SIMULATION OF FAULT-TOLERANT SCHEDULING ON REAL-TIME MULTIPROCESSOR SYSTEMS USING PRIMARY BACKUP OVERLOADING

Bindu Mirle and Albert M. K. Cheng

Real-Time Systems Laboratory
Department of Computer Science
University of Houston
Houston, TX, 77204-3010, USA
http://www.cs.uh.edu

Technical Report UH-CS-06-04

May 21, 2006

Keywords: Fault Tolerance, Real-Time Multiprocessors, Primary Backup, Overloading, Backup Overloading.

Abstract

Time-critical applications require dynamic scheduling with predictable performance. Tasks corresponding to these applications have deadlines to be met despite the presence of faults. This paper discusses and implements the combination of a successful fault tolerant algorithm with the EDF algorithm for multiprocessors running in parallel and executing real-time applications. The algorithm is based on a popular fault-tolerant technique called primary/backup (PB) fault tolerance. The algorithm has been extended to include and consider tasks arriving dynamically unlike the PB technique which only accommodated static tasks. The backup overloading Hengming-Farnam Algorithm implements mechanisms which have helped in overcoming the limitations of the primary/backup technique. It has been successfully shown by the simulation results that primary backup overloading increases system utilization and also increases efficiency.

*This work is supported in part by the UH Real-Time Systems Laboratory and the Institute for Space Systems Operations.

SIMULATION OF FAULT-TOLERANT SCHEDULING ON REAL-TIME MULTIPROCESSOR SYSTEMS USING PRIMARY BACKUP OVERLOADING

Bindu Mirle and Albert M. K. Cheng

Abstract

Time-critical applications require dynamic scheduling with predictable performance. Tasks corresponding to these applications have deadlines to be met despite the presence of faults. The paper discusses and implements the combination of a successful fault tolerant algorithm with the EDF algorithm for multiprocessors running in parallel and executing real-time applications. The algorithm is based on a popular fault tolerant technique called primary/backup fault tolerance. The algorithm has been extended to include and consider tasks arriving dynamically unlike just the PB technique which accommodated static tasks. The backup overloading algorithm (Hengming & Farnam [1]) implements mechanisms which have helped in overcoming the limitations of the primary/backup technique. It has been successfully shown by the simulation results that primary backup overloading increases system utilization and also increases efficiency.

Index Terms

Fault tolerance, Real-time multiprocessor, Primary Backup, overloading, backup overloading.

I. INTRODUCTION

A fault-tolerant system is one which produces correct results even in the presence of faults. Embedded and real-time systems have to operate despite the presence of faults, should they come from the hardware or from programming errors. To tolerate hardware faults, it is necessary to (i) detect errors, through the use of error detection codes or on-line diagnosis; (ii) to recover from the error, using either backward error recovery or task duplication. In particular, in hard real-time systems, the fault tolerance mechanisms (e.g. task duplication) have to be accounted for during the validation of the temporal behavior of the system.

Fault tolerance is even more important when multiprocessors are used to run real-time applications since many processors are running in parallel and failure of any one processor might hinder execution of others too.

In real-time systems temporal correctness is as important as functional correctness, i.e., the capability to support the timely execution of applications. In contrast with many existing systems, next-generation systems will require support for hard, soft, and non-real-time applications on the same platform. Many of the mission-critical systems are safety-critical, and therefore have fault tolerance requirements. In this context, fault tolerance is the ability of a system to support continuous operation in the presence of faults.

Although a number of techniques for supporting fault-tolerant systems have been suggested, they rarely consider the real-time requirements of the system. A real-time operating system must provide support for fault tolerance and exception handling capabilities for increased reliability while continuing to satisfy the timing requirements.

A. Fault Tolerance Techniques

1) TMR (Triple Modular Redundancy)

Multiple copies are executed and error checking is achieved by comparing results after completion. In this scheme, the overhead is always on the order of the number of copies running simultaneously.

2) PB (Primary/Backup)

The tasks are assumed to be periodic and two instances of each task (a primary and a backup) are scheduled on a uni-processor system. One of the restrictions of this approach is that the period of any task should be a multiple of the period of its preceding tasks. It also assumes that the execution time of the backup is shorter than that of the primary.

3) PE (Primary/Exception)

It is the same as PB method except that exception handlers are executed instead of backup programs.

