Modeling a Real-Time Multitasking System in a Timed PQ Net

Carl K. Chang, **Young-Fu Chang**, and **Lin Yang**, *University of Illinois at Chicago*
Ching-Roung Chou and **Jong-Jeng Chen**, *National Chiao Tung University, Taiwan*

Yesterday's performance-analysis models are inadequate for today's systems. This system integrates two popular approaches — timed petri nets and queuing networks — to model complex, real-time systems.

System performance concerns most system analysts. To model performance, you need a practical modeling method tailored to your environment. Existing modeling methods, which can model either queuing behavior or asynchronous concurrent behavior, are inadequate for today's complex systems. Many applications, including multiprocessor operating systems and distributed systems, require both modeling capabilities.

To fill this need, many modeling methods are being modified and extended, and some new methods have been introduced.[1] We present an approach that uses an enhanced model based on two familiar modeling methods, the queuing network and petri net. Our approach includes a graphical modeling tool, called TPQN,[2] a textual specification language, called TPQL, and a simulator, called TPQS.

TPQN can represent a system with both synchronization conditions and queuing behaviors, something that cannot be modeled with either the petri net or queuing network alone. We illustrate TPQN's capabilities with a performance analysis of a real-time, multitasking scheduler that had been previously implemented on top of SunOS on a Sun-3 workstation.[3] The scheduler was developed to control task scheduling in a distributed simulator of telephone switches, but we use a simplified abstract model in this article.

We have conducted some experiments with the TPQS simulator on a Sun-3. The results have helped us analyze the system's performance and validate our modeling tool.

Two approaches

The queuing network has been used widely as a purely mathematical technique for modeling sequential systems' random behavior.[4] In principle, a queuing network is suitable to model systems that can be represented as a network of queues. It pro-

0740-7459/89/0300/0046/$01.00 © 1989 IEEE

vides a compact representation that describes a system's random nature. However, it cannot model concurrency and synchronization conditions easily, something petri-net based models can do more naturally.

The petri net is a graphical modeling method that has sufficient mathematical foundation to support formal analysis. It is widely used to model asynchronous systems and concurrent processes. The petri-net model and its variations are graphical, mathematical modeling tools that can be applied to provide a formal, yet more visual representation of the properties of asynchronous, concurrent systems.[5] Petri nets can

- represent concurrent execution of multiple processes,
- represent nondeterminant and asynchronous executions,
- be decomposed and composed into many graphs, and
- reveal a model's structure and dynamics for analysis.

Figure 1 shows a petri-net model for a simple producer/consumer system. In the model, conditions are represented by place nodes and events by transition bars. You place a token on a place node to represent that a certain condition is held on that node. Directed arcs connect nodes to bars and bars to nodes. A transition bar (an event) can fire (occur) if all the nodes (conditions) that input to that transition bar have tokens (are holding conditions). When a transition bar fires, it removes one token from its connecting input nodes and places one token on its connecting output nodes.

You can determine the state of a petri net by observing the collection of names of the nodes that hold tokens (the number of tokens a node holds is equal to the number of instances of a node name in the state). This is called the marking of a petri net. The places item and receive_ready are the input set for the transition get_item. The output set of the transition get_item contains a single place ready_to_signal.

Conflicts arise when more than one transition is connected to a place node. For example, in Figure 1, item_lost and get_item are connected to the place item. Because only one transition can be fired when there is a token in place item, these two places are called a conflict set.

The original petri-net model has been much enhanced over the years. Among the enhanced models are the timed petri net,[6] which considers the firing frequencies and deterministic firing time of each transition in the net; the stochastic petri net,[7] which also represents time but can accommodate both continuous and discrete time; and the generalized stochastic petri net,[8] which associates an instantaneous or exponentially distributed firing time for each transition.

The petri net is a graphical modeling method that has sufficient mathematical foundation to support formal analysis.

These time-oriented petri net models have been applied to applications such as file systems and telecommunication systems to do performance analysis. Our modeling method, the timed PQ net, integrates the timed petri net's features with the queuing network.

