Supercharging Scientific Serverless: Slashing Cold Starts with Python UniKernels

By Jamie K

ILLINOIS TECH

Cold Starts

Container Cold Starts in HPC: Baseline

- Container systems are necessary for dependency management
- Containers have poor cold start times in HPC.

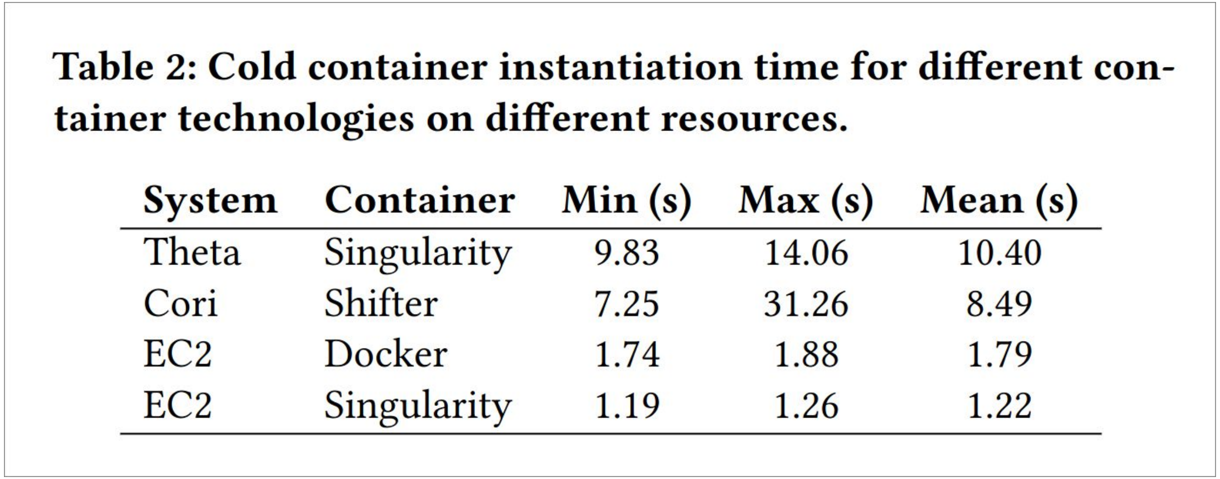


Table 2: Cold container instantiation time for different container technologies on different resources.

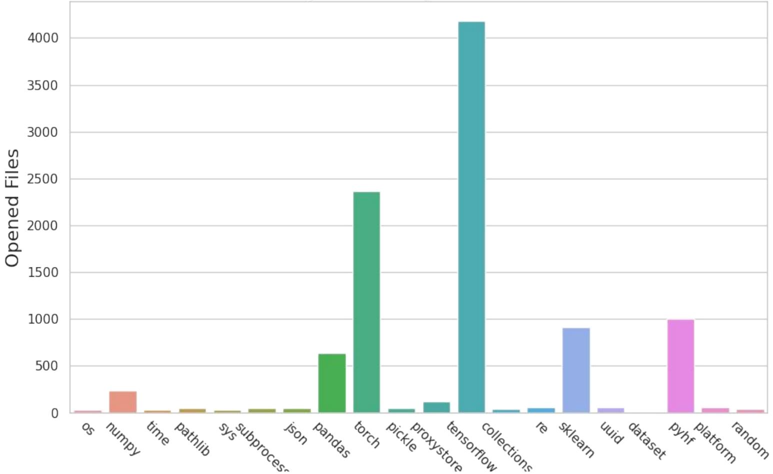
System	Container	Min (s)	Max (s)	Mean (s)
Theta	Singularity	9.83	14.06	10.40
Cori	Shifter	7.25	31.26	8.49
EC2	Docker	1.74	1.88	1.79
EC2	Singularity	1.19	1.26	1.22

