EPJ Web of Conferences **295**, 07046 (2024) https://doi.org/10.1051/epjconf/202429507046
CHEP 2023

CRIU - Checkpoint Restore in Userspace for computational simulations and scientific applications

Fabio Andrijauskas[1,*], *Igor* Sfiligoi[1,**], *Diego* Davila[1,***], *Aashay* Arora[1,****], *Jonathan* Guiang[1,†], *Brian* Bockelman[2,‡], *Greg* Thain[2,§], and *Frank* Würthwein[1,¶]

[1]SDSC - UC San Diego, MC 0505 | 9500 Gilman Drive | La Jolla, CA 92093-0505
[2]Morgridge Institute for Research | 330 N Orchard Street Madison WI

Abstract. Creating new materials, discovering new drugs, and simulating systems are essential processes for research and innovation and require substantial computational power. While many applications can be split into many smaller independent tasks, some cannot and may take hours or weeks to run to completion. To better manage those longer-running jobs, it would be desirable to stop them at any arbitrary point in time and later continue their computation on another compute resource; this is usually referred to as checkpointing. While some applications can manage checkpointing programmatically, it would be preferable if the batch scheduling system could do that independently. This paper evaluates the feasibility of using CRIU (Checkpoint Restore in Userspace), an open-source tool for the GNU/Linux environments, emphasizing the OSG's OSPool HTCondor setup. CRIU allows checkpointing the process state into a disk image and can deal with both open files and established network connections seamlessly. Furthermore, it can checkpoint traditional Linux processes and containerized workloads. The functionality seems adequate for many scenarios supported in the OSPool. However, some limitations prevent it from being usable in all circumstances.

1 Introduction

While executing scientific applications, it can be required to stop the process due to hardware problems or even end-of-life job problems. In that case, some applications can create files to save their current state from being loaded on a restoring process later. However, most of the software does not have this kind of feature. Some applications can create checkpoints; however, this process of dumping and loading software is a complex task due to the number of systems to control and the software's ability to control all the use case scenarios. This is a "long dream" of high performance computing and high throughput computing. Even

*e-mail: fandrijauskas@ucsd.edu
**e-mail: isfiligoi@sdsc.edu
***e-mail: didavila@ucsd.edu
****e-mail: aaarora@ucsd.edu
†e-mail: jguiang@ucsd.edu
‡e-mail: bbockelman@morgridge.org
§e-mail: gthain@cs.wisc.edu
¶e-mail: fkw@ucsd.edu

more, saving and starting an application again can save money and time. These scientific applications have been used to discover new materials [2], find black holes [10], and many others.

OSG [6] provides a distributed high throughput computing environment where campus research organizations can use their resources. This federation includes computing, data, and storage resources. The combination of HTCondor and Glidein Workflow Management System (GlideinWMS [7]) provides access to computational resources. When more resources are required, HTCondor job execution daemons (aka glidein pilots) are submitted to the computer resources at the Glidein Factory (GF) sites; HTCondor and GlideinWMS are the base of OSPool. The OSPool is a computing capacity source accessible to any researcher affiliated with a US academic institution. Capacity is allocated following a Fair-Share policy.

CRIU (Checkpoint Restore in Userspace, pronounced kree-oo) is a tool for checkpointing and restoring applications in GNU/Linux environments [11, 9]. With CRIU, it is possible to stop an application, save the working memory on disk, and restore the state later. One project that can use the CRIU functionality is the OSPool. This work aims to create use cases to test CRIU on a high throughput and high performance computing environment and the OSPool to check how these use cases can take advantage of CRIU features.

2 Test setup

To test CRIU's features, we created a list of use cases related to computational simulations and scientific applications and used two CRIU versions. One was the most updated version (3.17.1), and the second was a branch with non-root operations support [5]. This approach was necessary because scientific applications typically use an unprivileged account. The procedure was simple: executing a *dumping* and *restoring* CRIU command and checking the software output. After the restoring process, the system and kernel logs are inspected. We also used a well-known scientific application to emulate more complex scenarios and each case addresses a unique situation. In simple code scenarios, the objective was to create a simple and easy code to be debugged if necessary.

The operation of dumping and loading software is very straightforward. To dump a software using a PID 9191: `criu dump -shell-job -t 9191`, to load an application is `criu -shell-job load .`. Due to the necessity to load files, control network connection, and others, sometimes CRIU requires more parameters.

3 Tests and results

The goal with the use cases was to cover simple scenarios, such as in basic C code, until more complex scenarios related to networks and others. All the use cases and the results are in https://path-cc.io/GIL/criu_checkpoint_restore_userspace/. Table 1 shows an overview of all tests using CRIU.

