Intern. Journal of Modeling and Simulation,

INTRODUCTION TO A FAULT-TOLERANT DISTRIBUTED REAL-TIME SYSTEM SIMULATOR*

Kai Yu, Kiran Toutireddy, Israel Koren, and C. M. Krishna
Department of Electrical and Computer Engineering
University of Massachusetts
Amherst, MA 01003

Key Words—real-time, PVM, distributed, central clock

Abstract

This paper describes a distributed simulation package for real-time systems. The purpose of the simulator is to study recovery strategies for embedded systems, task allocation and scheduling algorithms, network protocols, and dynamic reconfiguration techniques.

The current version of the simulator runs on a network of workstations, connected by an Ethernet. The underlying communication mechanism is PVM (Parallel Virtual Machine) software.

1 Introduction

In this paper, we describe a simulation package for real-time systems. The package is meant to run on a distributed network of workstations interconnected by an Ethernet and using the Parallel Virtual Machine (PVM) software as the underlying communication mechanism. The simulator will be used to study the behaviour of embedded systems; in our project, it will be used to evaluate failure-recovery strategies.

A good review of techniques for distributed simulation can be found in [2,3]. Generally speaking, any distributed simulator can be divided into three components: logical processes (LPs), an

*This work was supported by ARPA, under order B855, and managed by the Space and Naval Warfare Systems Command under Contract N00039-94-C-0165.

interprocess message passing scheme for communication, and a mechanism for synchronizing the LPs. The purpose of the synchronization is to maintain sequential consistency in the simulation results.

Perhaps the most difficult (and important) part of building a distributed simulator is choosing a proper synchronization scheme. There is a large technical literature on synchronization techniques; in general they can be divided into two types: *conservative* [1,6], and *optimistic* [5]. In an optimistic scheme, processes are almost allowed to free-run, and any violation of consistency is corrected after the event. On the other hand, a conservative scheme does not allow violations of consistency to occur in the first place. Conservative schemes are, in some applications, a little slower than optimistic schemes; however, they are simpler to implement.

As mentioned earlier, message passing between the logical processes is supported by the PVM package [4]: this package offers an efficient and transparent communication mechanism for passing messages across a network of machines.

2 The Simulator Structure

A functional view of the simulator is shown in Figure 1. There are three classes of LPs; a clock server used for purposes of synchronization, some PVM communication links, and a user interface (in the shape of a system console).

The categories into which the LPs can be classified are as follows:

- Virtual Processors (VPs), which are meant to simulate individual processors in the system that is being simulated.

- Virtual Networks (VNs), that permit the simulation of various network topologies and protocols.

- Task and fault generators.

The simulator can be viewed as a layered structure as shown in Figure 2. Physical Layer 1 consists of the PVM software which, as pointed out earlier, is the underlying message passing mechanism. By using PVM, the virtual processes can communicate with one another.

Physical Layer 2 offers such basic functions as system initialization, fault tolerance, a logging system to record events, and a graphical user interface (GUI).

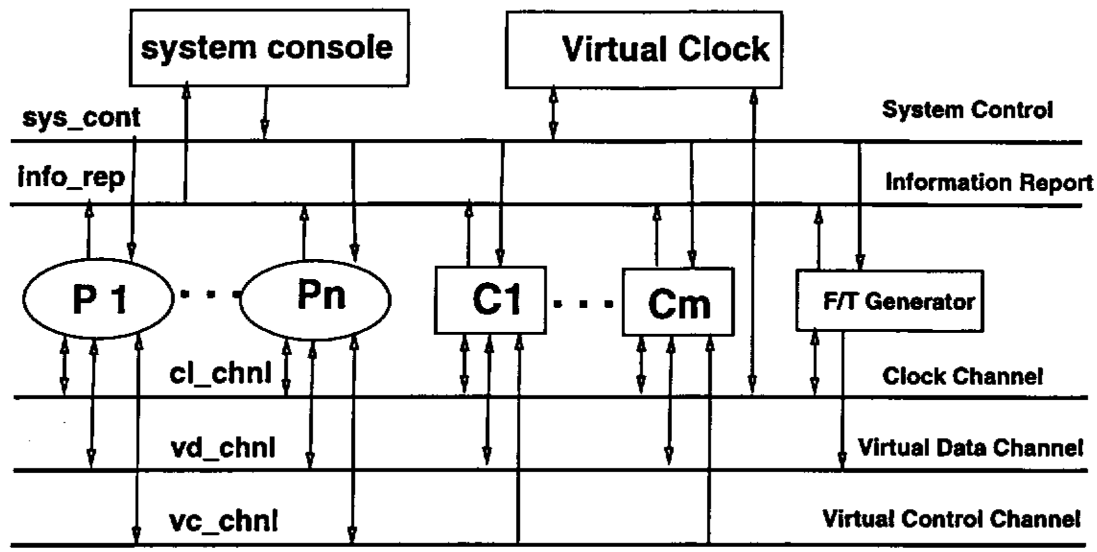

Figure 1: The Functional View of The Simulator

Atop Physical Layer 2 sits Virtual Layer 1, which is the central clock protocol. This clock provides synchronization for the LPs.

