Linux-CR:

Transparent Application Checkpoint-Restart in Linux

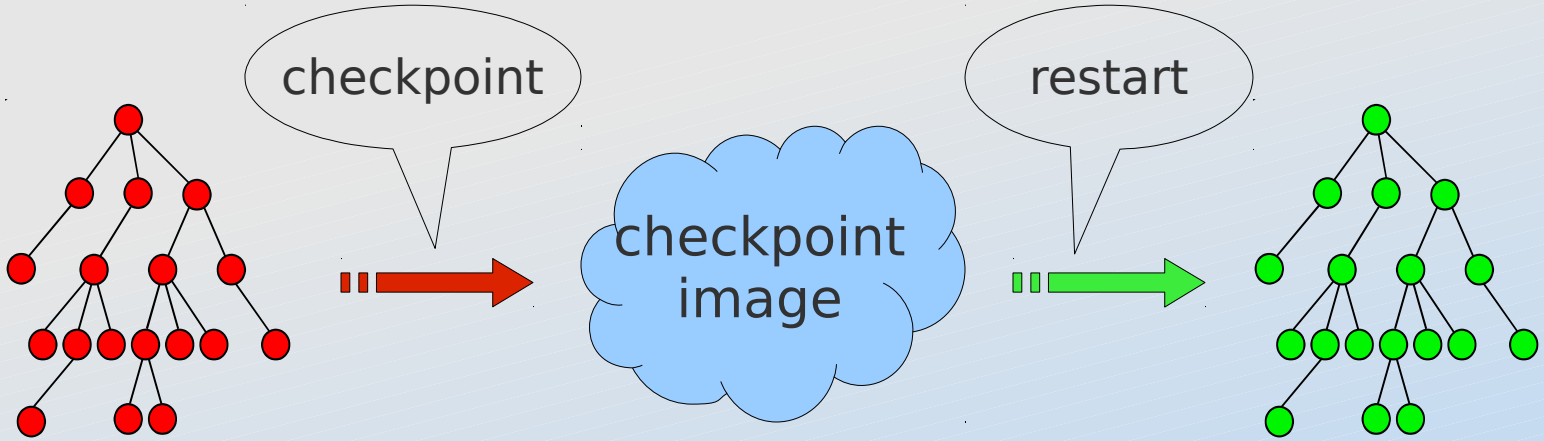
Oren Laadan
Columbia University
orenl@cs.columbia.edu

Application C/R

- Application Checkpoint/Restart

  a mechanism to save the state of running application(s) so that they can later resume execution from that point

Application C/R



original
hierarchy

restored
hierarchy

What is it good for ?

- Application roll back to the past
- Application suspend and resume
- Application migration

Application rollback

- Fault tolerance
- Effective debugging
- Fast application start-up
- Software testing
- Generic time-machine

Application rollback

- Fault tolerance
  - long running applications
  - cloud, HPC, at work, at home
- Effective debugging
- Fast application start-up
- Software testing
- Generic time-machine

Application rollback

- Fault tolerance
- Effective debugging
  - Super-core-dump
    - more details, multiple tasks
  - re-run from checkpoint
    - trace, profile, and instrument
- Fast application start-up
- Software testing
- Generic time-machine

Application rollback

- Fault tolerance
- Effective debugging
- Fast application start-up
  - from default/previous state (ccache...)
  - improve desktop boot time
- Software testing
- Generic time-machine

Application rollback

- Fault tolerance
- Effective debugging
- Fast application start-up
- Software testing
  - repeat from specific point(s)
  - distribute on multiple hosts
- Generic time-machine

Application rollback

- Fault tolerance
- Effective debugging
- Fast application start-up
- Software testing
- Generic time-machine
  - revive old server/desktop state
  - retry a move in a game

Application suspend/resume

- Improved OOM handling
- Better system utilization
- Suspend/resume a user's session

Application suspend/resume

- Improved OOM handling
  - suspend applications, don't kill
  - smart “swap” on embedded
- Better system utilization
- Suspend/resume a user's session

Application suspend/resume

- Improved OOM handling
- Better system utilization
  - suspend application to reduce load
- Suspend/resume a user's session

Application suspend/resume

- Improved OOM handling
- Better system utilization
- Suspend/resume a user's session
  - mobile desktop on USB key
  - linux based VPS/VDI systems

Application Migration

- Load balancing / resource sharing
- Zero-downtime maintenance
- High availability

Application Migration

- Load balancing / resource sharing
  - HPC (e.g. BlueWaters project)
  - cloud environments
  - linux-based VPS/VDI
- Zero-downtime maintenance
- High availability

Application Migration

- Load balancing / resource sharing
- Zero-downtime maintenance
  - live migration of applications
- High availability

Application Migration

- Load balancing / resource sharing
- Zero-downtime maintenance
- High availability
  - primary/backup in lock-step
  - frequent incremental checkpoints

Application vs Virtual-Machine

	Application C/R	Virtual Machine
granularity	specific applications	operating system as a whole unit
saved state	application state only	entire operating system state
overhead	none	visible
flexibility	application awareness	operating system is black box
deployment	linux only	same arch family

