Parallel Distributed Simulation

Sijian Tan

1 Traffic in Urban Area (Fall 2023)

Consider a discrete-event simulation application for studying traffic in a large-scale urban area, such as Los Angeles or Tokyo. Suppose the simulation directly models the road network, where each road segment is a logical process (LP) and events correspond to vehicle activities like arriving at a segment. (That is, the LP for, say, road segment A might process an arrival event with timestamp t for a car that arrives at one end of the road segment. That LP might then schedule a new arrival event for the same vehicle to arrive at a connected road segment B at some future timestamp $t + \Delta t$.)

Suppose this simulation has already been implemented on top of a simulation executive (or simulation engine) that uses the optimistic Time Warp synchronization protocol. Now suppose we wish to model the effects of a sudden unexpected event, like an earthquake: at an arbitrary point in the simulation, the earthquake "occurs" by taking out certain road segments. If this damage occurs at logical simulation time t_e, then any events scheduled or executed by the affected road segment at times $t > t_e$ must be undone.

(a) Explain how you would change the simulation application (not the underlying simulation executive) to implement this kind of earthquake phenomenon. (We want to see that you understand how optimistic synchronization works by exploiting its properties for this task.)

(b) Suppose you are allowed to modify the simulation executive. What would you do differently to accomplish this task?

(c) What are the strengths and weaknesses of your two approaches?

1.1 Question a

To implement an event to cause a rollback in **Time Warp algorithm**, we should take the following steps:

- **Define earthquake event:** Introduce a new event type specifically for earthquakes. This event should contain information about the timestamp at which the earthquake occurs and the road segments that will be damaged as a result.
- **Rollback mechanism:** For events scheduled after the earthquake on damaged segments, initiate a rollback using the following steps:

  - Identify the events to be rolled back.
  - Send anti-messages for these events to cancel them.
  - Revert the state changes made by these events, effectively undoing them.

- **Rescheduling:** After rolling back, reschedule the events by considering the new state of the road network. Events that were supposed to be processed on the damaged road segments will need to be discarded or rerouted.

1.2 Question b

To implement an event in **conservative synchronization**, we should take the following steps:

- **Define earthquake event:** Create a new event type for earthquakes with relevant details (timestamp, damaged segments).
- **Implement conservative synchronization mechanism:** Ensure that LPs process events based on the LBTS (Lower Bound on Time Stamp, or minimum timestamp of safe events), ensuring they do not process events out of order. Use lookahead values to help determine the LBTS, which helps in deciding the safe events to process.
- **Blocking and releasing LPs:** If an LP is responsible for a damaged segment, it blocks further processing of events related to that segment until it can be sure that it is safe (i.e., no earlier events are in the system).
- **Safe state propagation:** Propagate information about the earthquake and damaged segments to all LPs. Each LP updates its event queue, removing or adjusting events that are no longer valid due to the earthquake.

1.3 Question c

The comparison between optimistic and conservative synchronization: Here.

2 Radio Communication (Fall 2023)

Consider a distributed simulation of a large number of people who are moving around and communicate using wireless radios. Each person carries a radio, and communication takes place whenever two radios are within some fixed distance (the range of the radio) of each other. Assume each individual is modeled by a distinct logical process (LP). Two LPs exchange messages (events) when the individuals are within radio range of each other. Would it be better to simulate this system using the conservative Chandy/Misra/Bryant protocol or the optimistic Time Warp? Explain your choice giving at least two reasons why.

2.1 Solution to Use Time Warp

Based on the prompt, this system is a highly dynamic system. First, we analyze the **CMB protocol**:

- **Causality Preservation:** The conservative protocol strictly avoids causality errors by ensuring that no event is processed out of order. Each logical process (LP) waits until it is guaranteed that no earlier events will arrive before processing the next event. This makes the simulation deterministic and avoids the need for rollbacks.
- **Communication Overhead:** In a scenario where people move and communicate based on proximity, ensuring that events are processed in the correct order can lead to significant waiting times. LPs must continuously check and communicate to ensure no causality violations occur, which can lead to inefficiencies, especially in highly dynamic environments where the radio range interactions frequently change.
- **Resource Utilization:** The conservative protocol may lead to **underutilization** of computational resources due to blocking and waiting. In a highly dynamic system, this waiting can significantly impact performance.

