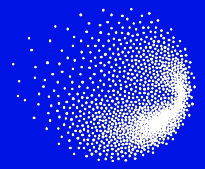


PSI

Efficient and fast container execution using image snapshotters

Max Fatouros, C. Lange, D. Feichtinger (PSI),
A. Thundiyil, V. Völkl, J. Blomer (CERN)
CernVM Workshop, 16th September 2024

Motivation

The use of containerised workloads is continuously increasing

- CMS centralised data processing has been using software containers for years
- Change of LXPLUS/LXBATCH from CentOS7 to AlmaLinux9 forced large number of physicists to use container images so that they can continue to run their analyses

Particle physics workloads nowadays largely run using Singularity/Apptainer

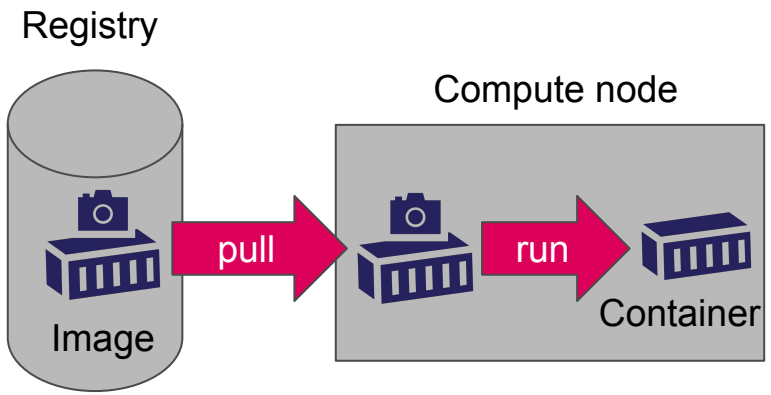
Idea: use **fully OCI-compatible container engines** such as containerd

Container image distribution

Typical particle physics analysis workloads consist of hundreds to thousands of batch jobs

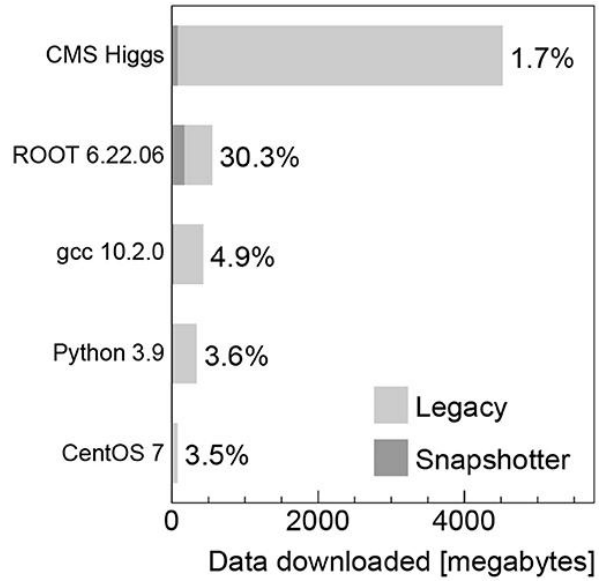
Historically, container images would have to be pulled (=downloaded and extracted) from the registry before each job can start

→ Huge load on container registries



Furthermore, only small fraction of container image content actually used/needed

Need to distribute/pull images "lazily",
i.e. download what's needed when it's needed



[Frontiers in Big Data Vol. 4 (2021) 673163]

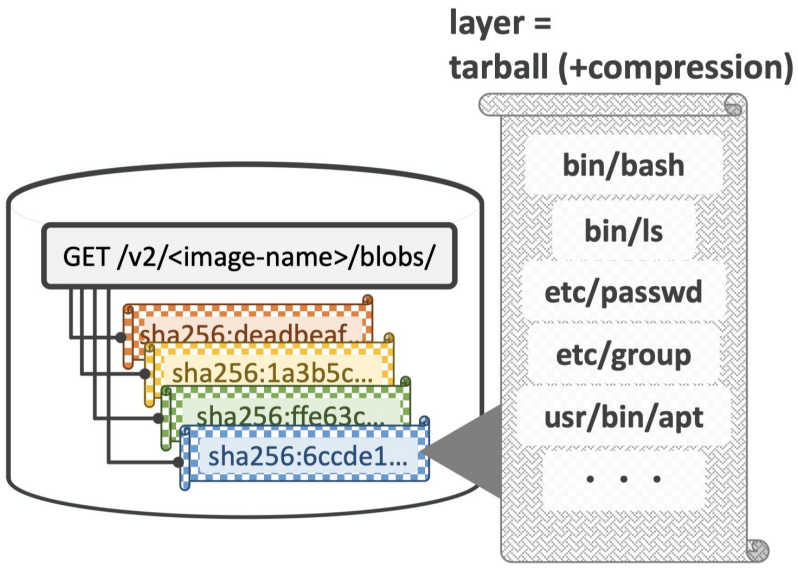
Lazy-pulling of images

PSI

Idea: pull only what is needed

Container reminder:

- A container is a set of tar-balls plus a manifest (list)
- Downloading and extracting the layers builds the container file system

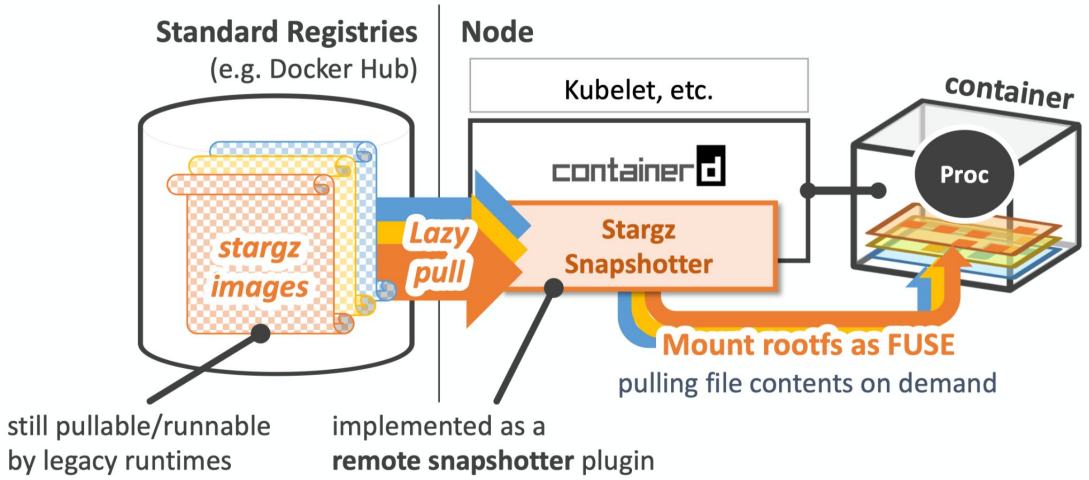


[K. Tokunaga]

Lazy pulling mounts (rootfs snapshots as FUSE and downloads) accessed file contents on-demand

→ Can start container almost immediately

→ Can be slower during execution



Implementations of lazy pulling

PSI

Solutions are implemented as so-called **image snapshotters** for use with containerd

Evaluated tools:

stargz snapshotter: use images in searchable tar.gz format

soci snapshotter: add separate index artifact to image (hosted in registry)

CVMFS snapshotter: use unpacked images on CVMFS

Overlayfs snapshotter: the default, non-lazy-loading snapshotter.

