Title of the Paper

First Last First Last First Last
Note: Make sure all info is hidden for double-blind submissions Harvard University

ABSTRACT

We present an abstract!

1. INTRODUCTION & MOTIVATION

Serverless Computing. Serverless computing, also known as *Functions-as-a-Service* (FaaS), has become an increasingly important and desirable platform in the cloud ecosystem. Stemming from the grand vision of computation as a utility, serverless computing offers users the ability to run application code directly on a black-box infrastructure. In comparison to current cloud computing platforms, serverless users (1) no longer have to manage virtual machine environments, (2) are billed only for application computation performed in response to requests, and (3) function instances are auto-scaled to handle dynamically changing request rates. Options are available from all of the major cloud players, including Amazon Lambda, Azure Functions, and Google Cloud Functions. Several significant online companies have implemented parts of their services on serverless platforms, notably the news site The Guardian.[1].

Serverless Infrastructure. The typical implementation of a serverless computing platform places user-submitted functions onto dynamically created Virtual Machines (VMs). These functions are intended to be light-weight stateless single-process programs. Because of the relatively short run-times of serverless functions, a critical performance constraint in serverless infrastructure implementations is the scheduling latency of functions – mainly comprising of the creation time for instance VMs. A standard optimization employed for this start-up time is to keep instance VMs running for a period of time, and schedule function requests onto existing VMs. This leads to a number of other properties, including limits on function run-time and the potential for functions to be terminated at any time. This is necessary for the scheduler to auto-scale instances efficiently. Another property is the clear split in individual function start-up latencies between *warm-starts*, where a function is scheduled to an already running instance, and *cold-starts*, where a new VM must be created for the function. Functions from different users are usually not placed in the same VM for security and isolation reasons, but one VM can host several instances of one user's function.

[1]https://aws.amazon.com/solutions/case-studies/the-guardian/

The Problem with Serverless: Coldstart. A central promise of serverless computing services is rapid scalability. Meeting this demand at scale requires that new function instances can be started very quickly to service incoming requests. This is easy when there exist currently running *warm* instances but more difficult when a new *cold* instance must be started. A major bottleneck in achieving low latency instance start-up and fast auto-scaling is the cold-start latency [19]. For cold-start, the major bottleneck is the creation of a new VM. After scheduling and provisioning, when a VM is created the host Hypervisor/Virtual Machine Monitor (VMM) must initialize virtual resources including CPUs, memory, and other devices, then the guest kernel and file system must be loaded from disk and initialized in guest memory, then the guest kernel is booted, and finally the function runtime/process is started and the request is handled.

Flash Cloning. The most significant parts of VM creation are (1) copying the kernel and file system into memory, (2) booting the guest kernel, and (3) loading potentially large libraries/runtimes. Management operations within the VMM are relatively cheap compared to the start-up within a guest and the copying of guest memory. Optimizing these steps could dramatically reduce the startup latency of new serverless functions. One technique to do this is to employ *flash cloning*. Instead of loading VM images from disk and booting a kernel, we propose cloning existing reference VMs in memory. Additionally, we propose a copy-on-write mechanic to reduce both the copy time overhead and the memory pressure of packing many VMs onto one host. This method can be compared to the Unix fork abstraction.

> To create an isolated virtual machine, rather than re-create the entire world, we should only re-create the necessarily distinct VMM components and clone identical guest memory and execution context from ready-to-go reference VMs.

From this insight we present *HyperFork*, a KVM-based VM cloning implementation for the context of serverless computing.

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

Contributions. Our contributions are as follows:

- A KVM based virtual machine cloning implementation which outperforms standard VM creation latencies by up to AMAZING%
- A thorough discussion of the trade-offs related to a number of potential implementations for VM creation in the context of serverless computing.
- A thorough analysis of the performance of Hyperfork for both VM creation latency as well as VM co-location density and performance degredation with respect to copy-on-write memory sharing.

2. HYPERFORK ARCHITECTURE

2.1 Firecracker VMM

2.2 Host KVM

2.3 Guest Kernel Driver

3. EXPERIMENTAL EVALUATION

This is the experiments section First summarize the findings in an implicit way. For example: "In this section, we demonstrate that System X achieves Z, Y, K" where Y, Z, K are our main contributions in the paper as well.

Experimental Setup. The experimental setup includes information about the following 5 things: (i) *Experimental Platform*, i.e., the machine we used, (ii) *Implementation*, i.e., whether it is a prototype system, based on an existing one, and any other relevant detail, (iii) *Configuration*, i.e., the system-specific configuration and tuning that might have been needed, and discuss the possible values for any parameter. (iv) *Workloads*, i.e., dataset characteristics, and query characteristics, and (unless the whole paper is a single experiment this here just gives a summary of the workloads used and each experiment later on gives the exact setup) (v) *Metrics*, typically latency and/or throughput with some details and the methodology.

For the experimental platform we do not want to use exactly the same phrasing in every paper, however, the content should be the same.

Explaining Each Figure. We first put the most important figures, i.e., the ones that have the most important results in terms of the contributions. Each figure should be a single message. Each figure comes with two paragraphs. One paragraph for the setup and one paragraph for the discussion. The set-up paragraph should start by saying "In this experiment we show that... We setup this experiment as follows...".

The result paragraph explains in detail why we see what we see. Explanations should be based on facts and logic. Numbers that back up the explanations should be provided whenever possible. The paragraph should finish by repeating the main message again.

4. RELATED WORK OR BACKGROUND

Our work in this paper draws concepts from several lines of past research.

