Lecture 6

A Survey of Checkpoint/Restart Implementations

Eric Roman - Lawrence Berkeley National Laboratory Berkeley, CA

Advanced Operating Systems

15 November, 2012

- Save the current state of the system/process periodically or before critical code sections, providing sufficient information to recover it in case of a system failure
- It's usually an operating system feature
- Not a new research area
- Commercial/production implementations are emerging (high availability systems, clusters, virtualization)

- **Process migration:** transparent process migration is used for distributed load balancing and job controlling systems
- **Crash recovery and rollback transaction:** a process can easily return to a previously checkpointed state (useful for long-running applications - scientific computation)
- **System administration:** system administrators can checkpoint processes before shutting down a machine and restart them (on the same of another machine)
- High performance systems, embeded systems, servers, medical devices, etc.

- Save the state of a running process in a file
  - Memory (stack, heap, data, bss)
  - Registers
  - File descriptors
  - Optional: pending signals, signal handlers, accounting records, terminal state
- Restart the process by recreating the objects described in the saved file

- **Full checkpoint:** saves the entire state of the program
- **Incremental checkpoint:** only saves data that changed from the previous checkpoint (minimizes the costs/time and space)
- **Checkpoint with fork:** duplicates the existing process (the original process continues to execute while the child saves its state)

Checkpoint

Checkpoint Implementations

Conclusion

Keywords

Questions

- Application-implemented checkpoint
- Library linked with the application
- Operation system implementation

- Highest degree of control
- The operating system may remain unmodified and completely unaware of checkpoints and restart
- Difficult to implement - it may not be possible to change the application source code
- Delay between the time of the checkpoint command and the time the application decides to save its state
- Lack of a common restart mechanism (different applications may use different checkpoint implementations)

- Avoids most the of underlying application source modifications
- Typically use a signal handler to accomplish checkpointing (reduces the delay between the checkpoint command time and the checkpoint decision time)
- Common restart procedure
- Imposes restrictions on which system calls the application may use (system calls like open file handles and memory mapp are intercepted)
- Interprocess communication is forbidden – scripts and parallel applications may not be checkpointed

Operating System Level Checkpoint

- Special support in the operating system kernel
- Avoids replicating kernel data structures (e.g. opened files)
- Data like process id, session id or original parent can be managed only in kernel level implementation
- Allows applications to be checkpointed at any time
- Very few implementations

- Process comunication: shared memory, pipes, local domain sockets
- During checkpoint, the operating system must suspend all processes and save their states
- During restart, the operating system must reconstruct all processes and IPC channels

Multinode Application Checkpoint

- Requires active involvement of the checkpointing processes or coordination with a remote kernel
- Ensures consistency: nodes cooperation
- Ensures all sent messages have been received or buffered

- Process address space
  - Library implementation
    - Obtains the start and end addresses for each region using system calls interception and kernel specific knowledge
    - Problems for mapped regions: mmap system call cannot be intercepted - it is used before checkpoint library is initialized; alternative: /proc filesystem
  - Operating system implementation - direct access to data structures describing the mapped regions
  - Optimization: application level implementation allow to designate "unimportant" data regions
- CPU registers - IP, SP, general purpose registers, etc
  - Library implementation: uses a signal handler (when a signal is received, the kernel stores the registers on stack)
  - Operating system implementation: direct access to data structures that store the process registers

Checkpoint Implementation Details (2)

- Signal handlers and pending signal state
  - Library implementation: sigaction or signal system calls, sigpending
  - Operating system implementation: direct access to data structures that save the signal handler and pending signals
- Files and file descriptors
  - Issues
    - Files may change between the checkpoint and the corresponding restart
    - Application interactions with the filesystem (if the application closed a file descriptor, there are no available data structures to recover file's state)
  - Improvements: save hidden copies of all the opened files
  - Reestablish the association between file descriptors and terminals
  - Opened directories: no existing implementation has addressed this issue
  - Sockets: shutdown and restart sockets through callback; message bufering mechanisms

- Library
  - libckpt
  - Condor
  - libtckpt
- System
  - VMADump
  - CRAK

- One of the first library implementations for UNIX
- Provides a number of special optimizations to reduce the size of checkpoint files
  - Memory exclusion (mark unused pages or pages that will not be modified)
  - Incremental checkpoint using mprotect()
  - Forked checkpointing
  - Synchronous checkpoint

libckpt (2)