All snapshotters will fallback to legacy pulling if image (or layer) not available in required format

- Mind: this is something Apptainer cannot do

Benchmarking approach

Use typical particle physics tasks and images, e.g.:

- ROOT
- Python

Using nerdctl to run workloads with the various snapshotters:

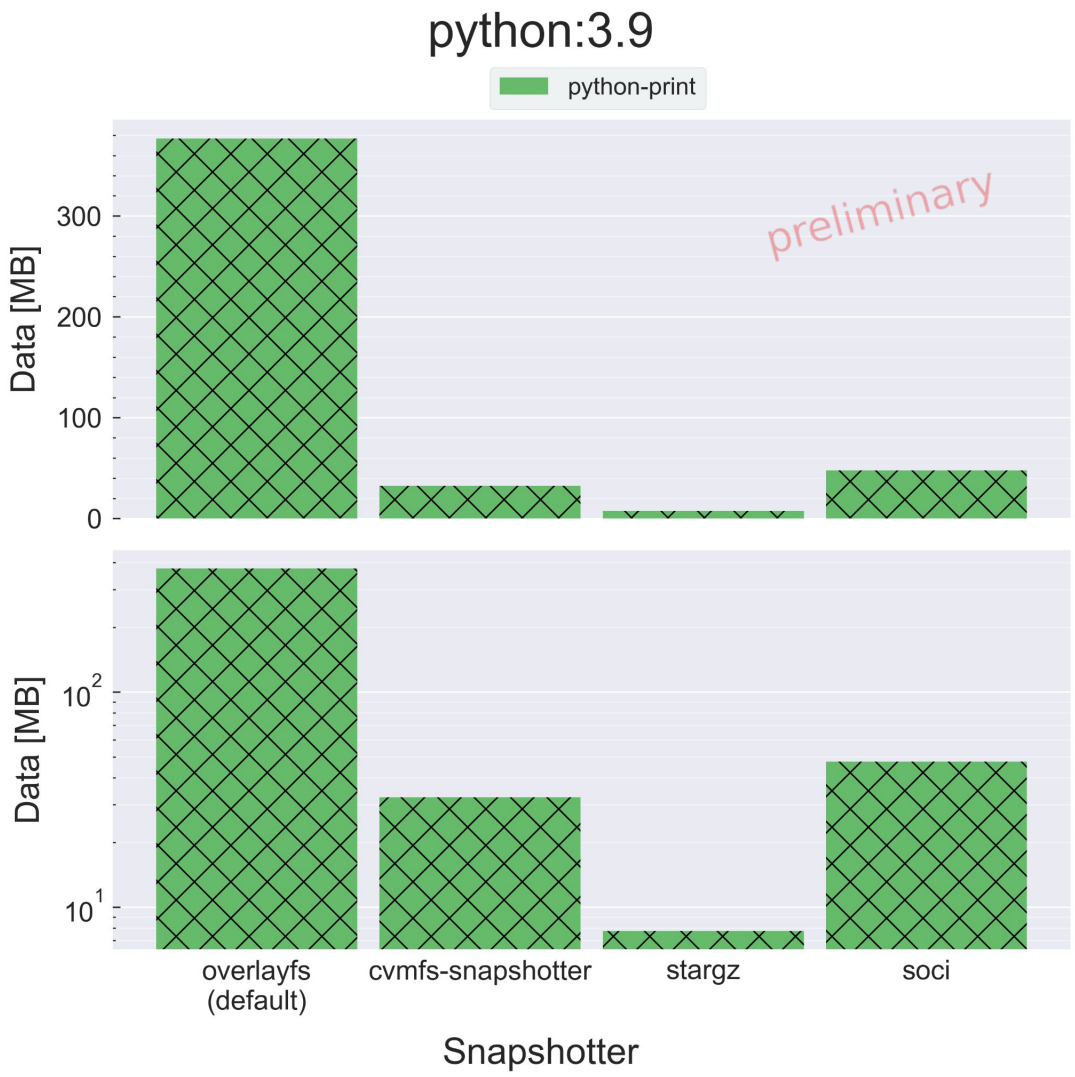
- Parse execution log files to extract timestamps
- Monitor traffic using network monitoring tools
- Repeat process several times, clear cache in between runs

Also compare to “legacy” approach pulling entire image before execution

Using local PC in connection with both a local (same machine) registry, and Harbor registry (in the same network).

Using squid proxy for CVMFS caching (with images pre-cached)