Above Virtual Layer 1 is Virtual Layer 2, which consists of the heart of the simulator. This layer consists of the following:

- Task generation modules: Periodic and aperiodic (sporadic) tasks can be defined by these modules. This module receives information from Virtual System Layer 3 as to the execution time (in terms of a probability distribution function), the period (if it is periodic), the deadline, and the communication pattern (i.e., the messages it sends out and receives) of each task.

- Fault injection module: The fault injection module can inject both permanent and transient faults. The user specifies (through the user interface) the fault rate at each processor as well as the duration of the transient faults.

- Virtual Processor: Master: The master function consists of task allocation, system recovery, system configuration control, and fault detection. Guidance for each of these is provided by Layer 3. Currently, the utilization-balancing approach to task allocation has been implemented. When a processor fails, the recovery technique chosen can be one of the following: rollback to the previous checkpoint and retry, replace the failed processor with

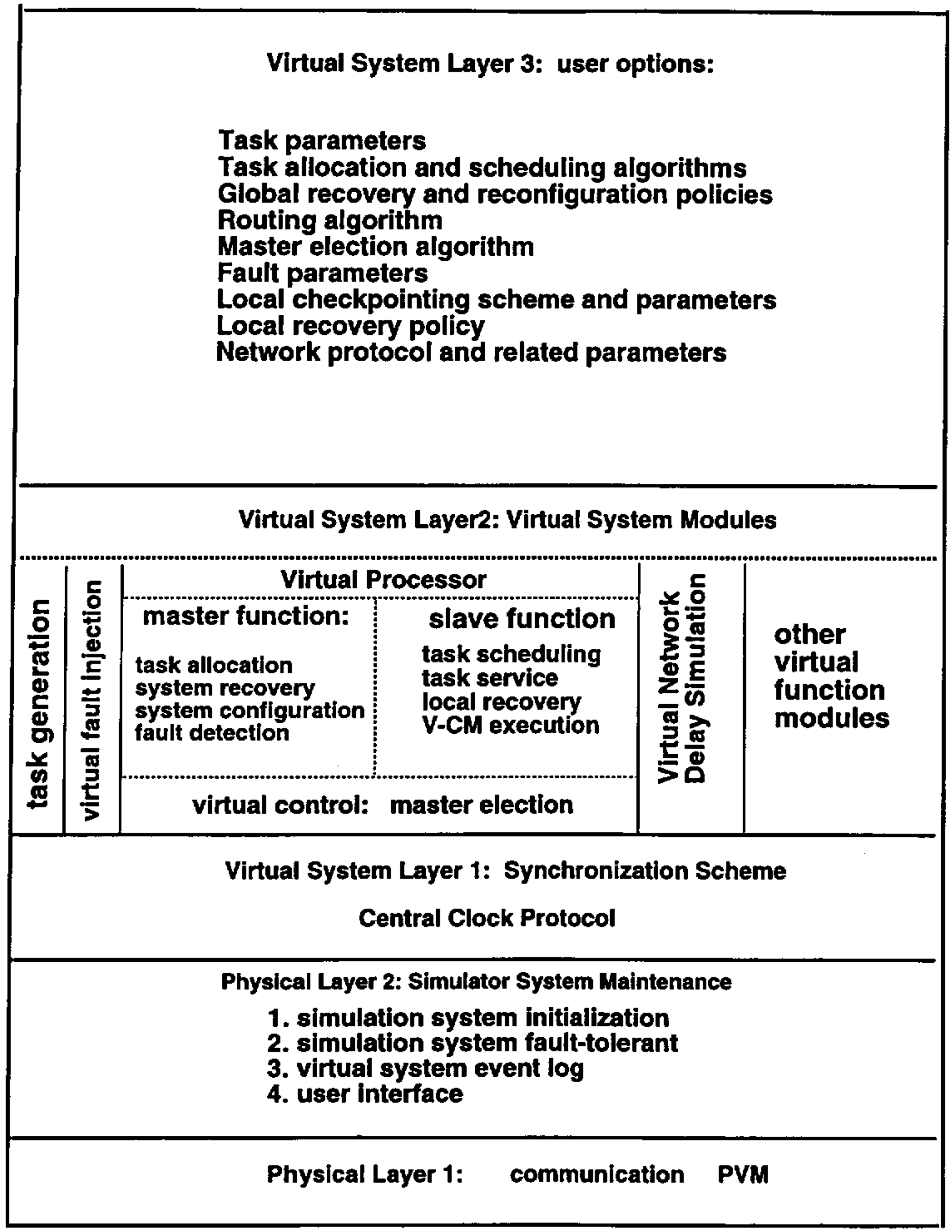

Figure 2: The Layered Structure

a spare if one is available (i.e., move all the tasks on the failed processor to the spare), and disconnect the failed processor and distribute its tasks among the functional processors. The appropriate recovery action will be specified by Virtual Layer 3.

- Virtual Processor: Slave: These processes simulate individual processors of the simulated system. They have functions such as task scheduling and execution, and local recovery.
- Virtual Network: The virtual network module allows the user to simulate the desired interconnection network and protocol. At present, we have implemented the FDDI and IEEE 802.3 protocols on a ring network, as well as point-to-point networks.

Our simulator is specifically designed to allow easy extensions. For example, a user may decide to augment the network module to handle other network topologies and protocols.

