High-speed Checkpointing for High Availability

Brendan Cully
brendan@cs.ubc.ca

Department of Computer Science
The University of British Columbia

Xen Summit 5, November 2007
High availability in a nutshell

- The ability to tolerate *fail-stop physical* failure
- *Not* software failures
- *Not* non-fatal errors (memory errors etc)
- *Not* cold-start (recovery should be seamless)
High availability is hard

- Customized hardware is expensive and inflexible
- Operating systems are complex and ever-changing
- Libraries are restrictive
- Applications infinitely reinvent the (square) wheel
The Xen solution

- Machine state is readily available
- Interface is narrow and stable
- Performance is good
The REMUS High Availability Service

A checkpoint-based service providing

- Redundancy
- Enhanced
- Moderately
- Unreliable
- Servers

- Generality
- Transparency
- Seamless failure recovery
- Multiprocessor support
- Active-Passive configuration
Outline

Introduction

Design and Implementation
  High-speed checkpointing
  Network buffering
  Disk replication
  Failure detection

Evaluation

Conclusion
Approach

- Encapsulate execution in a virtual machine
- Perform frequent lightweight checkpoints
- Execute *speculatively* between checkpoints
- Propagate checkpoints asynchronously
High-level overview
### General operation

- The primary and backup begin with identical disk images.
- Attach disk and network proxies to the protected VM when it begins execution.
- At frequent intervals (≈ 25 ms) take a checkpoint of memory and disk state and propagate it to the backup.
- When the checkpoint has been acknowledged at the backup, buffered output is released to external clients.
Virtual machine checkpointing

- Modification of existing code supporting live migration
  - In essence, it moves the virtual machine to a new location, but also leaves it running at the old location
  - The remote node does not allow the image to execute until a failure occurs at the primary
- Required several changes
  - Performance optimizations
  - Changes to Xen to allow checkpointed images to resume execution (now in the upstream codebase)
  - Changes to ensure that a consistent image is available at all times on the backup
Live migration in a nutshell

- Xen puts the virtual machine into *shadow paging mode*
  - Guest page tables are replaced at the hardware level with versions in which all pages are marked read-only
  - Write faults allow Xen to maintain a map of dirty pages before restoring read-write access to pages (or propagating page faults)
- Live migration is performed by copying dirty pages to the new location without pausing the guest
- This occurs in rounds: the migration process chases the virtual machine
- A final round before migration pauses the domain in order to capture a consistent image of up-to-date state before activating the VM at the new location
- The original VM is destroyed
Checkpointing support

- Checkpointing is the repeated execution of the final stage of live migration: all state changed since the previous epoch is propagated.
- To allow repeated checkpointing, new functions were added to Xen to mark a domain as runnable after suspend.
- The migration process was converted into a persistent daemon.
- The process receiving migration data was modified to buffer checkpoint rounds in memory and apply them only after they had been completely received.
- It was also modified to loop waiting for new checkpoint data unless the connection to the sender times out.
Performance optimizations

- Checkpoint data is buffered locally and propagated after the guest has resumed
- Special signalling is used to request guest suspension and receive notification upon completion
  - This reduces the time required for this operation from an average of 30-40ms (worst-case over 500ms) to roughly 100us
- The guest suspend process is simplified. Devices are no longer disconnected on suspend or reconnected on resume
Network buffer principles

- IP networks are unreliable
  - They may lose, duplicate or reorder packets
  - Applications either tolerate this or use a layer above IP to provide stream semantics (i.e. TCP)
- Replication does not need to preserve network data to ensure correctness
  - If network output is lost due to failover, applications will recover
- Network output representing *speculative* state must be buffered
  - In the case of failure, the state that produced this output is lost, and not likely to return
Network buffer overview
Network buffer implementation

- Implemented as a custom-built queueing discipline
  - Queueing disciplines regulate outbound traffic from network devices. Commonly used to rate-limit (token-bucket) or provide better fairness under congestion (SFQ)
  - Have two basic operations: enqueue and dequeue. In Remus, packets are only dequeued when the state that generated them has been checkpointed
  - Remus sends a message via RTNetlink to the queueing discipline to mark a checkpoint
- Installed over the IMQ device
  - *Outbound* traffic from the guest VM is *inbound* traffic for the host
  - Linux queueing disciplines only queue outbound traffic
  - IMQ is a third-party virtual device that accepts inbound traffic and reinjects it specifically to allow inbound queuing
Disk replication principles

- The active disk must be crash-consistent at all times
- In case of failure, disk state at the time of the most recent checkpoint must be available
- At all times, only one physical disk represents the most recent state of the host
Disk replication overview

1. Disk writes are issued directly to local disk.
2. Simultaneously sent to backup buffer.
3. Writes released to disk after checkpoint.
Disk replication implementation

- Implemented as a *block tap* module
  - The block tap is a Xen device that allows a user-space process to interpose on the block request/response stream between a virtual machine and its devices

- Operates as a client/server pair

- The client
  - Propagates disk write requests to the server at the same time that it passes them to the underlying device
  - Forwards checkpoint messages from Remus to the backup
  - Forwards checkpoint commit messages from the backup to Remus
Disk replication server

- The server maintains two separate buffers
  - The *speculation buffer* receives the write request stream forwarded from the client
  - When a checkpoint message arrives in the stream, it moves the speculation buffer into the *write-out buffer*
- The contents of the write-out buffer are written to disk asynchronously
- In the event of failure, the speculation buffer is discarded
- Execution does not begin on the backup until the write-out buffer has been flushed to disk
  - Once execution has begun, the backup represents externally visible state — its disk image must be at least crash-consistent
  - When the write-out buffer has drained, an activation record is written to disk and execution may resume
Failure detection

- Failure detection is managed by a simple in-stream heartbeat
  - If the primary times out writing to the backup, or does not receive checkpoint commit acknowledgment, it disables replication
  - If the backup times out reading checkpoint data from the primary, it activate the replicated VM from the most recent completed checkpoint.
- Currently there is no provision for fencing in the case of network partition
  - Bonded NICs on the replication channel may suffice
Test environment

UBC netbed

- Each node is equipped with
  - 1 3.2 GHz Pentium 4 CPU (2 hyperthreads)
  - 3 1Gbps NICs
  - 1024 MB RAM (mostly)
  - 1 80 GB SATA drive

- Nodes are networked
  - One link is for application traffic
  - One link provides administrative access
  - One link is for replication traffic
Failover test

- **Test procedure**
  - Ping primary every 200ms to measure response time
  - SSH to node, begin kernel compilation
  - Disconnect power to primary node

- **Results**
  - SSH session remains open
  - Kernel compilation continues to successful completion
  - Ping reports 6 lost packets (1.2 seconds unavailable)
The effect of checkpointing on kernel compilation time

![Bar chart showing the effect of checkpointing on kernel compilation time](chart.png)
Future work

- Hardware virtualization support
- Introspection optimizations
- Copy-on-write checkpoints
- Deadline scheduler
- Replication stream compression
- Disaster protection
Thank you