Chard et al. 2019

We see many diverse dependencies

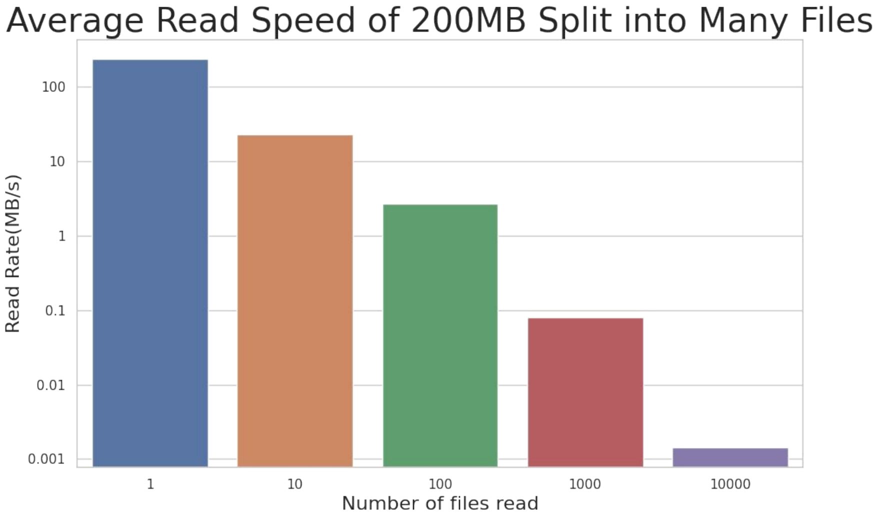
- FuncX users create apps that depend on many different libraries
- Some of those libraries require many files

Number of Files Opened by Most Common Libraries



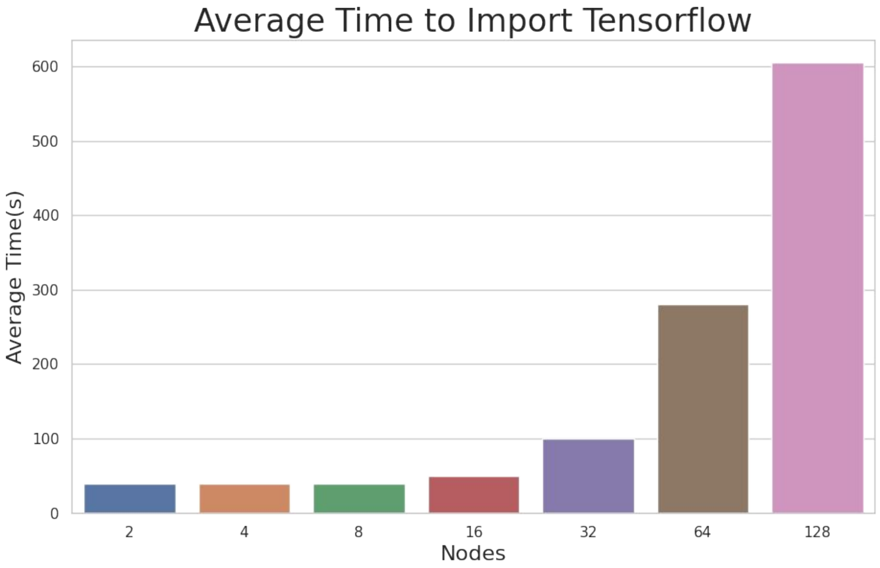
Shared File system Performance

- Shared File systems read large files quickly.
- Performance degrades quickly as file count increases
- Metadata store becomes bottleneck with many small files



Import tensorflow in HPC

- Can take up to 10 minutes to import tensorflow
- Consequence of reading many small files on HPC system
- Containers depend on underlying file system, thus would have the same problem

