

Time Warp Simulation on Multi-core Processors and Clusters

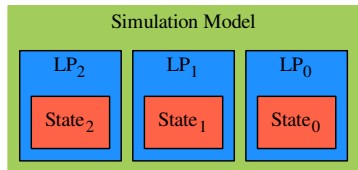
Douglas A. Weber
Masters Thesis Defense
Advisor: Professor Philip A. Wilsey

University of Cincinnati

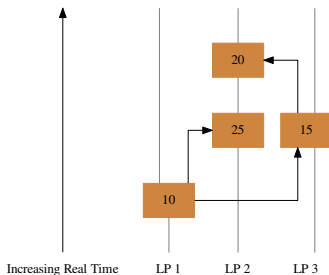
March 8, 2016

- ▶ Three main components
 - ▶ State variables
 - ▶ Simulation clock
 - ▶ Pending event set
- ▶ Unprocessed events stored in pending event set
- ▶ Events processed in time stamp order
- ▶ Simulation clock and state variables updated only when event occurs

- ▶ Model system as a set of Logical Processes (LPs)
- ▶ Events exchanged between LPs

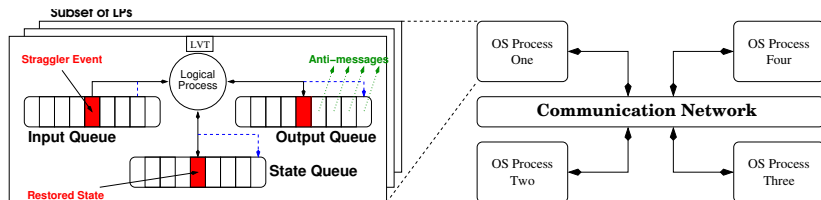


Possible Causality Violations

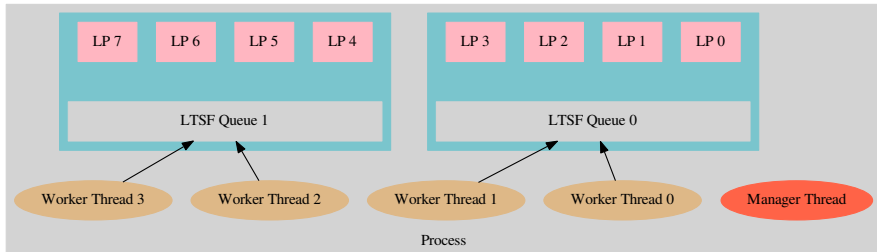


- ▶ Events can be received *and* processed out of order
- ▶ Two solutions
 - ▶ Conservative
 - ▶ Optimistic

Optimistic Mechanism



- ▶ Rollback Mechanism
 - ▶ State Restoration, Anti-Messages
- ▶ Local Virtual Time (LVT) & Global Virtual Time (GVT)
- ▶ Fossil Collection



- ▶ Worker Threads and Manager Thread
- ▶ LTSF Queues/Worker Thread: Contention vs Rollbacks
- ▶ LP Partitioning

Worker Threads

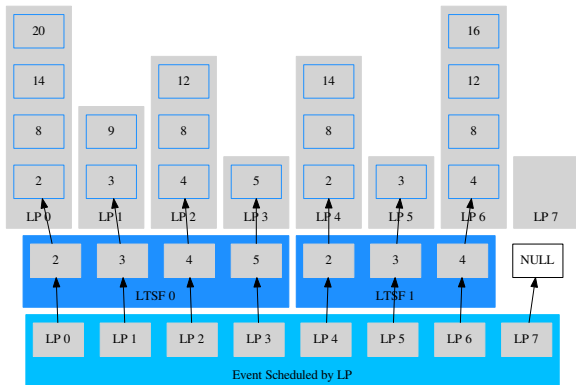
```
while termination not detected do
  e ← getNextEvent()
  lp ← receiver of e

  if e < last processed event for lp
  then
    └ rollback lp

  if e is an anti-message then
    └ cancel event with e
    └ schedule new event for lp
    continue

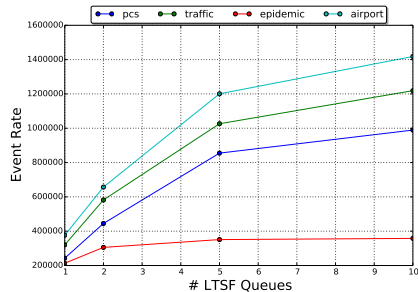
  process event e
  save state of lp
  send new events

  move e to processed queue
  replace scheduled event for lp
```

