# VIRTUAL MACHINE-BASED SIMULATION OF DISTRIBUTED COMPUTING AND NETWORK COMPUTING

Richard T. Wang and J. C. Browne
Department of Computer Science
The University of Texas at Austin
Austin, Texas 78712

## ABSTRACT

This paper proposes the use of virtual machine architectures as a means of modeling and analyzing networks and distributed computing systems. The requirements for such modeling and analysis are explored and defined along with an illustrative study of an X.25 link-level protocol performance under normal execution conditions. The virtualizable architecture used in this work is the Data General Nova 3/D.

## 1.0 INTRODUCTION

The great upsurge in interest in computer networking and distributed systems has not yet been accompanied by an equivalent increase in experimental or empirical research. The basic reason is clearly the high cost of establishing experimental systems and executing the experiments. It is further the case that a given physical realization of a distributed system or computer network can probably support only a small subset of the very wide class of interesting and significant experiments.

Models which are analytically soluble are very effective in supporting studies at high levels of abstraction but cannot support analyses of the often very important detailed execution patterns. Models detailed enough to require simulation evaluation can be constructed to execute at arbitrary levels of resolution of detail. The programs for realization of simulation models often approach the complexity of the actual system's code resolved to the same level of detail. It is also the case that the execution time for the evaluation and detailed modeling and simulation can be very time consuming. They may run in many times real time. This precludes the studying of long sequences of events.

A basic problem in both analytic and simulation evaluation of models of distributed computing and computer networks is the need for a baseline of empirical data upon which to expound the analysis, parameterize and, above all, to validate the model structure. Canon, et al [1] suggested the use of virtual machine architectures for capacity planning and configuration analysis. This paper proposes the use of virtual machine architectures as a means for modeling distributed computing systems and network computing systems. In this approach the several virtual machines supported by the virtual machine monitor execute the actual software of the distributed computer or network system while the virtual machine monitor emulates the interconnection of the network structure. The advantages of this approach are as follows.

1. The actual software implementing the function of the distributed architecture or network computer system will be used.

2. The actual data dependency and other subtleties of the software are reproduced in the model system. Of course simplified and/or modelled software can also be used if desired.

3. The execution time is, of course, bounded by real time.

4. Parametric or multiple copies of the same software can be used, thus lowering development efforts.

5. The virtual machine monitor can emulate any type of external interaction including such factors as transmission errors, lost data, synchronization errors, failed processors, etc. A properly designed virtual machine monitor isolates each factor so that the interaction can be studied separately. Additionally, the set of factors which can be studied is easily extended or modified.

6. The virtual machine monitor can execute the several virtual machines at different rates thus allowing the modeling of processors of different power.

7. The number of nodes and processors in the distributed system or network system could easily be extended by adding to the number of virtual machines executing under the control of the virtual machine monitor.

## 2.0 TIME RESOLUTION

The principle problem associated with the use of virtual machine monitors for modeling of distributed computing and network systems is the maintenance of consistent time and sequencing relationships. A message must not be received by one virtual machine before it is sent by another virtual machine [4]. The basic definition of virtual machine systems advances the time-of-day clock of a given virtual machine (VTOD) only when that virtual machine is in execution on the physical processor [1]. Thus a virtual machine which gets a small share of the physical processor may perceive a slower passage of real time than an identical virtual machine which obtains a larger share of the physical processor. It is also the case that interesting studies of distributed computing and computer networks must include a capability of defining and studying systems of different processing power. It is necessary, therefore, to maintain different processing rates for different virtual machines executing on a single real processor. The UT Virtual Machine Monitor includes a capability for executing virtual machines at different rates. During systems initialization, the operator assigns a value between 1 and 10 to the instruction execution time of each virtual machine. A virtual machine with shorter instruction execution time can execute more instructions per unit virtual time, and is thus more powerful than those with longer instruction execution time. A virtual-time-of-day clock is maintained for each virtual machine. This time-of-day clock is incremented at a rate proportional to the system's quantum time. The quantum time is in turn defined as an integer multiple of the hardware real-time clock pulses. The hardware clock tick rate is program selectable from 10 times a second to 1,000 times a second. Throughout these experiments, the clock tick rate was set at 1,000 times a second, and a quantum was defined (by the operator at the system's initialization time) as two clock pulses, giving a quantum time of 2 ms. The virtual-time-of-day clock for the running virtual machine is incremented at the end of each time quantum by an amount equal to the instruction execution time of that machine. Each external interaction of a virtual machine is stamped with the time-of-day clock value at the time the interaction takes place. Thus, if virtual machine i sends a message to virtual machine j, then the message is stamped with the VTOD of i, and machine j will not be allowed to receive the message until VTOD of j exceeds the time stamp of that message. This can easily be monitored and enforced by the virtual machine monitor. In order to minimize the span between VTOD, the virtual machine monitor selects for execution that virtual machine which currently has the smallest time-of-day clock.

## 3.0 THE UT DATA GENERAL VIRTUAL MACHINE MONITOR

The virtual machine monitor implemented at the University of Texas is a product of three efforts. These efforts have been described in the theses of Renaud [5], Herrington [3], and Wang [6]. The system is almost entirely virtualizable. The two defects are in the switching of context state and in the interrupt handling. Both exceptions to virtualizability of these can be handled in reasonable fashion to preserve an exact replica of the normal machine interface.

All of the normal features found in virtual machine architectures are preserved. There is double mapping of virtual memory, mapping of interrupts, the management of time as previously described and the usual other features.