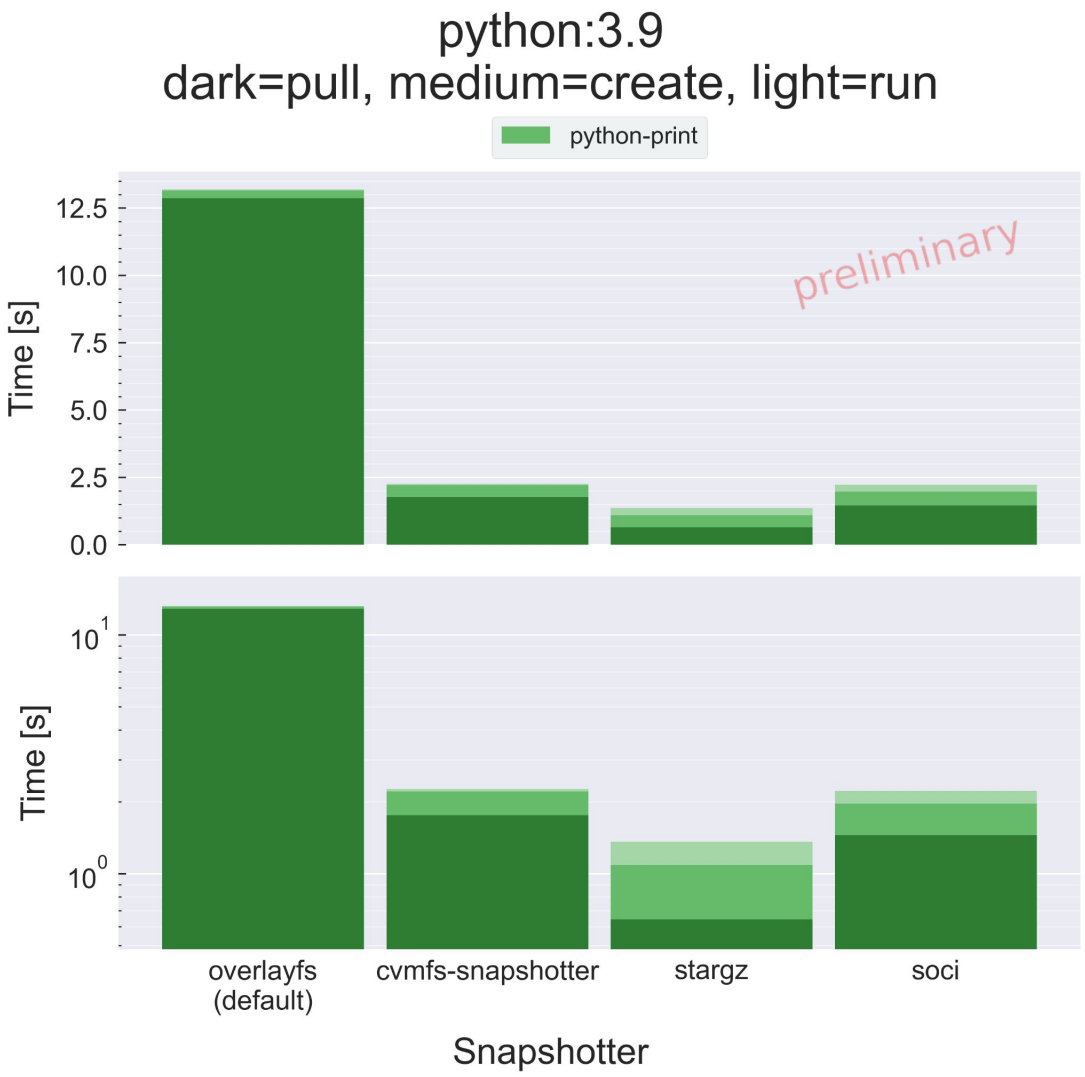
Results (data)



- Significant improvements for all three snapshotters.
- Overlayfs results verified by comparing to image side.
- CVMFS results verified by a CVMFS command (cvmfs_talk).

Runs **`print(...)`** in python

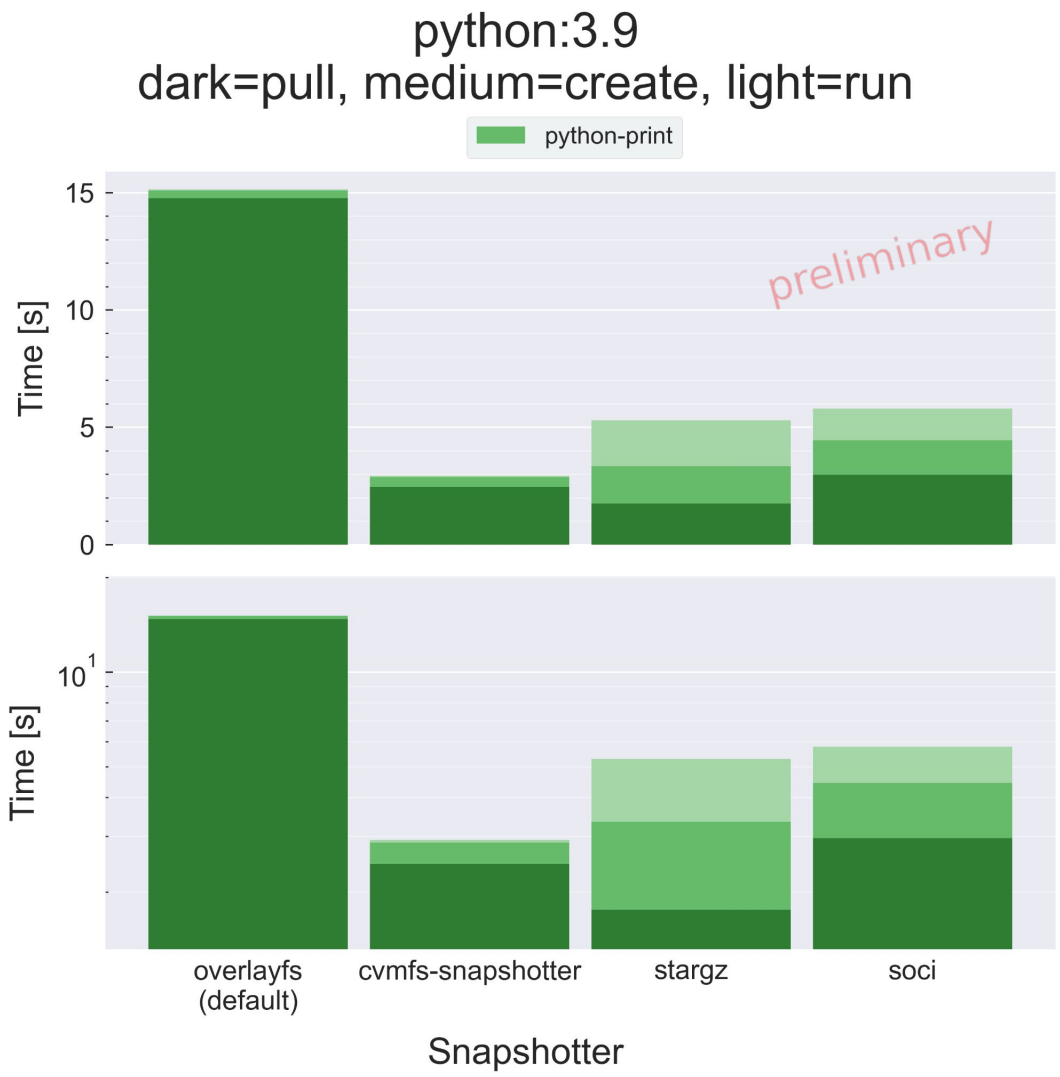
Results (time, registry on same machine)



- Similar outcomes to the data-usage
- Stargz and SOCI relying on local registry
- CVMFS still gets most of its data from the proxy server (off-machine).