Now we take a look at **optimistic Time Warp**:

- **Parallelism and Resource Utilization:** The optimistic protocol allows LPs to process events speculatively without waiting for guarantees that no earlier events will arrive. If a causality error occurs, the protocol rolls back to a previous state and corrects the error using anti-messages. This approach can fully utilize computational resources and exploit parallelism, leading to potentially higher performance in dynamic systems.
- **Handling Dynamic Interactions:** In a system where individuals frequently move and enter or leave each other's radio range, the optimistic protocol can handle dynamic changes more fluidly. While rollbacks may occur, the system can quickly adapt to changes without the significant delays caused by waiting in the conservative approach.

In conclusion, we should choose optimistic Time Warp.

3 Lookahead (Spring 2023)

The efficiency of any conservative scheme for parallel discrete-event simulation is limited by *lookahead*.

(a) Explain what lookahead is and why it limits efficiency.

(b) Give examples of four different real-world simulation problems that illustrate how lookahead affects efficiency. At least one example should show an instance of lookahead enabling abundant parallelism, and at least one example should show

an instance of lookahead severely restricting parallelism. For full credit, include an example where lookahead might vary within the same simulation; furthermore, don't just write in general terms, but try to be specific and give illustrative examples. (For instance, to justify that a given simulation will experience lookahead that either limits or enables parallelism, you will need to be specific about the time-scale of lookahead compared to other features of the system being simulated.)

3.1 Question a

Based on the definition, lookahead refers to the minimum amount of simulated time into the future that a logical process (LP) can guarantee it will not send a message to another LP. This guarantee allows LPs to process events safely without waiting for potential earlier messages, thus enabling parallel execution.

If lookahead is small or zero, LPs must frequently wait to ensure no earlier events will arrive. This waiting reduces parallelism and increases idle time for processors.

3.2 Question b

3.2.1 Traffic Simulation

- **Scenario:** Simulating traffic flow at a busy intersection.
- **Large Lookahead:** If lookahead is large (e.g., a traffic light cycle duration), LPs representing different segments of the road network can process events independently for the duration of the light cycle, enabling abundant parallelism.
- **Small Lookahead:** If lookahead is small (e.g., cars approaching the intersection need constant updates), LPs frequently wait for updates on car positions, severely restricting parallelism.

3.2.2 Manufacturing Simulation

- **Scenario:** Simulating an assembly line with robotic arms.
- **Large Lookahead:** If lookahead corresponds to the fixed cycle time of robots (e.g., robots perform actions in synchronized steps), LPs can process several steps in parallel, enabling high parallelism.
- **Small Lookahead:** If lookahead is limited by the need for real-time adjustments (e.g., responding to sensor data), LPs must frequently wait for updates, reducing parallelism.

3.2.3 Battlefield Simulation

- **Scenario:** Simulating movements and interactions of units in a military exercise.
- **Large Lookahead:** In structured movements (e.g., units moving in formation with predetermined plans), lookahead can be large, enabling LPs to process events in parallel for extended periods.

- **Small Lookahead:** In dynamic, reactive scenarios (e.g., units responding to enemy movements), small lookahead due to constant interaction and reaction limits parallelism.

3.2.4 Weather Simulation (Varying Lookahead)

- **Scenario:** Simulating weather patterns over a large region.
- **Large Lookahead:** In stable weather periods (e.g., high-pressure systems), lookahead can be large because changes are slow and predictable, allowing parallelism.
- **Small Lookahead:** During rapidly changing weather (e.g., storm fronts), lookahead becomes small as frequent updates are needed, limiting parallelism.

4 Memory Management (Spring 2023)

Consider an optimistic parallel discrete-event simulator based on the classical TimeWarp algorithm. A critical issue is memory use, which can grow in an unbounded fashion.

(a) Explain how memory can grow in this way, that is, when and why would memory use grow without bound?

(b) Describe how to modify TimeWarp so that it can "gracefully degrade" as memory grows. That is, suppose the user supplies an *a priori* memory limit; then, rather than crashing the program when the simulation's memory exceeds this limit, your modified synchronization scheme allows the simulation to recover and proceed, albeit possibly more slowly.

