The ROme OpTimistic Simulator: A Tutorial

Alessandro Pellegrini
Francesco Quaglia

High Performance and Dependable
Computing Systems Group
Sapienza, University of Rome

PADABS 2013

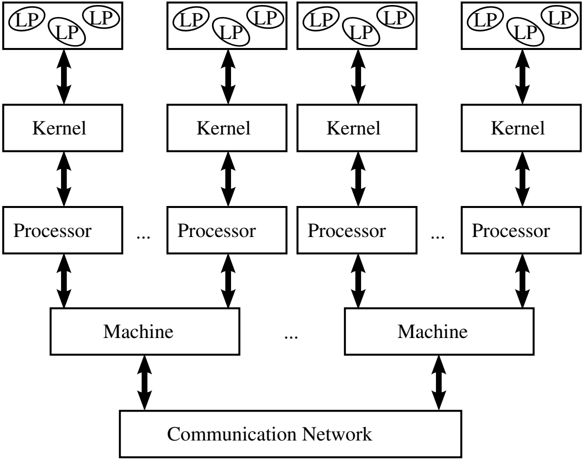
Discrete Event Simulation (DES)

- A simulation model is described in terms of:
  - *Simulation state*, describing the current state of the system
  - *Events*, associated to particular actions/changes in the system
  - *State transitions*, which modify the state depending on the executed events
- Based on event-driven programming
  - Events are dispatched to the associated event handlers which implement the model's logic
- A *discrete event* occurs at an instant in time, producing a change in the system
- DES represents the operation of a system as a chronological sequence of events
- An event cannot be scheduled in the past

DES Building Blocks

- **Clock**
  - Keep track of the current simulation time (independently of the measuring unit)
  - Being *discrete*, time hops to the next event's time
- **Events List**
  - At least the *pending event set* must be maintained by the simulation architecture
  - Events can arrive at a higher rate than they can be processed
- **Random-Number Generators**
  - Simulation often rely on distributions, in order to model real world's aspects
- **Statistics Collection**
- **Ending Condition**
  - Real systems can often run forever, so the designer of the model must decide when the simulation will halt

Going *parallel*: PDES Logical Architecture



Going *parallel*: PDES Object Model

- Simulation model is partitioned into *simulation objects*
- Simulation objects model a portion of the space and/or agents
- Disjoint States: Message Passing to represent interactions

$$S = \bigcup_{i=1}^{numLP} S_i \qquad \forall i, j \; i \neq j \; : \; S_i \cap S_j = \emptyset$$

- Logical Processes (LP) implement event handlers, used to dispatch events to simulation objects
- A simulation kernel schedules the execution of a particular LP for processing an event at a certain simulation object

Going *parallel*: Optimistic PDES

- Events are executed speculatively: processed events can be committed or uncommitted
- The *commitment horizon* is associated with GVT value

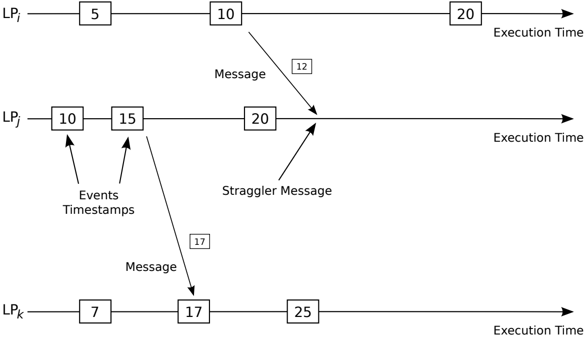
- Relatively independent of lookahead
- Resource utilization approaches 100%
- It is faster than the critical path

The Synchronization Problem

- The greatest opportunity arises from processing events from different LPs concurrently on different processors
- Is *correctness* always ensured?

The Synchronization Problem

- The greatest opportunity arises from processing events from different LPs concurrently on different processors
- Is *correctness* always ensured?

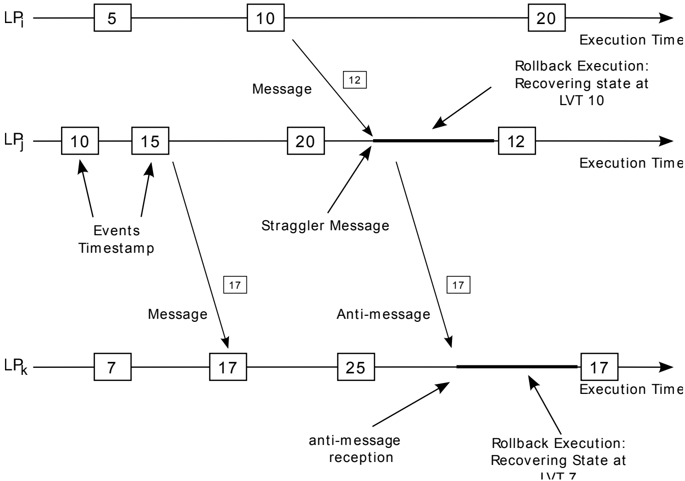


Optimistic Synchronization: Time Warp

- There are no state variables that are shared between LPs
- Communications are assumed to be reliable
- Messages might not be received in timestamp order