Virtualization Technology. Machine virtualization technology is a complex and diverse space. Classical Hypervisors/Virtual Machine Monitors (VMMs) relied on the fundamental primitive of *Trap-and-Emulate*, where sensitive instructions in the guest would be traped by the hardware, and emulated safely within the VMM using shadow structures for privaleged state [17]. In x86 however, not all sensitive instructions are privaleged, meaning they can be trapped, and other techniques must also be used to enable virtualization. *Full virtualization* of unmodified guest kernels was enabled through binary translation, where all sensitive instructions could be translated into privaleged instructions. *Paravirtualization* used modifications to the guest operating systems to ensure sensitive operations were trapped. Today, modern hardware architectures include special virtualization instructions which remove the need for binary translation, and additionally remove the need for performance critical shadow structures using two-dimensional hardware page tables [6]. Current virtualization technologies offer a mix of all these techniques, including binary translation for full nested virtualization using Oracle's HVX [10], paravirtualization with Xen [8], and classic trap-and-emulate utilizing modern hardware extensions and the QEMU x86 emulator [9] within the Linux kernel with KVM [14]. Xen is highly used within the research community because of its relatively simple software-only techniques, and KVM is valued for its tight integration with the Linux kernel.

Serverless Computing. Serverless computing has become an increasingly important and desireable platform in the cloud ecosystem. Stemming from the grand vision of computation as a utility, serverless computing offers users the ability to run application code directly on a black-box infrastructure. Serverless has the potential to offer an easy to program, auto-scaling, cost efficient way to utilize cloud infrastructure for users, without the need to manage machine provisioning and configuration or service orchestration [13]. Although there exist many popular industry serverless computing platforms today [1][2][3][4], serverless is still an active area of research, with many improvements to be made [19][7][11].

Flash Cloning / MicroVMs / Coldstart reduction efforts. Include Potemkin and Snowflock, as well as that short one about caching python modules. Also maybe include unikernels/Denali/Firecracker. [18][15][16][20][5]

VM Live Migration. Cite some state-of-the-art research / survey, maybe some tools/implementations. Focus text on explaining difference from flash cloning. [12]

5. SUMMARY

In this paper we present *HyperFork*, it's great guys.

6. ACKNOWLEDGMENTS

This is the acknowledgments section Note: Decide whether to add ack's in the submission

Note: Make sure we have the updated version of the conference style

7. REFERENCES

[1] https://aws.amazon.com/lambda/, accessed 2019.

[2] https://cloud.google.com/functions/, accessed 2019.

[3] https://azure.microsoft.com/en-us/services/functions/, accessed 2019.

[4] https://openwhisk.apache.org/, accessed 2019.

[5] https://lwn.net/Articles/775736/, accessed 2019.

[6] K. Adams and O. Agesen. A comparison of software and hardware techniques for x86 virtualization. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XII, pages 2–13, New York, NY, USA, 2006. ACM.

[7] I. Baldini, P. Cheng, S. J. Fink, N. Mitchell, V. Muthusamy, R. Rabbah, P. Suter, and O. Tardieu. The serverless trilemma: Function composition for serverless computing. In *Proceedings of the 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2017, pages 89–103, New York, NY, USA, 2017. ACM.

[8] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 164–177, New York, NY, USA, 2003. ACM.

[9] F. Bellard. Qemu, a fast and portable dynamic translator. pages 41–46, 01 2005.

[10] A. Fishman, M. Rapoport, E. Budilovsky, and I. Eidus. HVX: Virtualizing the cloud. In *Presented as part of the 5th USENIX Workshop on Hot Topics in Cloud Computing*, San Jose, CA, 2013. USENIX.

[11] J. M. Hellerstein, J. M. Faleiro, J. E. Gonzalez, J. Schleier-Smith, V. Sreekanti, A. Tumanov, and C. Wu. Serverless computing: One step forward, two steps back. *CoRR*, abs/1812.03651, 2018.

[12] M. R. Hines and K. Gopalan. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '09, pages 51–60, New York, NY, USA, 2009. ACM.

[13] E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. Yadwadkar, J. Gonzalez, R. Ada Popa, I. Stoica, and D. A. Patterson. Cloud programming simplified: A berkeley view on serverless computing, 02 2019.

[14] A. Kivity Qumranet, Y. Kamay Qumranet, D. Laor Qumranet, U. Lublin Qumranet, and A. Liguori. Kvm: The linux virtual machine monitor. *Proceedings Linux Symposium*, 15, 01 2007.

[15] H. A. Lagar-Cavilla, J. A. Whitney, A. M. Scannell, P. Patchin, S. M. Rumble, E. de Lara, M. Brudno, and M. Satyanarayanan. Snowflock: Rapid virtual machine cloning for cloud computing. In *Proceedings of the 4th ACM European Conference on Computer Systems*, EuroSys '09, pages 1–12, New York, NY, USA, 2009. ACM.

[16] A. Madhavapeddy and D. J. Scott. Unikernels: Rise of the virtual library operating system. *Queue*, 11(11):30:30–30:44, Dec. 2013.

[17] G. J. Popek and R. P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, July 1974.

[18] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. C. Snoeren, G. M. Voelker, and S. Savage. Scalability, fidelity, and containment in the potemkin virtual honeyfarm. In *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles*, SOSP '05, pages 148–162, New York, NY, USA, 2005. ACM.

[19] L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift. Peeking behind the curtains of serverless platforms. In *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '18, pages 133–145, Berkeley, CA, USA, 2018. USENIX Association.

[20] A. Whitaker, M. Shaw, and S. D. Gribble. Denali: A scalable isolation kernel. In *Proceedings of the 10th Workshop on ACM SIGOPS European Workshop*, EW 10, pages 10–15, New York, NY, USA, 2002. ACM.

APPENDIX

A. APPENDIX SECTION [RENAME]

This is a section in the appendix

B. LIST OF FIXMES

List of Corrections