B. Primary Backup Fault Tolerance

This is the traditional fault-tolerant approach wherein both time as well as space exclusions are used. The main idea behind this algorithm is that (a) the backup of a task need not execute if its primary executes successfully, (b) the time exclusion in this algorithm ensures that no resource conflicts occur between the two versions of any task, which might improve the schedulability. Disadvantages in this system are that

(a) there is no de-allocation of the backup copy,

(b) the algorithm assumes that the tasks are periodic (the times of the tasks are predetermined),

(c) compatible (the period of one process is an integral multiple of the period of the other process) and execution time of the backup is shorter than that of the primary process.

II. BACKUP OVERLOADING (EXTENSION OF PB)

This scheduling scheme is based on the following observations: (a) in real-time systems, tasks must be memory resident at the time of execution; (b) a processor functioning as a spare will be idle throughout the life-time of the system; and (c) reservations of resources for backup copies must be ensured, but backup copies can have a different scheduling strategy than primaries. The backup version is executed only if the output of the primary version fails the acceptance test, otherwise it is de-allocated from the schedule. PB with Overloading is a dynamic scheduling algorithm. It considers dynamic systems in which tasks are aperiodic. So the tasks are scheduled during run time since we do not know when they appear. The execution times of both primary and backup are equal. Here n interconnected identical processors are assumed to have a central controller which maintains the global schedule.

These are the reasonable things to do for efficient utilization of the CPU and the algorithm takes form based on these two techniques.

1) Backup Overloading

Scheduling backups for multiple primary tasks during the same time period in order to make the efficient utilization of available processor time is called backup overloading. As shown in the Figure 1, the back ups of primary 1 and primary 2 have been overloaded.

2) De-allocation of Backup tasks

Removal of resources reserved for backup tasks when the corresponding primaries complete successfully is called de-allocation.

Overloading with de-allocation ensures high schedulability while providing fault tolerance. The time slots on which the primary and backup copies are scheduled are called primary and backup slots. The Figure 1 gives an example of overloading.

The purpose of overloading is to utilize the available time slot better because the chances that the backup copies, Bk1 and Bk2 have to run simultaneously, is remote. Such situation will not arise because we are considering only one processor failure at a time. So if Pri 1 on P1 fails, Bk1 on P2 runs. There is no chance that P3 also fails at the same time. Thus overloading is a reasonable thing to do.

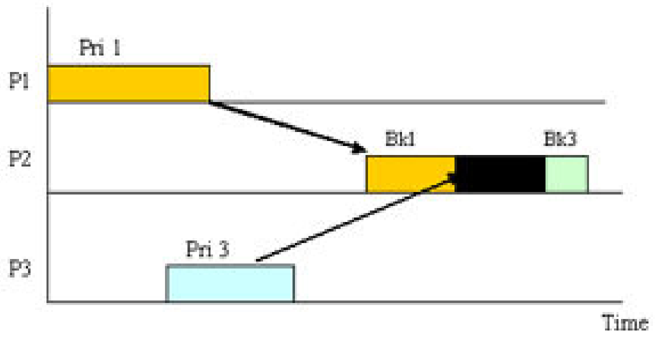

Fig. 1. Overloading Example, the backup copies of Pri 1 and Pri 2 are Bk1 and Bk2. Bk1 and Bk2 are scheduled to execute on the time slot as can be seen. The black region is the overloaded region.

A. *Backup Overloading Scheduling Algorithm*

The following steps form the procedure used to implement the backup overloading algorithm.

1) *Arriving task*

A task has four properties when it arrives, arrival time (ai), Ready time (ri), Deadline – (di) and worst case computation time (ci) represented as Ti = (ai, ri, di, ci)

2) *EDF schedulability*

Check if all the tasks can be scheduled successfully using the earliest deadline first algorithm. If the schedulability test fails, then reject the set of tasks saying that they are not schedulable.

3) *Searching for timeslot*

When task Ti arrives, check each processor to find if the primary copy (Pri) of the task can be scheduled between ri and di. Say it is scheduled on processor Pi.

4) *Try overloading*

Try to overload the backup copy (Bki) on an existing backup slot on any processor other than Pi.

Note: The backups of 2 primary tasks that are scheduled on the same processor must not overlap. If the processor fails, it will not be possible to schedule the two backups simultaneously since they are on the same time slot (overloaded).

