Research Collaboration Project - MSc Thesis Project

Live Migration of Container Deployments Using CRIU

Carlos Segarra (carlos.segarra@estudiant.upc.edu) - Jordi Guitart (jguitart@ac.upc.edu)

Project Description - TL;DR

The aim of this project is to implement live migration for all sorts of container deployments. In particular, we want to use Checkpoint Restore in Userspace (CRIU) to perform live migrations of arbitrary Docker deployments (including swarms) with minimal performance overhead and obliviously to the underlying architecture.

It will be developed during the 2019-2020 academic year, resulting in a Master Thesis and a scientific contribution in the form of an article or paper.

Distributed CRIU

Checkpoint Restore in Userspace [1] (CRIU) is a Linux open-source [2] software that enables to freeze a running application, checkpoint it's state to disk, and restore it elsewhere at the exact same point it was frozen.

Live Migration and P.Haul

Checkpointing and restoring to a different physical machine without impacting client processes or applications is denoted as **live migration** [3]. **P.Haul** [4, 5] is an extension to CRIU specially suited for live migration. It ships as a Go library and is used to perform container migration.

CRIU in Docker

There already exists an integration of CRIU with Docker [6] since Docker 1.13. To enable it, the Docker daemon must run in experimental mode. Some demo apps are available in an open Github Repo [1].

Checkpoint/Restore of Distributed Applications

On the other hand, live migration of distributed applications has already been studied for a long time theoretically [7, 8], but has not seen a succesful implementation in practice. In particular, for distributed applications, rollback algorithms are usually the prefered ones in the literature. Among rollback algorithms the authors differentiate between global checkpointing algorithms (coordinated, uncoordinated, or communication induced checkpointing), where the classical Chandy-Lamport algorithm lies, or Message Passing/Logging Algorithms (Pessimistic, optimistic, and causal). Further, self-stabilizing CR for ever-running applications should also be considered given that we want to do CR on containers which could need to run for a long time without halting. For illustrative purposes, we could do a small review on the existing ones.

Related Work

Among relevant related work, here we highlight those we could use in practice and benchmark against. The development team at the CRIU Foudnation provide themselves a quick SoA on the different CR projects with a comparison chart [9]. Note that, in general, fault-tolerancy is the biggest driven factor to support checkpoint/restore in an applciation (specially a distributed one). On the contrary, the initial scope of this project is to implement and support C/R in a controlled environment, without fault-tolerancy or fault injection. This remains as an open topic for further benchmarking down the line.

Firstly, MPICH-V [10] presents a general framework to implement fault-tolerant in MPI. The authors provide three novel algorithms based on the message logging paradigm and an optimization of the Chandy-Lamport algorithm for coordinated checkpointing. Although the results were novel back then, they seem to be a little bit

[1] https://github.com/csegarragonz/criu-demos/looper-single

deprecated as of today. The evolution of fault-tolerant MPI is Berkeley Lab Checkpoint/Restart [11] (BLCR) a project funded by the US government with High Performance Computing (HPC) in mind.

Distribbuted MultiThreaded Checkpointing [12] (DMTCP) checkpoints single-host or distributed computations in user-space. We implement a very simple proof of concept [2]. There are two main differences between DMTCP and CRIU. First, the former requires the code to be dynamically linked to their headers in order to perform CR, whilst CRIU does not require that. Secondly, DMTCP works by wrapping certain syscalls, CRIU on the other hand expects the Kernel to support it's features.

Roadmap

Beneath included a tentative table of contents to progress with the project. They all contain a deadline but it's meant to be very orientative and to have a general idea on how work should be distributed.

1. **Familiarization & Bibliography:** **DL:** Nov-Dec 2019

   The aim of this first block is to get introduced to the different topics we want to tackle, understand the underlying concepts, research the state of the art and get used to the software we will be using. In particular, the topics to be researched can be enumerated as follows:

   (a) Checkpoint\Restore & Live Migration

   (b) Docker & Docker Swarm & Other Distributed Docker Deployments

   (c) Live Migration of Distributed Applications

   (d) CRIU: Deployment and Usage

   (e) Distributed Checkpointing Algorithms: consider doing a mini-suvey on those for academical purposes.

2. **CRIU for Live Migration:** **DL:** Jan-Feb 2020

   The second block of work could focus on the already implemented CR functionality for Docker. Starting from pure code inspection, parameter tuning, and adding some functionality to get comfortable with the development pipeline. Some work could also be done to leverage P.Haul in another test setting for demonstration purposes. Lastly, it would be good to perform an evaluation of the introduced overhead and compare it against other CR tools like DMTCP.

   (a) P.Haul

   (b) Live Migration of Docker Containers

   (c) Evaluation & Benchmarking

3. **Live Migration of Distibuted Docker Deployments:** **DL:** May-June 2020

   This third block would contain all the novelty in our development. Here we would implement CR for clusters of Docker containers. Initially we could start with Docker Swarm and a simple distribtued algorithm. Then we could try and implement further and more advanced algorithms, and make that a tunable parameter in the deployment compose file. Lastly, an in-depth evaluation of the overhead in simulated and real hardware would complete the work.

   (a) Docker Swarm & Basic Algorithm

   (b) Implement More Algorithms/ Algorithm Types

      i. Introduced Overhead

      ii. Simulation

      iii. Real HW

4. **Wish List - To Do If Time:** **DL:** TBC

   I leave this last block of work to include things we would like to have if time permits.

[2] https://github.com/csegarragonz/criu-demos/tree/master/dmtcp

(a) Support other Orchestrators (k8s for instance)

(b) Relation w/ Fault-Tolerancy and Fault Injection

References

[1] CRIU Foundation. Checkpoint Restore in Userspace - Wiki. `https://criu.org/Main_Page`, 2019. [Last accessed 14/10/2019].

[2] CRIU Foundation. CRIU – A project to implement checkpoint/restore functionality for Linux. `https://github.com/checkpoint-restore/criu`, 2019. [Last accessed 14/10/2019].

[3] Joab Jackson. Live Migration - CRIU Wiki. `https://criu.org/Live_migration`, 2019. [Last accessed 14/10/2019].

[4] CRIU Foundation. Process Haul - CRIU Wiki. `https://criu.org/P.Haul`, 2019. [Last accessed 14/10/2019].

[5] CRIU Foundation. Process Haul - Github. `https://github.com/checkpoint-restore/go-criu`, 2019. [Last accessed 14/10/2019].

[6] CRIU Foudnation. Docker Integration - CRIU. `https://criu.org/Docker`, 2019.

[7] Michel Raynal. *Distributed Algorithms for Message-Passing Systems*. Springer Publishing Company, Incorporated, 2013.

[8] Ajay D. Kshemkalyani and Mukesh Singhal. *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge University Press, New York, NY, USA, 1 edition, 2008.

[9] CRIU Foudnation. Comparison with other CR Projects - CRIU. `https://criu.org/Comparison_to_other_CR_projects`, 2019.

[10] A. Bouteiller, T. Herault, G. Krawezik, P. Lemarinier, and F. Cappello. Mpich-v project: A multiprotocol automatic fault tolerant mpi. *IJHPCA*, 20:319–333, 08 2006.

[11] Computer Language and Systems Software Group. Berkeley Lab Checkpoint/Restart (BLCR) for LINUX. `https://crd.lbl.gov/departments/computer-science/CLaSS/research/BLCR/`, 2013.

[12] DMTCP. Distributed MultiThreaded Checkpointing. `http://dmtcp.sourceforge.net/`, 2019.