VIRTUALIZING TIME

CS6410

Ken Birman

Is Time Travel Feasible?

- Yes! But only if you happen to be a distributed computing system with the right properties
- Today's two papers both consider options for moving faster than the speed of light in distributed computing settings, without loss of consistency
- They look at a practical issue that also has a nice theory side; we'll focus on these "engineered artifacts" now, then revisit the theory later

Central shared theme

- Time is a kind of performance barrier in many kinds of applications and systems
- Assume a system of many processes or threads that interact via messages or events
  - Any complex distributed application
  - Event driven simulation code
- Both papers ask whether some form of “optimistic” or “speculative” execution can be beneficial

Idea behind speculation

- Suppose we have some task and "think" we have the needed inputs to perform it
  - For example, we have a guess as to the inputs for some computational step in a simulation. We could already start to run that step
  - Or we want to run an application in a risky, not-backed-up mode. If something crashes, we would need to role it back. But checkpoints are costly and the cost of frequent checkpoints would be prohibitive.
- Speculation can let us leverage "spare" CPU power

Computing faster than the speed of light

- Normally, wait for the data before computing
  - Speed of light = fastest that computation can be done with the full data
- But if we can somehow guess the data we can precompute the result
  - If we got things right, we win and break light-speed
  - If wrong... we paid an overhead. But if we had idle CPUs lying around, that cost may be trivial!

Speculation is a broad tool

- Work has been on using speculation to roll back applications that crash on certain inputs
  - Let the application eat the input and run ("optimism")
  - If it succeeds, great...
  - ... but if it crashes, then role it back to the state prior to seeing that input and rerun either
    - Without the input (e.g. sever the connection)
    - Or with the input but in a "careful" mode (i.e. watching for buffer overruns or other kinds of faults)
- Database transactions are a powerful speculation tool, very widely used in large systems
- Speculation is key to speed in modern chips

What limits the potential for speculation?

- In systems that interact with external resources, a speculative action could leave visible traces
  - Consume inputs on I/O channels
  - Display information visible to users
  - Modify files
  - Launch other actions, send messages to other programs, launch the rocket, dispense the cash, etc
- Clearly, when a task is running speculatively we need to prevent these kinds of actions

Speculation: Broad pattern

- Reach a point at which it becomes feasible to start a costly computation a bit early
- Inhibit any outside impacts (ideally including slowdown for the base system)
- Once knowledge is complete, either permit it to make progress, or “erase” the speculated state

Time Warp O/S

- A real system built to support event-oriented simulations of complex systems
- Developed at the NASA research center in California where they do extensive simulations of spacecrafts and other mission-related applications
- Often these have multiple stages that interact via events, so event-based simulation is natural

Core of any event-based simulator

- Create a single big task queue, ordered by time, call it "the worldline of the simulator"
  - Implemented in a distributed way, but if we merge the many event queues, we would have the worldline
- Simulation steps are triggered by events: the first event, at time T_0 is simply "start"
  - Think of events as asynchronously dispatched procedure calls: "compute F(x), then tell me the answer"
  - Reply is an event too
  - Event occurs at a time that would reflect things like network latency, time to operate a camera, etc.

Central problem in time warp

- Suppose that some simulator thread is on a lightly loaded CPU core and basically idle

- The thread knows the current state of simulator component Φ, at time T, and knows of an event that Φ should process (e.g. we should run Φ.F(X)) at time T+δ. When can we safely perform this task?

Possible answers

- We should wait until every simulator component has reached time $T+\delta$:
  - Events are always placed on the worldline "now" or "in the future", at time now+ε for some ε
  - Once every component reaches some point in time, we definitely know the correct inputs to every action that should occur at time $T+\delta$.
- But this is very conservative and may leave our system mostly idle

Why “mostly idle”?

- Recall that this is a *simulator*, not a real system

- Suppose that simulation of some physical thing is just very costly to do
  - Perhaps, it is very costly to accurately simulate deployment of the Mars rover parachute at ballistic speeds in the upper atmosphere
  - A computation that takes hours to compute seconds of physical events

Impact of that slow task?

