SFT: Scalable Fault Tolerant Runtime and Operating Systems

Jarek Nieplocha, Fabrizio Petrini, Vinod Tipparaju, Oreste Villa, Juan Fernandez, Manoj Krishnan
Pacific Northwest National Laboratory
Josep Torrellas, Yuanyuan Zhou
University of Illinois
David Addison, Ashley Pittman
Quadrics
Motivation

- Component count in high-end systems has been growing.
- How do we utilize large ($10^3$-5 processor) systems for solving complex science problems?
- Fundamental problems
  - Scalability to massive processor counts
  - Application performance on single processor given the increasingly complex memory hierarchy
  - Hardware and software failures
Multiple FT Techniques Needed

- **Application drivers**
  - Multidisciplinary, multiresolution, and multiscale nature of scientific problems drive demand for high end systems
  - Applications place increasingly differing demands on the system resources: disk, network, memory, and CPU
  - Some of them have natural fault resiliency and require very little support, others do not

- **System drivers**
  - Different I/O configurations, programmable or simple/commodity NICs, proprietary/custom/commodity operating systems

- **Tradeoffs between acceptable rates of failure and cost**
  - Cost effectiveness is the main constraint in HPC

- **Therefore, it is not cost-effective or practical to rely on a single fault tolerance approach for all applications and systems**
SFT Project

- Develop scalable and practical techniques for addressing fault tolerance at OS and RT level
  - Design based on requirements of DoE applications
  - Evaluate integrated solutions in applications
- Prior Work of the Team
  - Buffered Coscheduling (LANL)
  - ARMCI runtime (PNNL)
  - ReVive project (U. Illinois)
- Project started March 2005
- 3 publications to date under SFT project
  - SC’05, HPCA’06 OSR April 2006
Key Elements of SFT

- **BCS**: Buffered Coscheduling provides global coordination of system activities, communication, CR
- **ReVive (I/O)**: ReVive and ReVive I/O provide efficient CR capability for shared memory servers (cluster node):
- **TICK**: Incremental CR for Linux optimizes volume and performance of checkpointing data
- **FT ARMCI**: Fault-Tolerance module for ARMCI runtime system
- **XEN**: Hypervisor to enable virtualization of compute node environment including OS (external dependency)
Approach and Features

OS
- TICK, XEN
- ReVive

Runtime
- FT-ARMCI

NIC
- BCS
- FT-ARMCI

Host
- ReVive

Transparent
- TICK
- ReVive

User/Compiler
- FT-ARMCI
Buffered Coscheduling

- BSP-like system running MIMD applications
- Data transmission happens within strictly determined time window (SIMD style): simplifies system behavior
- Provides safe points and global coordination that we need for checkpoint restart
Non-blocking primitives: MPI_Isend/Irecv
Designing a Parallel Network OS

• Least common denominator of system and application software ⇒ Core Primitives

- Resource Management
- Parallel Application
- Parallel File System

Global control and coordination

Core Primitives
Xfer-And-Signal, Test-Event, Compare-And-Write

Hardware
Performance Evaluation of BCS MPI


TICK: Kernel based Incremental Checkpointing

- Implemented as a kernel module for Linux 2.6.11 (small piece statically compiled in)
- Incremental checkpointing can be triggered in less than 4µs on the 2GHz Opteron
- Can be used in globally coordinated checkpointing strategy based on BCS
- User kernel level thread that shares address space with user task.
  - High priority, SCHED_FIFO policy
  - It checkpoints the user task
- Checkpoints only dirty pages modified since the last checkpoint
- Prototype tested with serial version of NAS and ASCI SAGE benchmarks, SC’06 paper
Transparent Fault Tolerance

- BCS enforces a global recovery line at the end of each timeslice
- We are exploring a fault-tolerance model based on frequent checkpoints (every few seconds)
- Two main classes of algorithms, based on the type of storage device
- Initial performance evaluation of the bandwidth requirements of scientific applications


Xen-based Cluster Architecture

- **XEN NODE 1**
  - VM-01 (mng)
  - VM-U1 (appl 1)
  - VM-U2 (appl 1)

- **XEN NODE 2**
  - VM-01 (mng)
  - VM-U1 (appl 2)
  - VM-U2 (appl 1)

- **XEN NODE 3**
  - VM-01 (mng)
  - VM-U1 (appl 3)
  - VM-U2 (appl 3)

- **XEN NODE 4**
  - VM-01 (mng)
  - VM-U1 (appl 2)
  - VM-U2 (appl 1)

- **Storage**
  - VM-U1 (appl 2)
  - VM-U2 (appl 1)

- **QsNet^II**
Xen 3.0 Node Architecture

VM0 – Dom0

Device Manager & Control s/w

GuestOS (XenLinux)

Back-End

Device Driver

Daemon

MPI-Lib

BCS - Lib

Quadrics

BCS Module

Quadrics Driver (Elan 4)

Xen Virtual Machine Monitor

Hardware (SMP, MMU, physical memory, Ethernet, SCSI/IDE)

VM-DomU

Unmodified User Software

GuestOS

Front-End Device Drivers

Control IF

Safe HW IF

Event Channel

Virtual CPU

Virtual MMU
Implementation on ELAN4: FIFO

- No Pipeline
  - 4.5µs
  - Poll
  - Main Memory
  - BCS Desc
  - Done?
  - PCI-X
  - NIC Memory
  - BCS Desc

- Pipeline (10-slots FIFO)
  - 2µs
  - Poll
  - Main Memory
  - BCS Desc
  - FIFO Full?
  - Notification
  - Thread
  - BCS Desc
  - BCS Desc
  - NIC Memory
  - BCS Desc
  - Thread
  - BCS Desc
  - FIFO
  - BCS Desc
  - BCS Desc
  - ELAN4
Kernel level RDMA

One-sided RDMA

ITANIUM2 – Node 1

Process

Main Memory

BCS Desc

Src Buffer

Event Triggered?

