Running Multi-Process Applications on Unikernel-based VMs

Minoru Kanatsu (Student and Presenter), and Hiroshi Yamada
TUAT
mkanatsu@asg.cs.tuat.ac.jp, hiroshiy@cc.tuat.ac.jp

In virtualized cloud platforms such as Amazon EC2 and Google Compute Engine, the performance of the virtual machine (VM) launch is an important factor of the service quality. The cloud user typically creates VMs from their images uploaded in advance. It is better to launch VMs as soon as possible since the cloud user can quickly handle an increased workload to the service even if he or she misses capacity planning of the VM number. The miss of capacity planning for the current big memory applications such as relational databases and in-memory KVSes is critical since it takes long time for them to perform the regular throughput due to buffer-pool misses. To effectively do so, some researchers propose VM fork [3, 1] that spawns a new VM inheriting the same running states of the running VM.

Unikernel [5] is an attractive architecture for swift VM launches. Unikernel is a library operating system (LibOS) that supports OS-like functions and can link with an application. We can build a bootable instance image by compiling a LibOS and the target application, and execute it directly on the cloud platform instead of running an OS kernel. This type of the software stack is suitable for cloud platforms since cloud users typically run a single application on each instance such as web servers and databases. Since the memory footprint of the running applications can be minimized by pruning unnecessary OS functions, Unikernel-based VM launches are much more lightweight than general purposed-based VM launches such as Linux.

However, unikernel is not applicable to all types of applications; it assumes *single-process* applications. To run multi-process applications on unikernel, we have to largely modify their code to be suitable for the target unikernel. Although some LibOSes [6] support the `fork()` system call, the spawned VMs are placed only on the same physical host as the parent VM. Since the privileged software such as hypervisor supports inter process communications (IPCs) such as `pipe()` and `signal()`, the system co-locates the new VMs with the parent VM even if the physical host is heavy-loaded. Although VM fork mechanisms [3, 1] allow us to place child VMs on the different physical hosts, the child unikernel-based VM does not perform IPCs to its parent VM. Since the VM fork mechanisms run at the hypervisor level and spawn a new VM transparently to the parent VM, running applications inside unikernel-based VMs cannot still invoke `fork()` and IPCs.

Our goal is to (1) achieve lightweight VM launches, (2) support as-is multi-process applications, and (3) locate child VMs on different hosts. We present *Unikernel fork*, a mechanism to satisfy our three goals. Unikernel fork allows us to run multi-process applications on Unikernel-based VMs while parent and child VMs are on different physical hosts. The challenge here is how to support IPC functionality between the unikernel-based VMs on different physical hosts. To do so, we prepare management module, running at the hypervisor-level, that intermediates interactions between the unikernel-based VMs on different physical hosts. We note that the target applications of our system is typical server applications hosted on cloud platform such as web servers. In such applications, the parent process typically spawns its child process to handle a coming connection and hardly communicate with the child processes. Therefore, the performance degradation caused by the application execution over the network not severe very much.

Our current system, based on Xen hypervisor and Rumprun unikernel [2], are as follows. We add the `fork()` hyper call Xen hypervisor as a hypercall, and expose it to Rumprun. We also prepare a hyper call for xenstore to share instance physical locations, and add `pipe()` and hypercall to Xen. In addition, we offer a mechanism to accelerate unikernel-based VM lauches. In particular, we use hot standby instances, like Android OS's Zygote [4].

To show our approach's advantage, we preliminary measure boot time of various OSes and unikernels. We define the boot time in this measurement as $\Delta T = T_e - T_s$, where T_s is boot start time, when instance is kicked by `xl` command and T_e is boot end time, when application code on instance print time stamp counter. This preliminary experiment assumed that many VM instance boot on time. We measured launch time for each VM with 10 instances at the same time. The result shows that unikernel has advantage for boot time compared to Linux, existing OS. And also unikernel has advantage compared to **Mini-OS** that is minimize OS for the Xen PV mode. **Linux** has the longest boot time in this case. The boot time is 19.5 s. In **Rumprun**, the boot time is 1 s. **Mini-OS** booted up in 5.4 s.

Reference

[1] R. Bryant, A. Tumanov, O. Irzak, A. Scannell, K. Joshi, M. Hiltunen, H. A. Lagar-Cavilla, and E. d. Lara. Kaleidoscope: Cloud Micro-Elasticity via VM State Coloring. In *Proc. of the 6th ACM SIGOPS/EuroSys European Conference on Computer Systems*, (*EuroSys '11*), pages 273–286, Apr. 2011.

[2] A. Kantee and J. Cormack. Rump Kernels: No OS? No Problem! *;login:* 39(5):11–17, Oct. 2014.

[3] H. A. Lagar-Cavilla, J. A. Whitney, A. Scannell, P. Patchin, S. M. Rumble, E. d. Lara, M. Brudno, and M. Satyanarayanan. SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing. In *Proc. of the 4th ACM SIGOPS/EuroSys European Conference on Computer Systems*, (*EuroSys '09*), pages 1–12, Mar. 2009.

[4] J. Levin. *Android Internals::Power User's View*, volume 1 of *Android Internals*. 1st edition, Jan. 2015, pages 1–250. ISBN: 978-0991055524.

[5] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft. Unikernels: Library Operating Systems for the Cloud. In *Proc. of the 18th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, (*ASPLOS '13*), pages 461–472, Mar. 2013.

[6] C.-C. Tsai, K. S. Arora, N. Bandi, B. Jain, W. Jannen, J. John, H. A. Kalodner, V. Kulkarni, D. Oliveira, and D. E. Porter. Cooperation and Security Isolation of Library OSes for Multi-process Applications. In *Proc. of the 9th ACM SIGOPS/EuroSys European Conference on Computer Systems*, (*EuroSys '14*), 9:1–9:14, Apr. 2014.

CONF 'yy, Month d–d, 20yy, City, ST, Country.
Copyright © 2017 ACM 978-1-nnnn-nnnn-n/yy/mm…$15.00.
http://dx.doi.org/10.1145/nnnnnnn.nnnnnnn