Evaluation Of Real-time Distributed Systems Using Ada Concurrency

1-1-1990

Michael Bassiouni
Ming Hsing Chiu
Jack Thompson

Find similar works at: https://stars.library.ucf.edu/istlibrary

University of Central Florida Libraries http://library.ucf.edu

Recommended Citation

Bassiouni, Michael; Chiu, Ming Hsing; and Thompson, Jack, "Evaluation Of Real-time Distributed Systems Using Ada Concurrency" (1990). Institute for Simulation and Training. 81.
https://stars.library.ucf.edu/istlibrary/81

This Research Report is brought to you for free and open access by the Digital Collections at STARS. It has been accepted for inclusion in Institute for Simulation and Training by an authorized administrator of STARS. For more information, please contact lee.dotson@ucf.edu.
Evaluation of Real-Time Distributed Systems
Using Ada Concurrency

M. Bassiouni and M. Chiu
Computer Science Department
J. Thompson
Inst. for Simulation and Training

University of Central Florida
Orlando, FL 32765

Abstract

Recent breakthroughs in computer and communications core technologies have made possible the interconnection of large number of real-time training devices via local area networks. The effectiveness and degree of realism achieved by team-training via distributed simulation are greatly influenced by the choice of the network topology and protocol used to interconnect the simulation devices. In this paper, we describe the design approach used to build Ada models for the performance evaluation of network protocols suitable for real-time distributed simulation. The paper describes the structure of the Ada software, highlights the important task inter-communication and synchronization aspects used in the design of the models, and presents examples of the performance results obtained via these models.

Key Words: distributed simulation, Ada concurrency, network protocols, real-time training systems.

1. Introduction

Recent breakthroughs in several computer and communications core technologies have made possible the interconnection of large number of real-time simulators (special purpose hardware) via local area networks. Two main applications/advantages of such networks are:

(1) To provide a low-cost effective tool for the training of personnel in applications involving interactions among mobile vehicles. Examples of such applications include training exercises for police forces, fire/ambulance services, and military combat fighting.

(2) To provide an effective "test before you build" development tool to be used for evaluating proposed modifications in existing systems, as well as an aid in designing/developing new systems. Tactics and coordination strategies might also be simulated and evaluated before they are adopted in real-life.

We shall use the terms "distributed simulation", "simulation networks", and "real-time training networks" interchangeably to denote the networking of a large number of real-time simulators for the purpose of training [6]. Each simulator consists of specialized hardware (a high-speed microcomputer, computer image generation subsystem, and sensor/control devices) bearing resemblance to the interior of the simulated vehicle (e.g., tank or police car). Each simulator has its own local copy of the database describing the simulated environment (e.g., city streets, buildings, terrain). As the crew of the simulated vehicle operate as they would in the real-life vehicle, the appropriate visual scenery is displayed on the CRT screens of their vehicle, as well as those of other vehicles in its sight range. It is obvious that the simulators participating in a training session must communicate with each other while carrying out the simulation. It is the responsibility of the underlying local area network (LAN) to provide each simulator with a reliable and fast mechanism to send and receive the information pertaining to the simulated activities.

The networking of real-time interactive simulation training systems departs from the traditional use of a computer network, whose function would normally be to provide sharing of computing resources among multiple users (nodes) on the network. When used to interconnect real-time simulators, the network is used almost exclusively for communication of process state information between the simulators engaged in the training exercise.

The Institute for Simulation and Training (IST) at the University of Central Florida has established a Network and Communications Technology Laboratory (NCTL/IST) dedicated to performing research for the purpose of enhancing the networking capabilities of distributed simulations. This laboratory houses a number of real-time
simulators (different types of simulated ground and air vehicles) and is the center of several research projects dealing with the various aspects of real-time distributed simulation. In this paper, we describe the design approach used to build Ada models for the performance evaluation of network protocols suitable for real-time distributed simulation. The Ada models have been developed to provide a tool that can predict the performance of simulation networks and to gain valuable insight into the problems associated with the interconnection of real-time simulation devices. The models are also being used to complement the experimental network testbed at NCTL/IST.