4.1 Question a

The memory use can grow without bound due to the following reasons:

- **State Saving:** The Time Warp algorithm maintains multiple copies of the simulation state to allow for rollbacks. Each logical process (LP) periodically saves its state. As the simulation progresses, these state copies accumulate in memory.
- **Anti-Messages:** When a rollback occurs, anti-messages (which cancel previously sent messages) must be managed. Both the original messages and their corresponding anti-messages consume memory.
- **Rollback Frequency:** Frequent rollbacks can exacerbate the problem, as more states and events need to be saved and managed. If rollbacks occur frequently due to high levels of causality violations, the memory footprint can grow rapidly.

4.2 Question b

When memory reaches the limit, we could calculate the GVT and then free up the memory for the previous timestamp. Here are the details:

- **Monitor memory usage:** Implement a mechanism to continuously check the current memory usage against the predefined limit.
- **Calculate GVT:**
  - Each LP reports the minimum timestamp of its unprocessed events.
  - Include the minimum timestamp of all messages in transit.
  - Compute the global minimum of these values.
- **Discard Obsolete States and Events:** Once GVT is calculated, iterate through the stored states and events, and remove those with timestamps less than the GVT.

5 FIFO Queue (Fall 2022)

In this problem, you will develop a scheme to avoid the problem of *deadlock* that can occur in a parallel discrete-event simulation.

Recall how deadlock can arise in the basic structure of a simulator consisting of P logical processes (LPs) where any LP may generate remote events for any other. Assume each LP maintains a separate first-in, first-out (FIFO) queue for receiving event messages from every one of the other LPs; and suppose each LP executes the canonical simulation loop shown in Algorithm:

Algorithm 2.1 Canonical discrete-event simulation loop, executed by every logical process

Let `now` be the current logical simulation time on the executing LP.

```
1  while simulation is not over do
2      Wait until each incoming FIFO queue has a message event.
3      Remove the message event M with the smallest timestamp.
4      now := timestamp of M.
5      Process event M.
```

Figure 1: FIFO Queue Algorithm

Line 2 can lead to deadlock: there can be a cycle of waiting among the LPs. While one can use a *null message* synchronization scheme (e.g., Chandy/Misra/Bryant) to avoid deadlock, the performance of such schemes depends strongly on whether the simulation can exploit *lookahead*. (Lookahead refers to whether every LP can reliably estimate a lower bound on the timestamp of the next message that it might send to each of the other LPs.)

Instead of using null messages, suppose you are allowed to use a separate central process to coordinate the action of all LPs. Assume this central process does not otherwise participate in the simulation, i.e., it does not process any events.

(a) Suppose this central process can *detect* whether the simulation is in a deadlocked state. In particular, suppose it knows which LP has the event with the smallest timestamp. Explain how it can recover from the deadlock, i.e., enable execution of the simulation to resume.

(b) Give a scheme to detect deadlock. You must assume LPs execute *asynchronously*. However, you may assume that each LP has an additional FIFO queue for communicating with the central process.

(c) Analyze the tradeoffs of the overall deadlock detection and recovery scheme compared to a null message scheme like CMB.

5.1 Question a

Deadlock detection process:

- The central process monitors the state of all LPs.
- It identifies a deadlock when all LPs are waiting and no events are being processed.
- The central process keeps track of the smallest timestamp event in the system.

Deadlock recovery process:

- Once deadlock is detected, the central process will broadcast the global minimum time back down to all LPs.
- All processors with an event(s) with time stamp equal to this global minimum can safely process that event.

5.2 Question b

The scheme could include:

- Each LP has an additional FIFO queue for communication with the central process.
- LPs periodically send their status (e.g., waiting, processing) and the smallest timestamp of their events to the central process.
- The central process continuously monitors these messages.
- When the central process detects that all LPs are in a waiting state and no new events are being processed, it identifies this as a deadlock situation.
- The central process then follows the recovery steps.

5.3 Question c

The advantages of the deadlock detection and recovery scheme include:

- **Efficiency:** By using a central process to monitor and recover from deadlock, the simulation can potentially avoid the overhead associated with null message schemes.
- **Simplicity:** This approach can be simpler to implement and reason about compared to more complex null message schemes.

There are also some disadvantages:

- **Scalability:** The central process could become a bottleneck in large-scale simulations with many LPs.
- **Single Point of Failure:** If the central process fails, the entire simulation could be compromised.

5.4 Deadlock in CMB

Deadlock in the CMB algorithm could still occur, when **all LPs are waiting for null messages from other LPs before proceeding, creating a cycle of dependencies with no LP able to advance.** This is usually caused by **cyclic dependence** and **inadequate Lookahead**.

