Container Checkpointing

John Mehnert-Spahn
University of Duesseldorf
Germany
Overview

- Introduction – where do we want to go?
- Containers & Ghosts
- Container Checkpointing
Checkpointing in XtreemOS

- Kernel Checkpointer: saving states of nodes and Kerrighed clusters
- System Checkpointer: periodic incremental chkp. & garbage collection
- Grid Checkpointer: scalable hierarchical chkp., failure detection & recovery

WP3.3

WP2.1

WP2.1

WP2.2
Checkpointing in Kerrighed

- belongs to WP2.2 of XtreemOS
- Kernel Checkpointer: saving state of a process
  - shared memory: UDUS
  - open files and network communication: IRISA
- System Checkpointer:
  - WP2.1 code will be extended
  - a cluster appears as a single grid node
  - LinuxSSI/Kerrighed manages periodic checkpointing, failure detection and recovery of a cluster in interaction with the grid Checkpointer
WP2.2: container-based checkpointing in Kerrighed
- simplified checkpointing of different resources
- ghosts for saving & restoring kernel states
- checkpointing strategies for large scale clusters

WP3.3: grid-level checkpointing & recovery strategies
- adaptive strategies (coordinated versus independent ones)
- hierarchical approaches for applications spanning multiple clusters (interaction of Kerrighed System Checkpointer and Grid Checkpointer)
- heterogeneous environments (mobile, PC, clusters)
Containers

- Containers: for sharing data objects cluster wide
  - transparent access to remote data
  - MESI-like protocol for consistency
  - building block for Single System Image

- Linkers
  - Defines the type of objects to be managed by the linked container
  - Interface between containers and host OS resources
  - For memory, network streams, files, ...
Ghosts: for process migration

- handle kernel data structures of a process
- dynamically interweaving containers for resources of a process
Container Checkpointing

- coordinated checkpointing approach (synchronize processes, start checkpointing, resume work)
- what can happen within synch phase (yellow bar) in Kerrighed?
  - Case 1: change of ownership (grab page request, page eviction to a remote node)
  - Case 2: swapping pages to local disk

Distributed application - working on the same container

Diagram:
- Process on node A
- Coordinator
- Process on node B
- Synchronisation phase
- Saving phase (multiple objects after another)
Container Locking

Distributed application - working on the same container

Thread on Node A

page X deleted

Thread on Node B

page X added

coordinator

grab

synchronisation phase

object saving (all OWNER pages of one's node memory)
Case 1: Change of ownership

- caused by:
  - application unit B, stopped after application unit A
  - Message(s) in transfer

- risk: owner object can be left without saving it
  - owner object is not sent immediately after grab to requesting node
  - might be forgotten to save on requesting node ...
    - ... if object arrival follows decision which data to be saved has already been made

- => consistency issue
Case 2

- I/O operation required to retrieve objects from disk during the checkpointing operation

- Does not cause faults but a performance issue
Realising CP – Approach I

- **solution**: insert new state into state machine
- **define that ownership changes and evictions** must NOT be executed within new state – **block requests**
- **approach**: “An efficient and scalable approach for implementing fault-tolerance DSM architectures” (Morin, Kermarrec, Banatre, Gefflaut)
  - Extended Coherence Protocol (Precommit, Shared-CK, Inv-CK)
  - recovery data in memory, use for computation
- **PRO**: solves case 1 and case 2
  - new state ensures “undisturbed” synch phase
    - if extended: use replica data for computation
- **CON**: implementation; performance overhead
  - state machine modification
Realising CP – Approach II A

- Idea: stop senders and wait until container event queue is empty
- avoid impact of container protocol actions on objects on the recipient side
- wait until container event queue gets empty
- PRO: no modification to state machine
- CON: at what time will queue be empty?
  - not all processes, that could send container msg's can be stopped, otherwise system halts
  => queue is not guaranteed to be empty
Realising CP – Approach II B

- solution: avoid impact of protocol actions on sender side

- do not send protocol actions for certain containers

- realisation:
  - stop processes using signals (SIGSTOP, SIGCONT)
  - wrapper for protocol actions – do not block all containers
  - export objects
  - create disk structure (page data & meta data for recovery)

- PRO: solves case 1 and case 2
  no modification of state machine
Conclusion

- Container code is complex
- Still a lot of work ahead