Understanding Container Isolation:
An Investigation of Performance Implications of Container Runtimes

Naman Bhaia, Robert Cordingly, Ling-Hong Hung, Wes Lloyd
nbhaia@uw.edu

School of Engineering and Technology
University of Washington Tacoma
9th International Workshop on Container Technologies and Container Clouds
Middleware 2023

Outline

- Background and Motivation
- Research Question
- Methodology
- Results
- Conclusions

Motivation: Why Containers?

- Modern CPU designs have ever increasing number of CPU cores to improve performance.
- Dilemmas:
  - How should these powerful servers be best shared to run multiple-user workloads concurrently?
  - Which abstractions minimize performance interference and "sandbox" overhead?
- Containers vs VMs: Containers are more lightweight and efficient
  - Allows more containers to run on the same host.
- Containers vs running applications on the host: Containers provide improved security and isolation.
- Identifying Containers as best of both extremes, this study focuses on diving deeper into containers to identify the container runtime with the best resource isolation capabilities.

Container Runtime Performance Overhead

- Virtualization hypervisors abstract the system CPU, memory, and I/O devices introducing overhead resulting from sharing the hardware.
- Container overhead is from the Linux kernel at the OS level.

Prior work

1. Benchmarks run in containers performed better than VMs due to the minimal overhead added by containers. [1]
2. Performance of runc, gVisor, and Kata containers were found to be in the following order: runc > Kata > gVisor. [2]

This paper extends the prior work by comparing concurrent performance of popular container runtimes.

[1] R oberto Morabito, Jimmy Kjällman, and Miika Komu. 2015. Hypervisors vs. Lightweight Virtualization: A Performance Comparison. In 2015 IEEE International Conference on Cloud Engineering. 386–393. https://doi.org/10.1109/IC2E.2015.74
[2] Xingyu Wang, Junzhao Du, and Hui Liu. 2022. Performance and isolation analysis of RunC, gVisor and Kata Containers runtimes. Cluster Computing 25 (04 2022), 1–17. https://doi.org/10.1007/s10586-021-03517-8

Differences in Container Runtimes

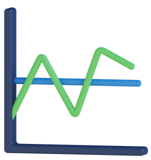
- Our study focuses on 3 runtimes: **runc** (Docker), **runsc** (gVisor) and **crun**.
- gVisor's container runtime engine, runsc, implements it own application kernel (Sentry) and file system (tempfs).
  - Pro: Improved isolation between container runs.
  - Con: Increased time for disk I/O and system calls.
- Previous work found that gVisor was at least 2.2x time slower at making system calls, and 11x slower at reading small files as compared to Docker.[3]
  - This performance loss is because gVisor's architecture has considerable duplication of functionality in the Sentry and tempfs.
- crun was developed in C to improve on the efficiency of Golang based container runtimes (like runc and runsc). [4][5]
  - As a result, crun's compiled binaries are 50 times smaller than runc.

[3] Ethan G. Young, Pengfei Zhu, Tyler Caraza-Harter, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2019. The True Cost of Containing: A gVisor Case Study. In 11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19). USENIX, Renton, WA.
[4] Giuseppe Scrivano Dan Walsh, Valentin Rothberg. 2020. An introduction to crun, a fast and low-memory footprint container runtime. https://www.redhat.com/sysadmin/introduction-crun.
[5] Redhat. 2023. crun Source Code. https://github.com/containers/crun.

Outline

- Background and Motivation
- Research Question
- Methodology
- Results
- Conclusions