TPQN

Formally, a PQ net is defined as a six-tuple:

PQ = (P, T, Q, I, O, M′) where

$P = p_1, p_2, \ldots p_n$

$T = t_1, t_2, \ldots t_m$

$Q = q_1, q_2, \ldots q_l$

$I = \subseteq P \times (T \cup Q)$

$O = \subseteq (T \cup Q) \times P$

$M' = m'_1, m'_2, \ldots m'_n$

As in a general petri net, P is the set of places; T is the set of transitions; I is the input function, which maps a transition (or a queue) to a collection of places,

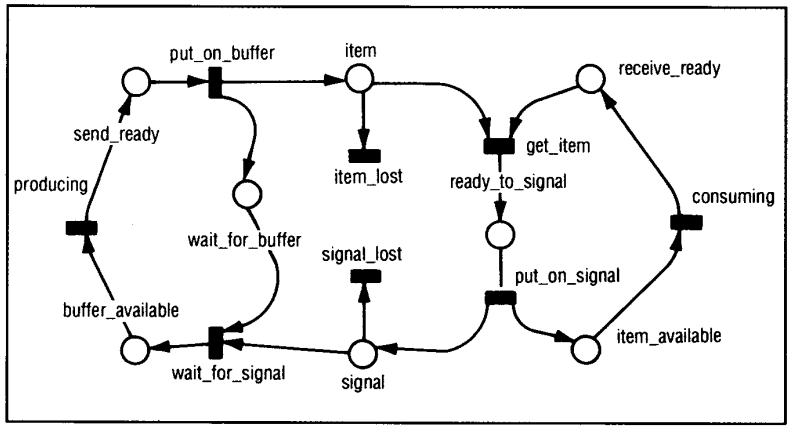

Figure 1. A petri-net model of a producer/consumer system.

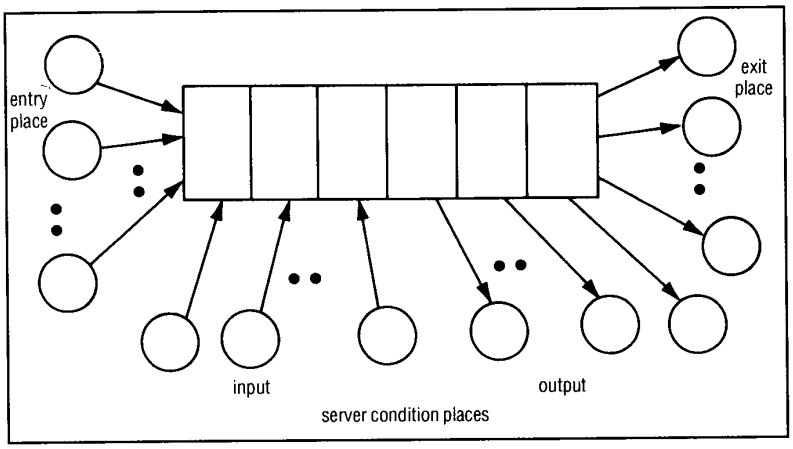

Figure 2. A graphical representation of a queue in a PQ net.

known as the input places of the transition; O is the output function, which maps a transition (or queue) to a collection of places, known as the output places of the transition; M′ is is the initial marking, which gives the number of tokens in each place. However, Q is a special set that denotes a finite set of queues.

Each element q_i in Q represents a queue specified by a five-tuple:

$(N_i, M_i, L_i, qd_i, k_i)$

where N_i is a set of n pairs of input/output places—the queue's entry and exit places, M_i is a set of m places to specify server input conditions, L_i is a set of l places to specify server output conditions, qd_i is a queuing discipline (which can be first-come, first-served; last-come, first-served; last-come, first-served with preempt resume; random choice; or process sharing), and k_i specifies the size of the queue.

Figure 2 shows a graphical representation of a queue. A queue is a special transition that can hold several waiting jobs. A token enters the queue immediately after it goes into one of the entry places, provided the queue is not full. The token is stored in the queue in an order defined by the queuing discipline qd_i. A queue is enabled to fire whenever there are tokens in all its input places and in the queue itself. After it fires, one token is taken from each input place and from the queue. These tokens are added to the output places, which are in one-to-one correspondence with the input places.

A timed PQ net is a two-tuple $<PQ, F>$ where PQ is the PQ net and F is a set of triples $<e_i, f_i, r_i>$. Each e_i specifies the density function of the enabling time, the time required to start an event. Each f_i specifies the density function of the firing time, the time required to finish an event. And r_i specifies the relative firing frequency of transitions or queues that are in conflict. Furthermore, f_i can be defined as a set of functions $f_{i1}, f_{i2}, \ldots f_{in}$, with each element specifying the service time density function of one class of tokens in q_i.