2. Network Configuration Models

Various choices exist for the implementation of a LAN [1, 2, 3, 5, 7] (e.g., transmission medium, topology, access protocols, etc.) to interconnect simulation devices. In this paper, we limit our discussion to token-based network protocols. Specifically, we describe a simulation approach using Ada which is suitable for token-ring and FDDI local area networks. The token ring configuration has a loop topology and employs a non-contention protocol that avoids collision by a token-passing mechanism [3, 4]. Figure 1 gives a block diagram of the basic configuration of the token-ring LAN. Simply stated, a token-passing ring is a LAN with a loop topology in which a token (a unique bit sequence in a data packet) is passed around the network, in a round-robin fashion, from one node to the next. Contention for transmission is resolved by stipulating that only the node currently in possession of the token is allowed to transmit a packet, or a sequence of packets, onto the ring. When the transmission is finished, the token is passed to the node downstream which then gains the right to transmit. Since there is a single token on the ring, only one node can be transmitting at a time. Other (non-transmitting) nodes, however, continuously receive the bit stream, examine it and repeat it onto the network (i.e., place it on the medium to the next station). A station repeating the bit stream may copy it into local buffers or modify some control bits if appropriate. A prototype token-ring scheme for real-time simulators has been recently completed at NCTL/IST.

The architecture of FDDI specifies a dual (counter rotating) token ring configuration that provides point-to-point connections between every pair of adjacent nodes on the ring. In normal operation, data packets circulate between the nodes using a single ring. The purpose of the dual ring is to gracefully recover from failures due to breaks in the transmission medium. Figure 2 shows a typical FDDI configuration that might be used to connect real-time simulation devices. Stations (i.e., simulators) are shown as nodes and connected in the form of a loop. Normally each simulator will be directly attached to the ring, but FDDI also allows the use of concentrators which can connect a number of stations to the ring. In simulation networks, these concentrators might be used to connect either a group of nodes which belong to the management and control of the training exercise, or to connect a group of regular nodes that are physically located near each other.

3. Properties of Networked Training

The application of networks to interconnect real-time simulators (for the purpose of training) has a

![Fig. 1 Token ring network topology](image-url)
number of characteristics and requirements. Recall that the main function of the LAN in this application is to communicate state update messages. When the state of a simulator changes (e.g., due to change in position or velocity, physical destruction, etc.) the simulator broadcasts a data packet of type "state update". This message is delivered by the network to every other simulator or node on the network. Upon receiving a state update message, each simulator updates its own local database and displays any appropriate changes on its screens. To accomplish this function under real-time constraints, the design of a simulation LAN must satisfy the following requirements:

1. The network must provide connectionless data transfer services (datagram services) that include point-to-point transfer, multicasting, and broadcasting.

2. The transmission delay incurred by a packet should be minimal [6].

3. The percentage of lost packets should be kept as close to zero as practically possible. The impact of lost packets on the fidelity of the training exercise depends on the type/contents of the packet. For example, the loss of a single state update packet from a slowly moving vehicle can be usually tolerated and would not much degrade the animated imagery displayed by other simulators. This is because the simulator of this vehicle will soon broadcast another state update message after a small time interval and, hence, its coordinates in the database of other simulators will be corrected.

4. The Ada Simulation Model

In this section, a high-level description of an Ada simulation model used in evaluating and predicting the performance of token-ring local area networks [3, 4] will be given. A similar simulation model has also been developed for FDDI LANs [1]. Since FDDI is primarily a token-based scheme, the basic design strategy described below for the token ring simulation model applies also to the FDDI counterpart. The concurrency mechanism in Ada has been used in our models to simulate the different concurrent activities within a simulation (training) network. A task in Ada may have entries which can be called by other tasks. Synchronization between two tasks occurs when the task issuing an entry call and the one accepting it establish a rendezvous. Communication in both directions is achieved via the parameters (input parameters) passed to the task accepting the entry call and those (output parameters) returned to the task issuing the entry call. This powerful synchronization facility has provided us with a convenient and elegant tool for modeling the parallel activities of the simulation network and the underlying networking protocol. The process interaction model of Ada has been used in our simulation to map the different entities and activities of the simulated network to corresponding Ada tasks. The following task types are the major generic entities used in the simulation of token ring LANs. Fig.3 depicts the interactions among these different tasks.