Runs **`print(...)`** in python

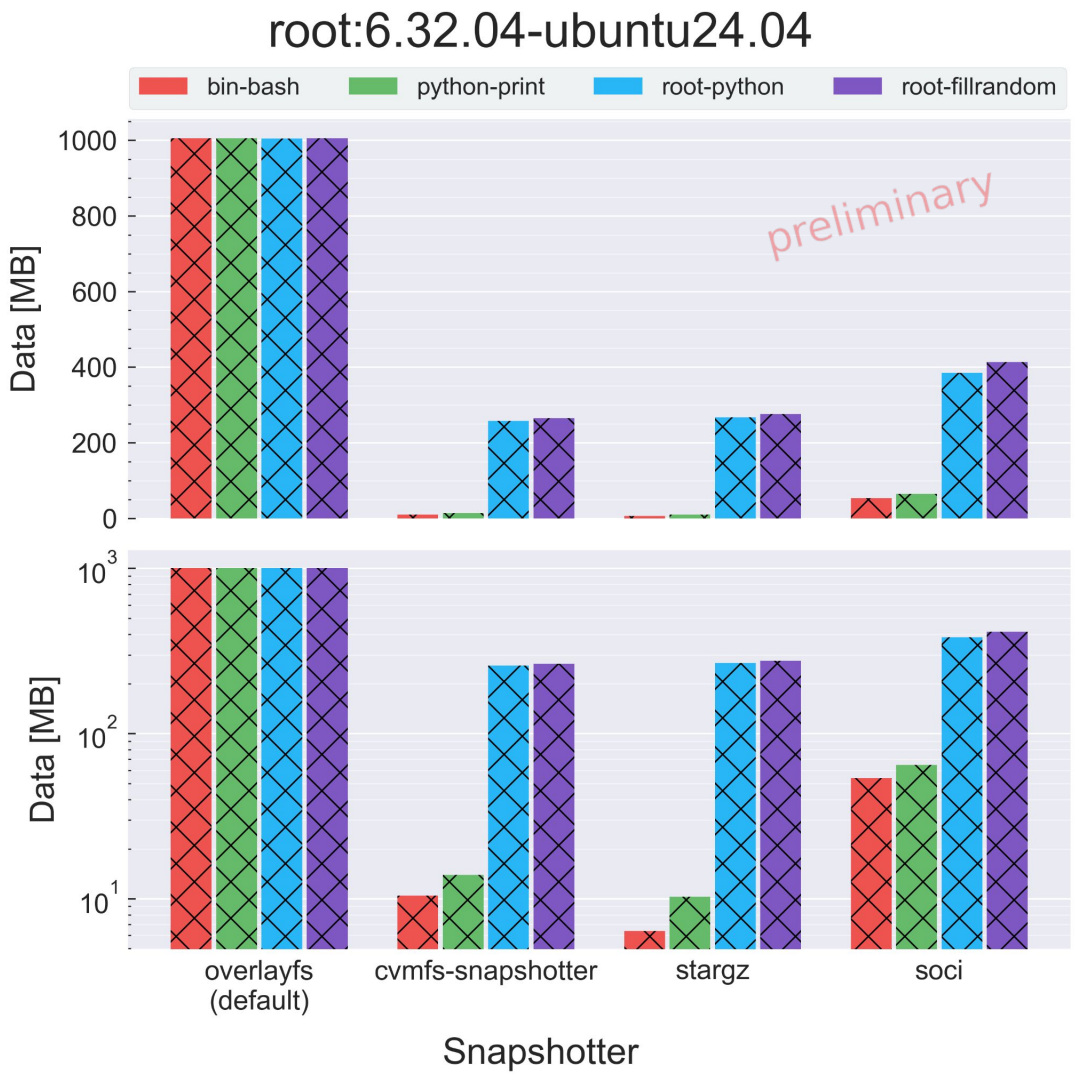
Results (time, registry on different machine)



- CVMFS performs just as well.
- With all three snapshotters pulling from images off-machine, CVMFS performs best.

Runs **`print(...)`** in python

Results (data)



- Simple scripts require much less data to run.
- CVMFS-snapshotter performs best for more complex scripts.
- Stargs performs best for simpler scripts.

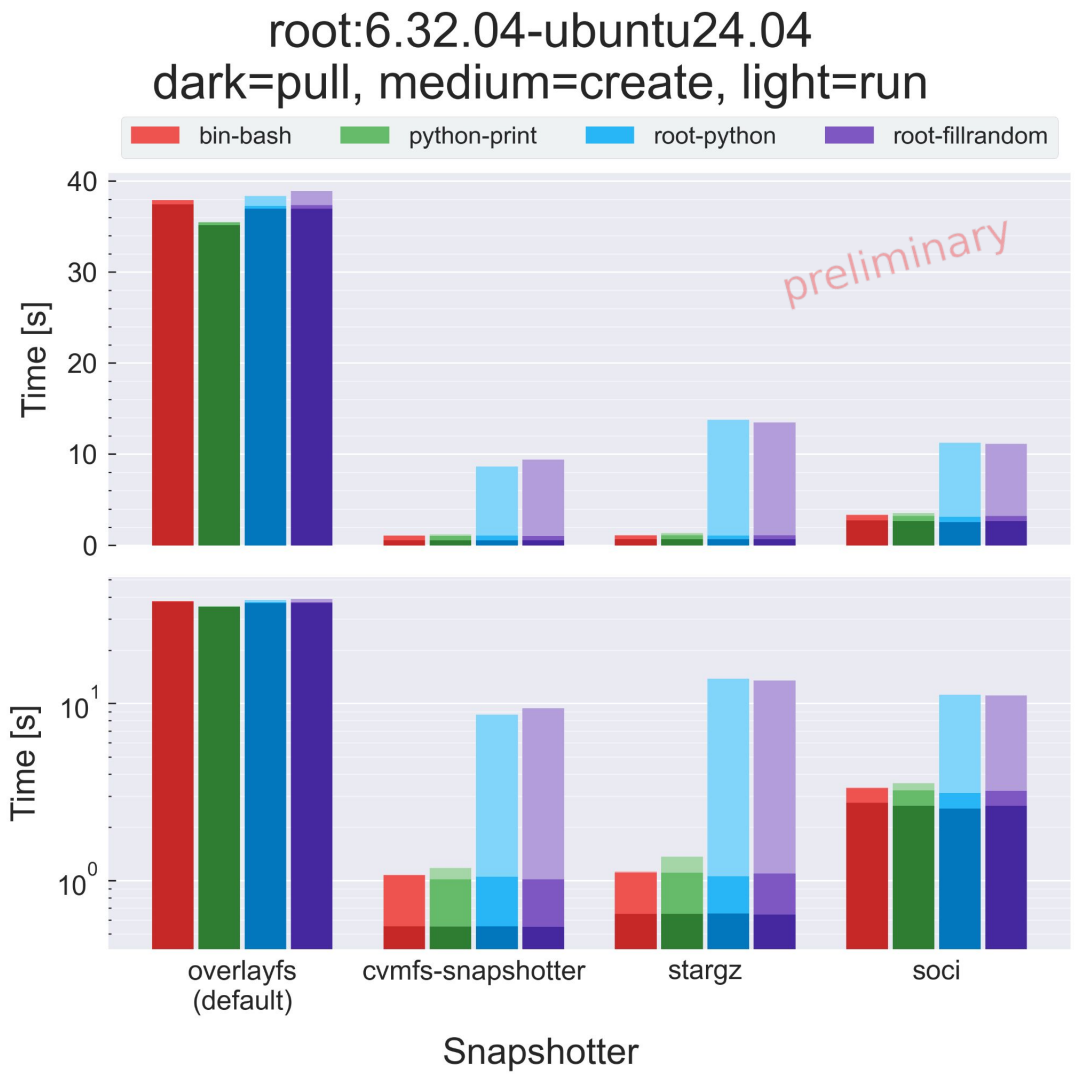
Runs **/bin/bash**

Runs **print(...)** in python

Runs **import ROOT** in python

Runs **fillrandom.py** (ROOT tutorial)

