Advanced Simulation

PDES: Time Warp Mechanism

Other Mechanisms

Outline

- Rollbacks idiosyncrasies and remedies
  - » Error Handling
  - » Dynamic Memory Allocation
- Event Retraction
- Improving the cost of rollbacks
  - » Lazy Cancellation
  - » Lazy Re-Evaluation
- Memory Management
  - » Mechanisms
  - » Storage optimal protocols
  - » Artificial Rollback
- Other optimistic protocols

Optimistic Execution Questions

- How to handle error handling in an optimistic simulator?
  - » Why is this a problematic?
- How to manage dynamic memory allocations?
  - » Why is this problematic? Remedies?
- How to make rollbacks more efficient?

Error Handling

- *Typically Errors such as divide by zero, are handled by aborting program. Is this appropriate for TimeWarp simulations? Why or Why not?*
- Problem: What if an execution error is rolled back?
- Solution: Do not *abort* program until the error is committed (GVT advances past the simulation time when the error occurred).
  - » Requires Time Warp executive to “catch” (flag) errors when they occur
  - » Countermeasures depend on error type

Error Types

- Program detected
  - » Logic errors, e.g., some variables never negative
  - » Treat “abort” procedure like an I/O operation, prevent error from propagating and flag error to see if it erased by rollback.
- Infinite loops
  - » Interrupt mechanism to receive incoming messages
  - » Poll for messages in loop
- Benign errors
  - » Errors that impact only checkpointed state (e.g., divide by zero)
  - » Trap mechanism to catch runtime execution errors
- Destructive errors
  - » Difficult to detect these…
  - » Example: overwrite state of Time Warp executive)
  - » Runtime checks (e.g., array bounds)
  - » Strong type checking, avoid pointer arithmetic, etc.

Dynamic Memory Allocation

`malloc()`, `free()` How should these be handled?

Issues:

- Roll back of memory allocation (e.g., `malloc()`)
  - » Problem: Memory leak (when check pointing a pointer to a previously allocated memory location). Run out of memory…
  - » Solution: release memory if malloc rolled back
- Roll back of memory release (e.g., `free()`)
  - » Problem: Reuse memory that has already been released. The LP did not really mean to free the memory …
  - » Solution:
    - – Treat memory release like an I/O operation
    - – Only release memory when GVT has advanced past the simulation time when the memory was released

Event Retraction

- **Goal:**
  - » Need a primitive to un-schedule a previously scheduled event (application level primitive)
- **Example:**
  - » ORD Schedules an arrival at JFK
  - » Need to re-route aircraft to Boston (due to congestion at JFK)
- **Observation:**
  - » Cancellation retracts events at the ‘simulation kernel level’
- **Problem:**
  - » Need a mechanism to ***undo*** event retraction (cancellation) if the event computation that invoked the retraction is rolled back.

Event Retraction Approaches

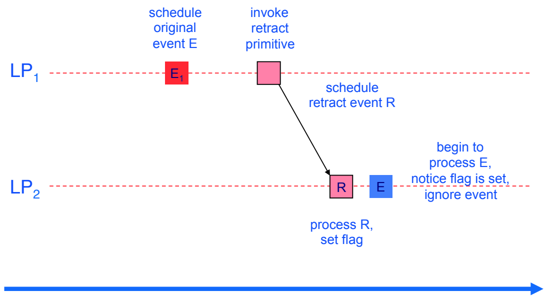
- **Application Level**
- **Kernel Level**

Event Retraction: Approach 1

Application Level Approach

1. Schedule a retraction event with time stamp earlier than (<) the event being retracted
2. Process retraction event: Set flag in LP state to indicate the event has been retracted.
3. Process event: Check if it has been retracted before processing any event

Example: *Application* Approach



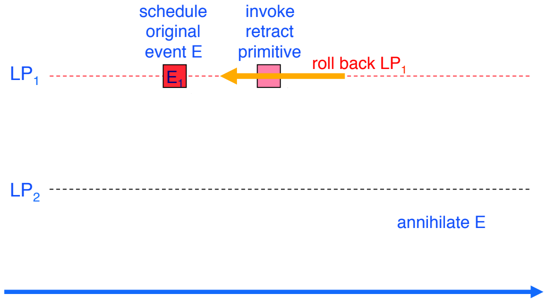
Retraction handled within the application