1. **Source task**: is used to represent a vehicle simulator on the network. A task of this type is created for each such simulator
2. **Node task**: is used to represent the point of contact of each network node with the ring medium. It performs the functions of the medium access control (MAC) layer protocol and ensures that the token-ring protocol is executed. A task of this type is created for each network node.

3. **Server task**: is used to implement and control the flow of data on the ring. A task of this type corresponds to one of the point-to-point connections of the token ring LAN.

4. **Scheduler task**: (not shown in Fig. 3) is used to order timed events and control the sequencing of activities of the entire simulation.

In Ada, a task is composed of a specification and a body. The specification declares entries in which the data type and the input/output status must be specified clearly for each parameter. The body is the code that defines the activities of the task. Below, a brief functional description of each task is given. The description is written in pseudo code and only aspects related to the synchronization of the parallel activities within a token ring simulation LAN are considered.

**Source Task**

In this task, local traffic is generated according to a specified input method (e.g., using traces of real data or random stochastically generated inter-arrival times such as exponential, uniform, fixed with jitter, etc.). Whenever there is a packet in the queue, the Source task makes a request via an entry call to its Node task to transmit the packet to the network. The request may be blocked until the packet is accepted by the Node task.

```
-- task specification --
task type source_type is
  entry input (id, pktlength: in integer;
  sim_t, mean_iit: in float);
end source_type;

-- task body --
task body source_type is
  accept input (id, pktlength: in integer;
  sim_t, mean_iit: in float) do
    -- get input parameters
    end input;
    -- subscribe to the scheduler task
    sched.addUser;
    -- become a producer of the Node task
    nodes(my_node).addProd;
    -- main processing phase --
    -- get arrival time for the first packet
    t := erand(mean_iit);
    -- request delay to wait for arrival
    sched.reqDelay(t, my_id);
    -- transmit the first packet
    nodes(my_node).transReq(pktlength, t);
    while simulation time has not expired loop
      -- get the arrival time of next packet
      t1 := erand(mean_iit);
      sched.now(time) -- get current time
      if time < t+t1 then
        -- wait for arrival of packet
        sched.reqDelay(time-t-t1, my_id);
      end if;
      -- update the packet arrival time
      t := t + t1;
```

---

**Fig. 3 Simulation model for token ring LAN**
transmit this packet
nodes(my_node).transReq(pktLength.i);
end loop;

-- termination phase --
-- no longer a producer of Node task
nodes(my-node).dropProd;
-- no longer a user of Scheduler
sched.dropUser;
end source_type;

Node Task

The Node task acts like a server ready to accept
entry calls from its Source task or its neighboring
Server tasks. When a packet from the producer
(upstream) Server arrives, the Node task examines
the type of the incoming packet. If it is a token,
the Node task checks whether there is a pending
local packet and if so, appends the free token at
the rear of the packet and make them available to
the consumer (downstream) Server. If it is a
message packet, the Node may absorb it or make it
available to the consumer Server depending on
whether this packet was locally generated.

-- task specification --
task type node_type is
  entry input (sim_t, trans_rate: in float;
              FT_length, CT_length: in integer);
  entry addProd;
  entry dropProd;
  entry addCons;
  entry dropCons;
  entry transReq (length: in integer;
                  start_t: in float);
  entry putmsg (nodeItem: in nItem);
  entry getmsg (item: out nItem);
end node_type;