5) *EDF Algorithm*

If there is no existing backup slot that can be overloaded, then schedule the backup on the latest possible free slot depending upon the dead line of the task. The task with the earliest deadline is scheduled first.

6) *De-Allocation of backups*

If a schedule has been found for both the primary and backup copy for a task, commit the task, otherwise reject it. If the primary copy executes successfully, the corresponding backup copy is de-allocated.

7) *Backup execution*

If there is a permanent or transient fault in the processor, the processor crashes and then all the backups of the tasks that were running on this system are executed on different processors.

B. *A feasible overloading example*

Figure 2 shows 4 processes running on 3 different processors P1, P2 and P3. The backup copies of these processes are scheduled to run on the different processors.

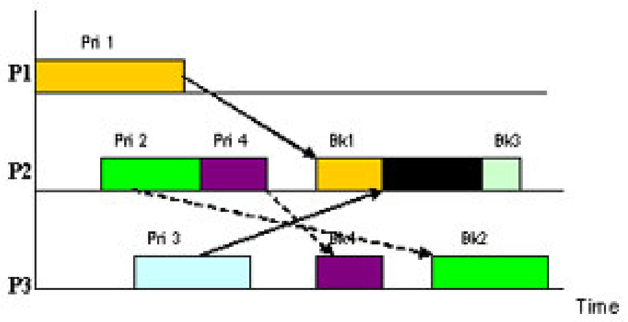

Fig. 2. View of the schedule. Bk1 and Bk3 have been overloaded. As explained earlier, we can overload only those processes whose primaries are running on different processors.

The arrows point from the primaries to their backup copies. Bk1 and Bk3 have been overloaded, i.e., they are scheduled to run at the same time, as discussed earlier, trying to perform backup overloading as far as possible when there is a chance to do it.

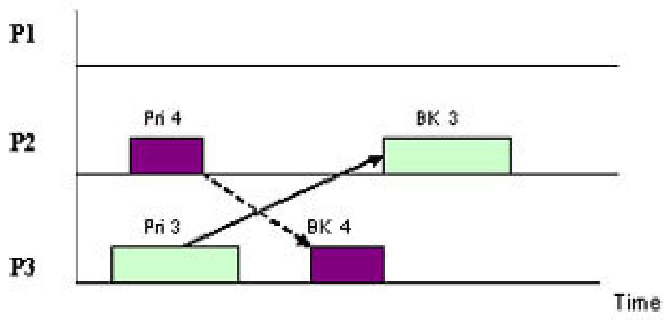

Fig. 3. View of the schedule. Pri 1 and Pri 2 have finished execution based on EDF algorithm. So Bk1 and Bk2 are removed automatically.

Figure 3 shows how scheduling proceeds when primary1 and Primary 2 finish executing. According to the EDF schedule Pri 1 and Pri 2 have earliest deadlines and are executed first. Since there is no more need for backup 1 and 2, they are de-allocated from the processors in order to use CPU efficiently. This increases CPU utilization.

Figure 4 shows the schedule after 2 new processes (5 and 6) arrive. Again overloading is accomplished by scheduling Bk3 and Bk5 on the same time slot. Care is taken while scheduling not to overload 2 backups whose primary copies are on the same processor. Here primary 5 and 3 are on different processors 1 and 3 respectively.

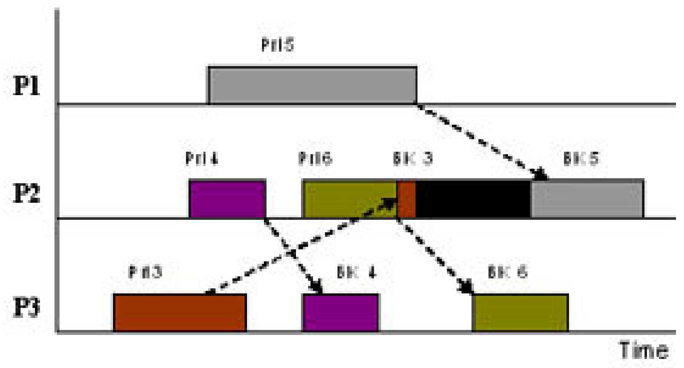

Fig. 4. View of the schedule. Backup copies of 5 and 6 are scheduled on P2 and P3. Again care is taken to see that backups of 4 and 6 are not overloaded.