There are several ways to avoid deadlock in CMB:

- **Increasing Lookahead:** By increasing the lookahead value, LPs can reduce the frequency of waiting for null messages, thereby reducing the chances of forming cycles.
- **Deadlock Detection and Recovery:** Implementing a deadlock detection mechanism where a central process or distributed agreement protocol detects and breaks deadlocks by allowing one or more LPs to proceed.

6 LCC (Spring 2022)

Problem setup: Consider a parallel discrete-event simulation consisting of logical processes (LPs) that interact exclusively by exchanging time-stamped messages. We regard the simulation as correct if it processes the events in a manner consistent with a sequential execution: a correct sequential execution would process those events in nondecreasing timestamp order, assuming unique timestamps. To attain correctness in the parallel case, the usual condition we try to enforce is the *local causality constraint*: each LP only processes events in nondecreasing timestamp order.

Suppose you are building a parallel discrete-event simulator but your customer requires *repeatable results* across the simulation runs. That is, if the customer runs your simulator twice—each time with the same external inputs (e.g., initial events), the same

initial state, and the same number of both logical processes (LPs) and physical processes (e.g., compute nodes or processors)—then she expects the final results of the two runs to be the same.
Questions: Is enforcement of the local causality constraint sufficient to ensure repeatability? Why or why not? If you believe it can be sufficient, explain what assumptions are needed to justify that claim. If you do not believe it is sufficient, explain what else your simulator needs to do to make its runs repeatable. (You should always assume that the simulation is stochastic, meaning any event handlers may invoke pseudorandom number generators.) If you believe the answer depends on whether the synchronization scheme is conservative or optimistic, explain the two cases separately; if you believe it does not, explain why not.

6.1 Sufficiency

Based on the definition, the local causality constraint (LCC) mandates that each logical process (LP) run the events in nondecreasing timestamp order. This ensures that the order of event processing within **each LP** is consistent with a sequential execution.

However, for the whole system, LCC **is not sufficient** to ensure repeatability. While it ensures that each LP processes events in a consistent order, **it does not guarantee that the parallel execution of multiple LPs will be deterministic or consistent across runs.**

6.2 Assumption

To make LCC to be sufficient, additional assumptions must be made:

- **Deterministic Event Handling:** The event handling within each LP must be deterministic. This means that given the same sequence of events, the LP should always produce the same output.
- **Synchronization Scheme:** If the synchronization scheme itself introduces nondeterminism, the results may vary across runs.

6.3 Synchronization

For conservative synchronization:

- **Deterministic Order:** In a conservative synchronization scheme, LPs only process events when it is safe to do so, ensuring that no causality errors occur. This scheme can lead to deterministic execution if the order of events and the safety conditions are deterministic.
- **Potential for Repeatability:** If all safety checks and event processing are deterministic, conservative synchronization can help achieve repeatability.

For optimistic synchronization:

- **Rollback Mechanism:** In an optimistic synchronization scheme, LPs process events speculatively and roll back if causality errors are detected. This introduces a higher degree of nondeterminism due to the potential for multiple rollbacks and reprocessing of events.
- **Challenges for Repeatability:** The inherent nondeterminism in rollbacks and event reprocessing can make it challenging to achieve repeatability.

6.4 Techniques to Ensure Repeatability

There are some techniques:

- **Deterministic Pseudorandom Number Generators:** Use deterministic pseudorandom number generators with fixed seeds to ensure that any stochastic behavior (e.g., event generation) is consistent across runs.
- **Consistent Event Scheduling:** Implement a deterministic event scheduling mechanism to ensure that the order of events processed is the same in each run.
- **Controlled Synchronization:** Ensure that the synchronization mechanism (conservative or optimistic) is implemented in a deterministic manner, with consistent handling of events and rollbacks.

7 Error Handling (Spring 2022)

Consider a parallel discrete-event simulation that implements an optimistic synchronization protocol, such as TimeWarp. A tricky issue is *error handling*, for errors like dividing by zero or executing an out-of-bounds array reference (e.g., reading or writing to an array location `x[i]` for an invalid `i` value). In particular, if an error occurs during the execution of an event handler, the program should **not** abort if that event is later rolled-back. (In such a case, the execution should appear as though the event *never* executed.) Explain what you might do to make an optimistic simulator more reliable, in the sense of reducing or eliminating the possibility of fatal program errors during events that are later rolled back. To help answer this question, you might consider different types of errors that could occur and give examples of strategies to handle those cases.