Research Question - Performance Isolation

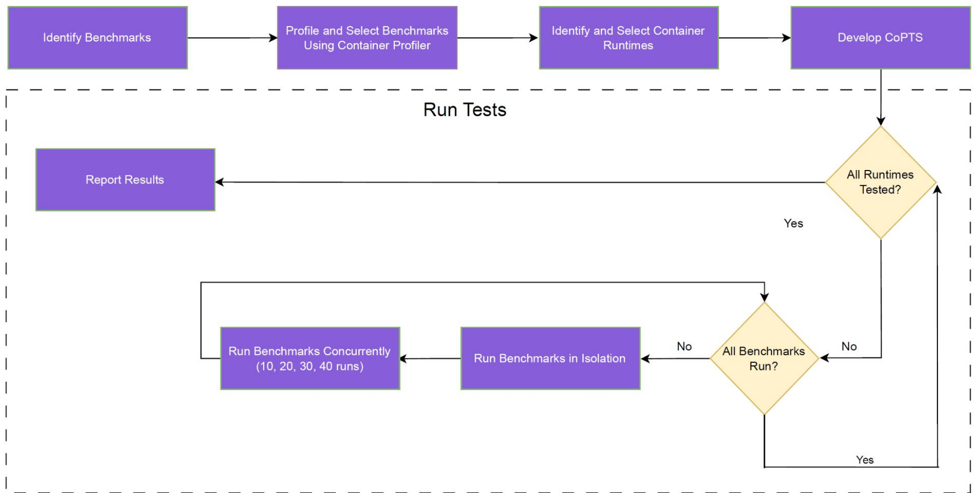


- What is the degree of performance isolation provided by current container runtimes?
  - Do some runtimes provide better isolation when containers compete for identical resources simultaneously (e.g., CPU, memory)?

Outline

- Background and Motivation
- Research Question
- Methodology
- Results
- Conclusions

Worfklow



Benchmark Profiling w/ Container Profiler

- To choose a complementary set of benchmarks, we first profiled the resource utilization of popular system benchmarks using the **Container Profiler**.
- The Container Profiler is a Linux-based tool that enables resource utilization profiling of scripts and container-based tasks.[6][7]
- It collects metrics related to CPU, memory, disk, and network utilization at the VM, container, and process levels.

[6] Hoang, V., Hung, L.H., Perez, D., Deng, H., Schooley, R., Arumilli, N., Yeung, K.Y., Lloyd, W., Container Profiler: Profiling Resource Utilization of Containerized Big Data Pipelines, GigaScience, Volume 12, (August) 2023, giad069.
[7] https://github.com/wlloyduw/ContainerProfiler

Container Parallel Test Suite (CoPTS)

- We next implemented CoPTS
- CoPTS uses Bash and Python scripts to orchestrate benchmark runs across container runtimes in parallel. [8]
- The configuration options are:
  - Choice of benchmarks: Bonnie++, Linpack, Noploop, Stream, Sysbench, Unixbench, and Y-Cruncher
  - Choice of runtime: Supported Runtimes: runc (Docker), runsc (gVisor), runnc (Nabla) and crun runtime
  - Test Configurations: “x y z”, where,
    - ‘x’ - number of processes to create
    - ‘y’ - number of containers to launch sequentially
    - ‘z’ - number of benchmark runs per container
- CoPTS outputs aggregated benchmarks results in a tabular format.

[8] https://github.com/namanbhaia/ContainerParallelTestSuite

Container Runtimes

- Container runtimes were selected based on their inherent differences, adoption in industry, and state of active maintenance.
- runc (Docker)
- runsc (gVisor)
- crun

- When reviewing container runtimes, we rejected Kata 1.0, Nabla, and RKT as they are no longer supported.

Benchmarks

For this paper we ran the following benchmarks using CoPTS, first in isolation, and then with 10, 20, 30 and 40 parallel runs.

Benchmark	Benchmark Configuration	Resource Tested
Linpack	Matrix size: 600x600	CPU
Noploop	6 Billion NOP instructions	
Sysbench-CPU	20 million prime numbers	
Stream	Array has 10 million elements	Memory
Sysbench-Memory	100GB written in 1KB blocks	
Y-Cruncher	100 million digits of pi	

Each container instance was allotted 2 cores and 4GB of memory.