III. SIMULATION

Simulation of the multiprocessor systems has been successfully accomplished using threads on a uni-processor system. Threads are a way for a program to split itself into two or more simultaneously running tasks. This multithreading occurs by time slicing where a single processor switches between different threads. Threads are similar to processes, but differ in the way that they share resources. In the simulation, threads are created representing the processors. Process objects are created using the process class in the C# language representing the processes that run on different processors. Process class retrieves various types of information for a process running on the system and this information is cached. Up-to-date information about process information is obtained by frequently refreshing the cache by calling the Process. Refresh method. Other information include

- Memory Statistics
- Process Times
- Thread statistics
- Handle Statistics

The system has been implemented on the Visual studio .net 2003 platform. The programs are written in the C# language. Thread class is utilized for processor and Process class is used to represent the processes. The language was chosen based on its capability to simulate multithreading on a single system, and threading enables C# program to perform concurrent processing so we can do more than one operation at a time.

A. Simulation with B/O

This part of the simulation was performed by implementing the backup overloading algorithm. After the EDF scheduler assigns tasks to different processors with respect to their deadline times, the backup copies are scheduled on different processors according to the B/O algorithm. Overloading is performed to increase efficiency. The user gets to decide which processor (thread) will crash during execution. After the crash of that processor, the processes which were scheduled to run on that thread are queued and their backups are scheduled to run. After the execution of all the processes, the failure rate is calculated.

B. Simulation without B/O

This part of the simulation was performed by implementing the primary backup algorithm. In this algorithm, after the primary tasks are scheduled, the backup copies are scheduled accordingly. However no attempt is made to perform overloading of the backup copies. Again the user gets to choose the processor to crash. The failure rate and efficiency are calculated. For the same efficiency, results indicate that more number of processes will not run successfully and thus the failure rate is higher.

Scheduling algorithm extensibility: The Earliest deadline first scheduling algorithm schedules the 30 different processes on 10 processors according to their deadlines. The combination of both EDF and B/O has been implemented keeping in mind the extensibility factor. If the RM (rate monotonic) or the LLF (least laxity first) scheduler need to be used in combination, it is easy to apply the changes since the both algorithms run simultaneously but on different threads.

IV. RESULTS

The simulation was performed with 10 processors (10 threads) and 30 processes (30 process objects) in the C# .net language. The failure rate σ at a given point of time t in a multiprocessor environment is given by the equation shown. It is the ratio of the number of processes that failed to execute to the number of processes that executed successfully.

$$(\sigma)_t = n\,(\textit{Failed Processes})_t \,/\, n\,(\textit{successful processes})_t$$

To be sure and to show that the simulation is working as expected, the same experiment was conducted on different number of processors every time and with a different number of processes scheduled every time. They were scheduled using the EDF algorithm.

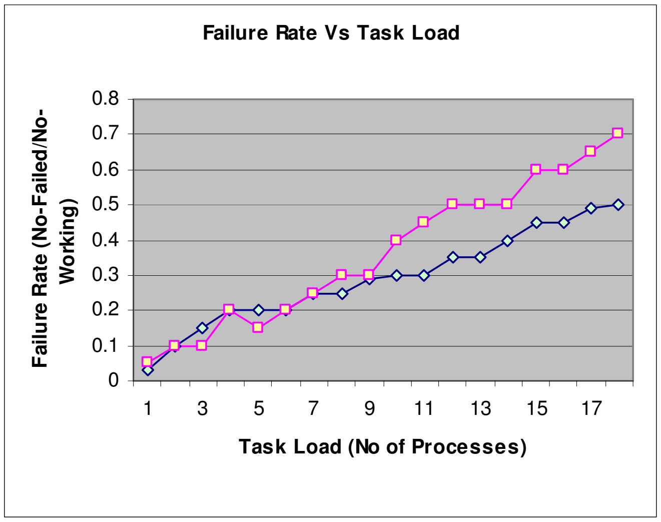

Fig 5. Failure rate Vs. Task load. Green dots – Failure Rate with B/O. Pink Dots – without B/O. The graph plots Failure rate Vs task Load. As can be seen, the pink line has higher failure rates only at higher task loads.