The UT Virtual Machine Monitor is at this time constrained to have only four virtual machines executing simultaneously. This is, however, a limitation of the current program rather than an intrinsic limitation. There were only four terminals available to serve as the system console. The UT Virtual Machine Monitor is coded mainly in PASCAL (4,000 lines of source code) and is readily extendible. A small kernel (200 words) of the VMM that does the initial handling of interrupts and traps processes is coded directly in assembly language.

## 4.0 AN EXPERIMENTAL STUDY

An example experimental study to illustrate the power of the virtual machine architecture for studying the characteristics of computer networks was executed. The study used the link level protocol of X.25 [2]. The goal was to study the performance characteristics of X.25 and the behavior of the machine executing it. Many experiments were run. The implementation of the X.25 protocol has two units: a transceiver unit which runs on each virtual machine to produce and receive messages in conformance to the datagram facilities of X.25, and a controller unit implemented in PASCAL as a part of the virtual machine monitor which implements the message handling, routing, and buffering characteristics and which does such things as inserting the transparency bits during transmission, introducing random transmission errors into the messages, and maintaining trace records for later performance analysis.

From the communication point of view, the TRANSCEIVER that runs on each virtual machine can be in one of two states at any given time: (1) transmitting/receiving messages and (2) producing/digesting messages. In the first case, the transmission is in use, while the transmission line is free in the second case. It is thus possible to compute the fraction of time the transmission line is busy for a given virtual machine. We shall call this the line usage. Line usage is not a parameter assignable by the operation, but can be varied by changing the speeds of message production and consumption in the TRANSCEIVERs.

At system initialization time, the operator can assign the following parameters: number of virtual machines (1 to 4), number of message buffers per virtual machine (1 to 32), injection of noise (yes/no), number of real-time clock pulses per quantum (1 to 16383), and instruction execution time for each virtual machine (1 to 10).

A representative experiment is one where the four virtual machines are given different execution rates, and the processing speeds were set to give relatively high line usage (about 50 per cent). The virtual machines are run to determine the number of messages sent and received by each virtual machine, the average number of messages waiting in the queue of a receiving virtual machine, the number of lost messages because of full queue, and the number of transmission errors of some well defined error types. A summary of the trace analysis for such an experiment is presented in Fig. 1. Results are organized into groups. System constants are those variables whose values are assignable by the operator at system generation time. In this experiment, four virtual machines are defined in the network with the following characteristics: maximum of 32 packets allowed to be in transit; eight packet buffers allocated to each VM; introduction of noise is desired; quantum time is 2 ms of real time; and the set of instruction execution times for each VM is 8, 7, 5, and 3 virtual time units respectively. The following general features were observed:

1. The number of messages sent is proportional to the speed (inverse of instruction execution time) of the virtual machine.

2. The number of messages received was inversely proportional to speed.

3. The average queue length at a VM was inversely proportional to speed.

4. The number of lost messages was inversely proportional to the speed of the receiving virtual machine.

5. The relative importance of transmission error detection mechanisms was evaluated.

## 5.0 FUTURE WORK

It is clear that the virtual machine monitor architecture can be used to replace the simulation of any computer network or distributed architecture. A number of further experiments with this system are planned.

## REFERENCES


1. Canon, M. D., Fritz, D. H., Howard, J. H., Howell, T. D., Mitoma, M. F., and Rodriguez-Rosell, J., "A Virtual Machine Emulator for Performance Evaluation", Seventh Symposium on Operating Systems Principles, Comm. ACM, p. 71, Vol. 23, No. 2, 1980.

2. CCITT, "Draft Revised Recommendation X.25; Interface Between Data Terminal Equipment (DTE) and Data Circuit-Terminal Equipment (DCE) for Terminals Operating in the Packet Mode on Public Data Networks".

3. Herrington, D. J., "The UT Virtual Machine Monitor Version 2.0", TR-119, Department of Computer Sciences, The University of Texas at Austin, May 1979.

4. Lamport, L., "Time, Clocks, and the Ordering of Events in Distributed Systems", Comm. ACM, p. 558, Vol. 21, No. 7, 1978.

5. Renaud, D. J., "The UT Virtual Machine Monitor", Systems Programming Laboratory Note 5, Department of Computer Sciences, The University of Texas at Austin, May 1978.

6. Wang, R. T., "The UT Network of Virtual Machines", TR-160, Department of Computer Sciences, The University of Texas at Austin, December 1980.


```
SYSTEM CONSTANTS:
   DATE: 0809
   NUMBER OF VM: 004
   NUMBER OF PACKETS: 032
   PACKETS PER VM = 08
   INJECTING NOISE :  YES
   QUANTUM: 002
   INST EX TIME FOR VM
           000 IS 008
           001 IS 007
           002 IS 005
           003 IS 003

000002736:  BEGIN SIMULATION.
000352294:  END SIMULATION.
000352558   TOTAL SIMULATION TIME.

LINE USAGES OF VM:
   0: 000259288 / 000352558 = 0.47
   1: 000264089 / 000352558 = 0.48
   2: 000280945 / 000352558 = 0.51
   3: 000295680 / 000352558 = 0.53

MESSAGES SEND:
                    TO
          0      1      2      3     SUM
 F   0  00000  00012  00012  00012  00036
 R   1  00048  00000  00049  00049  00146
 O   2  00199  00206  00000  00205  00610
 M   3  00557  00579  00578  00000  01714
    SUM 00804  00797  00639  00266  02506

LOST MESSAGES TO VM
   0: 023
   1: 000
   2: 000
   3: 000

AVERAGE QUEUE LENGTH AT VM
   0: 04.40
   1: 02.07
   2: 00.76
   3: 00.30

TOTAL ERROR COUNT = 381
           TYPE 2 = 09
           TYPE 3 = 07
           TYPE 4 = 41
           TYPE 6 = 324
```

Fig. 1: Summary of a Virtual Network Experiment