu Institute of Technology
kouta@ksl.ci.kyutech.ac.jp

Kenichi Kourai
Kyushu Institute of Technology
kourai@ci.kyutech.ac.jp

In IaaS clouds, users can reduce costs by optimizing instance deployment as necessary. The most popular optimization is scale-out and scale-in, which adjust the number of instances. However, when there exists only one instance, a user cannot reduce the number of instances even if the instance is under utilization. The optimization for only one instance is scale-up and scale-down, which adjust the amount of resources allocated to an instance. Unfortunately, most of the existing clouds achieve this optimization by switching instance types offline because they cannot dynamically change the amount of resources of an instance. When a user switches instance types of his instance, he has to stop all the applications in the instance. This causes service downtime. In addition, cost reduction is limited by the cost of the minimum instance provided by clouds.

For further optimization, a user can consolidate applications in multiple instances into one instance. This is called *application consolidation*. For example, when tightly coupled applications such as web servers and a database are running across multiple instances and they are under utilization, a user can run these applications in one instance and reduce the cost. Later, when the instance becomes over utilization, a user can de-consolidate the applications to multiple instances again and perform load balancing. Like scale-up and scale-down, however, this causes service downtime when a user moves applications between instances. This problem can be mitigated by using process migration, but a security issue arises due to consolidating applications. Since multiple applications run in the same instance, isolation among them becomes weaker than before consolidation.

To solve these problems, we propose *FlexCapsule*, which can dynamically optimize instance deployment by running each application in a lightweight virtual machine (VM) called an *app VM*, as illustrated in Fig.1. FlexCapsule uses *nested virtualization* to run app VMs inside an instance, which is also a VM. Since FlexCapsule can migrate an app VM with an application as necessary, it can reduce service downtime on application consolidation and de-consolidation. In addition, it guarantees security between consolidated applications thanks to strong isolation between app VMs.

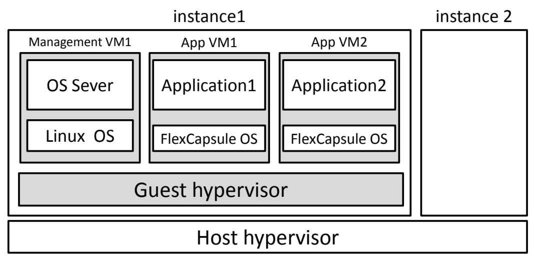

Figure 1. The architecture of FlexCapsule.

To reduce the overhead of extra virtualization, FlexCapsule uses an library operating system (OS) called *FlexCapsule OS*, which is optimized for app VMs. FlexCapsule OS is para-virtualized to avoid the overhead of full virtualization in nested virtualization. Therefore it provides support for VM migration and suspends and resumes devices by itself. Since FlexCapsule OS enables running only one application, it does not provide any protection. In addition, it needs only a small amount of memory.

FlexCapsule provides an OS server in each instance to manage multiple app VMs like traditional processes. Since each app VM is self-contained, the OS server manages app VMs from the outside. For example, the OS server provides information on app VMs to the FlexCapsule shell as if app VMs were processes. Furthermore, the OS server is used for cooperation between app VMs. For example, since FlexCapsule OS does not support multi-process, FlexCapsule achieves fork by duplicating app VMs. When an application in an app VM calls fork(), FlexCapsule OS communicates with the OS Server. Then the OS server creates a child app VM from the parent app VM and returns an identifier of the child VM to the parent VM. It returns zero to the child VM like the original fork().

We have implemented FlexCapsule in Xen 4.2. We have added support for VM migration and fork to Xen's Mini-OS. Using FlexCapsule, we measured migration performance of an app VM. As a result, the migration time was 2 times shorter in the minimum memory footprint, compared with migrating a VM where Linux is installed. The downtime was about 0.3 second and it was 0.1 second shorter than the VM running Linux.