Parallel and Distributed Simulation

1 Introduction

1.1 Overview

Some definitions:

1. **Parallel Simulation**: It involves the execution of a **single** simulation on a collection of **tightly coupled processors** (such as a shared memory multiprocessor).

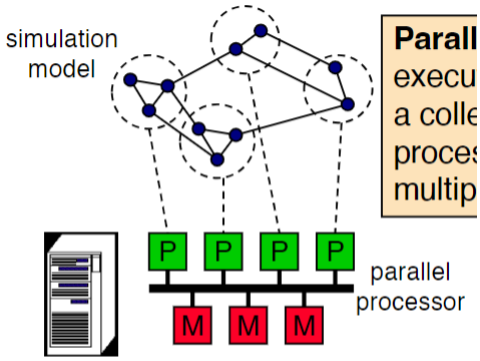

Figure 1: Parallel Simulation

2. **Replicated Trials:** It involves the execution of **several, independent but the same** simulation runs concurrently on different processors.

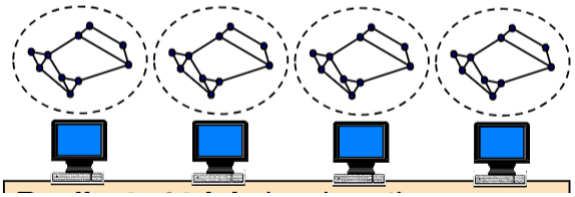

Figure 2: Replicated Trials

3. **Distributed Simulation**: It involves the execution of a **single** on a collection of **loosely** coupled processors, such as servers interconnected by a Local Area Network (**LAN**) and Wide Area Network (**WAN**).

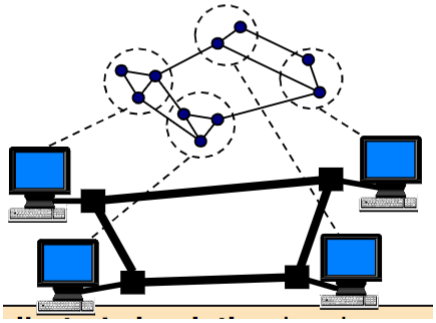

Figure 3: Distributed Simulation

1.2 Distributed Simulation

There are several properties of distributed platforms:

1. **Private Memory:** Each processing node has its own private memory, so one node could not directly modify the memory of another
2. **Message-Passing:** Nodes communication by exchanging messages
3. **High Cost:** The cost of sending messages is high compared to local computation.

1.3 Reasons for Parallel/Distributed Simulation

- Requires too much time to complete a single simulation
- Requires rapid response for real-time decision making to manage operational systems
- Requires more memory than available on a single computer
- Requires geographically distributed people or resources

1.4 System and Time

Usually systems could be divided into:

1. **Physical system:** the actual or imagined system being modeled
2. **Simulation system**: a system that emulates the behavior of a physical system

And the times could be categorized into:

1. **Physical time**: time in the physical system
2. **Simulation (logic) time**: representation of physical time in the simulation
3. **Wallclock time**: time during the execution of the simulation, usually output from a hardware clock

2 Parallel Discrete Event Simulation

2.1 Model Setup

Recall the airport example, first we transfer the physical process to **logical process**:

