This is to certify that the

dissertation entitled

TASK SCHEDULING AND COMMUNICATION SUPPORT
FOR PARALLEL AND DISTRIBUTED REAL-TIME SYSTEMS

presented by

Jong-Pyng Li

has been accepted towards fulfillment
of the requirements for

PhD degree in Computer Science

Major professor

Date January 4, 1994

Task Scheduling and Communication Support for Parallel and Distributed Real-Time Systems

By

Jong-Pyng Li

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science

1993

ABSTRACT

Task Scheduling and Communication Support for Parallel and Distributed Real-Time Systems

By

Jong-Pyng Li

Real-time applications are increasing in their complexity of control and computational demands. Parallel and distributed systems provide cost-efficient computing power and higher degree of fault tolerance that make these systems attractive computer systems for the next generation of real-time systems. As real-time systems move from uniprocessor systems to parallel and distributed systems, the design of real-time systems becomes more complex and new techniques are required.

This dissertation provides new approaches for solving two closely related problems on designing parallel and distributed real-time systems: dynamic scheduling of tasks with precedence relations and communication support for wormhole routed networks. The task scheduling algorithm combines task graph partitioning, least-laxity-first scheduling and branch-and-bound task allocation techniques to provide required real-time performance. Performance analysis show that the algorithm can efficiently schedule precedence-constrained tasks with low scheduling overhead. The parameters that affect the performance and hardware costs are studied to provide

system designers with the means for fine tuning the algorithm for different system configurations.

The flow control scheme of a direct network manages the network resources and directly relates to the system performance. Several flow control schemes are developed to support real-time communication on wormhole networks. The schemes differ in their priority mapping, priority adjustment, arbitration and message dropping strategies. A priority mapping scheme encodes the timing property of a message into a priority, which can be represented in a small number of digits. As the timing property of a message changes, a priority adjustment method modifies the priority to reflect the current status of the message. An arbitration function decides how to allocate bandwidth. Messages that miss their deadlines and lose their value are removed from the network by a message dropping method. With simple modifications to the existing wormhole routers and small additional costs, the flow control schemes deliver messages in a timely fashion to support real-time communication. The performance of the schemes is studied in environments in which the communication traffic is static or dynamic. The performance study verifies that the flow control schemes outperform the conventional flow control scheme implemented in most wormhole routers.

To my parents

ACKNOWLEDGMENTS

I wish to thank Prof. Matt Mutka, my advisor, who has provided persistent and generous support for this research. His patience, encouragement and insight has made the completion of this thesis possible.

Thanks are due to Prof. Lionel Ni who made perceptive criticisms that clarify and strengthen this thesis. Special thanks to Prof. Wen-Jin Hsu that he persuaded me to pursue the path that I am on. I would like to thank Prof. Richard Enbody and James Stapleton who squeezed in time to read the thesis and made many valuable comments.

Shieh-Chang Hsiung has been a constant source of support during these years of my graduate study. He deserves special thanks for offering his friendship. I thank Ramji Vaithianathan who contributed to the development of the graphical user interface of DRMS. I thank David Vogel, Linda Usack and C. Douglass Locke, the engineers and scientists at IBM Federal Sector Division, who provided their experience on the specification of DRMS.

My deepest appreciation goes to my parents for the support they have given me. Their constant love and encouragement has motivated me to be the best that I can. My wife Chen-Shun has been the greatest companion during my graduate study. Her different perspective of life has helped me through all of the difficult times. Her understanding and love for the last few years has been the force that pushing me forward. I thank her for all she has done.