Some examples

- HPC environments
  - can extend linux-cr → distributed-cr
- Cloud deployments
  - using linux containers
- Light-weight clusters of ARMs
  - combine LXC and linux-cr

Some concrete examples

- BlueWaters
  - NCSA's most powerful supercomputer
  - checkpointing based on linux-cr
- OpenVZ
  - VPS hosting with migration capabilities
- Canonical / Ubuntu
  - add LXC & linux-cr in UEC cluster stack

Who and Who

- Who is doing ?
  - Matt Helsley, Dan Smith, Serge Hallyn, Nathan Lynch, Sukadev Bhattiprolu, me
- Who is interested ?
  - IBM, Canonical, OpenVZ, HPC industry, Kerrighed, Google (?), ...
- Who else does/did ?
  - OS: AIX, OpenVZ, IRIX, Cray...
  - Systems: Moab, BLCR/Beowolf, Condor...

Linux-C/R design goals

- Transparency
- Reliability
- Security/safety
- Performance
- Maintainability

Linux-C/R design goals

- Transparency
  - applications oblivious to operation
  - allow notify of checkpoint or restart
  - allow application awareness
- Reliability
- Security/safety
- Performance
- Maintainability

Linux-C/R design goals

- Transparency
- Reliability
  - checkpoint succeeds → restart succeeds
  - report non-checkpoint-able reasons
  - checkpoint is non-intrusive
- Security/safety
- Performance
- Maintainability

Linux-C/R design goals

- Transparency
- Reliability
- Security/safety
  - ptrace capabilities to checkpoint
  - reuse kernel code to reconstruct state
- Performance
- Maintainability

Linux-C/R design goals

- Transparency
- Reliability
- Security/safety
- Performance
  - zero impact on performance
  - reasonable code footprint
- Maintainability

Linux-C/R design goals

- Transparency
- Reliability
- Security/safety
- Performance
- Maintainability
  - next slide ...

Maintainability

- Placement of C/R code
- Extensive test-suite
- Positive experience so far
- Impact on developers

Maintainability

- Placement of C/R code
  - generic code in kernel/checkpoint/...
  - most c/r code with or near subsystem code so subsytem maintainers sees it
  - c/r is well documented
- Extensive test-suite
- Positive experience so far
- Impact on developers

Maintainability

- Placement of C/R code
- Extensive test-suite
  - test large list (>120) of scenarios
  - test before/during/after behavior
- Positive experience so far
- Impact on developers

Maintainability

- Placement of C/R code
- Extensive test-suite
- Positive experience (2.6.27 → today)
  - can ignore most kernel changes
  - mainly need to add features
  - minor changes to prior c/r code
    - e.g. splice/pipe, syscalls #s, mm helpers
- Impact on developers

Maintainability

- Placement of C/R code
- Extensive test-suite
- Positive experience so far
- Impact on developers
  - understand what may affect c/r code
  - at least notify c/r people when needed
  - awareness will grow with exposure

Design Summary

- Save/restore state in-kernel
- Checkpoint container/subtree/self
- Image holds “user-visible” state
- Userspace image conversion
- Detailed error reporting

Checkpoint

(1) Freeze process hierarchy

(2) Save global data

(3) Save process hierarchy

(4) Save state of all tasks

} in-kernel

(?) Filesystem snapshot

(5) Thaw/kill process hierarchy

Restart

(1) Create container

(?) Restore (stage) filesystem

(3) Create process hierarchy

(4) Restore state of all tasks } in-kernel

(5) Resume execution

Current State

- Supported subsystems:
  - tasks (threads, signals, credentials, etc)
  - namespaces (all but mounts-ns)
  - sysvipc (shm, msg, sem)
  - files, dirs (regular, fifos/pipes, epoll, event, simple devices)
  - sockets (unix, ipv4, ipv6)
  - security (smack, selinux labels)

Current State

- What's missing
  - [reviewed] file locks, leases, owner
  - [reviewed] unlinked files/dirs
  - [wip] fanotify/inotify/dnotify
  - [wip] mounts, mount-ns
  - /proc filesystem
  - ptraced tasks
  - more devices

Current State

- Supported architectures:
  - x86-32
  - x86-64
  - s390x
  - PowerPC
  - ARM

Current State

- Code:
  - ~23K lines in total
    - ~1200 lines documentation
    - ~600 lines per arch (x5)
    - ~8K lines in kernel/checkpoint/* (base)
    - ~7K lines “near-place” files (*/checkpoint.c)
    - ~2K lines “in-place” save/restore
    - ~2K lines for LSM

Discussion ...

- Concrete path to mainline (mm/next?)
- Exposure to subsystem maintainers ?
- Image format tied to kernel version (userspace conversion tools)

Many thanks to those who reviewed, tested, and provided suggestions !

- Web page: http://www.linux-cr.org/
- Git tree(s): git://www.linux-cr.org/git/