7.1 Solution

The possible errors include:

1. **Program Detected Errors:** These are errors in logic and inconsistencies detected by the simulation program itself. For example, it may be that a certain state variable should never take on a negative value. This is the most straight forward one.

   Solution: Time Warp executive can provide an abort primitive that marks the logical process as being in an error state so that no more events are processed by that TWLP. If the error is erased by a rollback, the process could be returned to the normal state. If the error is committed, the program is then aborted.

2. **Infinite Loops:** It's possible that the system will be stuck into an infinite loop, normally in Time Warp algorithm.

   Solution: a mechanism is required to break out of the loop. This could be accomplished by using an interrupt mechanism to handle incoming messages. This call must check to see if any messages were received that would roll back the event that is now being processed. If such an event is detected, the processing of the current event should be aborted, and the rollback processed.

3. **Benign Errors:** These are errors that do not result in the incorrect modification of any memory other than checkpointed state variables. Examples include:

   - Dividing by zero
   - Taking the square root of a negative number
   - Illegal memory reference

   Solution: A mechanism is required to pass control to the Time Warp executive when such errors occur. Support must be provided by the operating system or the language's runtime system. Once the error has been detected and control is passed to the Time Warp executive, the error can be handled in the same way as a program-detected error.

4. **Destructive Errors:** Destructive errors are those that result in the modification of state that is not checkpointed. For example, a pointer reference could modify internal data structures within the Time Warp executive. Because TW is usually viewed by the operating system as part of the application program, such an error would occur if the program attempted to overwrite a memory location within the operating system. These are the most problematic types of errors because once the state has been erroneously modified, it is difficult to recover.

   Solution_1: A brute force solution is to checkpoint all memory that could be modified by the application without causing a trap in the operating system, such as all state used by the TW executive.

   Solution_2: Provide explicit runtime checks for such errors, effectively converting the error to a program detected error.

8 Exploit Property (Fall 2021)

One approach to modeling a system is to use a set of real devices for some components of the system and have them interact with simulation models of other components. For example, a model for a telecommunication network might include actual network components (routers, traffic generators, etc.) interacting with simulated routers and simulated traffic generators.

Suppose one wishes to use a parallel simulator based on Time Warp for the simulated portion. Describe at least two important challenges that must be overcome to use Time Warp in this fashion, and suggest an approach to overcome each of these challenges. Consider a parallel discrete-event simulator that uses the Time Warp algorithm on a class of applications with the following property: when the computation rolls back

and re-executes events, it is often—but not always—the case that the same events (messages) will be scheduled during the re-execution phase as were scheduled in the original execution.

Describe a scheme for Time Warp that can exploit this property to execute more efficiently than simply re-executing. Then give pseudocode for a Time Warp executive that implements your scheme.

8.1 Challenges

The challenges of using Time Warp with **real and simulated components:**

1. **Synchronization Between Real and Simulated Components:**
   - **Challenge:** Real devices operate in real-time, while simulated components can run faster or slower depending on computational resources. Synchronizing the two requires careful management to ensure that interactions occur at the correct logical time.
   - **Approach:** Implement a time management mechanism that aligns the logical time of simulated events with the real-time progression of actual devices. This might involve slowing down or pausing the simulation to wait for real-time events or buffering real-time events to be processed by the simulation at the appropriate logical time.
2. **State Consistency and Rollback Management:**
   - **Challenge:** The Time Warp algorithm uses rollbacks to correct causality errors, which can complicate the interaction with real devices that cannot be rolled back
   - **Approach:** Use checkpointing and state-saving techniques to manage the state of simulated components. For real devices, design the system to minimize the need for rollbacks or to handle rollbacks in a way that does not disrupt real-time operations. For example, use compensating actions to undo the effects of events on real devices when a rollback occurs.

8.2 Property Utilization

Given that the same events (messages) are often scheduled during re-execution, the Time Warp algorithm can be optimized to avoid redundant computations:

1. **Event Logging and Reuse (memoization):** Log the events and their outcomes during the initial execution. When a rollback occurs and the same events need to be re-executed, retrieve the outcomes from the log instead of recomputing them. This reduces the computational overhead associated with re-execution.
2. **Deterministic Event Scheduling:** Ensure that the event scheduling mechanism is deterministic. This means that given the same initial state and inputs, the events are scheduled in the same order. This determinism ensures that the outcomes logged during the initial execution remain valid during re-execution.