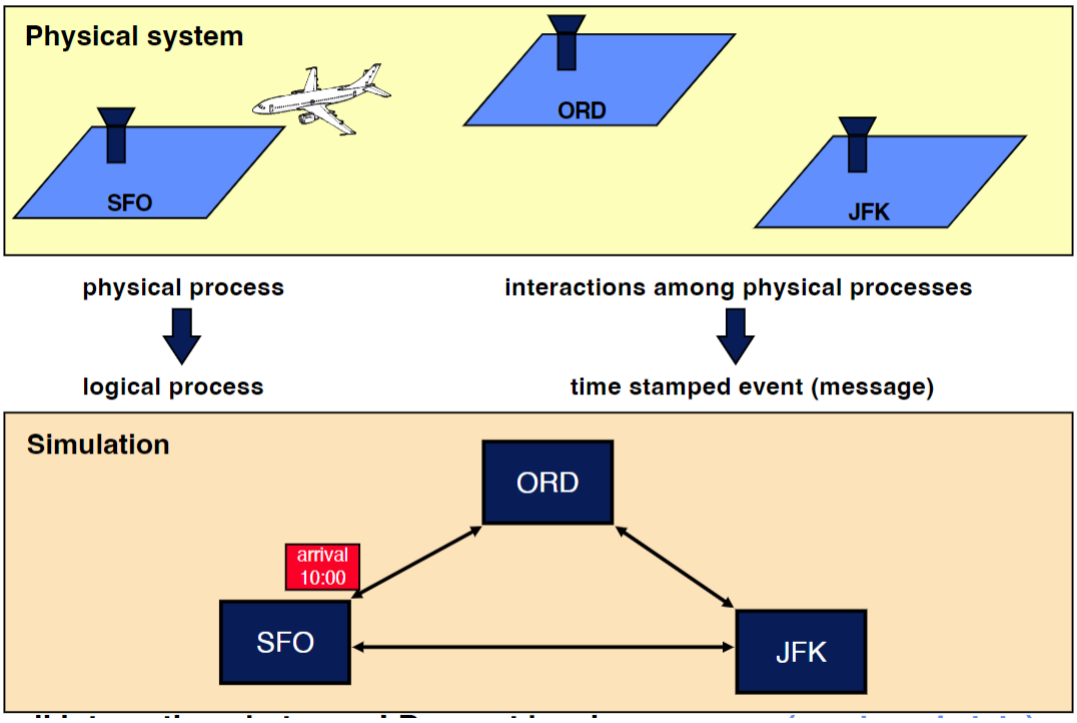

Figure 4: Physical to Logical Process

Some remarks:

1. Each airport simulator is a logical process (LP).
2. An LP can schedule events for other LPs by **sending messages**.
3. **No shared state** between LPs.
4. The **physical system** here is defined as the collection of interacting physical processes (airports).
5. The **simulation system** here is defined as:
   - A collection of logical processes (LPs)
   - Each LP models a physical process
   - The interactions between physical processes modeled by **scheduling events between LPs**

2.2 Parallel Implementation

To implement the simulation, we need to map LPs to different processors. It is allowed to have multiple LPs per processor. All the interactions will via messages. Some important rules:

2.3 Local Causality Constraint (LCC)

2.3.1 Definition

A model should be designed such that an event or state change at a given point in space and time can only affect, and be affected by, events and states in its **local neighborhood within a finite amount of time**. In other words, process incoming messages in time stamp order, including externally generated events.

2.3.2 Case Study

Assume we have several logical processes shown below:

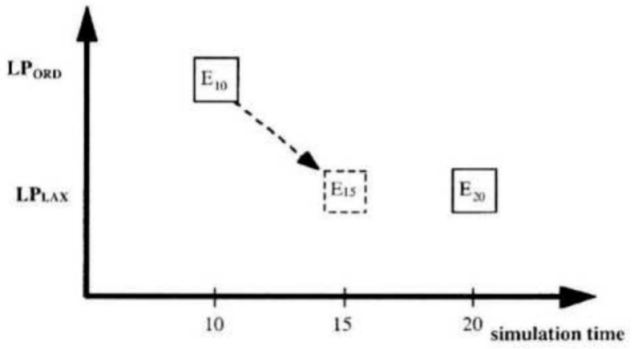

Figure 5: Airports Logical Process

Some clarifications:

1. ORD and LAX represent two local processing units (LPs) handling events.
2. E_{10} is an event in the queue at ORD with a timestamp of 10. E_{20} is an event in the queue at LAX with a timestamp of 20.
3. When ORD process event E_{10}, it may schedule a remote event E_{15} at LAX. **This means that E_{15} is queued at LAX but has a timestamp of 15, which is earlier than E_{20}.**

This is a good example of LCC. According to LCC, events must be processed in the order of their timestamps to ensure the simulation's accuracy. Therefore, **LAX could not process E_{20} before E_{15} because E_{15} might change the state or conditions that E_{20} depends on.** Therefore, LAX must wait to process E_{20} until after it processes E_{15}. This shows the challenge of **achieving concurrency** (the ability of multiple logical processes (LPs) to execute simultaneously) in distributed simulations.