Virtual System Layer 3 provides parameters and guidance for Layer 2. As we have already seen, it provides information about failure parameters, specifies the task allocation and scheduling algorithms, the network routing algorithm, the checkpointing scheme (to facilitate rollback), and the network protocol. It also includes the algorithm to elect the master virtual processor.

3 The Implementation

3.1 System Console

The console consists of a system initialization routine, a simulator controller (allowing the user to suspend and restart the simulator), and windows which display the simulation results. These results are displayed to the user as they are generated by the simulator.

3.2 Clock Mechanism

The synchronization scheme is a variation of Breathing Time Buckets technique [12]. A central clock server controls the Global Virtual Time (GVT). The clock server is a PVM process which broadcasts the current virtual time. The clock function is summarized in Figure 3.

The central clock algorithm can be informally described as follows. Each of the LPs sends to the clock its next event time (NET). The clock picks the minimum of these NETs, and broadcasts this time value as the Current Virtual Time (CVT). When an LP receives a new CVT, it commits all events until this time, and transmits its next event time to the central clock.

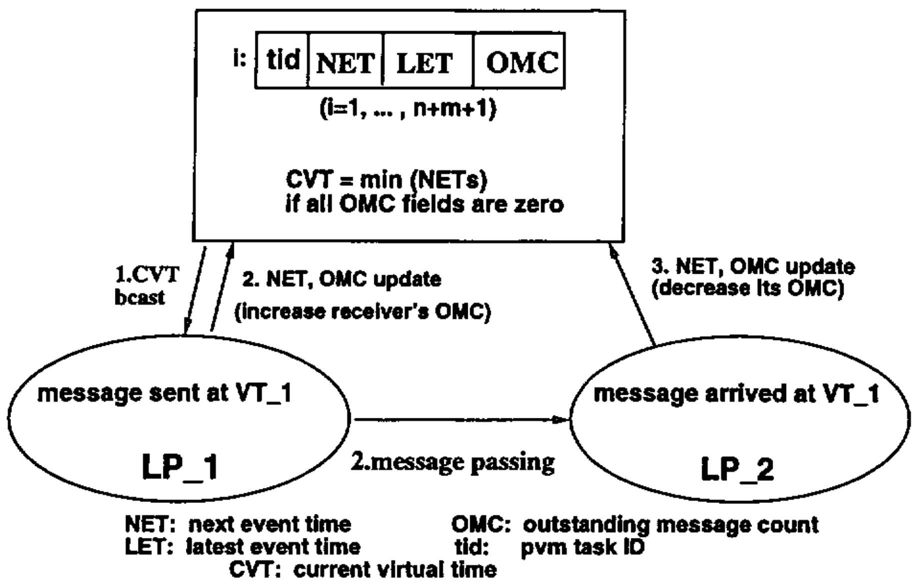

Figure 3: Central Clock

The above approach can give rise to a race condition due to delays in message passing. For example, suppose that processor A sends a message to processor B at time 10. Processor B does not know that a message is coming its way until it actually receives it: suppose its next event (that it is aware of) is at time 30. This next event time is reported to the central clock. However, some time after this report, processor B receives the message that A sent it; let us suppose this event should have happened at time 25. This results in the CVT being set incorrectly.

