Poster: Migrating Running Applications Across Mobile Edge Clouds

Andrew Machen*, Shiqiang Wang†, Kin K. Leung*, Bong Jun Ko†, Theodoros Salonidis†
* Imperial College London, United Kingdom, Email: {andrew.machen14, kin.leung}@imperial.ac.uk
† IBM T.J. Watson Research Center, Yorktown Heights, NY, USA, Email: {wangshiq, bongjun_ko, tsaloni}@us.ibm.com

ABSTRACT

Mobile edge clouds (MECs) are small cloud-like infrastructures deployed in close proximity to users, allowing users to have seamless and low-latency access to cloud services. When users move across different locations, their service applications often need to be migrated to follow the user so that the benefit of MEC is maintained. In this paper, we propose a layered framework for migrating running applications that are encapsulated either in virtual machines (VMs) or containers. We evaluate the migration performance of various real applications under the proposed framework.

CCS Concepts

•**Computer systems organization** → **Cloud computing;** •**Human-centered computing** → *Ubiquitous and mobile computing systems and tools;*

Keywords

Cloudlet; edge computing; service migration; containers

1 Introduction

Mobile edge clouds (MECs) represent a way of bringing the benefits of the cloud closer to the user, by installing small cloud infrastructures at the network edge. The close proximity of MECs to the user naturally bestows low-latency that allows for improvements in cloud services, and an altogether new breed of real-time applications that necessitate super low latency, such as instantaneous object recognition.

An important characteristic of MEC is the need for service migration triggered by user mobility. In short, the service applications need to be migrated (i.e., moved) to follow the user so that the performance benefit of MEC can be maintained [1]. We consider the *stateful* migration of applications in this paper, where a user receives service for a continuous period of time, and the service application needs to keep some internal state for the user, such as intermediate data processing results. After migration is completed, programs resume exactly where they left off before migration, thus the migration is classified as *live*.

An active research challenge is how to implement live migration in MECs. Most existing virtual machine (VM) live migration methods are built with data center environments in mind, assuming the use of local area networks and shared storage. Such shared storage with ultra-high speed network connection is generally not available in an MEC environment, where applications are deployed in geographically dispersed locations. For live migration in the MEC environment, some effort has been recently made with focus on VM migration [2]. *Container migration* is a relatively new area which has not been systematically studied in the literature. As containers usually have much smaller size than VMs, it can be very beneficial to run container-based applications on MECs which have limited storage and processing capability.

In this paper, we present a generic layered migration framework using incremental file synchronization, which works with applications that are encapsulated *either in a container or in a VM*. A benefit of this framework is that it is built based on functionalities readily available in most container and VM technologies that are popular today, so that one does not need to have extensive knowledge on the technical details of container and VM implementation in order to apply this framework. We also study how containers and VMs perform when live-migrated between MECs using our framework, for a variety of applications.

2 Layered Migration Framework

We use LXC (linuxcontainers.org) and KVM (www.linux-kvm.org) as representative technologies respectively for containers and virtual machines, which already have accessible tools that can be leveraged for migration, namely the *checkpointing* functionality of LXC and the *saving* functionality of KVM. Both methods suspend the guest OS (at which point the application is temporarily stopped) and save down the in-memory state into one or more files that can be easily transferred. Complementary tools exist to restore the guest OS from their checkpoint (or save).

The problem with migrating an encapsulated application directly is that the "package" contains the guest OS, virtualization data, and, for VMs, the system kernel, which are required to make the service application self-contained. A base package with no services installed can have a size of hundreds of megabytes and a few gigabytes for LXC and KVM, respectively. Migrating the whole suspended package directly would incur excessive service downtime.

In the proposed framework, we reduce the service downtime and overall migration time through the use of *layers*. We separate the base package (that includes the guest OS, kernel, etc., but with no applications installed) into a layer separate and distinct from its service applications. We call this package the *base layer*, which can be largely reused by different applications. *A copy of this base layer is stored on every MEC*, so that it does not need to be transferred in every migration. Then, we have an *application layer*

MobiCom'16 October 03-07, 2016, New York City, NY, USA

ACM ISBN 978-1-4503-4226-1/16/10.

DOI: http://dx.doi.org/10.1145/2973750.2985265

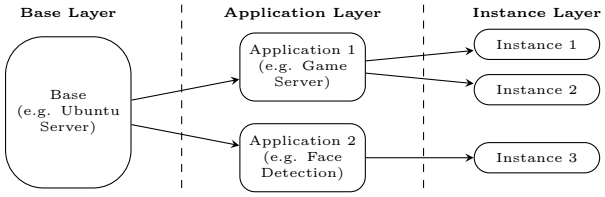

Figure 1: The three-layer framework.