9 HPC (Fall 2021)

In the classical description of a parallel discrete-event simulation algorithm (e.g., Chandy/Misra/Bryant, Samadi's method) we describe the algorithm in terms of the behavior of logical processes (LPs). When mapping such algorithms to a parallel computer, the most natural assignment of LPs to processors is one LP per processor. If there are more LPs than processors, then it is still "easy" to do the mapping: we map more than one LP to each processor, and the processor context-switches between its LPs in some fashion (e.g., round-robin). In either scenario, it is theoretically possible to get a speedup of $O(P)$, where P is the number of processors. (Recall that speedup is the sequential execution time divided by the parallel time.)

Suppose the system has more processors than LPs. That is, let k be the number of LPs and P the number of processors, where $k < P$. Describe a strategy to map the simulation to the parallel computer in a way that can still (in theory) get a $O(P)$ speedup. If no such strategy can exist, explain why.

9.1 Solution

For $O(P)$ speed up, every processor should be fully utilized, meaning each processor would need to handle part of the workload. However, if $k < P$, only k processors can be actively involved in executing the LPs, leaving $P - k$ processors idle, make the achievable speedup to $O(k)$.

10 Prove Execution Time (Fall 2020)

Prove or disprove the following statement: When executing a discrete event simulation program on a parallel computer, the Time Warp algorithm cannot yield an execution time less than the length of the critical path in the event graph. Be sure to state any assumptions you are making in your answer.

10.1 Solution

Based on the definition, **critical path** is the longest path through the event graph, representing the minimum time required to process all dependent events **sequentially**. Any delay on this path will delay the overall completion of the simulation.

The Time Warp algorithm allows events to be processed speculatively in parallel by different LPs. However, this is only for the **independent events**. Events on the critical path must still be processed in the specified sequential order because each event depends on the completion of its predecessors. Therefore, the statement is true.

11 Airport Traffic (Fall 2020)

Consider a simple airport traffic model for a network of airports. Assume each airport has a single runway, and when an airplane arrives, it must queue to use the runway, land, spend time at the gate for boarding, and then departs again. Furthermore, suppose we also wish to model airport closures, that is, sudden airport shutdowns

for some random period of time. Give a complete description of a message passing-based parallel discrete event simulation program for this model. (That is, specify the logical processes, events, propose a synchronization scheme, and be sure to state all assumptions and justify your design.)

11.1 Key Variables

Some key variables include:

1. **Logical Process (LP)**: In this -problem, each airport is a LP. An airport handles the key events.
2. **Events**:
    - **Arrival:** When an airplane arrives at an airport, it queues for the runway
    - **Landing:** After queueing, the airplane lands on the runway
    - **Gate Usage:** Once landed, the airplane moves to the gate for a certain period
    - **Departure:** After gate usage, the airplane departs
    - **Shutdown**: An airport can be suddenly shut down for a random period, during which no operations can occur.
3. **Message Passing**:
    - The LPs communicate via messages representing events.
    - Each event has a timestamp, and LPs process messages strictly in timestamp order to maintain causality.
    - To prevent deadlock and to ensure that LPs do not block indefinitely, LPs send "null messages" to their neighbors. A null message indicates that the sending LP will not send any events earlier than a specific timestamp, allowing the receiving LPs to unblock if they were waiting for earlier events.
4. **Conservative Synchronization**:
    - **Local Clock:** Each LP maintains a local clock and processes the earliest event in its event queue. The LP advances its clock to the timestamp of this event.
    - **Lookahead:** Before processing an event, an LP checks whether it is safe to do so. This check is based on the lookahead value and the knowledge of incoming events from other LPs.
    - **Blocking:** If an LP cannot guarantee that processing an event will not cause causality violations, it blocks, waiting for more information (e.g., null messages or additional events).

11.2 Assumptions and Justification

Some assumptions include:

- **Single Runway per Airport:** Each airport has exactly one runway, meaning only one airplane can land, take off, or use the runway at any given time. This limitation introduces natural queuing for landing and departure events.
- **Finite Gate Capacity:** Each airport has a limited number of gates available for airplanes to dock and load/unload passengers. If all gates are occupied, incoming airplanes must wait in a queue until a gate becomes available.
- **Random Airport Shutdowns:** Each airport can experience random shutdowns, where the airport cannot process any events (landings, takeoffs, gate operations) during this period.