Tests	CRIU 3.17.1	CRIU Branch non-root
Simple serial application	Working	Working
Pthreading and forking	Working	Working
Applications with open files	Working	Working
Applications running in containers	Partially working	Partially working
Checkpointing while running inside a container runtime	Not working	Not working
CPU-specific optimizations	Working	Working
Applications using GPUs	Not working	Not working
Network applications	Partially working	Partially working
Network file system	Working	Working
Parallel application	Not working	Not working

Table 1. Each test group using CRIU and the overwall results.

Simple serial application

Some scientific applications are purely compute-heavy, e.g., Monte Carlo simulations. After reading their inputs and getting additional input arguments, e.g., random seeds, they keep computing and do not interact with the environment until the end, when the outputs are created. We create this scenario with a simple C program that computes π and writes the result to the terminal and other software for Molecular Dynamics. The first test, a "Simple C" test, shows a perfect CRIU execution using the standard version and the version with non-root capabilities. The other test used a serial LAMMPS opening a basic input of Lennard-Jones simulation [8]. CRIU shows that it is possible to run/dump/load using a LAMMPS serial version with both CRIU versions. These two results show that CRIU can be used in simple scenarios. However, there are more complex scenarios related to more compute-heavy processing.

Pthreading and forking

One critical scenario is the utilization of multiple processes or threads; these types of programming techniques are used in molecular dynamics and other types of simulations. CRIU supports checkpointing threads or forks. To test this, two C codes were used. The first software was a PThread code showing a sequence of numbers and creating four threads; the other program created one fork to "print" a sequence of numbers. Saving and loading applications with forks and threads was possible using both CRIU versions (root and non-root).

Applications with open files

Finding software that loads the input and writes the output in a file is possible. An example of these features can be found on LAMMPS. LAMMPS loads files, and the result is written on the disk. CRIU can load and unload an application that uses files. However, keeping the same file structure is required, meaning the "directory tree" should be created on the computer to restore the process. This could be complicated if it is not known a priori where the application writes the file.

Applications running in containers

Many applications rely on containerization these days for ease of portability and reproducibility. HTCondor can run user applications inside a container runtime, e.g., apptainer (was singularity), and it is indeed the most frequent use case in the HTCondor pilot setup. We thus tested the simple C program running inside an un-privileged apptainer, mimicking what an HTCondor pilot does. CRIU could not checkpoint such a job by invoking it outside apptainer nor inside apptainer, using both CRIU versions.

We then repeated the test by invoking CRIU using root privileges, mimicking the behavior of HTCondor as the host batch system manager. Even with added privileges, CRIU failed to checkpoint the user job. Next, running as root, we tested if replacing apptainer with podman and docker. The result did not change; CRIU could not checkpoint the user jobs running in the container. That said, docker does support checkpointing but has to be initiated directly through the docker toolset.

It is possible to use Docker and podman with CRIU. However, it is necessary to use an interface on Docker or podman to stop or start containers, i.e., `docker checkpoint create looper checkpoint1` [3]. Singularity does not have this interface. Podman and Docker both have an interface to work with CRIU.

Checkpointing while running inside a container runtime

All the above tests were performed inside a virtual machine application, which closely mimics the behavior of a bare-metal setup. However, Some resource providers have started offering containerized resources, e.g., Kubernetes-based, for HTCondor pilots instead. Thus, we tested launching the simple C program inside the containerized environment and invoking CRIU in the same environment. Checkpointing failed in this setup.

CPU-specific optimizations

Compiling the application's code using special flags specific to a given CPU is a common practice to speed up scientific applications. Using our simple C code, we proved that CRIU can work in this scenario if the CPU family/type is the same across the checkpointing and restoring processes. The test software can only be restored on a computer with the same family/type of processor used to compile the software.

Applications using GPUs

To test the GPU with CRIU, a GPU matrix multiplication code was used. CRIU does not support GPU checkpointing; all the attempts failed. In fact, CRIU documentation explains this on the repository [4].

Network applications

Several scientific applications can connect to different hosts. This network connection could be related to a data set transfer, a user interface, or process communication. One example is the Matlab. Network applications were tested in different ways, starting with simple send-and-receive messages using TCP and UDP coded in C; cases related to starting and stopping using CRIU all in the same machine work very well. However, it is only possible to restore once on the same machine. If it is required to stop the software and change the machine, CRIU can not load the application again. The behavior of just being able to load the application is related to the firewall configuration to keep the connections alive during the unload/load process. Another case is the network file system.

Network file system

On the OSPool, we have several network file systems to provide data to the users, container images, libraries, and others. One example of this type of network file system is CernVM-File System (CVFMS) [1]. We use a docker and a simple bash script to test this scenario. The docker container image was mounting a CVMFS repository, and a script read files from the CVMFS mounting. With CRIU, stopping and restoring one software is possible without losing access to a remote file system.

Parallel application

The message-passing interface (MPI) is a technology used to run software across different machines using the network. Using a LAMMPS application with MPI support, we could not perform checkpointing with CRIU. The load process got ”hung“ on all the attempts using the two CRIU versions (root and non-root).