Event Retraction: Approach 2

Time Warp Executive Level Approach

- **Retraction: Mechanics: send anti-message to cancel the retracted event**
  - » **Retraction: invoked by application program**
  - » **Cancellation: invoked by Time Warp executive (transparent to the application)**
- **Rollback “retraction request” (visualization – coming up!)**
  - » **Idea: Reschedule the original event**
  - » **During retraction: a positive copy of message being retracted is placed in output queue**
  - » **Rollback: release messages this message from output queue (emulating that the message was not retracted)**

Example: Kernel Approach

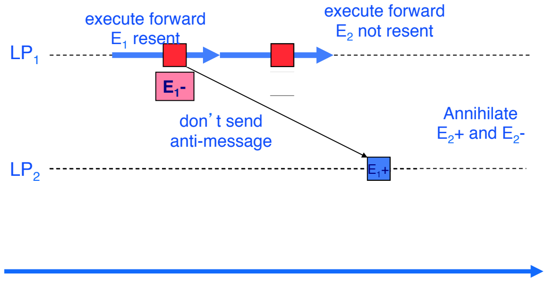


Retraction handled within Time Warp executive

Lazy Cancellation

- **Motivation**:
  - » re-execution after rollback may generate the same messages as the original execution
  - » in this case, need not cancel original message that were scheduled by rolled back event.
- **Mechanism**:
  - » rollback: do not immediately send anti-messages
  - » after rollback, re-compute forward
  - » only send anti-message if recomputation does NOT produce the same message again

Example: Lazy Cancellation



Lazy cancellation avoids unnecessary rollback

Lazy Cancellation: Evaluation

- **Benefit**:
  - » avoid unnecessary message cancellations which in turn eliminate secondary rollbacks.
- **Liabilities:**
  - » extra overhead (message comparisons)
  - » delay in canceling wrong computations
    - – may allow erroneous computations to spread further. -> more computations may need to rollback when anti-message is finally sent
  - » more memory required (delayed memory reclamation)
- **Conventional wisdom**
  - » Lazy cancellation typically improves performance
  - » Empirical data indicate 10% improvement typical

Lazy Re-evaluation

- **Motivation**:
  - » re-execution of event after rollback may be produce same result (LP state) as the original execution
  - » in this case, original rollback was unnecessary
- **Mechanism**:
  - » rollback: do not discard state vectors of rolled back computations
  - » process straggler event, recompute forward
  - » during recomputation, if the state vector and input queue match that of the original execution, immediately “jump forward” to state prior to rollback (the straggler didn’t change the state).

Lazy Re-evaluation

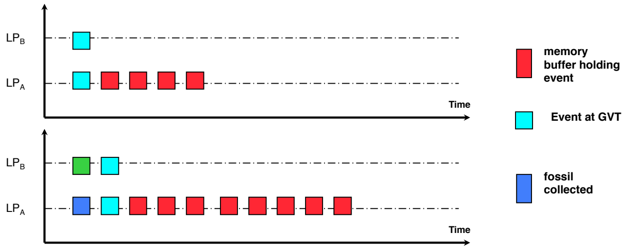
- **Benefit**:
  - » avoid unnecessary recomputation on rollback
  - » works well if straggler does not affect LP state (query events)
- **Liabilities**:
  - » extra overhead (state comparisons)
  - » more memory required
- **Conventional wisdom**
  - » Typically does not improve overall performance
  - » Useful in certain special cases (e.g., query events)

Memory Management in Time Warp

- Parallel execution using Time Warp tends to use much more memory than a sequential execution (even with fossil collection)
  - » State vector and event history
  - » Memory consumption can be unbounded because an LP can execute arbitrarily far ahead of other LPs
  - » “Overoptimism” lead to very long and frequent rollbacks, may waste computation time.

Memory Management in Time Warp

"Overoptimism" lead to very long and frequent rollbacks, may waste computation time.



Memory Consumption

- **Sequential Simulations**:
  - » aborts
- **Parallel Simulations:**
  - » abort?
  - » more memory?
  - » blocking?
  - » Memory:
    1) positive and
    2) anti-messages and
    3) state vectors

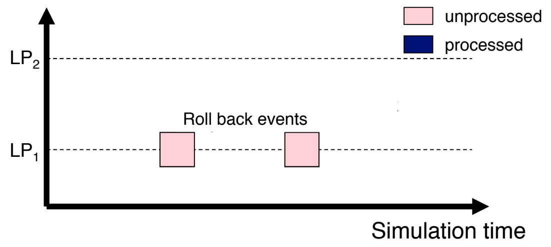
Memory Consumption Remedies

- **Infrequent / incremental**: state saving
- **Pruning**: dynamically release copy state saved memory
- **Blocking**: block certain LPs to prevent overly optimistic execution
- **Roll back** to reclaim memory
- **Message sendback**