It is relatively easy to avoid this race condition. We introduce the OMC field at the central clock. This consists of one word per processor. When A sends a message to B, it increments the OMC field associated with B. When B receives the message, it decrements this field. The clock does not advance until the OMC fields of all the LPs are zero.

3.3 Virtual Processor

The structure of the virtual processors (VPs) is shown in Figure 4.

The set of virtual processors can be broken down into four subsets:

- *System Maintenance:* During system initialization, this unit sets up the local VP. This action includes setting up the virtual routing tables, presiding over the master election algorithm in the virtual control function, initializing the task allocation and scheduler pro-

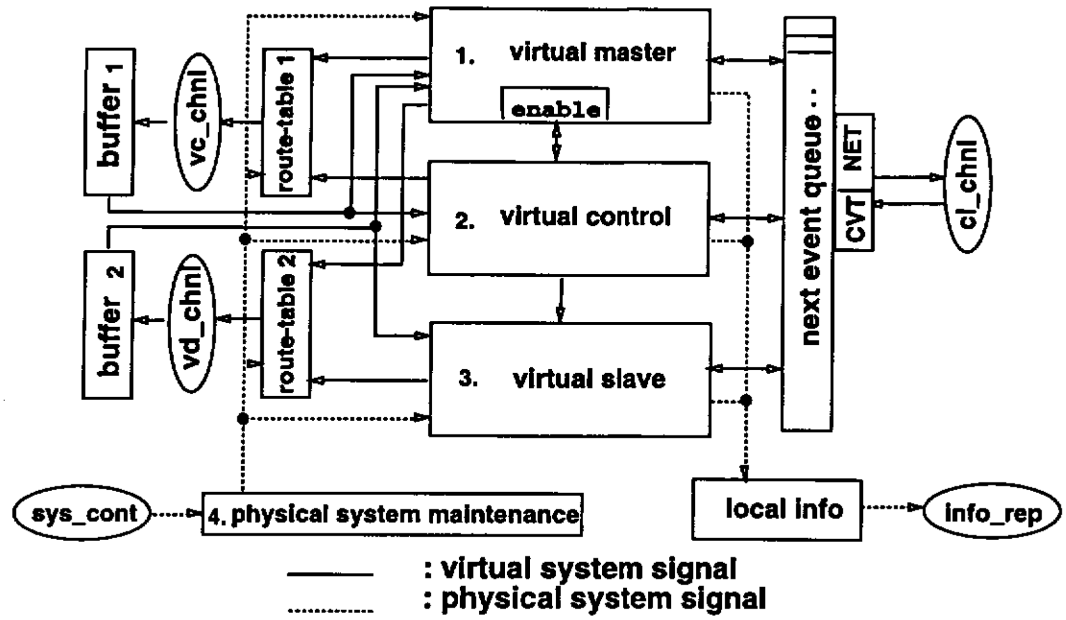

Figure 4: Virtual Processor

cedures, and other housekeeping tasks. It is also responsible for executing user commands such as process termination.

- *Master:* The job of the master is to allocate and reallocate tasks, run fault-detection algorithms, and carry out system reconfiguration when necessary.

- *Virtual Control:* Virtual control is responsible for running the master election algorithm, and for handling switchover to a new master if the current master fails. It also keeps up-to-date the routing tables to account for changes in the network structure.

- *Slave:* The slave simulates the individual processors in the simulated system. It creates the task structure, schedules tasks, updates status tables appropriately, and has the structure shown in Figure 6.

When the system is started, the virtual master is as yet unselected. This fact is detected by each VP by means of a timeout. A system-wide master election process then follows. After the election of the master is complete, the virtual tasks are sent to the master and allocated by it to the various slaves. When a slave receives its tasks, it creates task structures and places them in its dispatch queue. The user-prescribed uniprocessor scheduling algorithm is used to