Backup overloading was used in some processors but was purposefully removed in certain processors. Both kinds of processors performed almost well and normal when the number of processes they ran was < 9. When the number of processes was gradually increased above that, the processors with no backup overloading algorithm began to slow down and eventually crashed more than the others. Thus the plotting of the failure rate vs. task load gave us the graph shown in Figure 5. It can be clearly seen that the backup overloading algorithm improves the performance and efficiency of computers.

Figure 6 represents how the utilization factor is distributed among the 8 processors. The program was executed taking 8 processors, feeding 15 processes with different computation (Ci) and periods (Pi) and using

$$\text{Utilization } U = \Sigma\, Ci / Pi$$

for all the tasks on that processor. It gives an idea that not many processes were executing on processor 5, the graph is useful to determine the behavior of the scheduling algorithm.

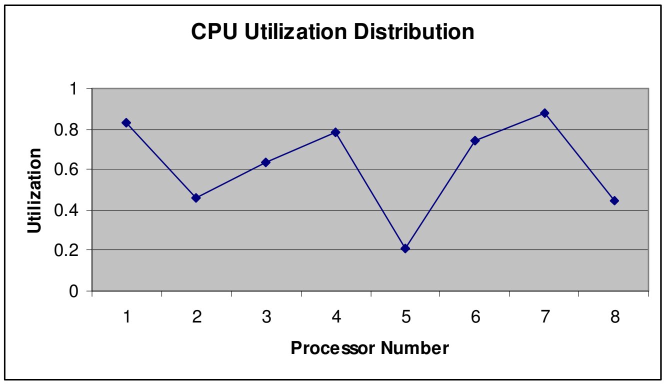

Fig 6. The graph represents the plot of CPU utilization vs. Processor number. This shows the behavior of the back up overloading algorithm and the EDF algorithm on a large scale.

V. LIMITATIONS

The algorithm can tolerate only a single fault: Since overloading is considered, the algorithm cannot afford the failure of more than one processor at the same time. If more than one processor crashes at the same time, the algorithm will fail to execute the tasks on time.

There are basically 2 kinds of faults considered here:

1. Transient Fault – This fault is only temporary and the system will start running correctly after some time. So, only that backup during which the processor had failed needs to be executed.
2. Permanent Fault – Here the processor undergoes a permanent flaw and all the processes scheduled on it crash. So, all backups and primaries on that processor have to be re-executed.

It has been deduced that the PB overloading algorithm has been quite successful in overcoming 86% of the transient faults but not the permanent faults.

VI. CONCLUSION

Based on the results, it is deduced that at higher loads, backup overloading yields less failure rate and thus better performance. The other algorithms show mare failure rate according to the simulation results. Backup overloading demands less processor time to provide fault-tolerance (i.e., schedule all backup tasks) hence increasing schedulability. Resource reclamation has been used previously for tasks that finish execution earlier than their worst case execution time but not for the primary/backup approach. Here resource reclamation has been used by which unwanted backups are de-allocated. The backup de-allocation is a combination of the two ideas. Thus overloading and de-allocation increase the schedulability of real-time tasks on the multiprocessor system.

The other advantage of the algorithm is that the tasks considered are dynamic and aperiodic. The algorithm is simple and easy to understand and implement. It also increases utilization and speed efficiency of scheduling. It can also be concluded that appropriate use of redundancy is important since too much redundancy increases reliability but potentially decreases the schedulability. Too little redundancy decreases reliability but increases schedulability. Also, designing, managing redundancy incurs additional cost, time, and memory and power consumption. Thus this algorithm can be efficiently used for fault tolerance in case where multiprocessors are used to run real-time applications.

ACKNOWLEDGMENT

I am grateful to my research director and advisor, Dr. Albert Cheng, Department of Computer Science, University of Houston, for helping me complete the project. This work would not have been possible without his timely encouragement. Professor Cheng has published many papers on self-stabilization of real-time systems and gave me all the necessary feedback including the Real-Time Systems Laboratory facilities.

REFERENCES

[1] Hengming Zou, Farnam Jahanian, "Real-Time Primary-Backup (RTPB) Replication with Temporal Consistency Guarantees", Department of Electrical Engineering and Computer Science, University of Michigan.

[2] Oscar Gonzalez, H. Shrikumar, John A Stankovic, "Adaptive Fault Tolerance & Graceful Degradation Under Dynamic Hard Real-time Scheduling", University of Virginia.