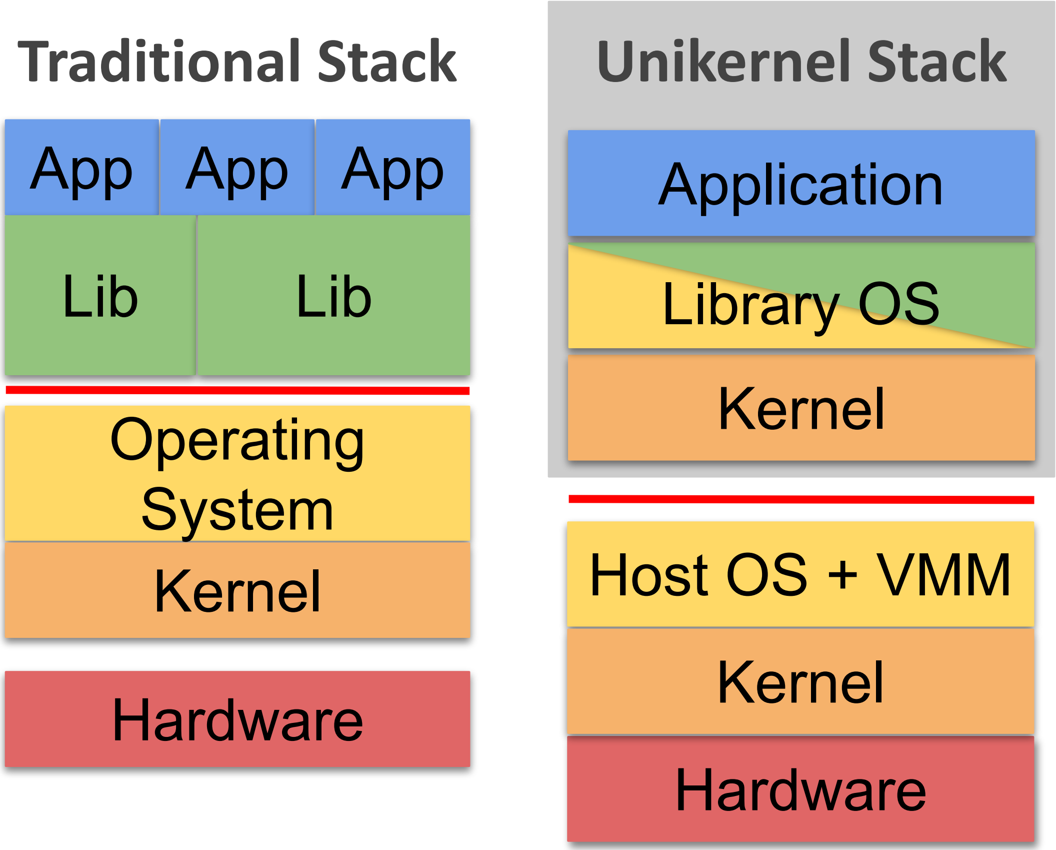


Kamatar et al. 2023

What are Unikernels?
How do they Help?

What's a Unikernel

- Unikernel is an operating system built for a singular application
- Traditional Systems can support many libraries and interact with many devices
- Unikernels support devices and libraries necessary for a singular application
- Red line indicates userspace-kernelspace barrier
- We use Unikraft to build unikernels



Traditional Stack

App App App

Lib Lib

Operating System

Kernel

Hardware

Unikernel Stack

Application

Library OS

Kernel

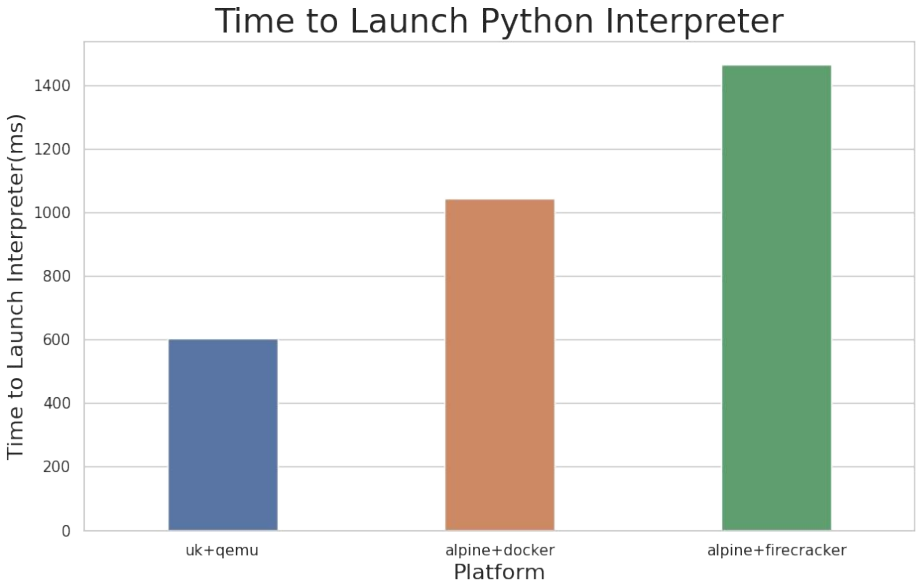
Host OS + VMM

Kernel

Hardware

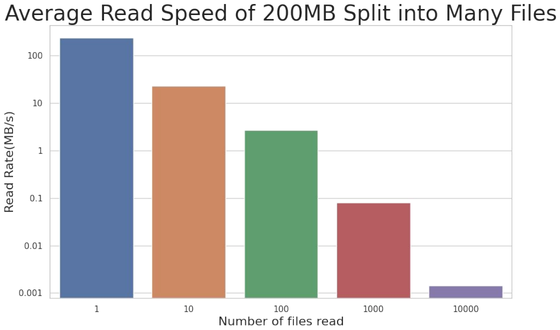
How do they help?