- All the other threads in our simulator pause waiting
- This is true even if it is unlikely that the parchute simulation task would cause events relevant to them
- E.g. is the “warmup the hover engine fuel unit” task likely to be impacted by the parachute task?
  - Perhaps, if there is a violent turbulence event
  - But probably not “often” or “usually”

So our vision is of...

- ... a massively parallel simulation, with thousands of hard computational tasks ready to run
- ... all waiting for the right time at which to run
- ... and yet one might be able to demonstrate that in general, they waited for hours of real-time only to run in the same state they were in hours earlier!

(back to) Possible answers

- This leads to option 2

- Suppose we simply run a task as soon as we can
  - Every component eagerly runs, acting as if the events *currently* on the worldline are the full event set
  - Thus we could aggressively run the engine warmup simulation for time T+δ *right now* if we think we know the state prior to T+δ.

Issues with option 2?

- It definitely keeps our HPC system humming!
- But our speculation may have been wrong, perhaps an event relevant to the engine warmup simulation task does occur
  - Perhaps, the "heat shield detach" task simulates breakup of a portion of the shield
  - A fragment-trajectory task runs and decides that this fragment hits an engine fuel-line component
  - That fuel-line simulation task decides that the fuel-line is now dented and has reduced flow capacity
  - This would matter to the engine warmup task so an event is sent to it at time $T+\alpha$, for some $\alpha<\delta$
- Our simulation of the engine warmup state was incorrect!

Proposed solution?

- We can just roll back the warmup simulation
  - Discard any state it created when speculatively running
  - Restart it in the prior state, but now run the event at time T+α
  - Unsend any messages it sent to other simulation components: send "anti-messages" on the worldline

- How do these work?
  - If the worldline has a message m and an anti-message ~~m~~ shows up, m and ~~m~~ cancel each other
  - If m was already consumed, ~~m~~ triggers a rollback by the consumer task, and so forth
  - Clearly, need a way to "name" events that will be unique and unambiguous, and to track the history of events
  - Time Warp O/S has an event-naming scheme that solves this

Visualizing a time-warp execution

- Waves of speculative execution overlap with waves of rollback
  - A speculative task runs, sending events to other tasks
  - These events trigger more speculative work
- Meanwhile, as slow tasks ("laggards") send events, rollbacks can be triggered, spawning waves of antimessages that in turn trigger further rollbacks
- Does progress occur at all? Or can this degenerate into a chaotic state?

Theory of Time Warp

- Jefferson argues that in fact, his system state can always be topologically sorted: a kind of forest of rooted trees (some inner nodes may have multiple roots, of course)
- Focus on those roots: tasks that cannot be forced to roll back because they are at the earliest clock time known in the system
  - To make these roll back, an event from the past would need to arrive. But no active task could generate such an event
  - Analysis can be extended to deal with events still in the communication channels of Time Warp

Downside to speculation?

- When we run a task in a speculative mode, we consume many kinds of resources
  - Network traffic is created, hence network is slower
  - Disk I/O occurs, hence disk will be less responsive for other uses
  - We create intermediary state checkpoints, which can be large and slow to store
  - We need lists of events that were consumed, and shadow copies in order to "unconsume" them if a rollback occurs
- Moreover, if a rollback occurs, this has costs too

I/O?

- One practical challenge involves files and other I/O occuring during the simulation run
  - Time Warp treats the file system itself as a kind of event-driven simulation task
  - Events create files, modify, delete them
  - Permits the same model to deal with elimination of files created speculatively, or modified speculatively
  - But does require that Time Warp keep logs of old versions or deltas, for use in rolling back, and these can easily get very large

Worst case?