-- task body --
task body node_type is
  -- set up parameters
  accept input (sim_t, trans_rate: in float;
               FT_length, CT_length: in integer);
  end accept;
  sched.addUser; -- subscribe to scheduler
  sched.passive; -- initial state
  -- wait for first producer
  accept addProd do
    increment producer count
    end addProd;
  end select;
  -- main processing phase --
  while there are clients loop
    select
      when Ok_local_packet =>
      or
      or
      or
      or
      or
      or
      when Ok_local_packet =>
        accept transReq (length: in integer;
                       start_t: in float) do
          -- get a new local packet
          -- make it a pending packet
          end transReq;
        or
        accept putmsg (Item: in nItem) do
          -- get token/message-packet
          if msg is the free token then
            begin
              record the token cycle time
              if there is pending packet then
                append FT to its rear; endif;
              msgReady := True;
              end
            or
            begin
              if packet is mine then absorb it
              else msgReady := True;
              end
            end putmsg;
        or
        when msgReady =>
          -- unblock consumer server
          accept getmsg (Item: out nItem) do
            if this was my pending packet then
              Ok_local_packet := True;
              end getmsg;
            or
            accept addProd do
              -- increment producer count
              end addProd;
            or
            accept addCons do
              -- increment consumer count
              end addCons;
            or
            accept dropProd do
              -- decrement producer count
              end dropProd;
            or
            accept dropCons do
              -- decrement consumer count
              end dropCons;
            end select;
            end loop;
        -- print statistics --
  end select;
  sched.dropUser -- withdraw
end node_type;

Server Task

The Server Task is used to implement and control
the flow of data in the network. Each Server task
takes the packet from its producer (upstream)
Node and delivers the packet to its consumer
(downstream) Node. The Server task also keeps track of the progress of propagation of the packets that it has transmitted and which have not yet arrived at the downstream node.

-- task specification --
task type server_type is
  entry input(id: in integer; sim_t, prop_time, trans_rate: in float);
end server_type;

-- task body --
task body server is
  -- initialization phase --
  accept input (id: in integer; sim_t, prop_time, trans_rate: in float) do
    end input;
  -- request to be a user of Scheduler
  sched.addUser;
  -- inform adjacent nodes
  nodes(upstream).addCons;
  nodes(downstream).addProd;
  -- main processing phase --
  while simulation time has not expired do
    -- request packet from producer Node
    nodes(upstream).getmsg(item);
    -- let ttrans be the time remaining for
    -- the transmission of current packet
    -- let tpropag be the remaining time
    -- until first traveling packet arrives
    -- at downstream node
    while tpropag < ttrans do
      begin
        sched.reqDelay(tpropag,my_id)
        nodes(downstream).putmsg(item);
        update ttrans and tpropag
      end;
      sched.reqDelay(ttrans,my_id);
    end loop;
  -- termination phase --
  nodes(downstream).dropProd;
  nodes(upstream).dropCons;
  sched.dropUser;
end server_type;

Scheduler Task

The Scheduler maintains a list of all the tasks that make delay requests and reactivates these tasks at the proper time. When there is no active task in the system, the Scheduler reactivates the task at the head of the list and advances the clock accordingly. A task may be in the waiting state, in the active state, or in the passive state. The latter state is used when a task is blocked indefinitely, for instance, when a Server task requests to take a packet from its producer node but the packet is not available.

-- task specification --
task sched is
  entry now (ctime: out float);
  entry reqDelay (dt: in float; tid: in integer);
  entry addUser;
  entry dropUser;
  entry passive;
  entry active;
end sched; -- specification

-- task body --
task body sched is
  -- initialization phase --
  -- accept the first user of Scheduler task
  accept addUser do
    increment client and active count
  end accept;

  -- main processing phase --
  -- accept requests while there are clients
  while client count > 0 loop
    select
      accept addUser do
        increment client and active count
      end addUser;
      or
      accept dropUser do
        -- decrement client and active count
      end dropUser;
      or
      accept passive do
        -- decrement active count when a
        -- task enters passive state
      end passive;
      or
      accept active do
        -- increment active count when a
        -- task in the passive state
        -- becomes active
      end active;
      or
      accept now (ctime: out float) do
        -- set ctime to current time
      end now;
    end select;

    -- handle a delay request
    accept reqDelay (dt: in float;
       tid: in integer) do
      -- decrement active count
      -- add task to the waiting list
    end reqDelay;
  end loop;

  -- when there is no active task and
  -- the waiting list is not empty
if active count = 0 and list not empty
advance clock
increment active count
reactivate and delete head of list
end if;
end loop;

-- termination phase --
end sched;

In addition to the above entities, several auxiliary tasks/packages are utilized for the purpose of:
collecting statistics, functions definition, user interfacing and task dispatching, etc. The software system is implemented in a modular fashion with emphasis on ease-of-modification and the use of parameterized values that facilitate the testing of a wide range of network characteristics and the simulation of different load conditions and different network parameters.