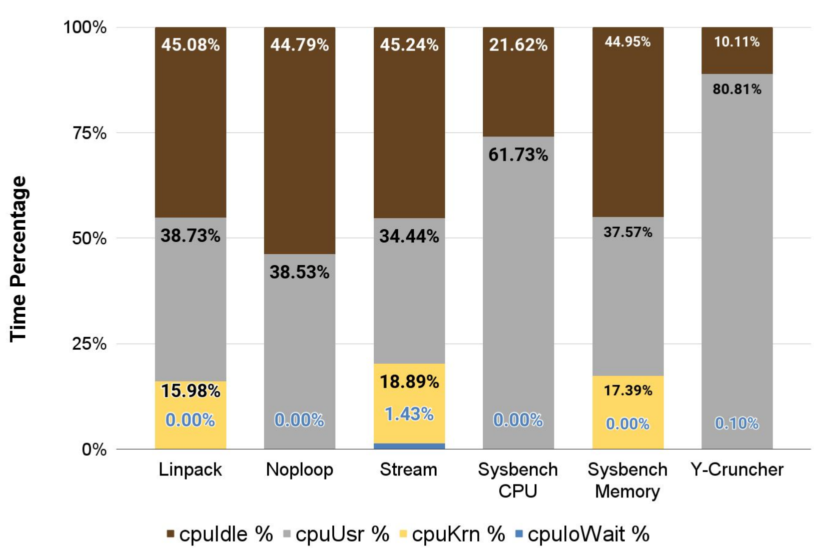
EC2 Configurations

- Initial measurements were taken on a t2.micro instance:
  - 1 vCPU
  - 1 GiB of memory
  - Intel Xeon Scalable
  - 3.3 GHz CPU Clock Speed
- Final measurements, we used a c5d.metal instance
  - 96 vCPUs
  - 192.0 GiB of memory
  - Intel Xeon Platinum 8275CL
  - 3 GHz CPU clock speed

Outline

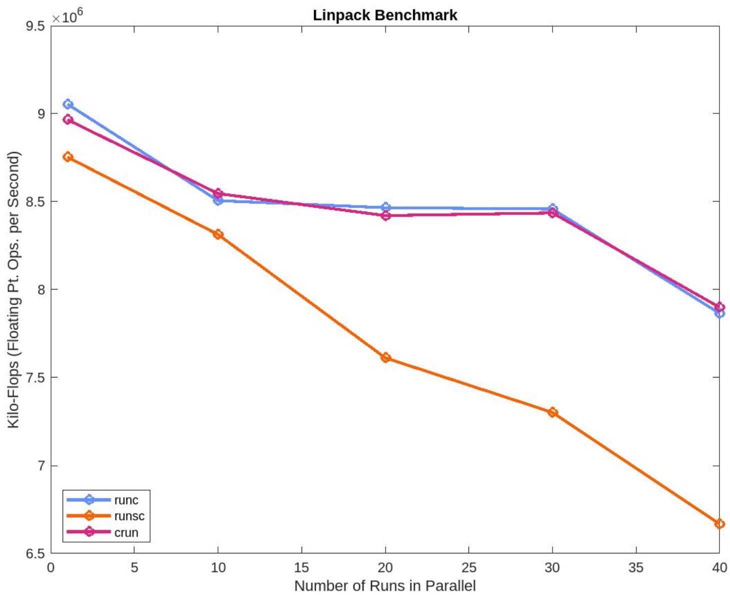
- Background and Motivation
- Research Question
- Methodology
- Results
- Conclusions

Experiment-0: Benchmark Linux CPU Metrics



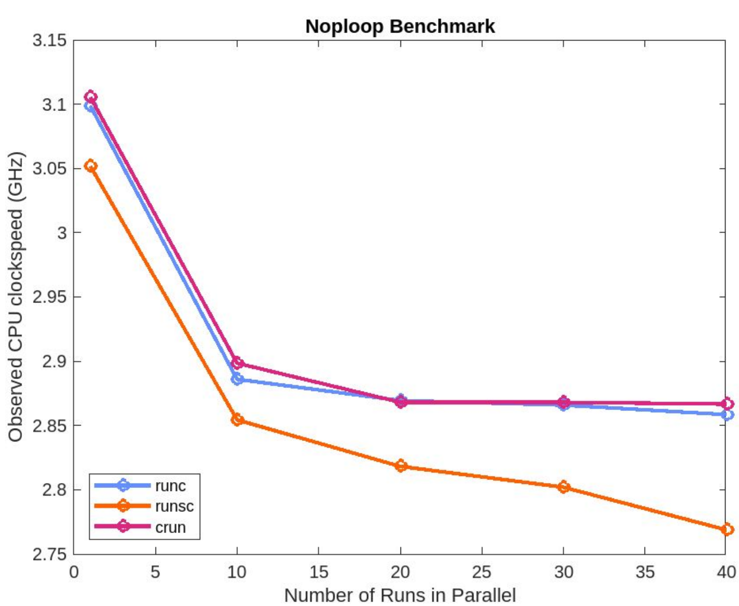
All benchmarks were profiled using Container Profiler on a c5.xLarge AWS EC2 instance with 4 vCPUs and 8 GiB RAM.