TABLE OF CONTENTS

LIST OF TABLES x

LIST OF FIGURES xi

1 Introduction 1

1.1 Design Issues of Real-Time Systems 2
1.2 Direction of The Thesis 5
1.3 Thesis Organization 6

2 Motivation and Problem Statement 8

2.1 Dynamic Scheduling on Distributed Real-Time Systems 9
2.2 Dynamic Scheduling on Mesh Connected Multiprocessors 11
2.3 Real-Time Flow Control for Wormhole Networks 13

3 Background Review 16

3.1 Static Scheduling Algorithms 17
3.1.1 Static Scheduling on Uniprocessors 18
3.1.2 Static Scheduling on Distributed Systems and Parallel Multiprocessors 19
3.2 Dynamic Scheduling Algorithms 21
3.3 Real-Time Communication Protocols and Flow Control 22
3.3.1 Multiple Access Networks 22
3.3.2 Point-to-Point Interconnection Networks 25

4 Dynamic Scheduling for Distributed Real-Time Systems 27

4.1 The System Model 28
4.2 Task Group Preprocessing 31
4.2.1 Task Graph Partitioning 31
4.2.2 Estimating Task Deadlines and Earliest Start Execution Time 32
4.3 Selecting Processors for Task Allocation 33
4.4 The Scheduling Algorithm 35

4.4.1 Local Scheduling 36
4.4.2 Global Scheduling 37
4.4.3 The BB algorithm 38
4.5 Performance Studies 41
4.5.1 The Simulation Model 41
4.5.2 Relationship between Maximum Subgroup Size and Rejection Ratio 43
4.5.3 Effects of Scheduling Overhead 45
4.5.4 Comparison of Preferred Set Expansion Strategies 47
4.6 Summary 49

5 Scheduling Mesh Connected Multiprocessors 51
5.1 The System Model 52
5.2 The Scheduling Algorithm 54
5.2.1 Local Scheduling 55
5.2.2 Global Scheduling 55
5.3 Performance Analysis 56
5.3.1 The Simulation Model 56
5.3.2 Relationship between The Rejection Ratio and Scheduling Overhead 57
5.3.3 Local Rejections 59
5.4 Summary 62

6 Virtual Channel Flow Control for Static Traffic 63
6.1 Wormhole Networks and Virtual Channels 66
6.2 Flow Control 68
6.2.1 Conventional Wormhole Network Flow Control — First Come First Served 71
6.2.2 Priority Mapping Schemes Based on Message Generation Times 71
6.2.3 Dynamic Priority Adjustment Schemes — Priority Climbing . 74
6.2.4 Priority Based Arbitration — Enhanced Priority Climbing . . 76
6.2.5 Frequency-Based Priority Mapping and Arbitration — RMS . 78
6.2.6 Reducing the Cost of the RMS Scheme — The One Bit Approach 79
6.3 The Simulation Model 80
6.3.1 The Model of Communication Traffic 81
6.3.2 Message Source Generation 83
6.4 Performance Analysis 84
6.4.1 Number of Virtual Channels 84

6.4.2 Priority Mapping 85
6.4.3 Priority Adjustment 87
6.4.4 Prioritized Virtual Channel Assignment and Arbitration Function 88
6.4.5 Latency 89
6.5 Summary 90

7 Virtual Channel Flow Control for Dynamic Traffic 95
7.1 Flow Control 96
7.2 Performance Analysis 97
7.2.1 The Model of Performance Study 97
7.2.2 Priority Based *vs.* FCFS Flow Control 98
7.2.3 Scalable Real-Time Network 99
7.3 Message Dropping 101
7.3.1 The Message Dropping Scheme 101
7.3.2 The Effect of Message Dropping 107
7.3.3 The Size of BC 108
7.4 A Router Design to Support Real-Time Flow Control 109
7.5 Summary 113

8 Conclusion and Directions for Future Research 115
8.1 Summary of Major Contributions 115
8.2 Priority Based Task Scheduling 117
8.3 Task Scheduling with Resource Requirement 118
8.4 Real-Time Flow Control in ATM Networks 118

A The DRMS Tool 120
A.1 Background of the Rate Monotonic Scheduling Algorithm 124
A.1.1 The Worst-Case Bound of the Rate Monotonic Scheduling Algorithm 124
A.1.2 Pre-Period Deadlines and Utilizations Larger than the Worst-Case Bound 125
A.1.3 Period Transformations for Critical Tasks 127
A.1.4 The Priority Inversion Problem 128
A.2 Requirements for the DRMS Tool 128
A.2.1 Task Specifications 129
A.2.2 Processor Specifications 132
A.2.3 Methods of Analysis 132
A.3 The Branch-and-Bound Algorithm for Task Allocation 135