5. Performance Results

The Ada simulation models have been used to gain insight into the performance of simulation networks under both the token-ring and FDDI protocols. The models have been used to predict the performance of these two schemes when used to support a large number of real-time simulators. In what follows, we give examples of the results obtained by these models.

For the token-ring model, the recreation of the "free token" onto the ring is assumed to follow the "early token release protocol". According to this protocol, the transmitting station (the one which removed the free token from the ring) recreates the free token and puts it onto the ring immediately after it finishes transmitting its packet. This protocol results in better LAN throughput and smaller packet delays than protocols that require the header (or the tail) of the transmitted packet to complete one cycle around the ring before the free token is recreated. Another factor affecting the performance of the token-ring scheme is the length of the free token. Although this length has been used as a variable in our various tests, the results reported in this section use a length of 24 bits (24 bits is the length used in many commercial token-ring implementations).

The FDDI scheme uses an underlying token passing mechanism with appropriate modifications to handle synchronous data. If the FDDI parameters (thresholds and initial timer values) are carefully chosen, the FDDI LAN operates smoothly without (or with very few) reinitializations. In this case, the performance of FDDI is basically that of a high speed token ring. Unlike ETHERNET [2] whose performance deteriorates at high traffic loads due to excessive packet collisions, the overhead of FDDI token management does not result in throughput decline when the traffic load on the ring increases. Figure 4 shows a typical

![Figure 4](image-url)
relationship between the throughput and traffic load in a token ring LAN. Since the token ring and FDDI protocols use a collision-free scheme, they do not suffer from the problem of declining performance at high loads. Throughput around 90% of the transmission medium bandwidth can be easily obtained in token ring and FDDI LAN's (compare this to the ETHERNET protocol whose throughput is usually limited to about 65% of the medium bandwidth). Figure 5 shows a typical relationship between traffic load and the average time required for a packet to be successfully communicated through the network.

6. Conclusions

In this paper, we have described the high-level details of Ada simulation models used to evaluate the performance of networked simulation (training) systems under token based network protocols. The Ada models are also being used to perform a comparison study and evaluate different design decisions. Some of the numerical performance measures that are being gathered by the models are: the impact of traffic load on network throughput, the utilization of the transmission medium, and the distribution of delay times of transmitted packets. Further work is underway to use these models in evaluating schemes to incorporate real-time voice services within simulation networks.

Acknowledgements

This work has been supported by the Defense Advanced Projects Research Agency (DARPA) and the Army's Program Manager for Training Devices (PM TRADE) under Broad Agency Announcement #88-01, Contract N61339-89-C-0043. The views and conclusions herein are those of the authors and do not represent the official policies of the funding agency, the Institute for Simulation and Training, or the University of Central Florida.

References


About the Authors

M. A. Bassiouni received his Ph.D. degree in Computer Science from Pennsylvania State University in 1982. He is currently an Associate Professor of Computer Science at the University of Central Florida, Orlando. His current research interests include computer networks, distributed systems, databases, and performance evaluation. He has authored several papers and has been actively involved in research on local area networks, concurrency control, data encoding, I/O measurements and modeling, file allocation, and relational user interfaces. Dr. Bassiouni is a member of IEEE and the IEEE Computer and Communications Societies, the Association for Computing Machinery, and the American Society for Information Science.

M. Chiu is a graduate student at the Department of Computer Science, the University of Central Florida. For the past three years, he has worked as a research assistant in the Communications and Networking Laboratory at IST/UCF. Mr. Chiu has played a key role in the design, coding and debugging of several simulation programs written in Concurrent C and Ada.

J. Thompson received his BS degree in Electrical Engineering from the University of Central Florida in 1978. He is currently a Research Associate for the Institute for Simulation and Training (IST) at the University of Central Florida, Orlando. Mr. Thompson has technical responsibility for all IST research activities involving computer and simulation networking.