Some justifications include:

- **Causality Preservation:** By processing events in strict timestamp order and using blocking when necessary, the CMB algorithm ensures that causality is preserved across the simulation.
- **Deadlock Avoidance:** Null messages help avoid deadlocks, ensuring that LPs do not wait indefinitely for events that cannot arrive.

11.3 Implementation

Some more details about the implementation:

1. **Initialization:** initialize LPs for each airport with their local clocks set to zero
2. **Event Queue:** Each airport LP maintains an event queue ordered by timestamp
3. **Arrival Event:**
    - When an airplane arrives, the airport LP checks if the runway is available.
    - If the runway is free, the airplane is allowed to land immediately, which triggers a Landing event.
    - If the runway is occupied, the airplane is placed in a queue to wait for its turn to land.
4. **Landing Event**:
    - Once the airplane lands, the next step is to assign it to a gate.
    - If a gate is available, the airplane moves to the gate, triggering a GateUsage event.
    - If no gates are available, the airplane waits until one becomes free.
5. **Gate Usage Event:**

- The airplane occupies the gate for a certain period. After the gate usage period, a Departure event is scheduled.

6. **Departure Event:**

   - The airplane departs from the airport, freeing up the gate and runway. The airport LP may then process the next airplane in the runway or gate queue.

7. **Shutdown Event:**

   - During a shutdown, the airport suspends all operations, and no events are processed. Events scheduled to occur during the shutdown are delayed until the airport reopens.

12 Cascaded Rollbacks (Fall 2019)

One problem with Time Warp is the so-called cascaded rollbacks problem. A cascaded rollback is a situation where one rollback causes a second rollback, which in turn causes a third, etc. Cascaded rollbacks can lead to an excessive amount of rolled back computation and poor performance.

(a) Design a variation on the original Time Warp algorithm where rollbacks can occur (when a logical process receives a message in its past) but cascaded rollbacks cannot occur. Describe in detail all changes to the original Time Warp algorithm that are needed to realize this mechanism. **Hint:** you can eliminate secondary rollbacks if you eliminate the need for anti-messages.

(b) Comment on the effectiveness of this approach relative to the original Time Warp algorithm. Specifically, describe the advantages and disadvantages of your approach relative to the original algorithm in terms of performance.

12.1 Question a

To design a variation of the Time Warp algorithm where rollbacks can occur, but cascaded rollbacks are eliminated, we can implement the following changes:

1. **Eliminating Anti-Messages:** Anti-messages in the original Time Warp are used to cancel events that have been scheduled due to a causality violation. However, these anti-messages are the primary cause of cascaded rollbacks.

   Solution: Instead of using anti-messages, implement a causality-preserving mechanism where each LP maintains a committed state and a tentative state. The committed state is the last known good state of the LP, free from causality violations.

2. **Checkpointing and State Saving** Introduce a checkpointing mechanism where the LP periodically saves its committed state. If a rollback is required, the LP reverts only to the last checkpoint, reducing the amount of recomputation needed. This could be done by the usage of GVT.

12.2 Question b

For these approaches, the advantages include:

1. By removing anti-messages and implementing selective rollbacks, the risk of cascaded rollbacks is effectively eliminated, leading to more stable and predictable performance.

2. The use of incremental state saving reduces the memory overhead compared to the original Time Warp, which often requires saving complete states for rollback purposes.

But there are also some disadvantages:

1. The variation introduces additional complexity, such as maintaining separate committed and tentative states, and implementing sophisticated checkpointing and lookahead mechanisms.

2. Each LP must perform consistency checks before processing events, which adds computational overhead compared to the original Time Warp, where LPs optimistically process events and rollback if needed.

13 Time Stepped Simulation (Fall 2018)

The parallel execution of a discrete event simulation program requires a synchronization algorithm, whereas the parallel execution of a time-stepped simulation, e.g., a cellular automata or continuous simulation, does not. Explain why.

13.1 Solution

For the DES, events are not uniformly spaced in time and can depend on the outcomes of previous events, there is a need to ensure that events are processed in the correct order to maintain causality. The synchronization algorithm ensures that no event is processed before all causally preceding events have been handled, preventing causality violations.