Results (time, registry on same machine)



- CVMFS leads, slightly, for all scripts.
- Startup times quick for all, even with larger scripts.
- Note: bash is being ran inside of bash, since we wrap the scripts in `date` commands to get times.

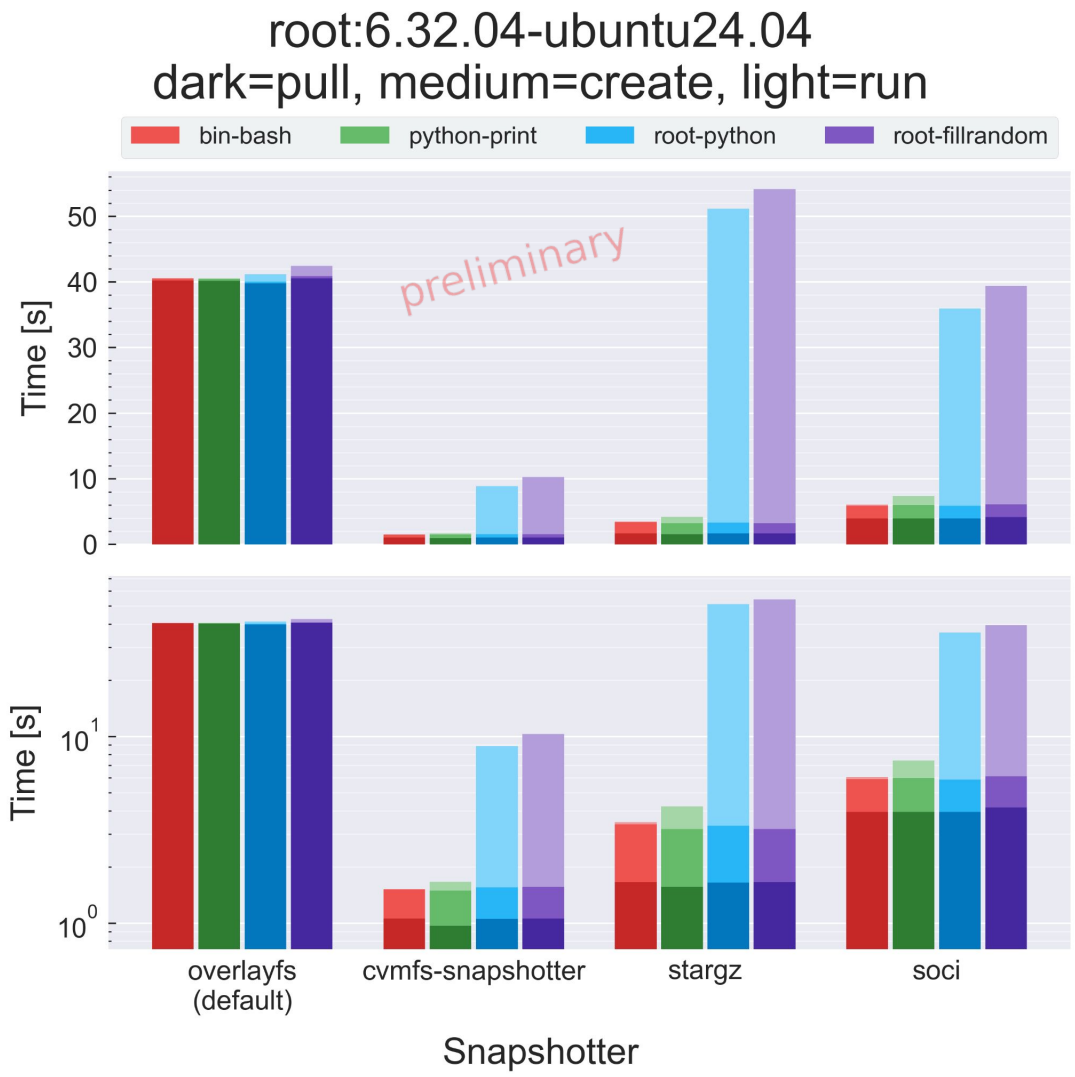
Runs **/bin/bash**

Runs **print(...)** in python

Runs **import ROOT** in python

Runs **fillrandom.py** (ROOT tutorial)

Results (time, registry on different machine)



- Strange behaviour for Stargz and SOCI?
- Startup times still much quicker.

Runs **/bin/bash**

Runs **print(...)** in python

Runs **import ROOT** in python

Runs **fillrandom.py** (ROOT tutorial)

Notes

Use of stargz snapshotter requires images to be converted (programatically) to a specific format → adoption might be slow/difficult

soci snapshotter only requires small addition to existing image—however, only certain registries support additional artifacts (Harbor ✅, GitLab ❌)

CVMFS snapshotter requires images to be “unpacked” → delay between building them and having them available (and they need to be added to the unpacker “sync” list)

Conclusions

Container image snapshotters open up new possibilities for image distribution and access