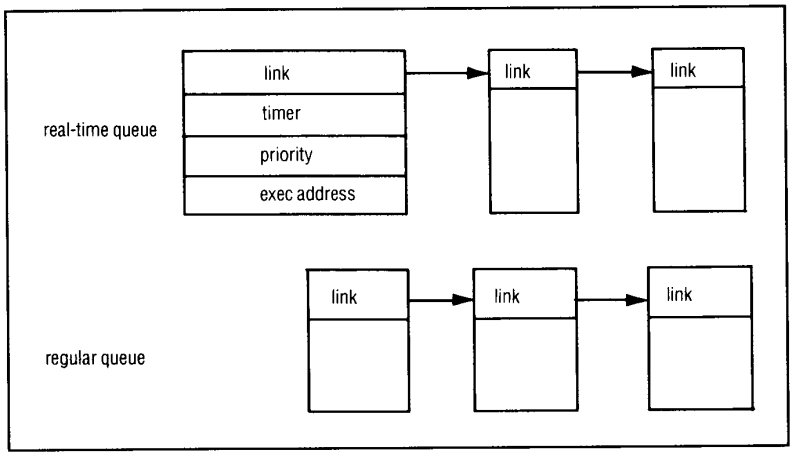

Figure 3. The structure of the two ready queues in an example real-time multitasking scheduler.

Real-time example

All operating systems that support multiprocessing do so through CPU scheduling: They switch the CPU between processes to make the computer more productive. Deciding how to share the CPU among processes involves three factors:

1. How should the CPU be shared? What mechanisms are required to guide a CPU that is executing one process to change its activity and execute another process?
2. When should the CPU be shared? As a result of what events should the CPU switch from one process to another?
3. Which process should get the CPU's attention?

Selecting which process to run is based on the relative importance of all eligible processes. Obviously, the process selected should be the one that is most important for the system's operation.

The problem is deciding which is the most important process. Process importance is usually stated in terms of priority: A more important process has a higher priority than a less important one. In addition to assigning priorities, a real-time scheduler must also switch among processes fast enough to observe stringent real-time requirements. Real-time processes, which normally demand real-time responses, are assigned the highest priority.

In a uniprocessor system, there is never more than one process running. If one process is running, the rest must wait until the CPU is free and can be rescheduled. The processes that are waiting to execute are listed in a ready queue, which is a linked list. A ready-queue header contains pointers to the first and last process-control blocks in the list. Each process-control block has a pointer field that points to the next process-control block in the queue. The ready queue's structure is determined by the scheduling algorithm; it is

not necessarily first-in, first-out. However, all of the processes in the ready queue are linked up, waiting for a chance to run on the CPU.

There are two queues in our real-time scheduler model, the real-time queue and the regular queue. Tasks in both queues are executed according to a scheduling policy, such as first-come, first-served.

The processes in the regular queue can be executed only when the real-time queue is empty and there is time left in the current time slice. These conditions mean the model must have both queuing and synchronization conditions. In cases like this, neither petri nets nor queuing theory alone can provide a complete model.

Our scheduler has two ready queues, as Figure 3 shows. We implemented the scheduler as fully interrupt-driven and defined the scheduling process to run in fixed time intervals. The interval between schedule-process activation defines the system's minimum integral delay. For example, if the processes are activated every 20 ms, other processes may delay themselves only by 20 ms, 40 ms, 60 ms, and so on.

When an application program needs service, it generates its own process-control block and sends it either to the real-time priority ready queue or the regular priority ready queue, according to its priority. The processes in both queues are ready to execute.

Figure 4 shows our real-time scheduler model with its two queues: a real-time priority queue and regular priority queue. Tasks arriving at the system, which can be either real-time or regular, join their respective queues. The branching probabilities for either queue are specified at the selection arcs.

According to the real-time scheduling policy, a real-time task must finish during each time interval. In the model, place 10 controls the switching between real-time and regular tasks. Because place 10 is connected to both queues, the regular queue is connected to place 10 with an inhibited arc. If a token is in place 10, only queue 1 (the real-time queue) is enabled. Queue 2 (the regular queue) is disabled by the inhibited arc. If there are real-time tasks in queue 1, tokens are accumulated in place 4. After the task executed, queue 1 outputs a token and fires transition 4. Whenever there are no tokens in place 4, the real-time tasks are finished and the system is switched to execute regular tasks. Places 8, 9, and 10 are timing-control systems. Transition 6 fires periodically and the token generated in place 10 is for task control.

