Enhancing Usability and Performance in Experimental Environments Management

Zahra Temori [1] Paul Marshall (Advisor) [2] Kate Keahey (Advisor) [2]

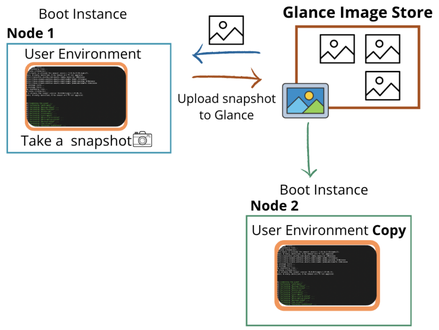
[1]University of Delaware [2]University of Chicago

Introduction and Project Overview

Reproducibility is essential in HPC:

- Allows experiments to be regenerated and supports collaboration
- Reduces effort of repeating resource-intensive work
- Provides a foundation for extending experiments

Snapshotting captures the complete state of a system so that environments can be rebuilt and restored automatically This project: improve snapshotting for HPC reproducibility by enhancing usability and performance, using the `cc-snapshot` tool on the Chameleon testbed.



Problem Statement and Goals

Reproducibility on HPC can be slow, difficult to maintain and cause errors due to intensive work and large environments.

Key problems of snapshotting tool that made it less practical for frequent use:

- **Usability:** Limited user controls, tightly coupled code, and lack of testing reduce flexibility and maintainability.
- **Performance:** Snapshot creation with **(QCOW2 + zlib)** is slow, especially for large HPC environments, limiting practical use.

Goal: To extend reproducibility on HPC environments we aimed to improve snapshotting by addressing these issues. This work shows the approach through two phases in snapshotting:

- **Phase 1 – Usability Enhancements:** Expand CLI options, modularize code, and integrate automated testing.
- **Phase 2 – Performance Optimization:** Compare compression algorithms **(zlib vs zstd)** and image formats **(QCOW2 vs RAW)** to lower creation times while maintaining reliability.

Phase 1: Usability Enhancements

To improve usability, we focused on both enhancing users and developers experiences.

Improved user control by adding more functionalities to CLI:

- **-u**, a flag that disables auto-update, user can choose when to update.
- **-d**, a flag for Dry-run mode to simulate runs with no side effects.
- **-s**, a flag to snapshot a custom source path for further testing.

Improved developer experience to ease maintenance & increase reliability:

- Modularized code into 5 different functions to ease future developments.
- Added automated tests via GitHub Actions to capture errors.
- Added tests for CLI options and tested check image size.

Phase 2: Performance Optimization

This phase addresses performance bottlenecks in the snapshotting process, which is slow with large environments. The current snapshotting uses (**QCOW2 + zlib**) which results in slow **creation time** and **boot times**. To improve snapshotting efficiency, we benchmark alternative **image formats** and **compression methods** to identify what method results in a faster, more time-saving approach.

Benchmarking Compression and Image Formats

To recognize the most time efficient configuration for **cc-snapshot**, we benchmarked three different combinations:

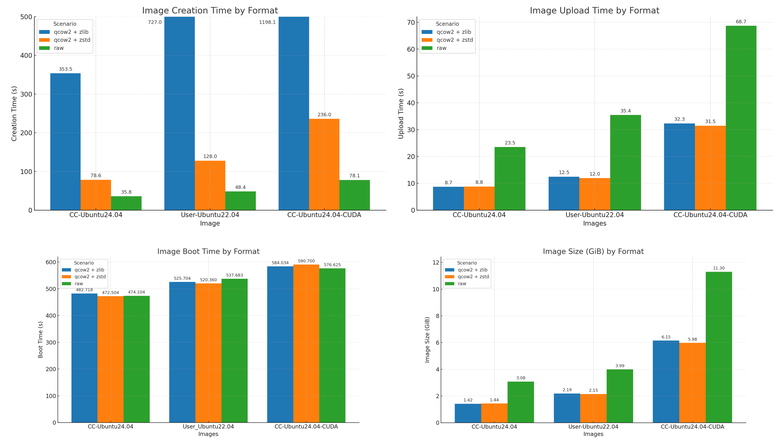
Snapshot were run on three images to represent small, medium, and large environments:

- **CC-Ubuntu24.04** – 4.47 GiB (small, baseline test).
- **User-Ubuntu22.04** – 7.62 GiB (medium, user-created).
- **CC-Ubuntu24.04-CUDA** – 12.7 GiB (large, CUDA-enabled).

Metrics Collected: Key stages that affect performance and user experience.

- **Creation Time** - produce snapshot image
- **Upload Time** - transfer image to storage
- **Boot Time** - launch instance from image
- **Final Image Size** - affects creation time and storage usage

Preliminary Findings



Format	Creation Time (s)	Upload Time (s)	Boot Time (s)	Image Size (GiB)
QCOW2 + zlib	759.52	17.83	530.82	3.25
QCOW2 + zstd	147.53	17.41	527.85	3.19
RAW	54.08	42.53	529.47	6.12

Table 1. Total time taken from experiments on three different image sizes using different compression methods.

Format	Creation Time	Upload Time	Boot Time	Image Size
ZSTD vs ZLIB	80.6%	2.36%	0.56%	1.85%
RAW vs ZLIB	92.9%	-138.5%	0.25%	-88.3%

Table 2. The percentage of improvement in performance and size compared to the current default (QCOW2 + zlib). Note: that the negative sign shows slower percentage and larger image size

Performance Summary

- **Creation Time:** **zstd** performed ~80% faster than **zlib**; **raw** is even faster but with the cost of size.
- **Upload Time:** **zlib** and **zstd** performed nearly the same; **raw** uploads ~140% slower due to the large file size.
- **Boot Time:** All formats show similar boot performance.
- **Image Size:** **zstd** produces images ~2% smaller than **zlib**; **raw** images are ~88% larger than zlib.

Conclusion

- **Summary:** This project improved **snapshotting** in two key areas, usability and performance.
  In Phase 1, we expanded user control, modularized code, and automated testing that made the tool easier and faster to use, maintain, and extend in the future.
  In Phase 2, we benchmarked and proved that zstd compression reduces snapshot creation time by ~80% compared to the current zlib default, however maintaining similar upload and boot performance.
- **Impact:** These changes make snapshotting faster, more usable, and more reliable. It will help researchers to capture, package and share HPC and computing environments efficiently and reduce barriers to reproducibility.

Acknowledgments

The results of this poster were obtained on the Chameleon Testbed funded by the National Science Foundation (Award No. 2431425).

Resources

https://chameleoncloud.org
https://reproduciblehpc.org