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/

ACM ISBN 978-1-4503-2138-9.

DOI: 10.1145/1235

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. The major bottleneck in achieving low latency instance start-up and fast auto-scaling is the cold-start latency. 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.[2]

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

[2]Cite Peeking Behind the Curtain? Or something to justify bottleneck description.

VM cloning implementation for the context of serverless computing.

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. A description of Paravirtualization (Xen), Binary Translation (Old VMWare), Virtualization Hardware, KVM and QEMU. Keep it brief and focus on aspects related to current KVM and our Hyperfork implementation, such as copy-on-write and virtualized hardware extensions. This is at least 4 or 5 citations.

Serverless Computing. A description of the general goals of serverless computing, the various aspects to serverless research, and the current state of the art industrial offerings. Include things like berkley serverless stuff, peeking behind curtain, formal models of computation, Firecracker, and a bunch of AWS/Microsoft/IBM citations for offerings. Again, brief text, relating mostly to the importance of serverless and interest of research community. Basically just cite the class reading list.

Flash Cloning / Coldstart reduction efforts. Include Potemkin and Snowflock, as well as that short one about caching python modules.

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

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

APPENDIX

A. APPENDIX SECTION [RENAME]

This is a section in the appendix

B. LIST OF FIXMES

List of Corrections