Distributed Simulation and the Time Warp Operating System

David Jefferson, $\boxed{\text{Brian Beckman}}$, Fred Wieland, Leo Blume, Mike DiLoreto, Phil Hontalas, Pierre Laroche, Kathy Sturdevant, Jack Tupman, Van Warren, John Wedel, Herb Younger, and Steve Bellenot

Circa 1987

Peter Reiher, too

Introduction

- What is the problem?
  - They want to enable concurrent execution for speedup on a multiprocessor machine for $\boxed{\text{discrete event simulation.}}$
  - The system exhibits irregular causal and temporal behavior.
- What is new or different?
  - They developed an operating system with a complete commitment to optimistic execution and rollback.
- What are the contributions and limitation?
  - Performs synchronization with a distributed process rollback mechanism.
  - There is ~~no~~ dynamic creation of processes at runtime.
  - There is ~~no~~ migration of processes for load management.

we know how to do both of these

Problem Details

Logistics (queuing models)
Supply-chain
Monte-Carlo Forecasting
Physics! pucks (2D pool)
general collision processing
(Mirtich in a games engine)

- The system was ~~specifically~~ designed for military simulations.
  - Expensive computational tasks
  - Composed of many interacting subsystems
  - Highly irregular temporal behavior
- This was not intended as a general purpose operating system.
- Nevertheless, there are many applications:
  - Discrete event simulations
  - Large, distributed databases
  - Real time systems
  - Animation systems

Approach

rollback is the ONLY synchronization mechanism

- Develop a new operating system with complete commitment to optimistic execution and process rollback
  - Don't treat rollback as a special case for exception handling, deadlock breaking, transaction abortion, etc.
  - Use rollback as frequently as other systems use blocking.
- But why a new OS?
  - Rollback forces a rethinking of all os issues (scheduling, synchronization, message queueing, flow control, memory management, error handling, I/O, and commitment.)
  - Building TW on top of an os would require two levels of synchronization, two levels of message queueing, etc.

Operating Environment

251 M processes

Around 2013, TW was applied on 2,000,000 cores for "speedup" > 2M × & 504 BN transactions/sec

LLNL
ROSS
Sequoia

- TWOS was developed for the Mark III hypercube.
  - Mark III was developed between 1983 and 1987.
  - Composed of 32 nodes, or processing units, which together have peak speed of about 512 million floating point operations per second (flops).
  - Later upgraded to 128 nodes.

(Not possible to compare to a sequential run)

- It was a single user system.
- Supported distributed applications composed of processes communicating by messages.

Time Warp and Virtual Time

- Concept of *virtual time* used to organize and synchronize distributed systems [Jefferson 85].
- Virtual time is a global, temporal coordinate axis defined by the application as a measure of progress and as a scale against which to specify synchronization
  - Real values
  - Totally ordered
  - May or may not be connected to real time
- Virtual memory is to demand paging as virtual time is to the Time Warp mechanism.

$vt \mapsto rt$

$rt \mapsto vt$

Virtual time is a kind of dual to Lamport clocks

Virtual Time Synchronization Problem

- Application is composed of many processes
- All messages are time-stamped
- All incoming messages are funneled into a single input queue per process
- How can the operating system control the execution of a process so that it receives its messages in nondecreasing time-stamp order and is guaranteed to make progress?

Time Warp Mechanism

- Takes an *optimistic* approach.
  - Assume the next message in the queue is the true next message
- Messages may arrive asynchronously
- When a message with time-stamp t less than what has executed, Time Warp must:
  1. roll back the process to a time just before virtual time t
  2. execute the new message at virtual time t
  3. start re-executing messages with time-stamp greater than t, again in time-stamp order, canceling all effects of any output messages that were sent after t during the last forward execution but were not re-sent in this one.

Antimessages

- How to support the cancellation described in (3)?
- Introduce the concept of *antimessages*
  - Every event, query, and reply message has a *sign*, + or -
  - Two messages identical in all fields with opposite signs are *antimessages* of eachother
- How do antimessages behave?
  - -(-m) = m
  - Insert(Insert(Q, m), -m) = Q

How to use Antimessages

- When a process P requests a message to be sent, TWOS creates a message-antimessage pair
- The positive message delivered to the receiver's input queue
- The negative message is retained in P's output queue
- As long as there are no rollbacks, the negative message stays in the output queue and is eventually garbage collected

Cancellation Mechanism

- To undo the affects of sending m from P to Q, remove -m from P's output queue, and send it to Q.
  1. If -m arrives in Q's future, then it will annihilate with the m in P's input queue and the cancellation is finished
  2. If -m arrives in Q's past, it will cause Q to roll back, but it will also annihilate with -m, so that when Q executes forward again, neither +m not -m exist. Q will not see either of them.

TWOS Programming Model

- Each process has a 20 character unique name
- Any process can send to any other process at any time
- No notion of a pipe, channel, or connection

TWOS Process

A TWOS Process is composed of 4 sections and state variables

- Initialization Section - executed once at initialization
- EventMessage Section - invoked when an event message is processed
- Query Message Section - invoked when a query message is processed
- Termination Section - invoked once at termination

TWOS System Calls

Several system calls unique to TWOS:

- Me(MyName)
- Virtual Time(VTime) "now"
- SendEventMessage(ReceiveTime, Receiver, Text)
- SendQueryMessage(Receiver, Text, Reply)
- SendReplyMessage(Text)
- MCount(n)
- ReadEventMessage(k, Text)
- ReadQueryMessage(Text)

Processor Scheduling

- Preemptive lowest virtual time first
- Arbitrary choice to break ties
- A process could run indefinitely
- If a new message arrives which causes a rollback, the process will be pre-empted

Process Synchronization

- A process only blocks if it has no unprocessed messages in its input queue, or if it is waiting for a reply to a query
- Does a full rollback immediately (even if executing) if a message arrives with a lower time-stamp
- A process can roll back out of a blocked state, then execute and re-enter the blocked state

Flow Control Challenges

Flow Control is challenging in TWOS

- Not only do incoming messages fill up memory, but also outgoing and saved state
- No explicit channels, so flow control cannot be done on a per-channel basis
- Messages cannot be deleted when they are received because of rollback
- ~~TWOS runs best when memory is almost completely full,~~ which strains on storage and flow control mechanisms

no longer thought true

Flow Control

now called
"cancelback"

Flow Control tool is *message sendback* [Gafni 85]

- Idea is that when memory is full, send a message back to make room, which may cause a rollback and annihilation
- Communication analog of process rollback

⇒ frees up memory BOTH sides

recomputation uses up real time
effectively throttling msg flow

Commitment

Some operations are *irreversible* and therefore require commitment.

- Use Global Virtual Time (GVT)
- GVT is the minimum virtual time of any uncompleted event or message transmission in the system
- Once GVT is is greater than some value t, TW can commit all output requests at virtual time less than t, and release all messages and state buffers with time less than t, and report any error outstanding with time less than t

Performance

most similar to
AMZN problems

- Primary criteria is time to completion of benchmarks.
  - Game of Life
  - Military command and control model

colliding
pucks

- Note that since there is no dynamic process migration, there were many different assignments of processes to processors.

COMMO Benchmark

- COMMO is a military simulation
- The evaluation shows:
  - Little improvement after 16 processors, best time with 24 nodes
  - Maximum speedup of 10.66 using 24 processors
  - Number of rollbacks increases with the number of processeors
  - Greater speedup occurs with more rollbacks (rollbacks aren't in the bottleneck code)
  - About 33.4 % of messages were antimessages

Eventually achieved speedup of 60x on 84 processors

Questions?