Message Sendback

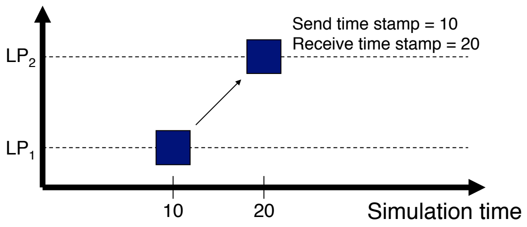
Basic Idea:

- **Send time stamp**
- **Reclaim memory used by a message by returning it to the original sender**
- **Usually causes the sender to roll back**



Event Time Stamps

- **Receive time stamp: time stamp indicating when the event occurs (conventional definition of time stamp)**
- **Send time stamp of event E: time stamp of the LP when it scheduled E (time stamp of event being processed when it scheduled E)**



Message Sendback

- **Causes sender to roll back to the send time of event being sent back**
- **Can any message be sent back?**
  - » **No! Can only send back messages with send time greater than GVT**
- **A new definition of GVT is needed**

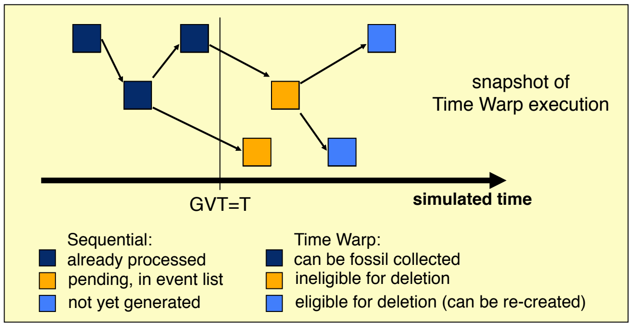
GVT(T) (GVT at wallclock time T) is the minimum among
- » **Receive time stamp of unprocessed and partially processed events**
- » **Send time stamp of backward transient messages at wallclock time T**

Storage Optimal Protocols

Storage Optimality: A memory management protocol is storage optimal iff it ensures that every parallel simulation uses memory O(M), where M is the number of units of memory utilized by the corresponding sequential simulation

- **Basic idea: if the Time Warp program runs out of memory**
  - » **identify the events (message buffers) that would exist in a sequential execution at time T, where T is the current value of GVT**
  - » **roll back LPs, possibly eliminating (via annihilation) all events except those that exist in the corresponding sequential execution.**

Classifying Events



snapshot of Time Warp execution

GVT=T

simulated time

Sequential:	Time Warp:
■ already processed	■ can be fossil collected
🟧 pending, in event list	🟧 ineligible for deletion
🟦 not yet generated	🟦 eligible for deletion (can be re-created)

Sequential execution: Which events occupy storage in a sequential execution at simulation time T?

Time Warp: For which events can storage be reclaimed?

Observations

- **In a sequential execution at simulation time T, the event list contains the events with**
  - » **Receive time stamp greater than T**
  - » **Send time stamp less than T.**
- **Time Warp can restore the execution to a valid state if it retains events with**
  - » **Send time less than GVT and receive time stamp greater than GVT.**
  - » **All other events can be deleted (as well as their associated state vector, anti-messages, etc.)**
- **Storage optimal protocols: roll back LPs to reclaim all memory not required in corresponding sequential execution**

Storage Optimality

- **Time warp on shared memory processor can achieve optimality… (extra number of buffers needed compared to a sequential simulation differ by a constant and are not in terms of number of LPs).**
- **Example:**
  - » **5 logical processes organized in a ring (communication network).**
  - » **Single message, when processed creates new message and sends it over using shared memory (no extra buffers).**
  - » **One event in pending event list a sequential simulator would only need two memory buffers one to hold current event and one to hold the new event (same as in the Time Warp protocol as GVT advances the old buffer can be reclaimed).**

- **Same protocol in a distributed environment is not optimal as memory buffers cannot be shared among processors - here each processors need to hold 2 buffers at all times so (2N buffers - where N is the number of processors in the system).**

Cancelback

- **Shared memory machine mechanism**
- **Storage optimal**

- **Global pool to hold free buffers**
- **Uses Message Sendback mechanism**
  - » **Send back messages TS > GVT**
- **Batching – into a salvage parameter**

```
1.Compute GVT
2.Fossil Collect
3.Message Sendback
   • Find eligible
     events
   • Send back messages
```