- It is easy to see how Time Warp can spawn a kind of exponentially branching world of speculative activities that all will need to roll back
- Should this occur, we'll spend all our time speculating and rolling back, none of our time doing useful simulation!
- Moreover, tasks will run slower because of cold caches: speculation/rollback will often flush caches simply by overwriting contents with other stuff used speculatively

Time Warp practical challenges

- Trick is to speculate wisely
- Today we would see this as a machine-learning problem:
  - Training data: For a given simulator, learn to categorize the tasks and, for each category, learn the probability of rollback for that category
  - Use of the training set: At runtime, only speculate if the predicted value greatly outweighs the predicted risk
- At the time, Jefferson used heuristics

Time Warp practical challenges

- Garbage collection is a big issue
- Time Warp accumulates a LOT of speculative state and rollback state
  - Task states, file system deltas, shadow copies of messages that have been consumed but might still be unconsumed, etc
  - Clearly a garbage collector is required
- Solution: periodically compute the "oldest time active in the system". Safe to garbage-collect rollback data *older than this computed time*: they won't be needed

Results obtained?

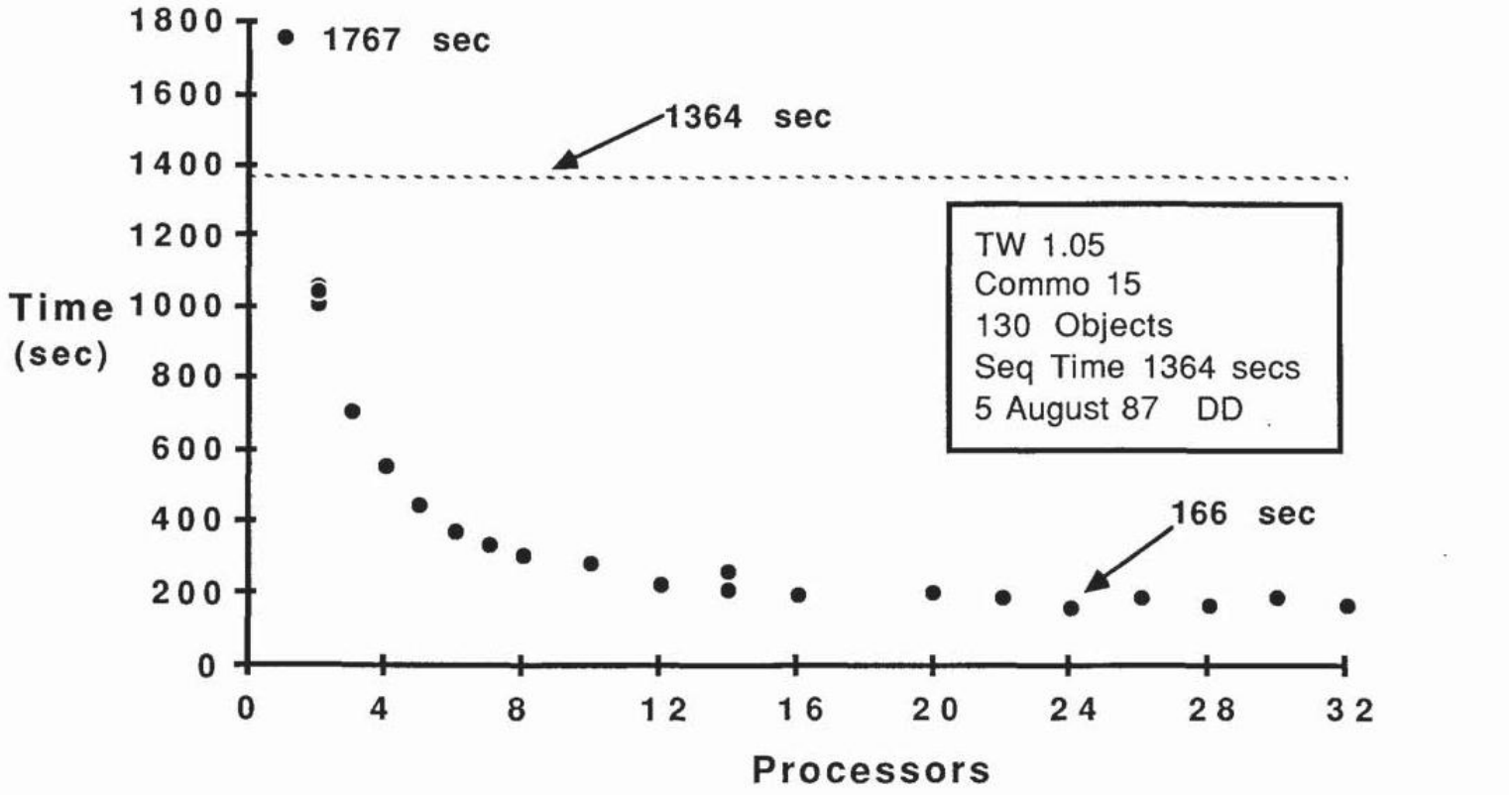
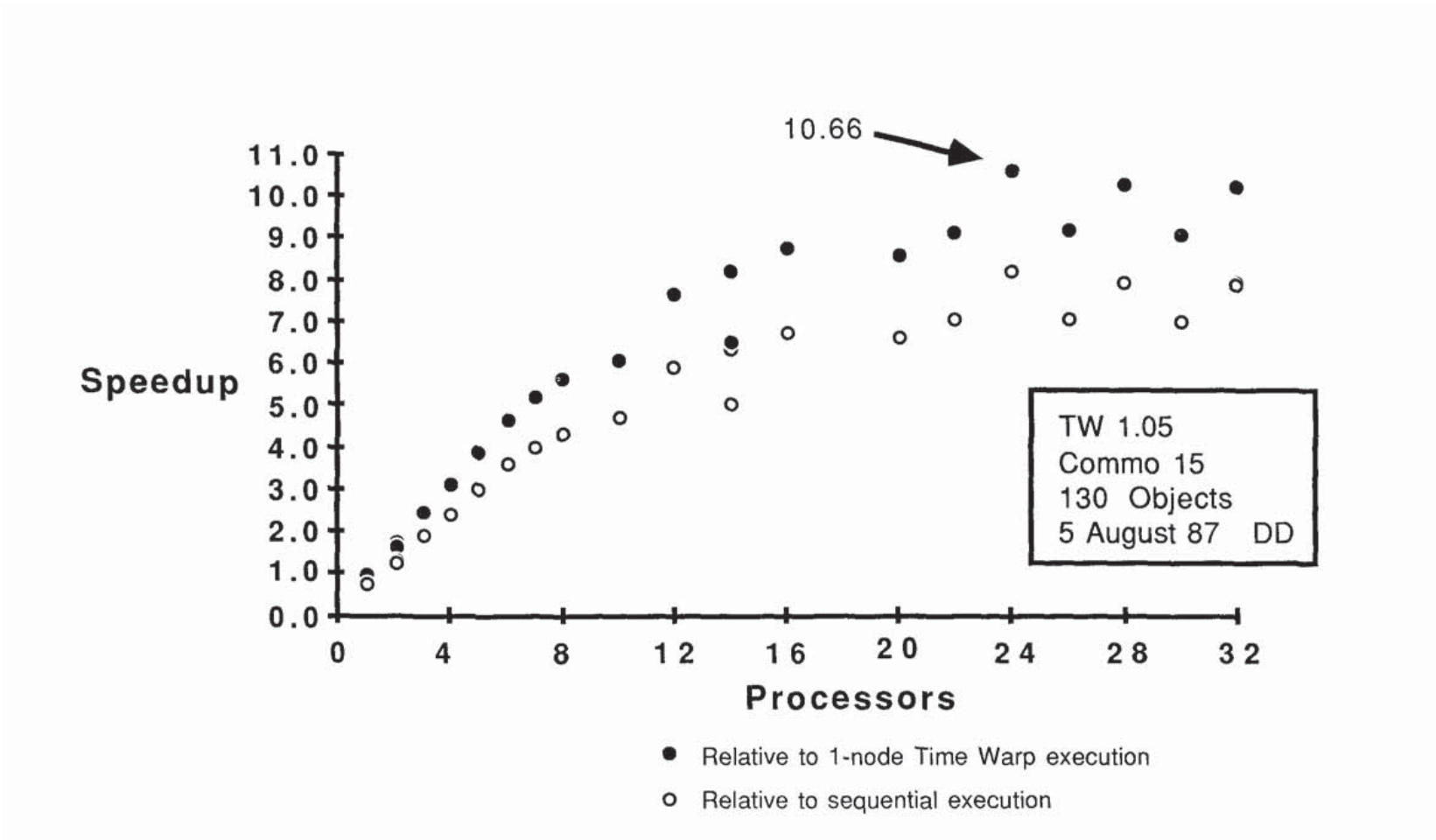
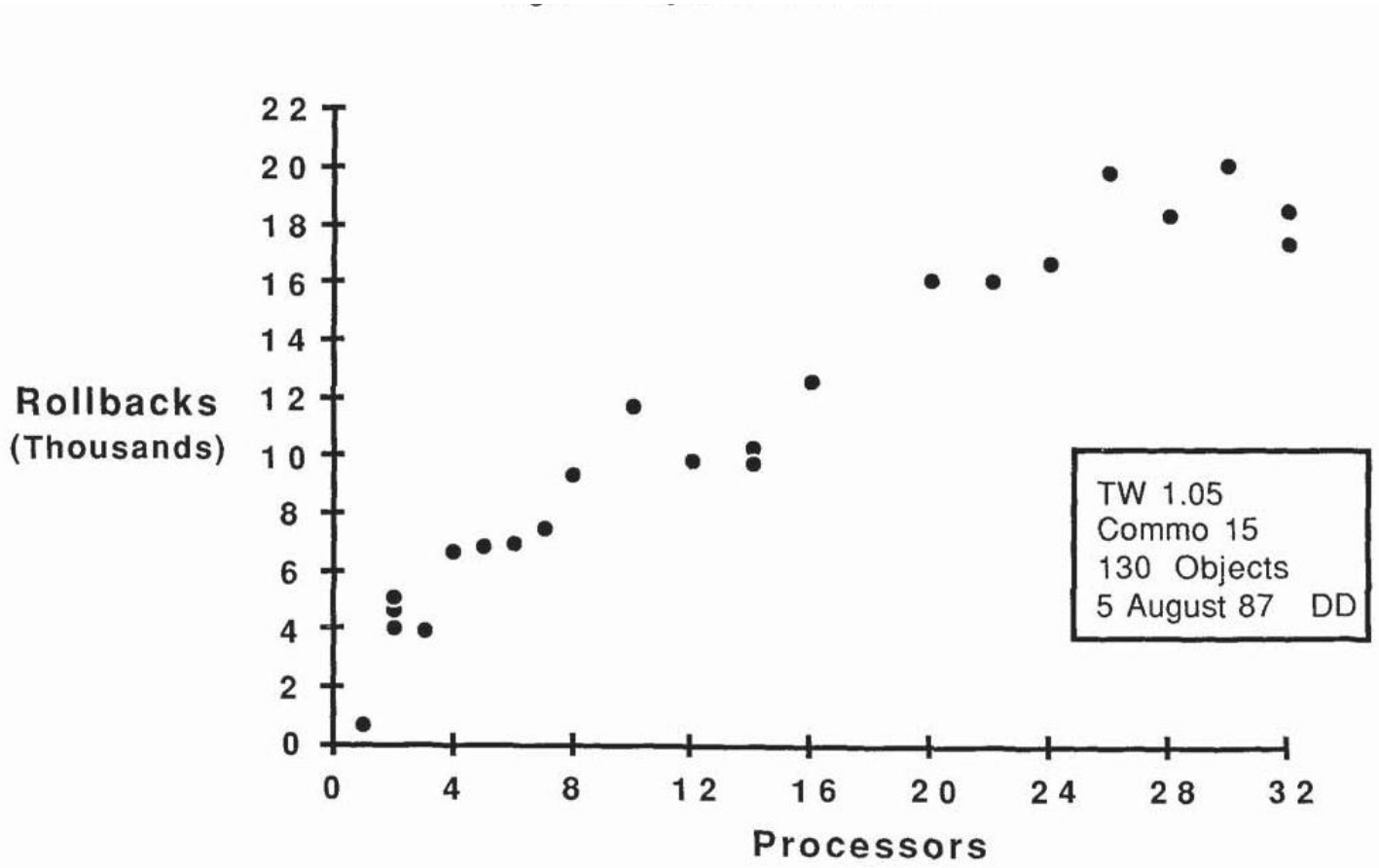