A.4 User Interface for DRMS 139
A.4.1 Input User Interface 139
A.4.2 Output User Interface 144
A.5 An Example Usage of DRMS for Scheduling Periodic Tasks with Communication Requirements 145
A.6 Summary 146

BIBLIOGRAPHY **149**

LIST OF TABLES

4.1 An example preferred list for a binary 4-cube. 34
4.2 The Branch-and-Bound algorithm. 40

6.1 A summary of the flow control schemes. 70
6.2 Parameters of the linear bounded arrival process. 82

A.1 Parameters for Task Specification. 130
A.2 Parameters for Processor Specification. 132
A.3 Processor Parameters. 139
A.4 Task Parameters. 140

LIST OF FIGURES

4.1 An example task graph of a task group. 28
4.2 The system model. 30
4.3 A task assignment tree. 38
4.4 Performance of the group scheduling algorithm and the baseline algorithm. 44
4.5 Sensitivity of the rejection ratio to the maximum distribution subset size. 46
4.6 Sensitivity of the scheduling algorithm to the preferred set expansion method. 48

5.1 The system model of the mesh connected multiprocessor 53
5.2 Task group Rejection Ratio for Different Unit Scheduling Time; DTF=5. 58
5.3 Comparisons of local rejection ratios with different unit scheduling time; DTF=5. 59
5.4 Percentage of local rejections; DTF=5. 60
5.5 Local rejections ratios for DTF=2, 5, 10, 100; system load=0.8. . . . 61

6.1 Flow control components and the resources of a physical link. 69
6.2 An 8-bit priority and block count carried by a message header. 75
6.3 An example of the virtual channel assignment and arbitration decisions in the EPC scheme. 77
6.4 Performance of FCFS scheme with different number of virtual channels. 85
6.5 Comparison of priority mapping schemes. 86
6.6 Comparison of priority adjustment schemes. 88
6.7 Comparison of the prioritized virtual channel assignment and arbitration with the other schemes. 89
6.8 Percentage of performance improved by adjusting priorities of nodes on the path. 90
6.9 Average latencies of the different flow control schemes. 91
6.10 The tradeoff between performance and extra hardware costs. 92

7.1 Performance comparison of the flow control schemes. 98
7.2 Deadline missed ratio increases as the network size scales upward. . . 100
7.3 An example of dropping a flit. 104
7.4 Dropping a tardy message. 106
7.5 The effect of the message dropping on the performance. 107
7.6 The size of BCs affects the rate of dropping messages and the performance. 108
7.7 A router design to support priority based flow control and message dropping. 109
7.8 The components of virtual channels. 110
7.9 Dropping a message using an additional drop signal line with each virtual channel. 112

A.1 Specifying Global Parameters. 141
A.2 Specifying Execution Preferences. 142
A.3 Execution Results. 143
A.4 RMS analysis on the Display Processor. 145
A.5 Scheduling periodic tasks and their communication traffic using DRMS. 147

CHAPTER 1

Introduction

A real-time system can be characterized as a system of which both the correctness of logical results and the timing constraints to obtain the results must be satisfied. Otherwise, severe consequences may damage the system, properties or human lives. Examples of real-time systems are autopilots for aircraft, automatic production lines of manufacturing factories, control systems of nuclear power plants, process control facilities and avionics systems of space shuttles [1, 2, 3, 4]. A typical real-time system consists of two subsystems: a controlling subsystem and a controlled subsystem. Humans are the controlling subsystems in traditional real-time systems. As the controlled job becomes complicated and the reaction time becomes short, a human cannot adequately control a job. Therefore, computers are replacing humans as the controllers of real-time systems. For example, real-time computers are commonly used in aircraft for flight control [5].

An increasing number of real-time applications operate in environments that require complicated control and flexibility to comply to the changing environment [6]. As the rapid advances in parallel and distributed systems continues, these systems are emerging as the new generation of real-time systems [7]. Parallel and distributed systems, which execute tasks concurrently, can efficiently handle complicated tasks by partitioning one task to several simple subtasks. The subtasks can be executed

on different processors. Also, parallel and distributed systems have a higher degree of fault tolerance than uniprocessor systems. These advantages of parallel and distributed systems make them the computer systems for the new generation of real-time systems.