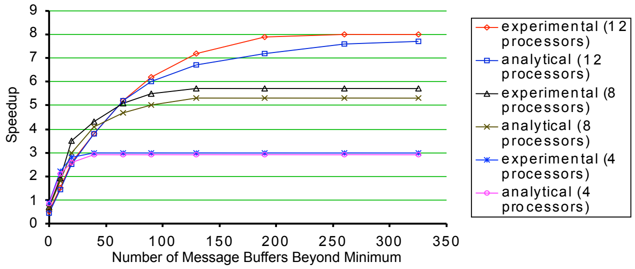
Artificial Rollback

- Similar to cancelback but reclaim using rollback instead of sendback to reclaim memory.
- Rollback the LP with largest simulation time clock is rolled back to the LP with the next largest clock value.
- Storage optimal when implemented in shared memory with a global pool.
- Simpler to implement.
- Batching.

Other Memory Mechanisms

- Prune-back
- Adaptive mechanisms: predicts memory the program needs on-line
- Trading performance and Memory
  - » Performance may DECREASE if memory is increased further – poorly balanced workloads
    - – limiting memory may provide a flow control mechanism that avoids overoptimistic execution.

Effect of Limited Memory on Speedup



- symmetric synthetic workload (PHold)
- one logical processor per processor
- fixed message population
- KSR-1 multiprocessor
- sequential execution requires 128 (4 LPs), 256 (8 LPs), 384 (12 LPs) buffers
- 25% to 75% extra buffer and beyond minimum did not improve performance

Performance (CPU) Hazards

- Chasing Down Incorrect Computations
  - » incorrect computation spreads while correcting/canceling erroneous computations
  - » dog chasing its tail
- Rollback Echoes
  - » Expensive rollbacks may cause length of rollback to expand at an exponential rate. Cost of rollback:
    1. Antimessage to all cancelled events
    2. Restore State
    3. Pointer updates of input queue
  - » 1&2 suggest cost is proportional to #events being rolled back
  - » What if rolling back T units of simulated time takes twice as long as going forward by the same amount?
    - – net rate of GVT progress decreases as the simulation proceeds! (this is referred to as an echo).

Other Optimistic Algorithms

Principal goal: avoid excessive optimistic execution

A variety of protocols have been proposed:

- window-based approaches
  - » only execute events in a moving window (simulated time, memory)
- risk-free execution
  - » only send messages when they are guaranteed to be correct (send TS > GVT), eliminates cascaded rollbacks.
- add optimism to conservative protocols
  - » specify "optimistic" values for lookahead

Other Optimistic Algorithms

Principal goal: avoid excessive optimistic execution

A variety of protocols have been proposed:

- introduce additional rollbacks
  - » triggered stochastically or by running out of memory
- hybrid approaches
  - » mix conservative and optimistic LPs
- scheduling-based
  - » discriminate against LPs rolling back too much
- adaptive protocols
  - » dynamically adjust protocol during execution as workload changes

Conservative Algorithms

Advantages:

- Good performance reported for many applications containing good lookahead (queuing networks, communication networks, war gaming)
- Relatively easy to implement
- Well suited for “federating” autonomous simulations, provided there is good lookahead

Disadvantages:

- Cannot fully exploit available parallelism in the simulation because they must protect against a “worst case scenario”
- Lookahead is essential to achieve good performance
- Writing simulation programs to have good lookahead can be very difficult or impossible, and can lead to code that is difficult to maintain

Optimistic Algorithms

Advantages:

- good performance reported for a variety of application
  - » queuing networks, communication networks, logic circuits, combat models
- offers the best hope for “general purpose” parallel simulation software
  - » not as dependent on lookahead as conservative methods
- “Federating” autonomous simulations
  - » avoids specification of lookahead
  - » caveat: requires providing rollback capability in the simulation

Disadvantages:

- state saving overhead may severely degrade performance
- rollback thrashing may occur (though a variety of solutions exist)
- Implementation:
  - » generally more complex and difficult to debug than conservative mechanisms; careful implementation is required or poor performance may result
  - » must be able to recover from exceptions (may be subsequently rolled back)

Summary

- Other Mechanisms
  - » Simple operations in conservative systems (dynamic memory allocation, error handling) present non-trivial issues in Time Warp systems
  - » Solutions exist for most, but at the cost of increased complexity in the Time Warp executive
- Event retraction
  - » Not to be confused with cancellation
  - » Application & kernel level solutions exist
- Optimizations
  - » Lazy cancellation often provides some benefit
  - » Conventional wisdom is lazy re-evaluation costs outweigh the benefits