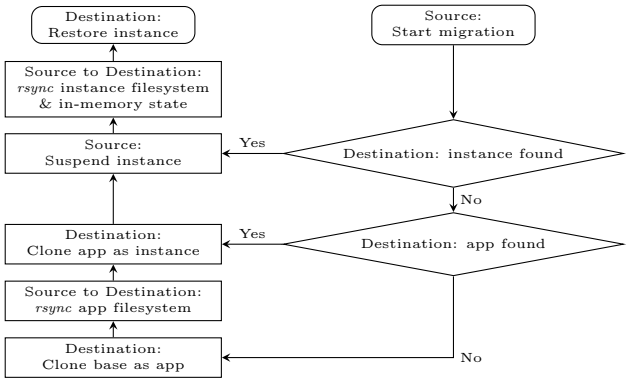

Figure 2: Flow chart of the migration mechanism.

which contains an *idle* version of the application and any application-specific data. The running state of an application is placed in a third layer called the *instance layer*. This *three-layer framework* is shown in Fig. 1.

When migrating a service, the application layer can be migrated first whilst the service is running, then the service is suspended and the instance layer is transferred to the destination. A running service can be reconstructed from a combination of the base, application, and instance layers. By doing this, we are able to transfer the majority of an encapsulated service's data before suspending the service, leaving only the running state to count towards the service downtime. The files are transfered with incremental file synchronization, where we use *rsync* (rsync.samba.org) in the current implementation. This ensures that only those files, or parts thereof, which are different from those already located at the destination, are being transfered. The complete migration process is shown in Fig. 2.

3 Evaluation

We ran experiments with different applications running in LXC and KVM. The detailed experimentation setting is omitted here due to space limitation. The results are shown in Table 1 and Fig. 3. We see that containers perform favorably over VMs, mainly because containers are more compact and the filesystem and in-memory contents of a container is mostly relevant to the application; whereas the filesystem and in-memory contents of a VM can be related to many other background processes irrelevant to the considered application, and *rsync* (or any other incremental file synchronization mechanism) needs to remotely compare a larger amount of data and may not be able to filter out everything that does not belong to the application. The total migration time is shorter when the application has been found (i.e., pre-exists) at the destination. The service downtime is related to transferring the runtime state (i.e., instance layer) and is only a fraction of the total migration time.

LXC Mig. Time / Transferred Data	App not found	App found	Service Downtime
No Application	11.0 s 1.9 MB	6.3 s 1.4 MB	2.0 s
Game Server	10.9 s 2.7 MB	6.4 s 1.6 MB	2.0 s
High RAM App	27.2 s 97.6 MB	19.8 s 97.1 MB	15.3 s
Video Streaming	37.3 s 184.6 MB	8.5 s 7.4 MB	3.3 s
Face Detection	70.1 s 365.0 MB	15.5 s 10.0 MB	3.7 s

KVM Mig. Time / Transferred Data	App not found	App found	Service Downtime
No Application	141.8 s 65.9 MB	79.7 s 65.2 MB	56.1 s
Game Server	142.0 s 72.3 MB	80.9 s 69.7 MB	58.7 s
High RAM App	152.2 s 178.4 MB	93.0 s 170.4 MB	72.0 s
Video Streaming	189.7 s 270.5 MB	86.4 s 87.3 MB	61.3 s
Face Detection	558.3 s 1,033.7 MB	152.3 s 99.0 MB	107.5 s

Table 1: Total migration time and transferred data for LXC (top) and KVM (bottom) setups.

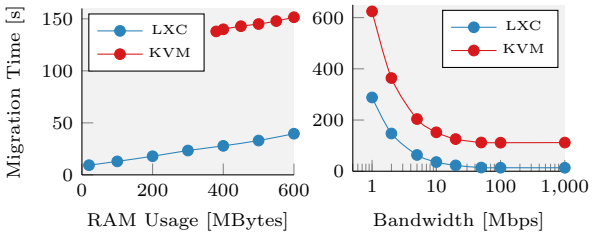

Figure 3: Total migration time under different RAM usage and bandwidth.

4 Conclusion

The proposed three-layer framework allows different applications to be efficiently migrated following a general approach. A highlight of this framework is that it supports containers, which is a promising emerging technology that offers tangible benefits over VMs. This framework also allows popular applications to be pre-distributed (or cached) at MECs, so that the overall instantiation/migration time can be reduced. As such, it enables rapid reconfiguration of MECs to cope with dynamics at the network edge.

Acknowledgement

This research was sponsored in part by the US Army Research Laboratory and the UK Ministry of Defense and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the US Government, the UK Ministry of Defense or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

5 References

[1] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge-clouds," in *Proc. of IFIP Networking 2015*, May 2015.

[2] K. Ha, Y. Abe, Z. Chen, W. Hu, B. Amos, P. Pillai, and M. Satyanarayanan, "Adaptive VM handoff across cloudlets," Technical Report CMU-CS-15-113, CMU School of Computer Science, Tech. Rep., 2015.