1.1 Design Issues of Real-Time Systems

The complexity of designing a real-time system increases as we move from uniprocessor systems to parallel and distributed systems. New problems for building a real-time system are introduced, *e.g.*, synchronization and communication, which do not exist in the traditional real-time systems. Also, the existing design principles may not be appropriate for the new systems. For example, task scheduling algorithms for uniprocessors cannot be directly ported to parallel and distributed systems.

Design issues of parallel and distributed real-time systems include *specification*, *communication*, *programming languages*, *task scheduling*, *resource management*, *fault tolerance* and *architectures* [1, 5, 8].

Specification: The specification of a real-time system is based on the properties of the environment and the required performance, so that the specified system can provide guaranteed performance. A system might deviate from its assumed properties under faulty conditions, a specification scheme should provide means for predicting the possible failures. A general purpose system becomes difficult to specify as parallelism is incorporated. The difficulty increases further as real-time constraints are introduced. Real-time systems not only require correct logical results, but also the timely execution of tasks. Specification techniques for general purpose systems are insufficient for real-time systems.

Communication: Parallel and distributed systems are the computers for tomorrow's real-time systems. The processors of such computers are distributed and

communicate by means of communication networks. The communication network of a real-time system should deliver messages in a timely fashion such that the communication delays will not cause tasks to violate their timing constraints. Specific real-time protocols and flow control schemes should be developed for different networks. The bus, the token ring and the direct network are examples of different types of networks.

Programming language support: A task is a software module that performs a certain function. Beside the functionality of describing parallelism and task synchronization, programming languages for parallel and distributed real-time systems should provide means for measuring the timing properties of tasks. Real-time applications perform in controlled environments such that the timing properties of tasks are predictable. Examples of timing properties of tasks include the computation demands, the deadline and the resources required for execution. The computation demands of tasks must be predictable so that the tasks can be scheduled before their deadlines. The features to support the timing controls of tasks are important requirements for real-time programming languages. New techniques are also required to verify and test the demands of the tasks.

Scheduling: Task scheduling is critical in real-time system design. Scheduling algorithms for real-time systems must guarantee that a scheduled task can meet its deadline. Therefore, the performance of scheduling algorithms for real-time systems is measured in terms of the task guarantee ratio instead of throughput as is done for general purpose systems [4]. Real-time tasks can be classified into three categories: critical, essential and unessential [9]. A catastrophe will occur if a critical task misses its deadline. Critical tasks are statically scheduled such that the required resources are reserved in advance of their submissions. A larger number of tasks that sporadically arrive at the system are classified as essential tasks. Essential tasks have timing constraints, and the system performance will seriously degrade if the timing

constraints of these tasks are not met. Since the arrival time of an essential task is not known beforehand, essential tasks are dynamically scheduled. Unessential tasks may or may not have timing constraints. These tasks execute when they do not affect critical or essential tasks. Any failure to meet a deadline associated with an unessential task is not crucial to the performance of the system. In practice, these three types of tasks may co-exist in the same system such that task scheduling becomes a very hard problem.

Fault tolerance: Real-time systems require fault tolerant ability in both hardware and software. Fault tolerance is usually statically built in real-time systems during the design phase to ensure tasks can be reliably executed. However, static fault tolerance means high cost and is inflexible with respect to the environment. The new generation of real-time systems requires the systems to adapt to the changing environment. Thus, new approaches, which dynamically adjust to the faulty conditions, are required.

Architecture: Real-time systems are usually special purpose systems, so the architectures to support the systems are specifically designed for the problems. Real-time capability and fault tolerance are the important features that are seldom found in general purpose systems. These features must be designed into the real-time systems. Examples of real-time capability include a short context switch time and timing control facilities. In most systems, hardware fault tolerance is achieved by using redundant components. For example, a redundant system may have several processors that perform as one unit and execute the same code. The results produced by the processors are compared to ensure the correctness. However, redundancy introduces an extra cost in building a system. Off-the-shelf components can be used to trim cost as long as the components serve the required functions. Recent development of reconfigurable parallel and distributed architectures [10, 11] enables real-time systems to automatically reconfigure to keep the system functioning under faulty conditions.