3 Synchronization Problem

Based on the previous discussion, an algorithm is needed to ensure each LP processes events in time stamp order. Based on the observations:

1. **Ignoring simultaneous events:** This means we ignore the events that have identical timestamps. These events are often considered separately because they can be processed in any order without affecting the overall outcome.

2. **Adherence to the LCC**: This is sufficient to ensure that **the parallel simulation will produce the same results as a sequential execution.** If LCC is followed, the results will be consistent with a scenario where all events across all LPs are processed sequentially in timestamp order.

4 Conservative Synchronization

4.1 Overview

Conservative synchronization avoids the risk of causality errors (events are processed out of their intended order, violating the natural cause-and-effect relationships that should be preserved) by **preventing any process from executing an event until it is certain that no other events with earlier timestamps can affect it.**

Some **key principles**:

1. **Blocking until safe (Change!!!):** Each processing unit (LP) blocks the execution of an event until it is guaranteed that no other event with an earlier timestamp can arrive.

2. **Lookahead:** This the minimum time into the future that an LP can predict events will not interfere. LPs can use lookahead to safely process events up to a certain point, knowing that earlier events will not occur.

    - **Link lookahead:** If an LP is at simulation time T, and **an outgoing link** has lookahead L_i, then any message sent on that link must have a timestamp of at least $T + L_i$.
    - **LP lookahead:** If an LP is at simulation time T, and has a lookahead L, then that LP must have a timestamp of at least $T + L$.

3. **Deadlock avoidance:** A common challenge in conservative synchronization is the potential for deadlock, **where all LPs are waiting for each other indefinitely.** Mechanisms such as **null messages** (messages that convey the absence of events within a certain time frame) are used to prevent deadlocks by providing LPs with the information needed to make progress.

The **advantages** of conservative synchronization include:

1. **Deterministic Execution:** Events are processed in a guaranteed correct order, eliminating the need for rollbacks and their associated complexities. The absence of rollbacks leads to more predictable and stable performance.

2. **Simplicity:** The algorithms for conservative synchronization are generally simpler compared to optimistic synchronization since they do not require mechanisms for state saving and rollback.

3. **Reduced Memory Usage:** Since there are no rollbacks, there is no need to save multiple states, resulting in lower memory usage.

But there are also some **disadvantages**:

1. **Potential Inefficiency:** Can lead to significant idle time for LPs, waiting for the assurance that it is safe to proceed, which can reduce overall simulation performance.

2. **Scalability:** May not scale well for very large and complex simulations due to increased communication overhead and potential for blocking.

4.2 Chandy/Misra/Bryant (CMB) Null Message Algorithm

4.2.1 Introduction

In CMB algorithm, LPs periodically send special messages, called **null messages**, to their neighboring LPs. Null messages indicate that **no events with timestamps earlier than a certain value will be sent in the future.**

4.2.2 Assumptions

This algorithm needs several **assumptions**:

- LPs exchanging timestamped events (messages).
- Static network topology (LP-to-LP), no dynamic creation of LPs.
- Messages sent on each link are sent in timestamp order.
- Network provides reliable delivery, preserves order.

4.2.3 Procedures

The procedures include:

1. **Initial State:** Each LP starts with its local event queue containing events it needs to process.

2. **Sending Null Messages:** When an LP processes an event, it sends a null message to its neighboring LPs. The null message contains a timestamp **indicating the earliest time at which the LP might send a new event in the future.** This timestamp is typically **the current simulation time plus some lookahead value**, representing the minimum time into the future before which no new events will be generated.