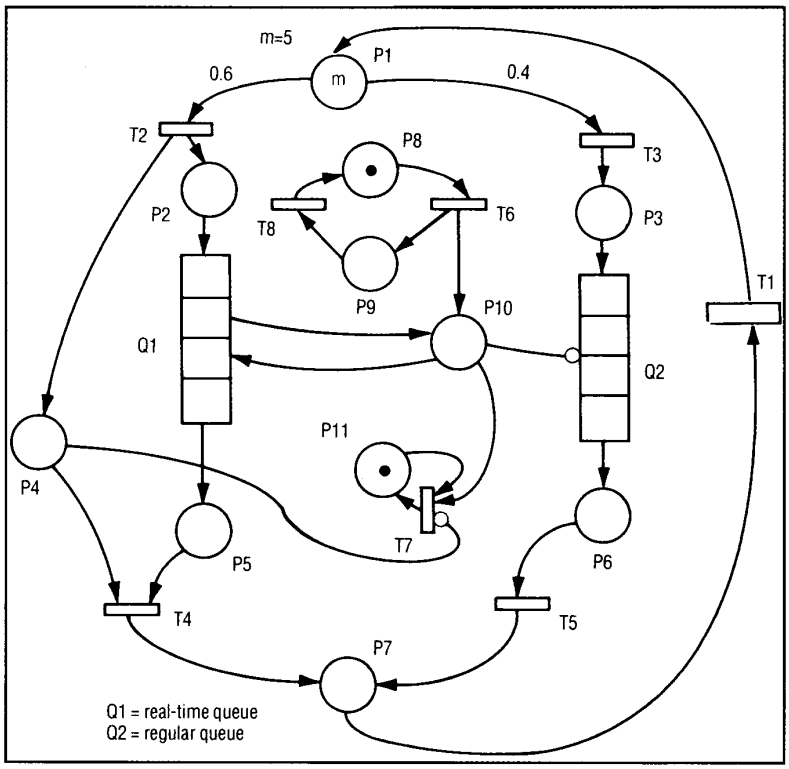

Figure 4. An example real-time multitasking scheduler modeled in TPQN.

This real-time multitasking scheduler exhibits two distinct characteristics: concurrency synchronization (synchronization between queues) and queuing (processes are queued in different queues with different priorities). It is difficult to find a single model that describes both these behaviors fully. TPQN does this by combining theories in petri nets and queuing networks.

TPQL and TPQS

This added description power makes TPQN very complex. To simulate a TPQN model, we provide an easy-to-use simulation language, TPQL,[9] to describe the user's model. TPQL provides 15 specification statements for places, transitions, queues, and so on. Figure 5 shows the TPQL textual representation of the graphical TPQN model in Figure 4. The textual representation usually complements the graphical representation because it can be processed more easily.

```
PLACES          11;
TRANSITIONS     8;
QUEUES          2;
TYPE            Q1:FCFS
                Q2:FCFS;
SIZE            Q1,Q2:INFINITE;
MARKING         P1:5
                P8:1
                P11:1;
CONDITION       T1:P7/P1
                T2:P1/P2,P4
                T3:P1/P3
                T4:P5,P4/P7
                T5:P6/P7
                T6:P8/P9,P10
                T7:P4',P10,P11/P11
                T8:P9/P8
                Q1:P10/P10
                Q2:P10';
IO              Q1:P2/P5
                Q2:P3/P6;
FIRE            Q1:CN(0.01,0.2)
                Q2:CE(0.3)
                T1:CE(0.1)
                T8:1;
PROB            T2:0.6
                T3:0.4;
RUN             5000;
```

Figure 5. The textual representation in TPQL of the TPQN model in Figure 4.