Experiment-1: CPU Benchmark - Linpack



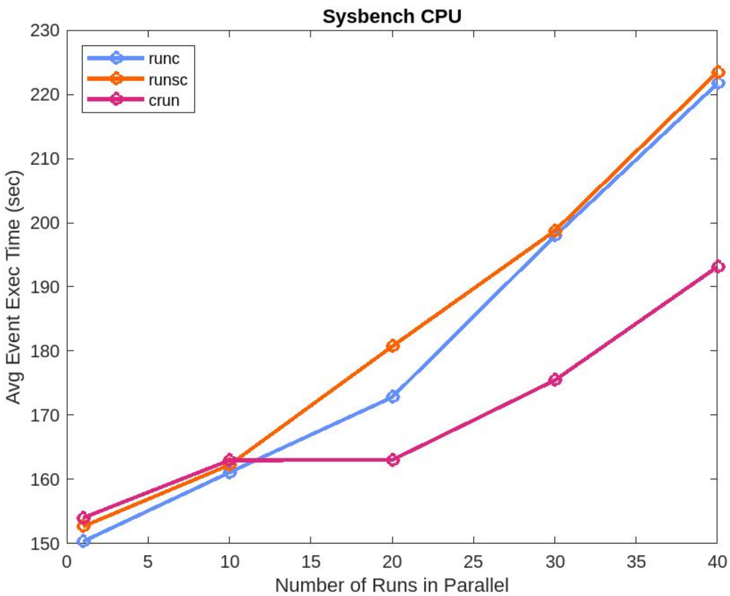
- runsc's performance loss at 40 concurrent runs vs. 1 isolated run was 2x greater than runc.
- Throughput and performance loss was almost identical for runc and crun.
- For Linpack we observed the following ordering of container CPU isolation: crun > runc > runsc.

Experiment-1: CPU Benchmark - Noploop



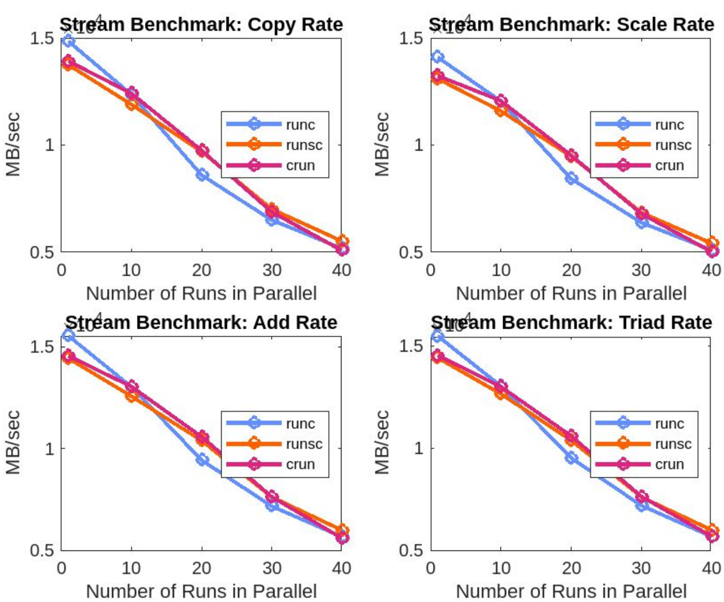
- Noploop performance and degradation were similar across all container runtimes.
- No inference on which container runtimes provided better CPU isolation.

Experiment-1: CPU Benchmark - Sysbench CPU



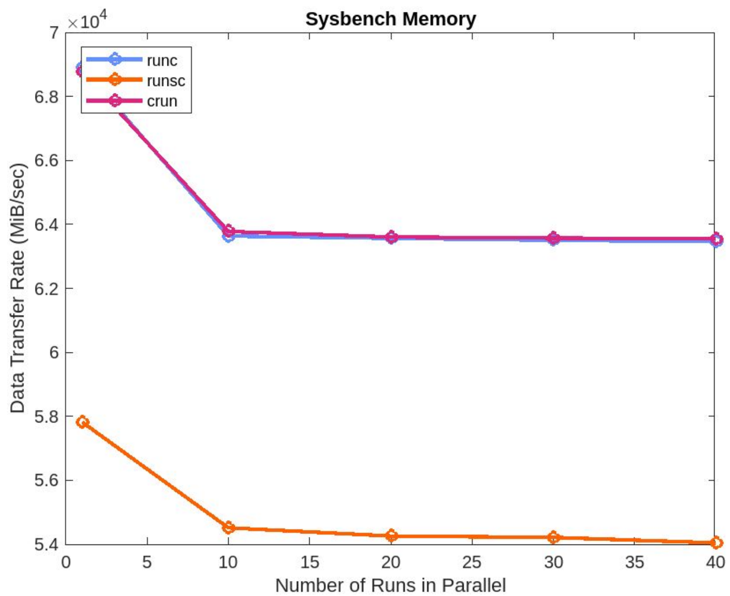
- For an isolated run, runsc and crun performed poorly compared to runc.
- However, for 40 concurrent runs, crun outperformed runsc and runc.
- For Sysbench CPU, we infer the following order for CPU isolation: crun > runc > runsc.