Significant bandwidth/data savings observed

Time saving depends on workload/image details

Overall, evaluated snapshotters all have advantages and disadvantages

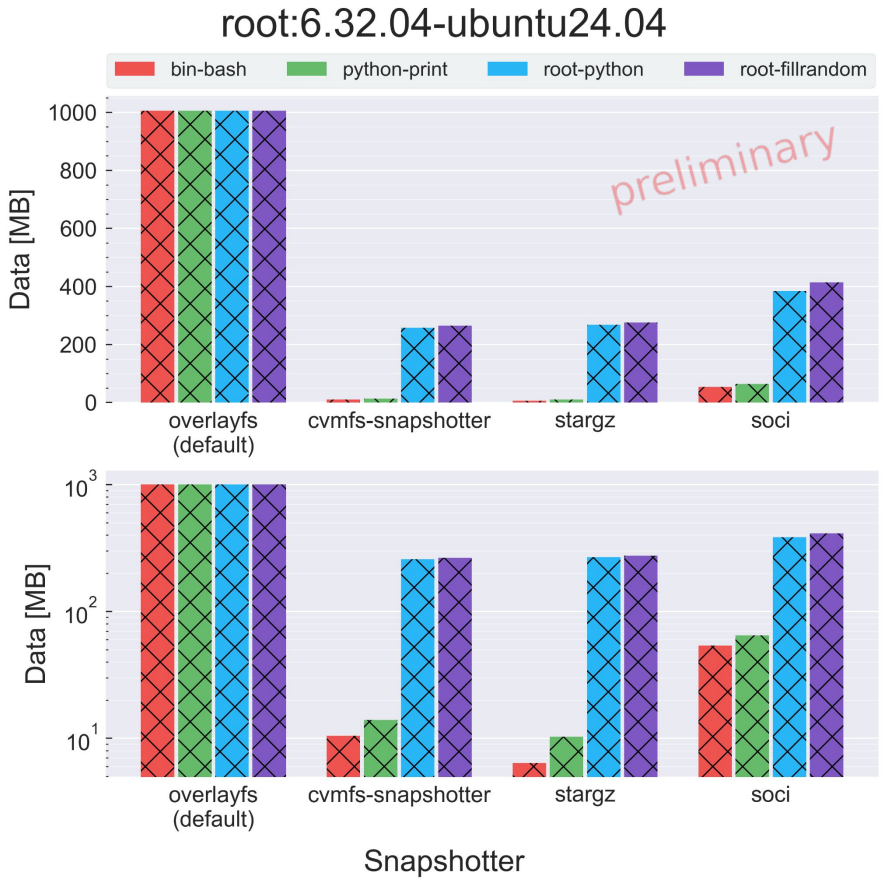
Performance similar, CVMFS snapshotter shows best overall performance so far for time-savings, and good results for data-savings.