Figure 7: Time to completion of the irregular model COMMO*

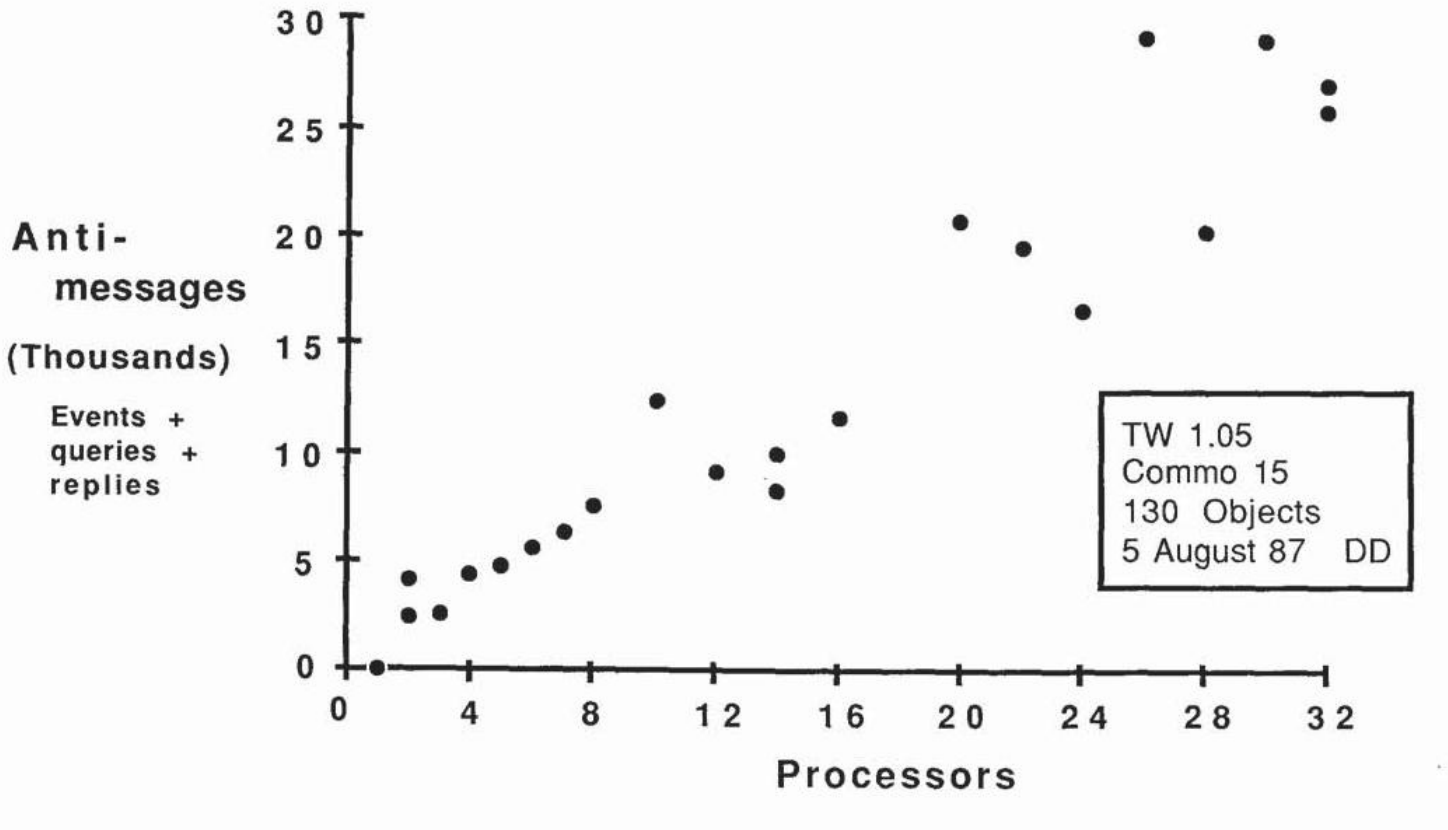
Results obtained?