- **Local Control Mechanism**
  - Events not yet processed are stored in an *input queue*
  - Events already processed are not discarded
- **Global Control Mechanism**
  - A-posteriori detection of causality violation
  - Event processing can be **undone** (*rollback*)
    - Reverse computation
    - Simulation State Checkpoint/Restore

Rollback


LP_i
5
10
20
Execution Time
Message
12
Rollback Execution: Recovering state at LVT 10
LP_j
10
15
20
12
Execution Time
Events Timestamp
Straggler Message
Message
17
Anti-message
17
LP_k
7
17
25
17
Execution Time
anti-message reception
Rollback Execution: Recovering State at LVT 7

The ROme OpTimistic Simulator (ROOT-Sim)

- Simulation Platform built according to the Time Warp Synchronization Protocol
- Supports ANSI-C programming
- Targets both simulation efficiency and model development transparency
- Comes bundled as a static library

http://www.dis.uniroma1.it/∼hpdcs/ROOT-Sim/

ROOT-Sim API: Objects/Model Description

- The ROOT-Sim API is based on a reduced set of call/callback functions:
  - `ProcessEvent()` (callback) – Used to give control to the simulation model (to a LP). It passes an event to be dispatched to some simulation object
  - `ScheduleNewEvent()` (call) – Allows to inject a new event in the system
  - `OnGVT()` (callback) – Used to perform analysis on a committed state, and for termination detection
- The simulation model is written in ANSI-C
- The LPs' simulation state is stored into dynamically-allocated memory
- The simulation is started via a special `INIT` event

An Example Simulation Model: Data Definition

```c
#include <ROOT-Sim.h> // The ROOT-Sim header file

#define PACKET 1 // Event definition
#define DELAY 120
#define PACKETS 1000000 // Termination condition

typedef struct _event_content_t {
   time_type sent_at;
} event_content_t;

typedef struct _lp_state_t{
   int packet_count;
} lp_state_t;
```

An Example Simulation Model: Event Processing

```c
void ProcessEvent(unsigned int me, time_type now, unsigned int
    event, event_t *content, unsigned int size, lp_state_t *state){

   event_t new_event;
   time_type timestamp;

   switch(event) {

      case INIT: // must be ALWAYS implemented
         state = (lp_state_t *)malloc(sizeof(lp_state_t));
         state->packet_count = 0;
         timestamp = (time_type)(20 * Random());
         ScheduleNewEvent(me, timestamp, PACKET, NULL, 0);
         break;
```

An Example Simulation Model: Event Processing (2)

```
     case PACKET: {
        state->packet_count++;
        new_event_content.sent_at = now;
        int recv = FindReceiver(TOPOLOGY_MESH);
        timestamp = now + Expent(DELAY);
        ScheduleNewEvent(recv, timestamp, PACKET, &new_event,
            sizeof(new_event));
     }
   }
}
```

An Example Simulation Model: Termination Detection

```c
bool OnGVT(unsigned int me, lp_state_t *snapshot) {
  if (snapshot->packet_count < PACKETS)
     return false;
  return true;
}
```

ROOT-Sim Facilities

Already Released:

- Simulation state can be scattered over dynamically allocated memory
- Supports Full, Incremental and Autonomic Logging
- Consistent and Committed Global State management
  - Termination detection
  - Consistent statistics collection
- Transparently-rollbackable statistical library
- Topology library

ROOT-Sim Facilities

Already Released:

- Simulation state can be scattered over dynamically allocated memory
- Supports Full, Incremental and Autonomic Logging
- Consistent and Committed Global State management
  - Termination detection
  - Consistent statistics collection
- Transparently-rollbackable statistical library
- Topology library

Currently under development:

- Shared state management (via global variables)
- Consistent output streams management
- Load Balancing
- Load Sharing (via multithreading)

It’s now time...

...for a live demo!

Thanks for your attention

Questions?

pellegrini@dis.uniroma1.it
http://www.dis.uniroma1.it/∼pellegrini
http://www.dis.uniroma1.it/∼ROOT-Sim