Theft-Induced Checkpointing for Reconfigurable Dataflow Applications

Samir Jafar, Axel Krings, Thierry Gautier and Jean-Louis Roch
Laboratoire ID-IMAG, France

axel.krings@imag.fr

This work has been supported by the Region Rhône-Alpes (Ragtime project)
the CNRS ACI Grid-DOCG and Damascus University
Presentation Outline

- Motivation and background
- Execution model
- Theft-induced checkpointing
- Experimental results
- Conclusions and Future Work
Target Application

- Large-Scale Global Computing Systems
  - (potentially) large number of nodes
  - volatility of nodes, e.g. dynamic run-time behavior
  - heterogeneous computing environment

- Dependability Problems
  - reliability issues of large number of nodes
  - without fault-tolerance mechanism application may be infeasible
    - MTBF may sink below application execution time
Unreliability in the absence of FT

- **Computation on Cluster**
  - MTBF = 2000 days (48,000h, approx. 5 1/2 years)
  - Unreliability of one node: \( F(t) = 1 - R(t) = 1 - e^{-\lambda t} \)
Fault-tolerance Approaches

- Redundancy
  - Duplication

- Checkpointing
  » uncoordinated
  » coordinated
  » communication-induced

- Message-logging
  » optimistic
  » pessimistic
  » causal
Comparing Protocols

- **Coordination**
  - processes coordinate to build consistent global state at time of checkpointing or recovery

- **Heterogeneity**
  - checkpoint state can be restored on variety of platforms

- **Scope of recovery**
  - local or global recovery
  - local recovery: only roll-back of crashed process is necessary
Roll-back Methods

- Log-based
  - relies on logging and replaying of messages
  - process can be modeled as sequence of interval states, each one representing a non-deterministic event [Strom & Yemini 1985]

- Checkpoint-based
  - periodically save global state of computation to stable storage [Chandy & Lamport 1985]
  - differ in the way processes are coordinated
  - and on the interpretation of a consistent global state
Checkpointing

- Coordinated checkpointing
  - coordination of all processes for building consistent state before writing checkpoint to safe storage
  - e.g. [Ftc-Charm++, CoCheck]

- Uncoordinated checkpointing
  - each process independently saves state
  - consistent global state is achieved in recovery phase
  - possibility of domino effect

- Communication induced checkpointing
  - compromise between coordinated and uncoordinated
  - consistent global state achieved by forcing additional checkpoints based on some information piggy bagged on application message [Baldone 1997]
Motivating Conclusion

- Lack of solutions for
  - large parallel applications
  - dynamic execution environment
  - heterogeneous processing environment
    » potentially SMP

- Portability
  - achieved by portable languages, e.g. Java
  - or compilation into application code, e.g. Porch
  - but not on the checkpointing method itself
Presentation Outline

- Motivation and background
- Execution model
- Theft-induced checkpointing
- Experimental results
- Conclusions and Future Work
Definitions and Assumptions

- Application represented by Dataflow Graph
  \[ G = (\mathcal{V}, \mathcal{E}) \]
  \( \mathcal{V} \) finite set of vertices \( v_i \)
  \( \mathcal{E} \) set of edges \( e_{jk} \) vertices \( v_j, v_k \in \mathcal{V} \)

- Two kinds of tasks
  - \( T_i \) Tasks in the traditional sense
  - \( D_j \) Data tasks inputs and outputs
KAAPI Execution Model

- **Kernel for Adaptive, Asynchronous Parallel Interface**
  - implemented as C++ library
  - schedule programs at fine or medium granularity in distr. environment
  - KAAPI reference: http://moais.imag.fr/

- Relationship between processors and processes
Live-cycle of a Task in KAAPI

- Work-Stealing
  - primary method of scheduling workload
  - represents only communication between processes

- The states of a task
  - from a local process’ point of view
  - in the context of work-stealing
Presentation Outline

- Motivation and background
- Execution model
- Theft-induced checkpointing
- Experimental results
- Conclusions and Future Work
Theft-Induced Checkpointing

◆ State of the execution
  – based on macro dataflow graph
    » dynamic: changes during execution
    » portable: graph or portions of graph may be moved during execution

◆ Definition
  – *The macro dataflow graph G describes a platform-independent, and thus portable, consistent global state of the execution of an application.*
Theft-Induced Checkpointing

Definition of a checkpoint

- Checkpoints are with respect to a process $P_i$.
- The checkpoint of $P_i$ consists of the entries of $G_i$, the process stack
  » i.e. its tasks and their associated inputs
and **not of the task execution state** on the processor itself

Important difference:

- one simply checkpoints the tasks and their inputs
  => platform independent
- one does NOT checkpoint the task’s execution state
  => process context is platform dependent

- Note: the content of a checkpoint $G_i$ is only the dataflow graph
  representing the “future of the computation”.
Two Types of Checkpoints

- **Local Checkpoint**
  - each process takes a “local” checkpoint
    » at the expiration of a checkpointing interval $\tau$
    ■ after completion of the currently executing task

- **Forced Checkpoint**
  - needed to address global consistency in the presence of communication
  - a checkpoint is takes as the result of work-stealing
  - actions on thief and victim are defined by protocol

- Both concepts will be used in the checkpointing protocol presented
Theft-Induced Checkpointing (TIC)

- **TIC Protocol**
  - victim P0 has ready-task(s)
  - thief P1 is created on idle resource and initiates a theft operation
  - each theft results in exactly 3 checkpoints
    » the checkpoints before events A and F contain only single task

![Diagram](image)
**TIC rollback**

- Strength of TIC: rollback of single crashed process

- Need to guarantee consistent global state of execution:

**Question 1:** What does a process do that needs to send a message to a crashed process?

- attempted communication with crashed process results in error
- manager identifies the replacement processor
Question 2:

How can a process that is rolled back receive messages that it received after the last checkpoint and before the crash?

- 1) loss of theft request (event A)
- 2) crash of thief after event E but before able to checkpoint theft
- 3) crash of victim after receiving result (event G) but before being able to checkpoint
Bound on TIC Rollback Loss

- What is the maximum computation time loss due to rollback?
  - $T_1$: execution time of “parallel” application on single processor
    - note: not the same as execution time of sequential application execution
  - $T_{\infty}$: execution time on unlimited number of processors
  - $p_i$: processing time of task $T_i$

  $$\text{Max loss} = \tau + \max(p_i)$$

- But how bad can this loss be?
  - in parallel application one can always assume $T_{\infty} \ll T_1$
  - and $p_i \leq T_{\infty}$
Presentation Outline

- Motivation and background
- Execution model
- Theft-induced checkpointing
- Experimental results
- Conclusions and Future Work
Experimental Results

- Application: DOCG
  - Combinatorial optimization, Branch & Bound algorithm
  - QAP: Quadratic Assignment Problem
  - Problem size: NUGENT 22

- Platform: iCluster2 at IMAG
  - 104 dual-processor Itanium2
  - 900 MHz
  - 100Base Ethernet
TIC Overhead

- Implemented using distributed checkpoint services
  - two checkpointing periods
  - max overhead observed: 1.5%
Relative TIC Overhead

- Differences observed
  - overhead increases as the number of processors increases
    » more forced checkpoints due to work-stealing
Conclusions

- Theft-Induced Checkpointing was introduced
- Requires only crashed processes to be rolled back
- State of application represented in portable fashion
  - macro dataflow graph
  - platform independent description of application state
- Roll-back possible in
  - dynamic environment
  - heterogeneous infrastructure
- Experimental results indicate low checkpointing overhead
- Max roll-back loss can be controlled
  - selection of suitable period, granularity of application
Questions?