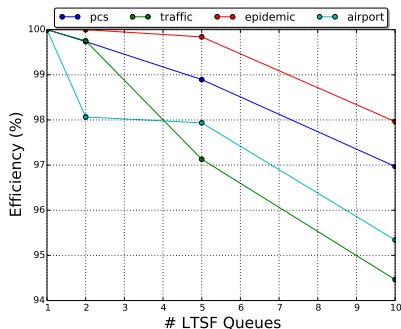


- Unprocessed and Processed Queues
- Scheduling to LTSF
- Rollbacks

Intel® Xeon® X5675 (6 cores/10 worker threads)



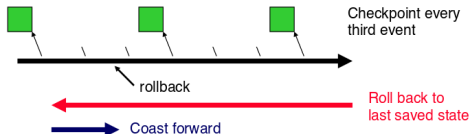
$$EventRate = \frac{CommittedEvents}{Second}$$



$$Efficiency = \frac{CommittedEvents}{ProcessedEvents} (\%)$$

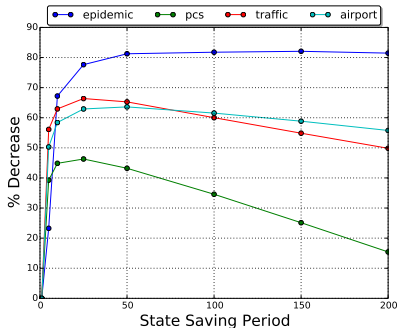
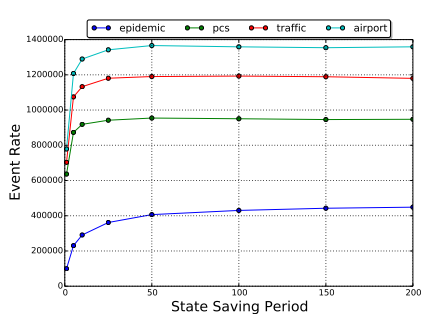
Reducing contention more important than reducing rollbacks

Save state only once every N events



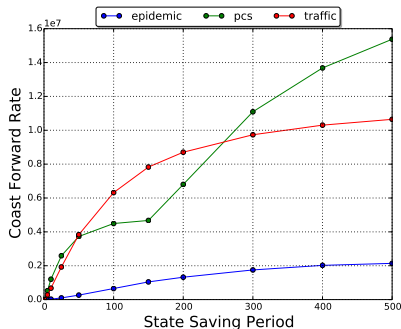
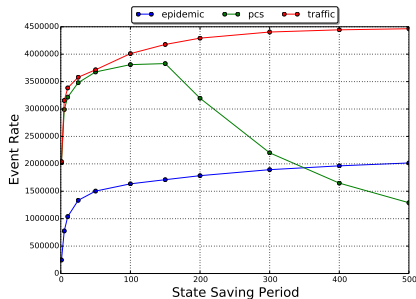
- ▶ Not all states available to roll back to
- ▶ Must “Coast Forward” to reproduce state
- ▶ Increases time to rollback
- ▶ Decrease time to save states and reduce memory footprint

Intel® Xeon® X5675 (6 cores/10 worker threads)



- ▶ Performance flattens
- ▶ Optimal memory footprint

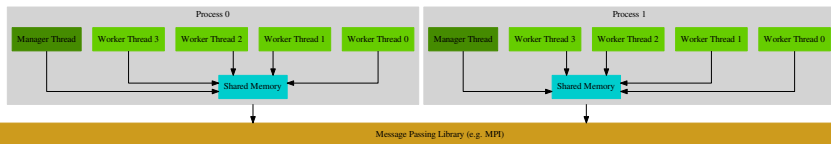
Intel® Xeon® E5410 (8 nodes)



$$CoastForwardRate = \left(\frac{CoastForwardEvents}{Rollbacks} \right) * \left(\frac{Rollbacks}{Second} \right)$$