Localhost vs Merlin registry data-usage

PSI

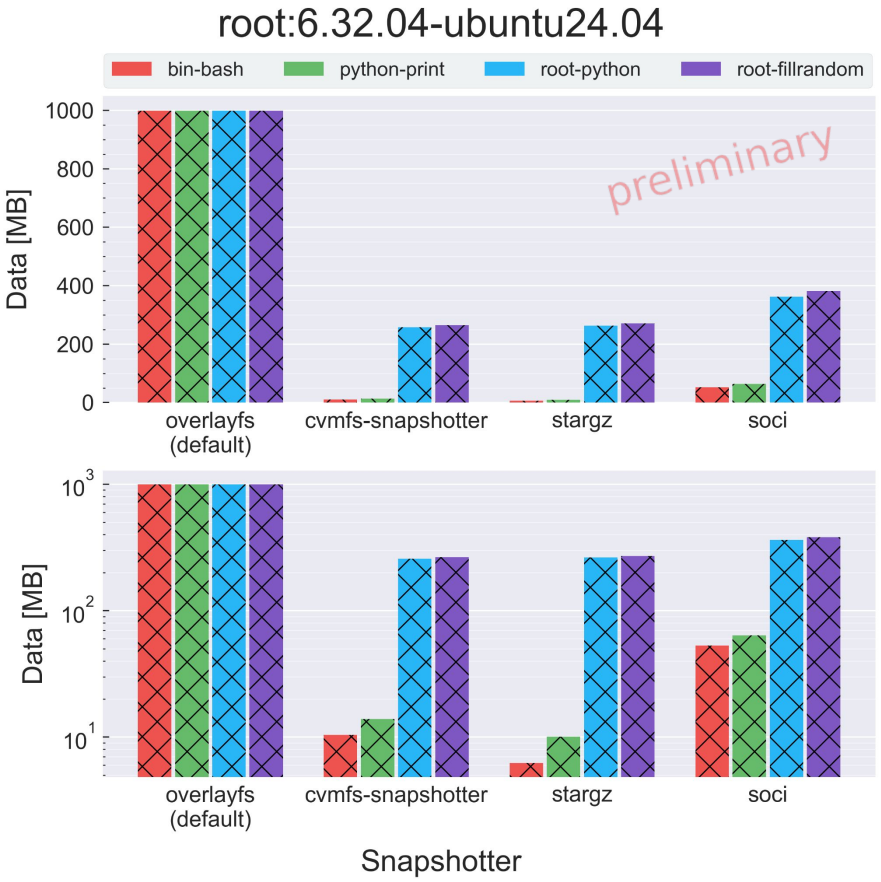
Harbor

Same network, different machine

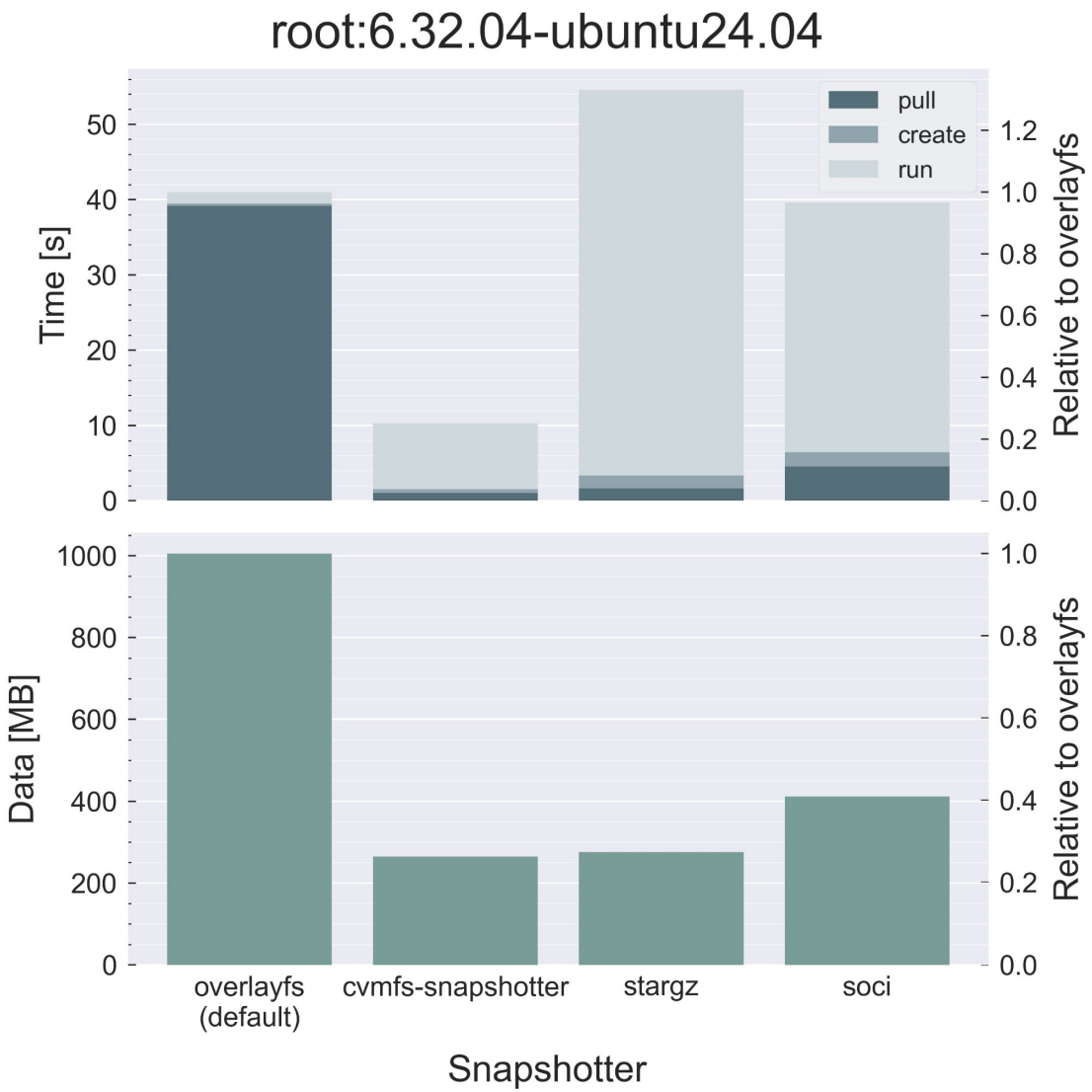


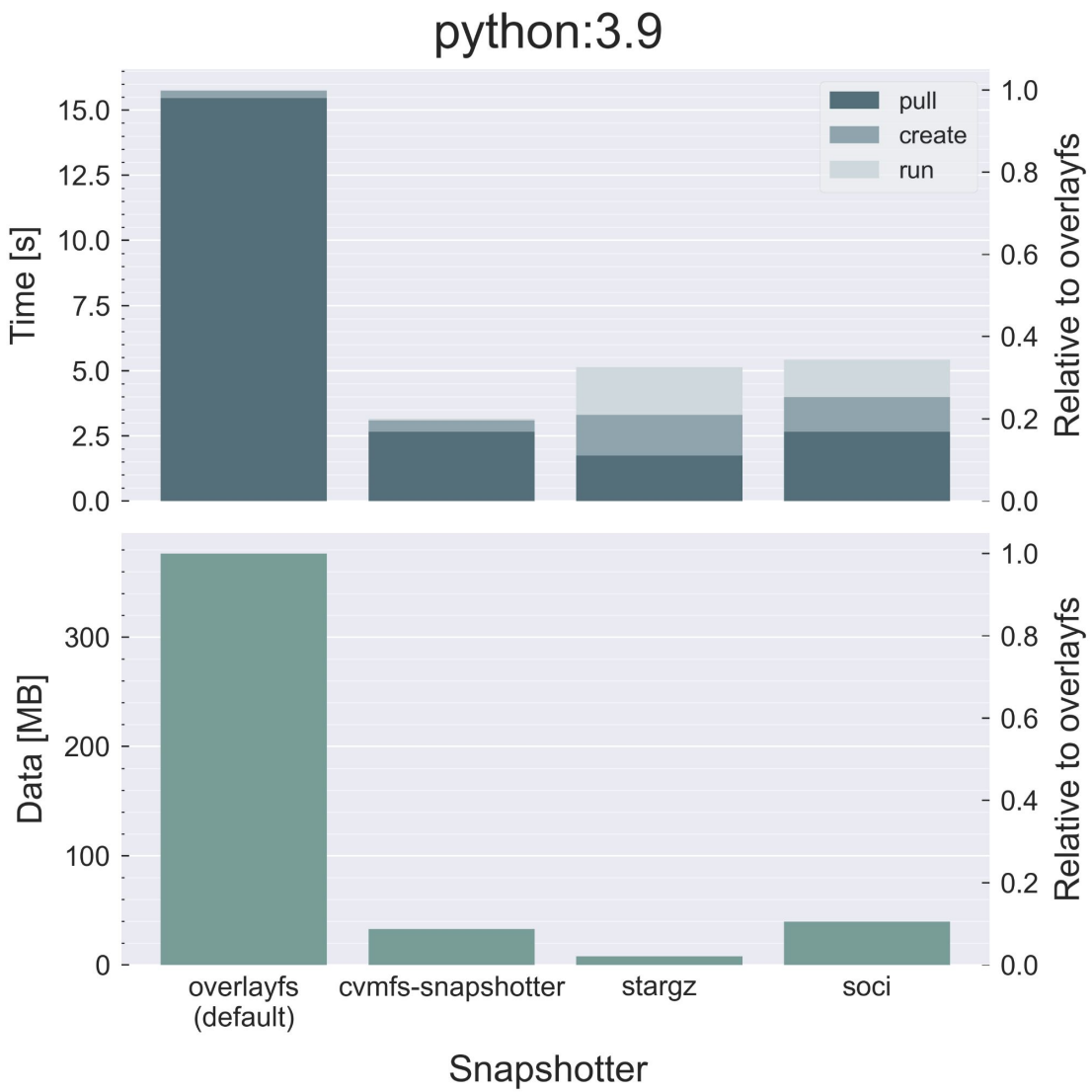
localhost

Same machine

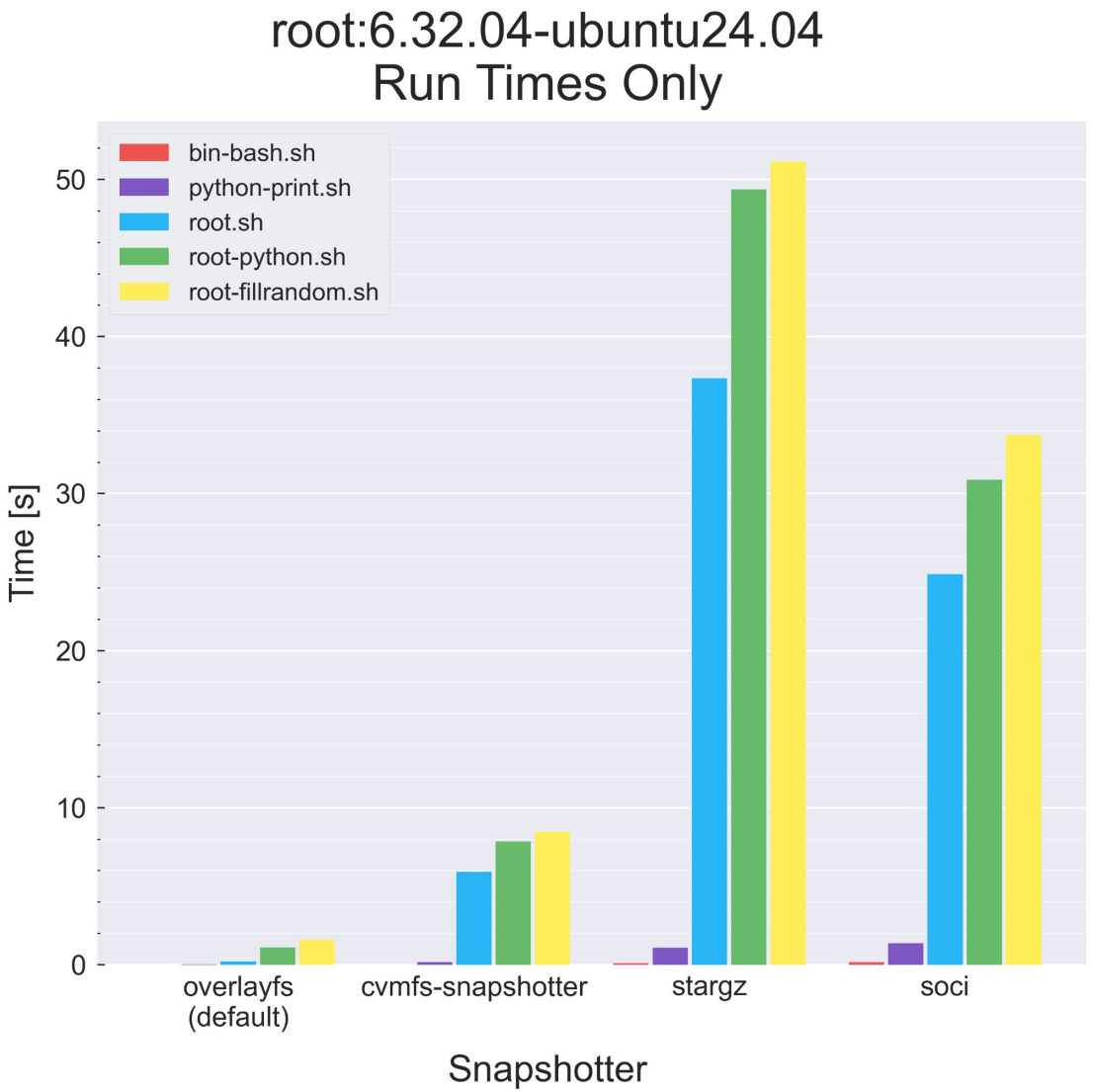


Main results





Run-time comparisons



Runs **/bin/bash**