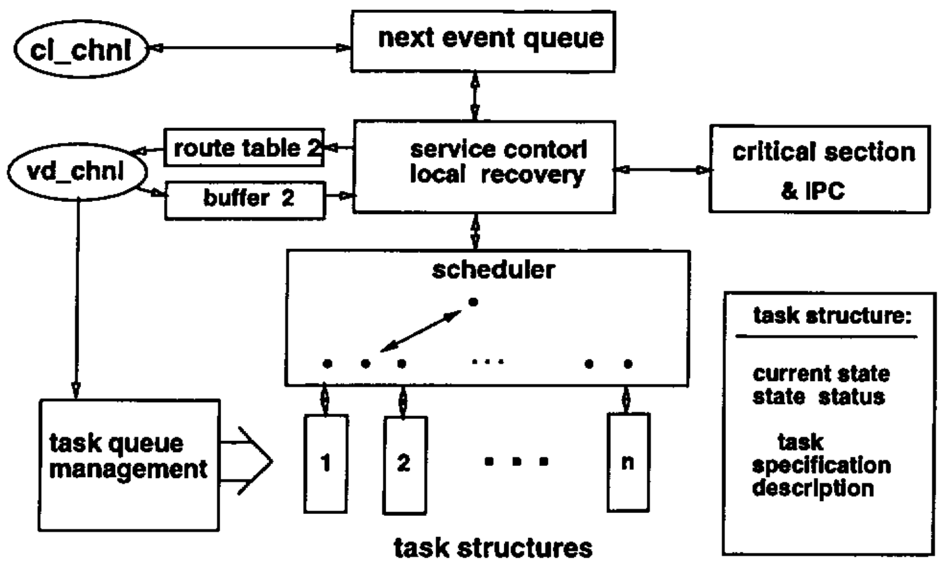

Figure 5: Virtual Processor, Slave Function

schedule these tasks. Messages between tasks are sent out on a virtual network, which simulates the network delay and then passes on the message to the destination VP. At present, Ethernet, FDDI, the fully-connected topology, and the IEEE 802.3 ring are implemented.

4 Conclusion

In this paper, we have briefly described a distributed simulator for real-time systems. The simulator is meant to be used in validating system recovery algorithms, and also permits the testing of task assignment and scheduling algorithms, network protocols, and dynamic system reconfiguration policies. In its present form, the simulator runs on four workstations connected by an Ethernet. It is coded in C and C++ for maximum portability.

5 Acknowledgement

We would like to acknowledge the technical assistance of Poornima Lalwaney, Steve Morin, and Rajeev Koodli in setting up the simulator.

Figures 1 to 5 previously appeared in [13]. They are reprinted by permission of the copyright holders, Simulation Councils, Inc.

References

[1] K. M. Chandy and J. Misra, "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs," *IEEE Trans. On Software Engin., 5*(5), 1979, 440-452.

[2] R. Fujimoto and D. Nicol. "State of The Art in Parallel Simulation", In Proceedings of the 1992 Winter Simulation Conference, 246-254.

[3] Fujimoto, R.. 1990. "Parallel Discrete Event Simulation," *Communications of the ACM, 33*(10), 1992, 30-53.

[4] A. Geist, A. Beguelin, et. al. *PVM: Parallel Virtual Machine, A Users' Guide and Tutorial for Networked Parallel Computing,* (The MIT Press, 1994).

[5] D. R. Jefferson and H. Sowizral, "Fast Concurrent Simulation Using the Time Warp Mechanism," *part I: Local Control. Technical Report N-1906AF,* (RAND Corporation 1982).

[6] V. Jha and R. L. Bagrodia, "Transparent Implementation of Conservative Algorithms in Parallel," *In Proceedings of the 1993 Winter Simulation Conference,* 1993, 677-688.

[7] J. Misra, "Distributed Discrete-Event Simulation," *Computing Surveys, 18*(1), 1986, 39-65.

[8] D. M. Nicol, "Global Synchronization For Optimistic Parallel Discrete Event Simulation," *In Proceedings of the 1993 Workshop on Parallel and Distributed Simulation,* 1993, 27-34.

[9] H. Rajaei, R. Ayani and L. E. Thorelli, "The Local Time Warp Approach To Parallel Simulation," *In Proceedings of the 1993 Workshop on Parallel and Distributed Simulation,* 1993, 119-126.

[10] R. G. Sargent, "A Historical View of Hybrid Simulation/Analytic Models," *Proceedings of the 1994 Winter Simulation Conference,* 1994, 383-386.

[11] T. J. Schriber, "Inside Simulation Software: How It Works and Why It Matters," *In Proceedings of the 1994 Winter Simulation Conference,* 1994, 45-54.

[12] J. S. Steinman, "Breathing Time Warp," *In Proceedings of the 1993 Workshop on Parallel and Distributed Simulation,* 1993, 109-118.

[13] Yu K. and Toutireddy K. K. "A Practical Distributed Test-Bed: Fault-Tolerant Distributed Real-Time System Simulator", *The Proceedings of The 1996 Summer Computer Simulation Conference*, 1996, pp. 97-102