Poll

PCI-X

NIC Memory

BCS Desc

Thread

BCS Desc

DMA Engine

Notification

ELAN4

ITANIUM2 – Node 2

Process

Main Memory

BCS Desc

PCI-X

QsNet™

ELAN4
RDMA Performance

Buffer size from 4B to 4 MB
Latency: 6 µs
Bandwidth: 888 MB/s
Shared-memory multiprocessor (cluster node) with in-memory incremental checkpointing such as Illinois ReVive [ISCA-2002]
- Can effectively roll back memory state
- Lacks solution for I/O undo/redo (disk and network)

Problem with I/O: output commit problem
- Output to external world cannot be rolled back
- An output should be committed only when the system is guaranteed not to roll back to a state prior to the corresponding output request
Solution: Pseudo Device Driver

- Pseudo Device Driver (PDD) software layer between the kernel and the device drivers. (Masubuchi et al., 1997)
- No need to modify the kernel or the applications.
- PDD delays output operations until next checkpoint to eliminate any inconsistencies after rollback.
- Implemented disk PDD and network PDD prototype on Linux 2.4 (HPCA-2006).
- Our disk PDD employs buffering or renaming technique to speed up synchronous writes.
- Goal is to have most faults handled through in-memory checkpoint/recovery.
Disk PDD – Buffering

- Data written to the memory and a fast disk buffer
- Disk buffer can be a small disk written sequentially
- If no fault, after the checkpoint, copy data from the memory buffer to the target disk

Faults 1 & 2: memory recoverable
Fault 3: non memory recoverable
Network PDD and TCP

- Can resend the packets after rollback → TCP eliminates duplicate packets

- Can avoid saving inputs for replay → In TCP, the sender timeouts and retransmits
Recovery from Memory-Recoverable Faults

- Checkpoint Error
- Detection Latency
- Self-check, Rerouting
- Rollback
- Reconstruct Lost Data

1. Reset Device (~1ms)
2. Reinitialize Device Driver (~10ms)
3. Re-issue Pending I/O in Background (~100ms)

Recovery related to I/O
Fault Tolerance in ARMCI

- ARMCI is runtime for global-address space programming models (GA, CAF, SHMEM, ..)
  - one-sided communication
- Fault Resilient Scheme (FRS)
  - program state not stored, recovery not transparent
  - light weight
- Fault Tolerant Scheme (FTS)
  - recovery is transparent to user
  - more heavy weight than FRS
- Both schemes rely on spare pool of nodes for reconfiguration and recovery
  - Virtualization of processor IDs
Reconfiguration Scheme

- Use spare set of idle nodes to replace the failed node.
- This approach is more realistic given the current operation practices of supercomputer centers than requesting them dynamically from the resource manager.
- For quicker restart the spare nodes can load a copy of the executable at startup.
Fault Resilient Scheme

- User specifies collection of data blocks to be checkpointed
- Data is saved through explicit calls
  - Automatic storage at regular intervals available
  - Several options to checkpoint data
    - option to not overwrite previous save is also provided
    - ARMCI_Protect_data(Data_Structure, SAVE_STACK (yes/no), ARMCI_Group)
- Fault notification through signals (from RM)
- All the protected data is restored
- User/compiler has to worry about program state!
FRS ARMCI Usage in GA

- User can protect Global Arrays and other critical stack and heap data for future restoration
- In memory through mapped files or on disk
- Each individual global array created on a group of processes can be independently checkpointed and restored by the group.
- User interfaces

```c
GA_Protect_data(Array_List, Num_of_Ele, User_Pointer_List, Num_of_Ele)
```
Processor Groups and Node Virtualization

- All ARMCI process ID's are virtualized
- Default Groups in GA
  - GA layer has been modified to transparently work on a process group without any modification to user program
- Subsequent GA calls after recovery work unmodified
- Global Array data structures are automatically updated
  - Spare process is able to allocate memory and inform the group it is in of the allocation
A popular method for solving Schroedinger equation \( H\Psi = E\Psi \).
The bulk of the work involves computing the 4-index elements \((\mu \nu | \omega \lambda)\). This is done by decomposing the quadruple loop into evenly sized blocks and assigning blocks to each processor using a global counter. After each processor completes a block it increments the counter to get the next block.

```
for i = 1 to N do 
  for j = 1 to M do 
    for k = 1 to P do 
      for l = 1 to Q do 
        F(i,j) = .. 
        Evaluate block 
      end for 
    end for 
  end for 
end for 
```

Accumulate results.
Preliminary Experimental Results

- Platform: dual Itanium-2, Elan-4, 2.6.11 kernel
- Checkpoints in every SCF iteration, 1 fault/run