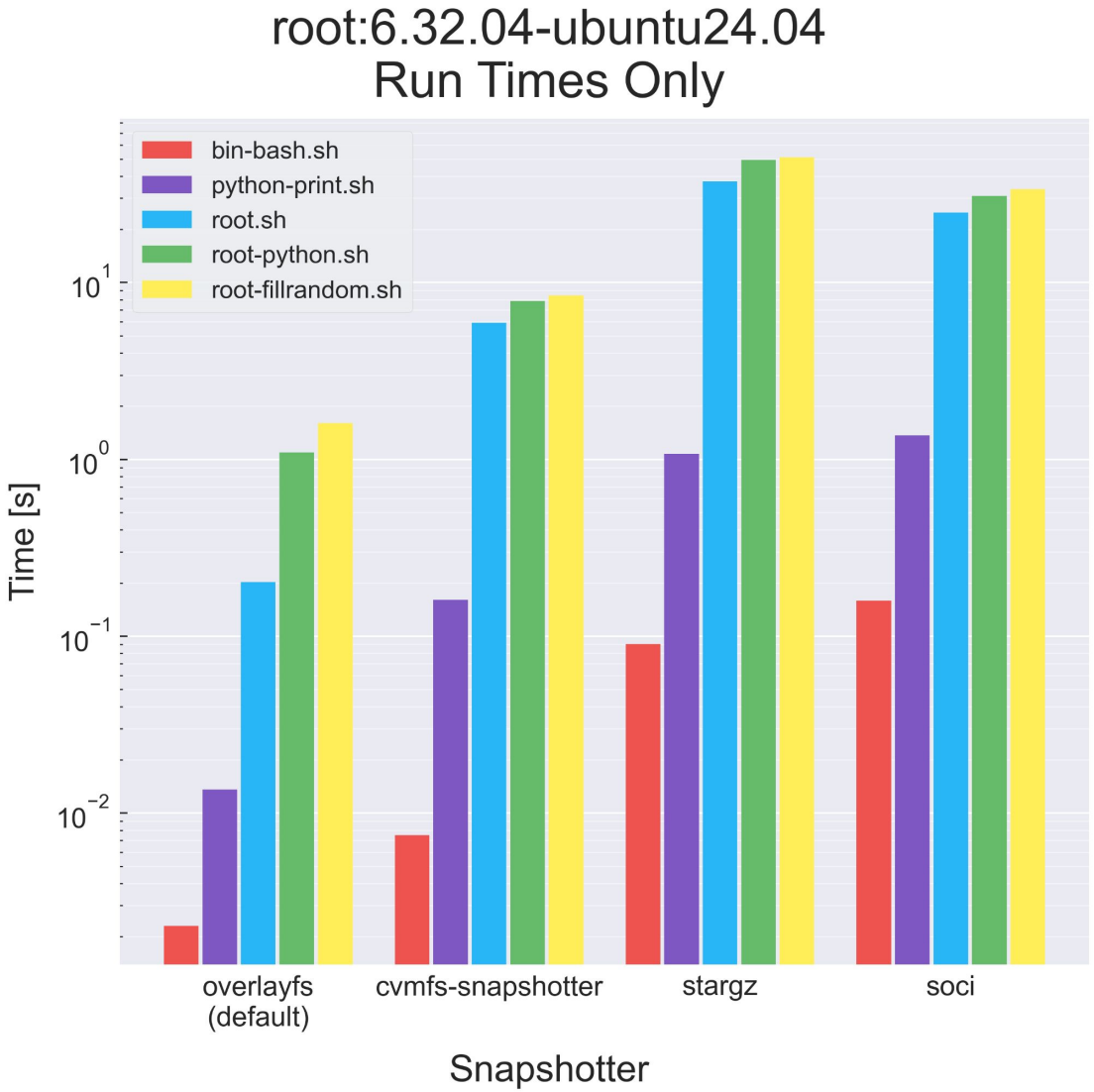
Runs **print(...)** in python

Runs **root**

Runs **import ROOT** in python

Runs **fillrandom.py** example that comes with ROOT.

Run-time comparisons (logarithmic)



Runs **/bin/bash**

Runs **print(...)** in python

Runs **root**

Runs **import ROOT** in python

Runs **fillrandom.py** example that comes with ROOT.