Master in Advanced Mathematics and Mathematical Engineering

Transparent Live Migration of Container Deployments in Userspace

Master's Thesis - Oral Defense

Carlos Segarra - `carlos.segarra@estudiant.upc.edu`
Advisor: Jordi Guitart - `jguitart@ac.upc.edu`

Barcelona, Tuesday July 7, 2020

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Session Outline

Problem Motivation:

- **Containers** have become the go-to alternative for sandboxing and deploying distributed applications, they build on the concept of **namespaces**.
- **Checkpointing** provides systems with **fault-tolerance** and **state consistency** enabling rollback and restore from previous stable versions.
- **Checkpoint-Restore in Userspace (CRIU)** is a tool to perform checkpoint-restore of processes, transparently to the user.
- **Live migration** consists in transparently relocating running services for improved resource-management and increased QoS.

Main Goal

Design, implement, and evaluate a tool for efficient live migration of running runC containers using **CRIU**.

Current limitations:

- Limited integration of C/R within container engines (none in terms of live migration).
- VMs are losing ground to containers, yet no replacement for VM migration tools.
- Existing live-migration libraries have lots of dependencies and are complex to set up.

Main Objectives:

- Implement a fully-featured container migration tool.
- Support network and memory intensive containers.

Contributions List

1. An exhaustive micro-benchmark of different CRIU functionalities.
2. An open-source library for live migration of runC containers using CRIU.
3. An easy-to-use binary to transparently migrate containers.
4. A comparison of our solution with traditional VM migration.

Linux Containers:

Definition

A linux container is a set of processes isolated from the rest of the system.

- Containers rely on **namespaces** and **control groups** for fine-grained resource control.
- A **container engine** transforms a *container image* into a process by means of a *runtime*.
- We choose `runC` as our container runtime as it has the best integration with CRIU.

Introduction to runC:

- **runC** is the Open Container Initiative's (OCI) reference runtime implementation.
- Rather than **images**, it relies on OCI bundles: `config.json` + `rootfs`.
- It sits at the core of most container engines as depicted in Figure 1.

Introduction to runC:

- **runC** is the Open Container Initiative's (OCI) reference runtime implementation.
- Rather than **images**, it relies on OCI bundles: `config.json` + `rootfs`.
- It sits at the core of most container engines as depicted in Figure 1.

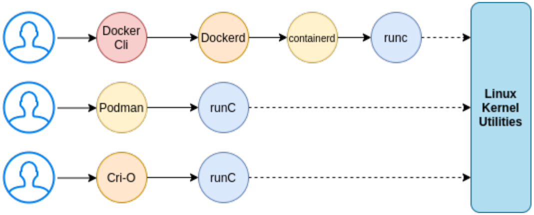

Figure 1: Placing runC in the call stack of different container engines.

Checkpoint-Restore:

Definition (Encyclopedia of Parallel Computing)

Checkpointing refers to the ability to store the state of a computation in a way that allows it be continued at a later time without changing the computation's behavior.

- The **state** is made up of: memory, pipes, sockets, ...
- This state is used to **restore** the process.
- Originated in HPC and popularized through VMs.
- **Live migration** consists in checkpointing in an environment, and restoring in a different one, **without affecting the application's availability**.

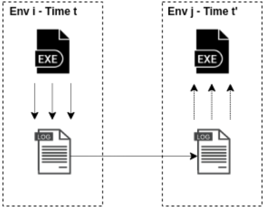

Figure 2: Basic working principle of live migration.

CRIU: Checkpoint-Restore in Userspace

- We rely on **CRIU** to perform C/R from userspace, **transparently** to the user.
- To **checkpoint**, CRIU gathers information from different resources and dumps the content to files.
- To **restore**, it morphs itself into the to-be restored process.
- To achieve **efficient live migration**, several design choices must be made.

	CRIU	DMTCP	BLCR
Target App.	Containers	HPC	HPC
Standard Kernel	$>$ `3.11`	Yes	Yes
Transparency	Full	Pre-Load	Pre-Load
Unmodified App.	Yes	Yes	No
Containers	Yes	No	No
Distributed App.	No	Yes	Yes
Open Files	Yes	No	No

Table 1: Comparison with other popular C/R tools.

.

How to reduce the impact of file I/O to disk?

- Avoid writing files twice (on dump + transfer) → Use CRIU's `page-server`.
- Avoid writing to disk at all → Use a `tmpfs` mount rather than disk.
- We compare a small application (100 kB, left) and a big one (1 GB, right).

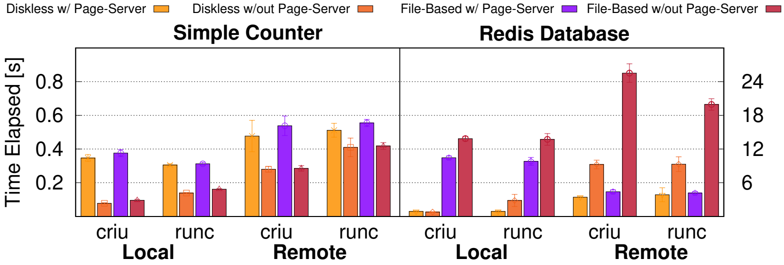

Figure 3: Diskless migration micro-benchmark.

How to minimize the application's downtime during migration?

- Perform iterative dumps → Use `pre-dump` and memory tracking.
- Ensure freshness upon restore → Parent directories linked list.
- For each app, we compare a setting w/out memory changes (left) and one with (right).