- Requires a modification to the application source code (renaming main routine)
- The application must be recompiled and statically linked to libckpt
- Support for shared libraries
- Can not restore segments mapped in by the application through mmap()

- Implements process migration for the Condor load balancing system
- Supports applications using memory mapped segments
- Mapped segments and dynamic libraries are read through the /proc filesystem
- Requires applications to be linked with a special checkpoint library
- No recompilation is necessary

- Checkpoints multithreaded applications using Linux or Solaris threads
- Adds a checkpoint thread to the application used to synchronize the other threads and invoke user callbacks
- User may install callbacks to be invoked before or after a checkpoint is taken or after a restart is performed

- Part of Scyld's Bproc system
- Designed mainly for this style of process migration
- Explicit cooperation from process
- System call for ckeckpoint and restore
- VMADump also allows process images to be executed directly through exec()
- Optimization in saving memory (saving shared libraries name and not the content)
- Drawbacks:
  - is application-initiated
  - ignores file contents and file descriptors
  - only individual single-threaded processes, not sessions, process groups, or multithreaded applications

VMADump (2)

Dump type
Process credentials
Umask
Trace flags
Priority
Resource usage
Current working directory
Bproc IO descriptor

BPRoc

VMADump Header
Command name
General purpose registers
Floating point registers
Blocked signals
sig_action structures
Address space descriptor
Virtual memory area descriptor
Shared library name
Modified pages in library
Virtual memory area descriptor
Pages

VMADump

- Designed for process migration
- Implemented as kernel module
- Minimal modifications to the operating system kernel
- Split between user space and kernel space
- User space is responsible for identifying the set of processes to be checkpointed, and for reconnecting open file descriptors and pipes
- Children can be save
- Signals to synchronize processes to be checkpointed

- Saves data in a manner quite similar with VMADump with the difference that CRAK checkpoint isn't necessary called by the process to be checkpointed
  - Cannot use current
  - Find the location of a checkpointing process' memory
- Saves files descriptors attached to sockets, unnamed pipes, and regular files
- Pipes between processes are reconnected in user space
  - Any data undelivered in pipes is restored in kernel space

CRAK (3)

- Sockets are restored in three phases
  - New socket is created in user space
  - In kernel space, local socket data structure is modified
  - The remote socket data structure is modified to reflect the restarting address
- CRAK is system-initiated, so no modifications are necessary to user code
- Drawbacks:
  - Cannot restart multithreading processes
  - No checkpoint handlers
  - Cannot block checkpoints
- Far from a general purpose checkpoint/restart

Checkpoint Implementation Comparison

Name	*Type*	*Scope*	*File Data*	*Resource Usage*	*Credentials*	*Checkpoint Handlers*	*Signals*	*File Descriptors*	*Address Space*	*Registers*
libckp	lib	Process	-	-	-	-	-	❍	❍	●
libckpt	lib	Process	❍	-	-	-	-	❍	❍	●
Condor	lib	Process	-	-	-	Δ	●	❍	❍	●
libtckpt	lib	Thread	-	-	-	❍	●	❍	❍	●
CRAK	sys	Child	-	-	❍	Δ	●	❍	❍	●
BPRoc	sys	Process	-	❍	Δ	-	●	-	❍	●
Score	lib	Parallel	-	-	-	❍	●	❍	❍	●
CoCheck	lib	Parallel	-	-	-	Δ	●	❍	❍	●
- = Missing. Δ = Weak. ❍ = Good. ● = Complete										

- Although checkpoint/restart is a useful technology, it is still mainly a research subject and has not come to production use. The reasons are:
  - Lack of support from popular operating systems
    - Most operating systems such as Unix were not designed for checkpoint/restart. It's very hard to add such functionality without significant change of the kernel
  - Lack of commercial demand
    - Checkpoint/restart is primarily used for high performance distributed systems
  - Transparency and reliability
    - Checkpoint/restart ought to be both transparent and reliable for general use, which is difficult

Checkpoint

Checkpoint Implementations

Conclusion

Keywords

Questions

Keywords

- checkpoint
- restart
- memory
- registers
- migration
- fault tolerance
- process communication
- libckpt

?