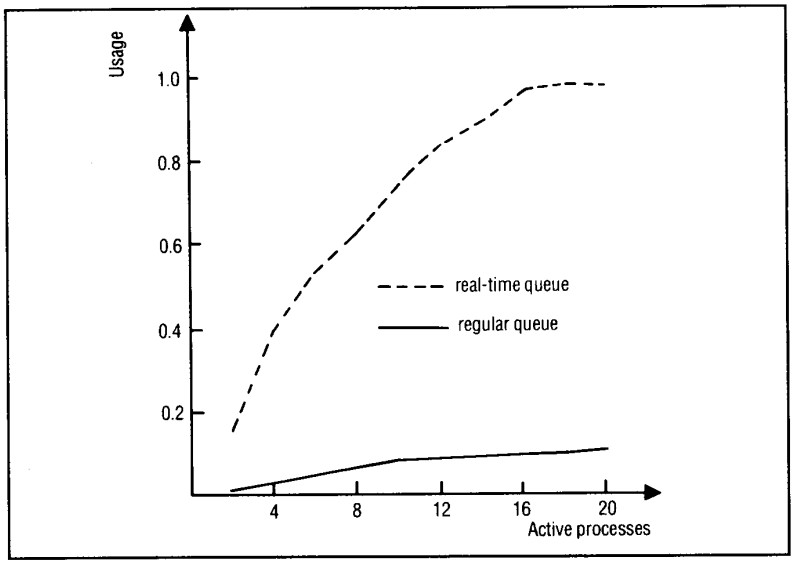

Figure 6. Queue usage in the real-time multitasking scheduler as modeled in TPQS.

TPQL is the input language of the TPQS simulation system,[10] which we developed to simulate TPQN models. We have implemented the five queuing disciplines in TPQS. Furthermore, TPQS provides 13 distribution functions for users to specify the enabling time of transitions and queues: Nine are for continuous distributions, four are for discrete distributions.

A parser in TPQS translates a model specified in TPQL into an intermediate file. The simulator uses this file to generate the simulation results, including performance attributes, and reachability sets. The system analyst can then use these results to evaluate and analyze system performance.

We are still investigating analytical solutions to many analysis problems based on TPQN. However, we have developed a simulation package for TPQN that lets us study the behaviors of various complicated systems under a designated queuing network topology (such as response time and mean queue length) before we implement the system. We have applied the TPQN simulator to the real-time multitasking scheduler model and have evaluated its performance.

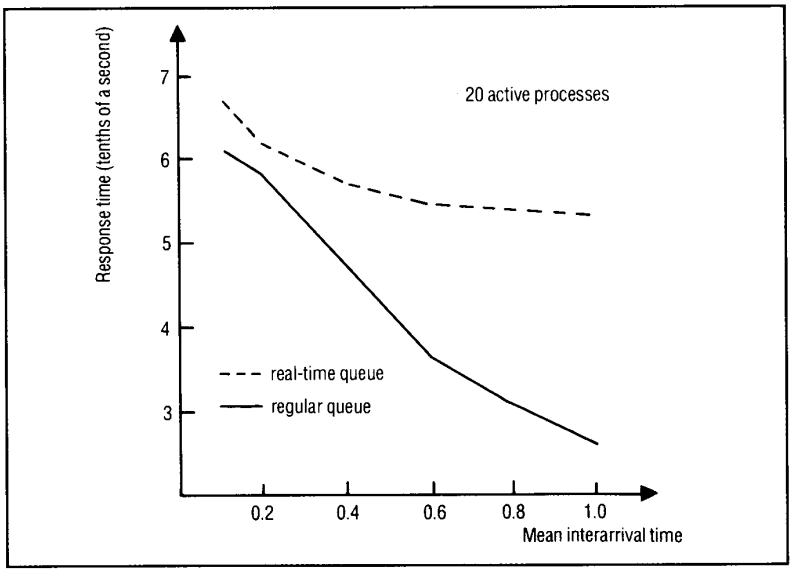

Figure 7. Average response time in the real-time multitasking scheduler as modeled in TPQS.

Figures 6 and 7 depict the curve according to the simulation result of TPQS.

In Figure 6, when the number of active processes (that is, the number of tokens in place P1) in the system increases along the *x* axis (that is, the system's customers increases), usage of the real-time queue approaches 100 percent rapidly, according to the parameters set in our specification. Because real-time tasks must be handled immediately, most of the system resources will be allocated to real-time tasks as the system load increases. Therefore, the simulation results shown in the figure are consistent with the service discipline of real-time tasks.

Figure 7 describes the average response time of both queues according to the average interarrival time of the customers (that is, the firing rate of transition 1). The curve shows that real-time tasks will be served at higher priorities, and the average response time is kept at a certain level. Because the customers in the regular queue use only the remaining time in each time slice (the time in which all customers in the real-time queue have to be served), it is more senstitive to the customer-arrival rate.

These simulation results confirm our intuitive observations. By combining the results shown in Figures 6 and 7, the TPQS package provides us with useful information on the average response time and queue utilization to analyze the system's performance.

It is hard to predict the performance of real-time software without a correct model and adequate tools. With the TPQN, we can create and verify the performance of a correct model of a real-time multitasking scheduler. Our real-time multiprocessor scheduler example illustrates that TPQN is suitable for modeling a system that involves complex synchronization and queuing analysis.