Results obtained?



Results obtained?



Applying similar ideas in real systems

- Time Warp was all about simulation
- But many real systems have lots of moving parts
  - Today's cloud computing systems are built by scripts that assemble complex components from smaller building blocks: objects that interact via events
  - Situation is this very much like Time Warp
- Do real systems present speculation opportunities?

Logging

- Main issue in Alvisi's paper centers on failures
  - Suppose a componentized system only makes checkpoints rarely, and components run by event-passing
  - How should we track "speculative states" that components enter, and how can they roll back when needed?
- He points out that there are three choices
  - Checkpoints
  - Sender-maintained event (message) logs
  - Receiver-maintained event logs
- Proposes a universal speculation model

What triggers rollback?

- For Alvisi, main concern is failure
  - If some component crashes and we want to restart it, perhaps in a slightly "fixed" state, we need to roll back to a prior checkpoint state
  - His goal is to ensure system-wide consistency by a mechanism similar to the one in Time Warp
    - Anti-messages that anihilate the prior message if it is still in the queue of the receiver, else trigger a new rollback
    - In general, we have several rollback options

Alvisi observations

- Treat interaction with the outside world as a special situation in which we need to be in a safe (non-speculative) state
  - External-interaction "barrier"
  - Must also treat interactions with many O/S features (clock, fork(), pipe I/O, ...) as managed events
- Employ aggressive task scheduling when risk of rollback seems sufficiently low.

Alvisi approach

- Seeks to offer a flexible framework in which we
  - Use rollback to a checkpoint if doing so is likely to have the least cost
  - Send anti-messages to trigger a more selective rollback if logs are available and the corresponding messages have not yet been consumed
- This unified model is elegant, but it does assume that messages linger on input queues for a long enough period of time to offer a benefit

Results obtained

- Theory of message logging
- Coverage of a wide range of possible behaviors including sender logging, receiver logging, checkpoints
- Careful proofs showing precisely when it is safe to garbage collect data, and precisely when a computation must roll back (and to what point)

Discussion

- Speculation buys us a great deal within model CPU architectures
  - Branch-prediction
  - Speculative cache prefetching
- Permits the chip designer to exploit parallelism at the lowest levels, but does require barriers to prevent speculative states from leaking into main memory

Speculation in protocols

- When we looked briefly at Van Renesse's Horus architecture, we saw another use of speculation
  - Pre-execute the non-data-touching code for the next message, under assumption that next event will be a message
  - If the Horus protocol stack sees some other event, Van Renesse had to roll back to a pre-speculation state

Speculation in systems

- Time Warp and Alvisi's unified logging scheme are just two of many examples
  - Speculation can be used in a file system, by guessing that if file A was touched, file B will be needed, and prefetching file B
  - Some modern systems use extra cycles to make a spare copy of B next to A, for quicker access if needed: a speculative form on on-disk replication
  - Speculation to consume a risky incoming message, roll back if it turns out to cause a crash or contain a virus

Broad tradeoff

- When we look at modern systems we often see large portions sitting idle waiting for permission to take some action
- Broadly speaking, speculation is about machine learning: if we can accurately predict what will happen next, we can sometimes jump the gun
- Cost is the overhead of rollback when needed