Experiment-2: Memory Benchmark - Stream



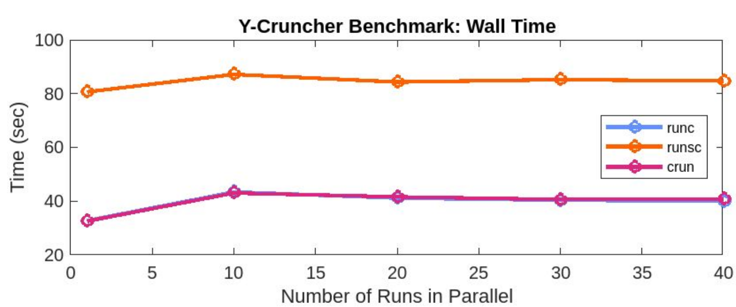
- runsc performed poorly vs. runc and crun when scaling up the number of concurrent runs.
- runsc's performance loss with 40 concurrent runs vs. 1 isolated run, however, was less than runc.
- For Stream we infer the following order for memory isolation: runsc > crun > runc.

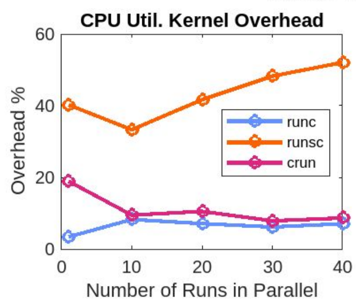
Experiment-2: Memory Benchmark - Sysbench Memory

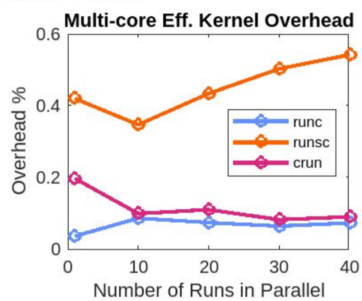


- Runsc performed poorly compared to runc and crun across all configurations.
- Runsc, however, had slightly less performance loss at 40 concurrent runs vs. runc and crun.
- We observed the following order of memory isolation: runsc > crun > runc.

Experiment-2: Memory Benchmark - Y-Cruncher







- runsc performed twice as poorly as runc and crun.
- Performance loss when scaling up to 40 concurrent runs was nearly five times less than runc and crun for runsc.
- For y-cruncher we infer the following order for memory isolation: runsc > runc > crun

Result Summary: Benchmark Performance

Benchmark	Resource and Metric	Performance Loss comparing 1 vs. 40 Parallel Runs (%)		
		runc (Docker)	runsc (gVisor)	crun
Linpack	CPU (KFLOPs)	~13	~24	~12
Noploop	CPU Clock Speed (Ghz)	~8	~7.5	~7.9
Sysbench CPU	CPU (Events/sec)	~51	~50	~25
Stream	Memory (MB/sec)	~65	~60	~63
Sysbench Memory	Memory (Mb/sec)	~8	~6	~7
Y-Cruncher	Memory perf. (sec) & overhead (%)	~22	~5	~25
Average	-	27.833%	25.4166%	23.316%

Outline

- Background and Motivation
- Research Question
- Methodology
- Results
- Conclusions

Conclusion Summary

- runsc's CPU and Memory performance was consistently poorer than runc and crun.
- crun and runc performance was mostly similar with crun outperforming runc marginally.

- crun had less performance degradation compared to runc for all benchmarks except y-cruncher.
- runsc provided better isolation only for memory benchmarks but crun and runc offered better CPU isolation.

Thank You!

This research has been supported by AWS Cloud Credits for Research.