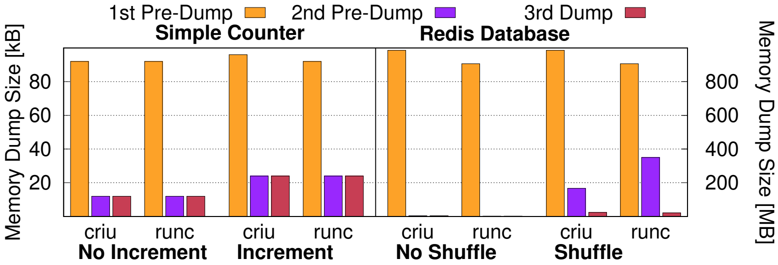

Figure 4: Iterative migration micro-benchmark.

How to migrate established TCP connections?

- Rely on `TCP_REPAIR` socket option.
- Re-create `iptables` on remote end.
- Re-use same IP address:
  1. Using network namespaces.
  2. Using locally scoped addresses.
- In the experiments we present:
  1. Downtime after an extended stop (top).
  2. Reactivity to immediate C/R (bottom).

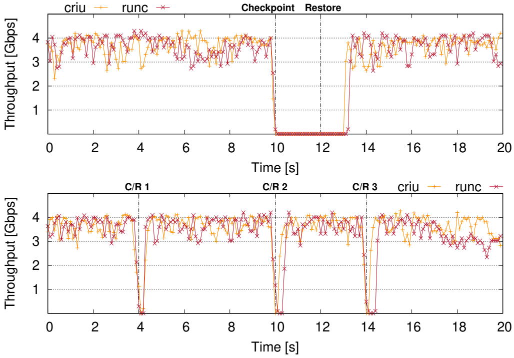

Figure 5: TCP connections micro-benchmark.

Efficient live migration of running containers:

- We employ a single process (no listening daemon on destination).
- We require `CAP_SYS_ADMIN` capabilities.
- We make two important assumptions:
  1. The OCI bundle is available on destination.
  2. The user has SSH access to both machines.
- To run with default parameters we require:
  1. A running `runC` container's name.
  2. An IP address where to migrate it.

```
while size_to_xfer > MEMORY_THRESHOLD do
    prepare_migration();
    start_page_server();
    pre_dump();
    transfer_intermediate_files();
    link_directories();
end while
start_page_server();
dump();
transfer_files();
restore_on_remote();
```

Algorithm 1: Main migration loop.

Impact of the threshold value in total downtime:

- **Downtime** (time container is unresponsive) is the key metric of success in live migration.
- The `MEMORY_THRESHOLD` becomes a decisive design parameter.
- We study its impact on downtime when migrating a Redis in-memory DB.

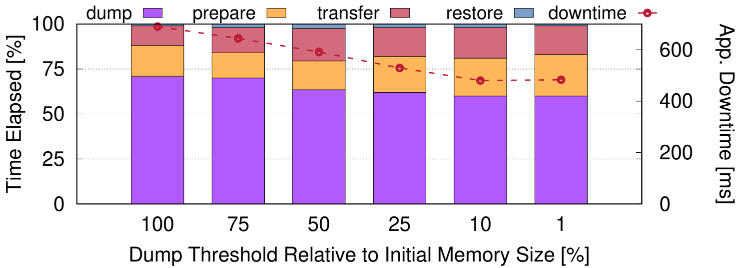

Figure 6: Application downtime macro-benchmark.

Comparison with other migration techniques:

- We compare our performance with two different settings:
  1. Naive migration: dump-transfer-restore.
  2. VM live migration: using VirtualBox's teleport.
- We measure:
  1. Scalability regarding the container's memory size.
  2. Overall time elapsed (and breakdown).
- We conclude:
  1. Higher baseline than naive due to increased set-up.
  2. $\times 1.5$ slowdown *vs.* $\times 2$ for VM and $\times 10$ for naive.

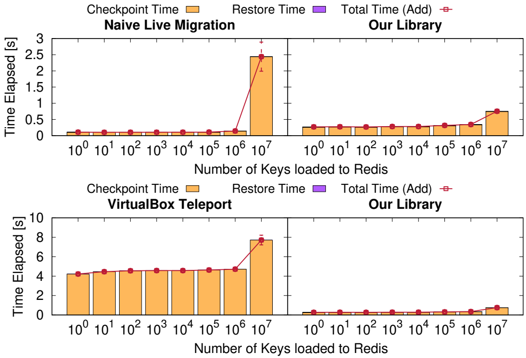

Figure 7: Comparison w/ naive migration and VM migration.

Conclusions & Future Work

Conclusions

+ Implemented an easy-to-use efficient solution.
+ Has minimal dependencies and requires minimal set-up.
- Limited comparison against other solutions (several dependencies and hard to set-up).
- Implementation does not support all features initially planned.

To-Do List

- Technical Details: upon completion we plan on submitting to a specialized conference.
  1. Evaluate against other live migration solutions for containers (*e.g.* `P.Haul`).
  2. Evaluation against other VM migration tools (*e.g.* `KVM`'s or `LXC`'s).
  3. Circumvent the pre-existance requirement for OCI bundles → Generate during **`pre-dump`** phase.
- Future lines of work:
  1. Initial goal was to support migration of distributed deployments.
     1. Implement distributed checkpointing algorithms.
     2. Integrate with container orchestrators.

Questions & Discussion

Thank you for your attention,

Observations, Doubts & Suggestions Welcome

Follow the development:

`https://github.com/live-containers/live-migration`

`carlos.segarra@estudiant.upc.edu`