[3] S. S. Kulkarni and A. Arora. "Automating the addition of fault tolerance, formal Techniques in Real-Time and Fault-Tolerant Systems", 2000. Available as a Technical Report MSU-CSE-00-13 at Computer Science and Engineering Department, Michigan State University, East Lansing, Michigan.

[4] Albert Mo Kim Cheng, "Self–Stabilizing Real –Time Rule–Based Systems", Proc. 11th IEEE-CS Symp. on Reliable Distributed Systems, Houston, Texas, pp. 172-179, Oct. 1992.

[5] Arora, P. C. Attie, and E. A. Emerson. "Synthesis of fault-tolerant concurrent programs". Proceedings of the 17th ACM Symposium on Principles of Distributed Computing (PODC), 1998.

[6] M. Gomaa, C. Scarbrough, T. N. Vijaykumar, and I. Pomeranz. "Transient-fault recovery for chip multiprocessors. In Proceedings of the 30th annual international symposium on a Computer architecture", pages 98–109. ACM Press, 2003.

[7] Sandeep S. Kulkarni and Ali Ebnenasir. "Enhancing the fault-tolerance of nonmasking programs". Technical Report MSU-CSE-03-2, Computer Science and Engineering, Michigan State University, East Lansing, Michigan, February 2003.

[8] Arora and M. G. Gouda. Closure and convergence: A foundation of fault-tolerant computing. IEEE Transactions on Software Engineering, 19(11):1015–1027, 1993.

[9] A.K. Somani and N.H. Vaidya, "Understanding Fault-Tolerance and Reliability," Computer, vol. 30, no. 4, pp. 45-50, Apr. 1997.

[10] T. Tsuchiya, Y. Kakuda, and T. Kikuno, "Fault-Tolerant Scheduling Algorithm for Distributed Real-Time Systems," Proc. Workshop Parallel and Distributed Real-time Systems, 1995.

[11] A.L. Liestman and R.H. Campbell, "A Fault Tolerant Scheduling Problem," IEEE Trans. Software Eng., vol. 12, no. 11, pp. 1,089- 1,095, Nov. 1986.

[12] D. Mosse, R. Melhem, and S. Ghosh, "Analysis of a Fault-Tolerant Multiprocessor Scheduling Algorithm," Proc. IEEE Fault-Tolerant Computing Symp. pp. 16-25, 1994.

[13] H. Kopetz, A. Damm, C. Koza, and Mulozzani, "Distributed Fault-Tolerant Real-Time Systems," IEEE Micro, pp. 25-41, 1989.

[14] S. Ghosh, R. Melhem, and D. Mosse, "Fault-Tolerance Through Scheduling of Aperiodic Tasks in Hard Real-Time Multiprocessor Systems," IEEE Trans. on Computers.

[15] K. Kim and J. Yoon, "Approaches to Implementation of Reparable Distributed Recovery Block Scheme," Proc. IEEE Fault-Tolerant Computing Symp., pp. 50-55, 1988.

[16] R. Rajkumar, M. Gagliardi, and L. Sha. The real- time publisher/subscriber inter-process communication model for distributed real-time systems: Design and implementation. In Proc. Real-Time Technology and Applications Symposium, 1995.

[17] Albert Mo Kim Cheng, Seiya Fujii, “Self-Stabilizing Real-Time OPS5 Production Systems”, IEEE Transactions on Knowledge and Data Engineering, Vol. 16, No. 12, pp. 1543-1554, Dec. 2004.

[18] G. Manimaran and S. R. Murthy, “A Fault Tolerant Dynamic Scheduling Algorithm for Multiprocessor Real-Time Systems and Its Analysis,” IEEE Trans. on Parallel and Distributed Systems, vol. 9, no. 11, pp. 1137–1152, Nov. 1998.

[19] K. Ahn, J. Kim, and S. Hong, “Fault-Tolerant Real-Time Scheduling using Passive Replicas,” Proc. on the Pacific Rim International Symposium on Fault-Tolerant Systems, Taipei, Taiwan, Dec. 15–16, 1997, pp. 98–103.

[20] A. Srinivasan and G. C. Shoja, “A Fault-Tolerant Dynamic Scheduler for Distributed Hard-Real-Time Systems,” Proc. on the Pacific Rim Conference on Communications Computers sand Signal Processing, Victoria, BC, Canada, May 19–21, 1993, pp. 268–271.