3. **Receiving Null Messages:** When an LP receives a null message from a neighboring LP, it updates its understanding of the earliest possible future event from that neighbor. **This helps the receiving LP determine the safe time horizon up to which it can process its local events without waiting for earlier events from its neighbors.**

4. **Processing Events:** Each LP processes its local events in timestamp order up to the minimum of the null message timestamps received from all its neighbors.

5. **Deadlock Avoidance:** The algorithm uses null messages to avoid deadlock situations where all LPs are waiting indefinitely for events that might never arrive.

4.3 Time Creep Problem

Time creep occurs when LPs advance their local simulation clocks very slowly because they are overly cautious. Each LP waits for confirmation that it can safely process an event, often resulting in significant idle times as LPs wait for messages from other LPs. This conservatism can lead to the overall simulation progressing very slowly, as LPs make minimal progress due to the need for constant synchronization and assurance of event order.

5 Optimistic Synchronization (Simplified, check textbook)

5.1 Introduction

Optimistic synchronization is a method used in parallel distributed simulation to allow multiple processes or logical processes (LPs) to progress at their own pace without strict adherence to global synchronization. It is based on the idea that processes can optimistically proceed with their simulations and later correct any inconsistencies that arise due to out-of-order events.

The advantages of optimistic synchronization include:

1. **Increased Parallelism:** LPs can proceed independently without waiting for synchronization barriers, maximizing CPU utilization and reducing idle times.

2. **Flexibility and Scalability:** Optimistic synchronization allows the simulation to adapt dynamically to varying workloads and event distributions without a rigid synchronization protocol. Also, it is well-suited for large-scale simulations where the communication overhead of conservative synchronization would be prohibitive.

3. **Efficient in Low-Conflict Scenarios:** If the events from different LPs rarely conflict, rollbacks are infrequent, leading to efficient parallel execution.

There are also some disadvantages of optimistic synchronization:

1. **Rollback Overhead:** For state saving, continuous saving of states consumes memory and adds computational overhead, especially in large-scale simulations. For reprocessing, rollbacks require reprocessing of events, which can negate performance gains if frequent.

2. **Complexity:** Developing an optimistic synchronization mechanism is complex, requiring sophisticated algorithms for state management, rollback, and conflict resolution.

3. **Resource Consumption:** The need to store multiple states for potential rollbacks can lead to high memory consumption. Also, Managing rollbacks and state saving can consume significant processor time, reducing the net benefit of parallelism.

4. **Unpredictable Performance:** The performance gains from optimistic synchronization are unpredictable and highly dependent on the frequency and cost of rollbacks. High rollback rates can become a bottleneck, causing performance to degrade, sometimes even below that of conservative synchronization methods.

5.2 Limiting Optimism

Limiting optimism refers to strategies used in optimistic synchronization to mitigate the overhead and complexity associated with excessive rollbacks. These techniques aim to balance the benefits of optimistic synchronization with mechanisms to control and reduce the negative impacts of frequent rollbacks.

5.2.1 Lazy Cancellation

Rather than immediately canceling and rolling back upon detecting an out-of-order event, the simulation continues for a short period to determine if the rollback is truly necessary.

This could reduce the frequency of rollbacks by avoiding premature cancellations, which can be beneficial in scenarios where events correct themselves shortly after being detected as out-of-order.

5.2.2 Adaptive Optimism Control

Dynamically adjusts the level of optimism based on the current state of the simulation, such as the frequency of rollbacks or the level of event conflict.

This allows the system to optimize performance by increasing optimism during low-conflict periods and reducing it when conflicts are frequent, thereby balancing efficiency and accuracy.

5.2.3 Bounded Time Windows

Limits the time window within which LPs can process events optimistically. If an event falls outside this window, the LP waits until it can safely process the event.

This reduces the likelihood of long-range rollbacks, thereby limiting the extent of reprocessing required.

5.3 Hybrid Approaches

Hybrid approaches in parallel distributed simulation combine elements of both conservative and optimistic synchronization methods to leverage the advantages of each while mitigating their respective drawbacks. These approaches aim to provide a balanced performance, ensuring efficient parallelism while maintaining accuracy and minimizing overhead.