Shared Memory vs Message Passing

- ▶ Sending Events, GVT, Termination

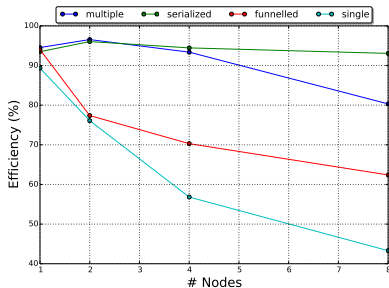
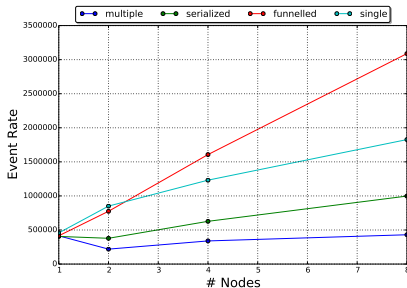


Thread Message Passing Models

- ▶ Single threaded
- ▶ Funnelled
- ▶ Serialized
- ▶ Multiple

Intel® Xeon® E5410 (8 nodes)

- ▶ Single: 1 thread (combining worker/manager threads)
- ▶ Others: 7 worker threads and manager thread



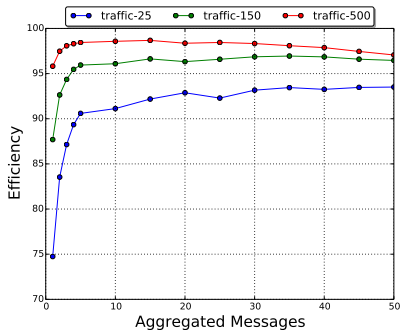
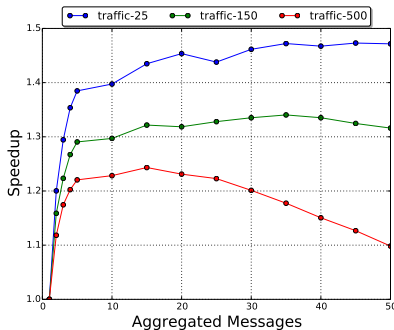
- ▶ Funneled/Single: Minimal Synchronization, High Latency
- ▶ Serialized/Multiple: Lots of Synchronization, Low Latency
- ▶ Performance will vary with model and partitioning.

- ▶ Wait for N messages with same **destination process** and send together

Num Msgs 4 bytes	Length 0 (L[0]) 4 bytes	Length 1 (L[1]) 4 bytes	Length 2 (L[2]) 4 bytes	...	Message 0 L[0]	Message 1 L[1]	Message 2 L[2]	...
---------------------	----------------------------	----------------------------	----------------------------	-----	-------------------	-------------------	-------------------	-----

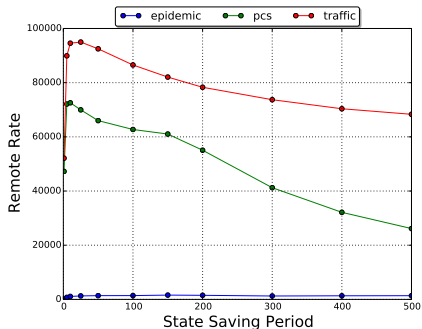
- ▶ Pros
 - ▶ Single buffer allocated for multiple messages
 - ▶ Latency shared by multiple messages (TCP/IP/Ethernet/...)
 - ▶ Events destined for same LP may be sent together
- ▶ Cons
 - ▶ May increase event latency and increase receiver rollbacks

Message Aggregation: Traffic Model

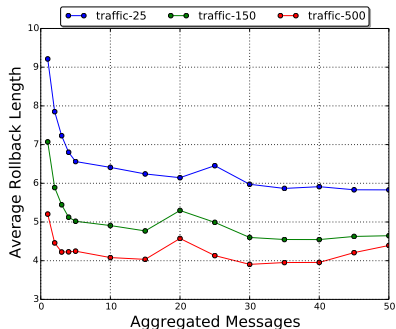


- ▶ Best with small number of aggregated events
- ▶ Different behavior with different state saving periods

Message Aggregation: More Observations



► State saving impacts message rate

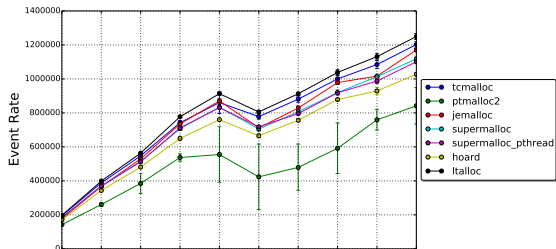


► Can lower rollback length?