We are now studying extensively the analytical capabilities of the TPQN model. We would like to compare our simulation results with these analytical results as they become available. ❖

Acknowledgment

The authors greatly benefited from a discussion with T.M. Jiang during the preparation of this article.

References

1. C.K. Chang, C.-C. Song, and Y.-F. Chang, "Integral: An Integrated Framework for Distributed Software Validation and Verification," *Workshop Future Trends of Distributed Computing Systems in 1990s*, CS Press, Los Alamitos, Calif., 1988, pp. 301-310.
2. C.R. Chou, "Modeling Asynchronous Concurrent System Using a Timed Petri Net with Queues," *Proc. Int'l Computer Symp.*, Nat'l Chen-Kung Univ., Tainan, Taiwan, 1986, pp. 607-615.
3. C.K. Chang, M. Aoyama, and Y.-F. Chang, "UICPBX: A Distributed Simulator of Switching Systems," *Proc. Summer Simulation Conf.*, Soc. Computer Simulation Int'l, San Diego, 1988, pp. 352-357.
4. L. Kleinrock, *Queuing Systems, Vol. I: Theory*, John Wiley & Sons, New York, 1975.
5. T. Murata, "Modeling and Analysis of Concurrent Systems," in *Handbook of Software Engineering*, C.R. Vick and C.V. Ramamoorthy, eds., Van Nostrand Reinhold, New York, 1983, pp. 39-63.
6. C. Ramchandani, *Analysis of Asynchronous Concurrent Systems by Petri Nets*, doctoral dissertation, Massachusetts Inst. of Technology, Cambridge, Mass., 1973.
7. M.K. Molly, "Performance Analysis Using Stochastic Petri Nets," *IEEE Trans. Computers*, Sept. 1982, pp. 913-917.
8. M.A. Marsan and G. Conte, "A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems," *ACM Trans. Computer Systems*, May 1984, pp. 93-122.
9. C.R. Chou and G.J. Yeh, "TPQL: A New Simulation Language for Timed PQ-Net Models," *Proc. Int'l Computer Symp.*, Tam-Kang Univ., Taipei, Taiwan, 1988, pp. 379-384.
10. C.R.Chou and D.L. Mao, "The Performance Evaluationof Design Alternatives for the Supermini Architecture Using TPQS," *Proc. IFIP Working Group 7.3 Int'l Seminar of Performance Distributed and Parallel Systems*, Elsevier, Amsterdam, 1988.

Carl K. Chang is an assistant professor of computer science and director of a software-engineering laboratory at the University of Illinois at Chicago. His research interests include specification languages, software testing, and knowledge-based software engineering.

Chang received a BS in mathematics from National Central University in Taiwan, an MS in computer science from Northern Illinois University, and a PhD in computer science from Northwestern University. He is a member of ACM, Sigma Xi, and Upsilon Pi Epsilon. He is a senior member of IEEE and serves on the *IEEE Software* editorial board.

Address questions about this article to Carl K. Chang at EECS Dept., M/C 154, University of Illinois, PO Box 4348, Chicago, IL 60680.

Young-Fu Chang is a research assistant in a software-engineering laboratory and a doctoral candidate in the Electrical Engineering and Computer Science Dept. at the University of Illinois at Chicago. His research interests include specification verification, real-time systems, and performance analysis.

Chang received a BS and MS in electrical engineering from National Taiwan University. He is a member of the IEEE Computer Society.

Lin Yang is a doctoral candidate in computer science at the University of Illinois at Chicago. Her research interests include the performance evaluation of concurrent systems and the specification and properties proof of such systems using temporal logic.

Yang received a BS in electrical engineering from Beijing Polytechnic University and an MS in electrical engineering from the Illinois Institute of Technology.

Ching-Roung Chou is an associate professor and chairman of the Computer Science and Information Engineering Dept. at National Chiao Tung University in Taiwan. His research interests include distributed and multiprocessor system design, performance modeling and evaluation, parallel processing, software engineering, and operating systems.

Chou received a BS in electrical engineering from National Taiwan University, an MS in computer science from National Chiao Tung University, and a PhD in computer science from Northwestern University. He is a member of the IEEE Computer Society and Phi Tau Phi.

Jong-Jeng Chen is a postgraduate student in the Computer Science and Information Engineering Dept. of National Chiao Tung University, Taiwan. His research interests include analysis and evaluation of system performance.

Chen received a BS in computer engineering from National Chiao Tung University.