However, for time-stepped simulation, time progresses in uniform increments (time steps). All entities or cells within the simulation are updated simultaneously based on the state at the current time step. This means that each update cycle occurs synchronously across the entire system. Since all updates happen simultaneously within each time step and the system advances uniformly, there is no risk of processing events out of order. The outcome of any update at a given time step is based on the state of the system at the previous time step, ensuring consistency and eliminating the need for synchronization between updates.

14 Advanced GVT (Fall 2018)

Consider a parallel discrete event simulation program executing on a parallel computer using Time Warp for synchronization. Assume this is the only program running on the machine, the machine has an unlimited amount of memory and the application program is correct, e.g., does not contain any infinite loops. Define global virtual time (GVT). Is it guaranteed that GVT will advance as the computation proceeds? If yes, prove that GVT is guaranteed to advance. If no, describe a scenario where GVT will not advance. If GVT is guaranteed to advance only if certain condition(s) hold, explain precisely those conditions and prove GVT will advance under the assumption these conditions do hold.

14.1 Solution

Based on the definition, GVT is defined as the minimum timestamp of all unprocessed events and messages in the system, including events that are scheduled but not yet executed, events that have been processed but could potentially be rolled back, and messages that are in transit between logical processes (LPs).

GVT will advance if there are no unprocessed events with timestamps smaller than the current GVT, and no LP is idle indefinitely while holding an event that could potentially have a smaller timestamp.

GVT might not advance if there is a logical process (LP) that is continuously rolling back or indefinitely delaying the processing of an event with a timestamp smaller than or equal to the current GVT.

Therefore, to ensure that GVT advances, the following conditions need to be met:

- **Progress:** Each LP must make progress in processing its events without indefinite delays.
- **Finite Rollbacks:** The number of rollbacks should be finite, and they should eventually be resolved, allowing the LP to commit events.
- **No Infinite Loops:** The simulation should not contain any infinite loops or scenarios where events are continuously generated with decreasing timestamps, which would prevent GVT from advancing.

15 CMB for GVT (Fall 2014)

The Chandy/Misra/Bryant (CMB) null message algorithm is used to compute the lower bound on time stamp (LBTS) of future messages that a logical process may receive. LBTS has some similarities to Global Virtual Time (GVT), a value computed in the Time Warp algorithm to enable operations such as memory reclamation and commitment of I/O operations. Propose a new GVT algorithm based on the CMB algorithm. Specifically:

1. describe any changes or constraints that must be included in Time Warp to use CMB for this purpose

2. show pseudo-code for your GVT algorithm, and

3. briefly comment on the efficiency of this approach to computing GVT.

Note that the Time Warp system will still utilize rollbacks, anti-messages, etc., and CMB is used solely for the purposes of computing GVT.

15.1 Solution

Design a center processor, which could periodically receive the null message from each processor (each LP) about the information of their existing minimum timestamp, then calculate the GVT based on this (find the minimum). And then, when needed, this center processor will broadcast this information using null message to each processor.

This process could simplify the GVT calculation process. In original design, when the system needs to calculate GVT, the center processor needs to broadcast the command to all the processors and then they will stop the current event (or do that after finish) to calculate the local minimum time stamp and then send it back to the center. After GVT calculation, the center will broadcast the GVT to all the other processors.

16 Video Game (Fall 2014)

Algorithms such as Chandy/Misra/Bryant or Time Warp are not used in distributed simulations used for training or video games, e.g., a collection of vehicle simulators used to train drivers. Explain why such algorithms are not used.

16.1 Solution

For training simulations and video games, these applications typically require real-time or near-real-time performance. They demand consistent and predictable response times to ensure that the user experiences smooth and realistic interactions.

CMB and Time Warp, however, involve significant computational overhead due to their focus on maintaining causal consistency:

- Time Warp, for instance, involves rollback mechanisms and anti-messages, which can introduce unpredictable delays. This unpredictability and the possibility of rollbacks make it unsuitable for real-time applications where delays or sudden changes in the simulation state (due to rollbacks) would disrupt the user experience.

- CMB requires strict conservative synchronization, which can lead to idle times as processes wait for messages. Time Warp requires extensive state-saving and rollback mechanisms. This overhead can degrade performance in real-time applications, where synchronization needs to be minimal and efficient to maintain a smooth experience.