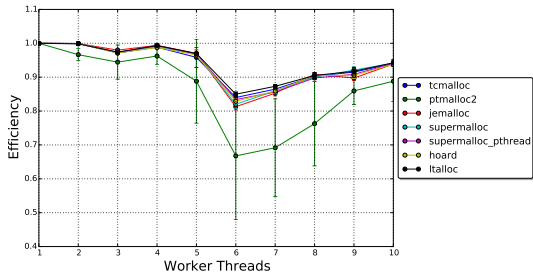
$$\text{AverageRollbackLength} = \text{RolledBackEvents} / \text{Rollbacks}$$

Memory Allocation: Traffic Model

Intel® Xeon® X5675 (6 cores/up to 12 threads)

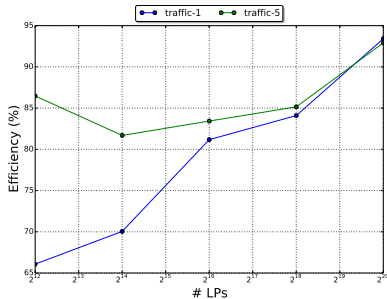
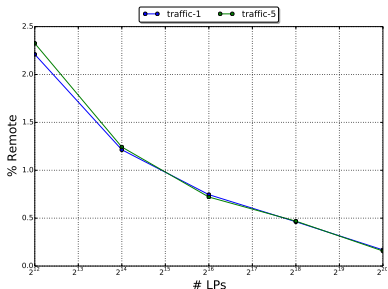


► Imbalance at 6 worker threads



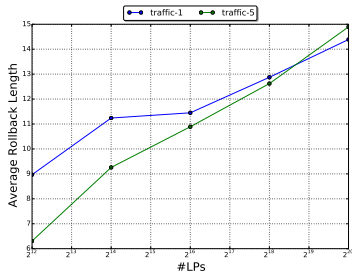
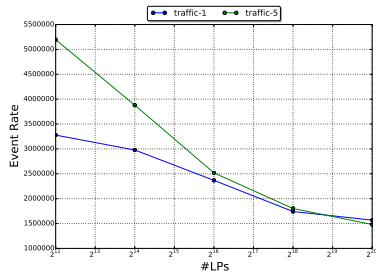
► ptmalloc2 (the default in GLIBC) is inefficient and unpredictable

Intel® Xeon® E5410 (8 nodes/7 worker threads)



- ▶ Coarser Granularity
- ▶ Higher Efficiency

Intel® Xeon® E5410 (8 nodes/7 worker threads)



- ▶ More LPs to fossil collect
- ▶ Longer Rollbacks

Best Time Warp Configurations

	Intel® Xeon® X5675	Intel® Xeon® E5410
State Saving Period	50	150
Aggregate Messages	N/A	15
Worker Threads	10	7
LTSF Queues	10	7
Processes	1	8
Total Worker Threads	10	56

Speedup over Sequential Simulation

	Intel® Xeon® X5675	Intel® Xeon® E5410
Maximum GVT	100,000	
Sequential Runtime	182.11 s	465.95 s
Time Warp Runtime	49.50 s	11.486 s
Speedup	3.68	40.57

- ▶ Conclusions
 - ▶ Synchronization should always be avoided
 - ▶ Interprocess communication: Higher latency over synchronization
 - ▶ 1 LTSF queue per worker thread
 - ▶ Avoid communication ... If possible
 - ▶ If it cannot be avoided, aggregating messages may help
 - ▶ Hard to calculate GVT efficiently
 - ▶ Memory consumption: May need flow control
 - ▶ Alternative multi-threaded memory allocators better than GLIBC default
- ▶ Future Research
 - ▶ Optimistic Fossil Collection
 - ▶ Scaling up