4 Conclusion

Checkpointing could solve the problem of long simulations that are essential for advancing science, which, unfortunately, often fail to fit within the time constraints of batch computing providers. From the point of view of batch system administration, stopping and restarting an application process on a different machine could open the door to support preemption without incurring low efficiency due to CPU waste time. Finally, it would help save the applications' progress affected by unplanned maintenance.

CRIU can provide several options to stop and restore applications, it is possible to control applications with multiple threads and processes, and it is possible to maintain network connections. CRIU supports containers using docker and podman. It requires a “form” of root access: sudo, SUID Bit, or Kernel capabilities, and to use Kernel capabilities requires a specific version of CRIU and Linux. Restoring a previously checkpointed process requires the same directory paths used during restoration as during checkpointing. Restoring a previously checkpointed process requires the same directory paths used during restoration as during checkpointing. There is no support for GPUs, and this is an excellent feature to be tested in the future.

From the OSPool point of view, CRIU can not be used due to the limitation of the ”container“ interface. The container itself can not checkpoint itself. To do that is required to use the interface between docker and CRIU. That prevents CRIU from being used on the OSPool. This is another desired feature to be implemented on CRIU.

5 Acknowledges

This material is based upon work supported by the National Science Foundation under Grant No. 2030508. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] Alexandre F. Boyer et al. “A Subset of the CERN Virtual Machine File System: Fast Delivering of Complex Software Stacks for Supercomputing Resources”. In: *Lecture Notes in Computer Science*. Springer International Publishing, 2022, pp. 354–371. DOI: 10.1007/978-3-031-07312-0_18. URL: https://doi.org/10.1007%2F978-3-031-07312-0_18.

[2] Vitor R. Coluci, Fabio Andrijauskas, and Sócrates O. Dantas. "8 - Modeling thermal conductivity with Green's function molecular dynamics simulations". In: *Modeling, Characterization, and Production of Nanomaterials (Second Edition)*. Ed. by Vinod K. Tewary and Yong Zhang. Second Edition. Woodhead Publishing Series in Electronic and Optical Materials. Woodhead Publishing, 2023, pp. 171–187. ISBN: 978-0-12-819905-3. DOI: https://doi.org/10.1016/B978-0-12-819905-3.00008-7. URL: https://www.sciencedirect.com/science/article/pii/B9780128199053000087.

[3] CRIU. *CRIU*. 1999. URL: https://criu.org/Docker (visited on 05/07/2023).

[4] CRIU. *CRIU*. 2018. URL: https://github.com/checkpoint-restore/criu/issues/534 (visited on 05/07/2023).

[5] Adrian Reber - GitHub. *CRIU non-root*. 1999. URL: https://github.com/adrianreber/criu/tree/non-root (visited on 05/07/2023).

[6] The Open Science Grid Executive Board on behalf of the Osg Consortium:Ruth Pordes et al. "The open science grid". In: *Journal of Physics: Conference Series* 78.1 (July 2007), p. 012057. DOI: 10.1088/1742-6596/78/1/012057. URL: https://dx.doi.org/10.1088/1742-6596/78/1/012057.

[7] Igor Sfiligoi et al. "The Pilot Way to Grid Resources Using glideinWMS". In: *2009 WRI World Congress on Computer Science and Information Engineering*. Vol. 2. 2009, pp. 428–432. DOI: 10.1109/CSIE.2009.950.

[8] Aidan P Thompson et al. "LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales". In: *Computer Physics Communications* 271 (2022), p. 108171. ISSN: 0010-4655. DOI: https://doi.org/10.1016/j.cpc.2021.108171. URL: https://www.sciencedirect.com/science/article/pii/S0010465521002836.

[9] Ranjan Sarpangala Venkatesh et al. "Fast In-Memory CRIU for Docker Containers". In: *Proceedings of the International Symposium on Memory Systems*. MEMSYS '19. Washington, District of Columbia, USA: Association for Computing Machinery, 2019, pp. 53–65. ISBN: 9781450372060. DOI: 10.1145/3357526.3357542. URL: https://doi.org/10.1145/3357526.3357542.

[10] Derek Weitzel et al. "Data Access for LIGO on the OSG". In: *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact*. PEARC17. New Orleans, LA, USA: Association for Computing Machinery, 2017. ISBN: 9781450352727. DOI: 10.1145/3093338.3093363. URL: https://doi.org/10.1145/3093338.3093363.

[11] Adityas Widjajarto, Deden Witarsyah Jacob, and Muharman Lubis. "Live migration using checkpoint and restore in userspace (CRIU): Usage analysis of network, memory and CPU". In: *Bulletin of Electrical Engineering and Informatics* 10.2 (2021), pp. 837–847.