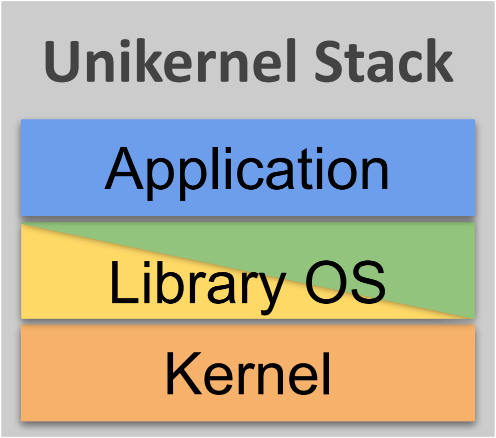
- Unikernels are lightweight and thus boot faster than other containers and VMMs
- Figure shows comparison of UniKernel on Qemu-KVM, Alpine on Docker, and Alpine on Firecracker
    - Alpine = Lightweight Linux
    - Firecracker = Micro VMM



Unikernel on Shared FS

- Since a Unikernel image contains all dependencies using applications with many dependencies(e.g. tensorflow) does not tax the metadata store
- Working towards performing this experiment*
- Current Unikernel image size ~ megabytes


Average Read Speed of 200MB Split into Many Files


Unikernel Stack

Application

Library OS

Kernel

The Big Picture: Future Work

- We envision users treating unikernels like docker containers. This entails:
  - A build system wherein users specify python dependencies
  - Build VMM? qemu-kvm and firecracker work now, but do we need better?
- Immediate Future Work
  - Make unikernel that can import python libraries
  - Measure unikernel cold start time on shared file system
- Medium Term Future Work
  - Unikraft is modular, allowing system calls to be rewritten, we need to investigate how this can help
  - Do we need to modify the VMM to enable snapshotting?

Shout-out to the Homies(Acknowledgements)

- Kyle Chard => Helping with the story
- Valerie Hayot-Sasson => Getting in the mud with me(making stuff work)
- Kyle Hale => Helping me with kernel stuff
- Alexandru Orhean => Setting up a cluster
- Ryan Chard => dataset on common python libraries

References

- Kuenzer, S., Bădoiu, V.A., Lefeuvre, H., Santhanam, S., Jung, A., Gain, G., Soldani, C., Lupu, C., Teodorescu, ., Raducanu, C., & others (2021). **Unikraft: fast, specialized unikernels the easy way.** In *Proceedings of the Sixteenth European Conference on Computer Systems* (pp. 376–394).
- Kamatar, A., Sakarvadia, M., Hayot-Sasson, V., Chard, K., & Foster, I. (2023). **Lazy Python Dependency Management in Large-scale Systems.** *To Appear in Proceedings of the International Conference on eScience*. **Slide 4**
- Shaffer, Tim, and Douglas Thain. **Taming metadata storms in parallel filesystems with metaFS.** Proceedings of the 2nd Joint International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems. 2017.
- Chard, R., Babuji, Y., Li, Z., Skluzacek, T., Woodard, A., Blaiszik, B., Foster, I., & Chard, K. (2020). **Funcx: A federated function serving fabric for science.** In *Proceedings of the 29th International symposium on high-performance parallel and distributed computing* (pp. 65–76). **Slide 1**
- Babuji, Y., Woodard, A., Li, Z., Katz, D., Clifford, B., Kumar, R., Lacinski, L., Chard, R., Wozniak, J., Foster, I., & others (2019). P**arsl: Pervasive parallel programming in python.** In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing* (pp. 25–36).
- Wanninger, N., Bowden, J., Shetty, K., Garg, A., & Hale, K. (2022). **Isolating functions at the hardware limit with virtines.** In *Proceedings of the Seventeenth European Conference on Computer Systems* (pp. 644–662).
- Agache, A., Brooker, M., Iordache, A., Liguori, A., Neugebauer, R., Piwonka, P., & Popa, D.M. (2020). **Firecracker: Lightweight virtualization for serverless applications.** In *17th USENIX symposium on networked systems design and implementation (NSDI 20)* (pp. 419–434).
- Oakes, E., Yang, L., Zhou, D., Houck, K., Harter, T., Arpaci-Dusseau, A., & Arpaci-Dusseau, R. (2018). **SOCK: Rapid task provisioning with Serverless-Optimized containers.** In *2018 USENIX annual technical conference (USENIX ATC 18)* (pp. 57–70).
- Cadden, J., Unger, T., Awad, Y., Dong, H., Krieger, O., & Appavoo, J. (2020). **SEUSS: skip redundant paths to make serverless fast.** In *Proceedings of the Fifteenth European Conference on Computer Systems* (pp. 1–15).