User Space Memory Management for Post-copy Migration

Mike Rapoport
IBM Research – Haifa
rapoport@il.ibm.com

Joel Nider
IBM Research – Haifa
joeln@il.ibm.com

CCS Concepts

•**Software and its engineering** → **Memory management;** *Virtual machines;* Cloud computing;

Keywords

Operating Systems; Memory management, Virtualization, Containers

1. PROBLEM

Post-copy migration allows reduction of application downtime and reduces overall network bandwidth used for application migration [4]. Migration can be used to help optimize several aspects of operations such as power efficiency[3]. The userfault technology recently introduced to the Linux kernel allows post-copy migration of virtual machines. However, this technology is missing essential features required for post-copy migration of Linux containers.

2. POST-COPY VM MIGRATION

The userfault technology [1] recently introduced into the Linux kernel allows a user space application to process page faults. It may process its own page faults (cooperative userfault), or on behalf of another process (non-cooperative userfault). The KVM hypervisor utilizes the cooperative userfault technology to implement post-copy migration of virtual machines. In the KVM solution, a virtual machine can be migrated to a destination host without copying the memory upfront. The guest execution is restarted on the destination host and each memory access causes a page fault. These page faults are delivered by the userfault to the user-space part of KVM hypervisor (qemu), which requests the required memory page from the source node hypervisor, copies the received page into the guest memory and resumes the guest execution.

3. POST-COPY CONTAINER MIGRATION

A similar technique may be used for post-copy migration of Linux containers. However, unlike a virtual machine, there is no hypervisor that has complete control over those processes. Therefore, container migration uses an external tool called CRIU [2] that collects the information necessary to migrate a container, including its memory dump, and then restores the container process tree on the destination host. Since CRIU is an independent process and does not share address space with the restored process it must use non-cooperative userfault. In this mode it is necessary not only to track process memory accesses but also changes to the process virtual memory layout.

In Linux, there are several system calls that allow modification of the virtual memory layout of a process. For instance, *mremap()* system call changes the virtual address of certain memory range inside the application address space, or *madvise(MADV_DONTNEED)* drops pages from a virtual address range and any subsequent accesses to this range will generate zeros.

We implement extensions for userfault technology that add hooks to these system calls and provide the means for the monitor application to take an appropriate action and adjust the non-cooperative process virtual memory layout accordingly.

Another system call that was modified to support non-cooperative userfaults is *fork()*. When a new process is created using the *fork()* system call, its virtual memory layout is identical to the memory layout of its parent. The userfault monitor has to detect the creation of the new processes so it will be able to fill their memory with the appropriate contents.

4. ACKNOWLEDGMENTS

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688386.

5. REFERENCES

[1] A. Arcangeli and D. A. Gilbert. Memory externalization with userfaultd. In *KVM Forum*, 2014.

[2] CRIU - Checkpoint Restore in Userspace. https://criu.org.

[3] J. Nider and M. Rapoport. Cross-isa container migration. In *Proceedings of the 9th ACM International on Systems and Storage Conference*, SYSTOR '16, pages 24:1–24:1, New York, NY, USA, 2016. ACM.

[4] E. Zayas. Attacking the process migration bottleneck. *SIGOPS Oper. Syst. Rev.*, 21(5):13–24, Nov. 1987.

SYSTOR '17 Haifa, Israel

© 2017 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-5035-8/17/05.

DOI: http://dx.doi.org/10.1145/3078468.3078490