The use of the reconfigurable architectures also reduces the hardware cost.

1.2 Direction of The Thesis

Our interest in designing parallel and distributed real-time systems is concentrated on two closely related problems: dynamic scheduling of tasks with precedence constraints and communication support. Dynamic scheduling for tasks with precedence constraints on parallel and distributed systems has not received much attention, which is much needed for the next generation of real-time systems. We developed a new scheme for efficiently solving the dynamic task scheduling problem. This scheme combines several techniques, *task graph partitioning*, *branch-and-bound* and *least-laxity-first*, to reduce the complexity of the scheduling to linear order. Due to the different design considerations of distributed systems and parallel multiprocessors, the scheme is specialized for different computer systems. In this dissertation, we present the algorithms for both distributed systems and mesh-connected multiprocessors.

In parallel and distributed systems, tasks communicate to achieve a common goal. When scheduling tasks, we must consider the communication cost for interchanging messages among tasks distributed to different sites. In general purpose systems, communication subsystems are designed to achieve higher message throughput. The timely delivery of real-time messages is not directly supported in such communication systems. In order to estimate communication cost, real-time communication subsystems must be specifically designed to meet the timing constraints of messages. Otherwise, scheduled tasks may violate their deadlines due to unpredictable message delays. As a result, real-time systems cannot provide guaranteed performance. To achieve the required performance of a real-time system, a communication subsystem that supports real-time message transmission is needed.

Direct networks are the communication subsystem found in most parallel systems.

Conventional communication schemes implemented in general direct networks cannot satisfy real-time requirement, *i.e.*, delivering messages in a timely fashion. The wormhole network is a promising switching mechanism for direct networks that adopted by many of the new parallel systems. The advantage of the wormhole network include simple router design and cost-efficient routers. We develop several priority based flow control schemes and a message dropping scheme for wormhole networks to support real-time communication. The flow control schemes can be easily implemented by simple modification of the existing wormhole routers. The message dropping scheme efficiently reduces the network contention by removing messages that miss deadlines and lose their value.

1.3 Thesis Organization

In this chapter, we overview the design issues of parallel and distributed real-time systems and the direction of our research work. The rest of the dissertation is organized in seven chapters and an appendix. The motivation and the detailed description of the research are presented in Chapter 2. Chapter 3 surveys related work of real-time scheduling and communication problems.

Chapter 4 studies a scheme for dynamic scheduling real-time tasks on distributed systems and the factors that affect the efficiency of the scheduling algorithm. Chapter 5 presents a dynamic scheduling scheme for mesh connected multiprocessors.

Chapter 6 describes the new flow control schemes for the wormhole network. The performance of the schemes in static environment is also discussed in the chapter. Chapter 7 considers the flow control schemes in dynamic environment and presents a message dropping scheme which removes messages that miss their deadlines from the network. Chapter 8 concludes the dissertation and layouts the future plan for enhancing the current work.

Appendix A is a tool, DRMS, that we designed to assist real-time system designers to schedule periodic tasks on distributed systems. In addition to scheduling tasks, DRMS can also be used to evaluate feasibility of assigning periodic traffic to networks. The extensive usage of DRMS for scheduling periodic tasks with communication requirements is described.

CHAPTER 2

Motivation and Problem Statement

Task allocation, which is a non-existent problem in uniprocessor scheduling, needs to be integrated with new scheduling algorithms. A large number of existing real-time scheduling algorithms are static and give optimal solutions. However, the next generation of real-time systems must handle those tasks that arrive sporadically in the changing environment which cannot be scheduled by static algorithms. Dynamic scheduling algorithms are required for the next generation of real-time systems.

In addition to the ability of handling sporadical arrivals, real-time systems are increasing in their complexity of control, and thus the demand of computing capacity is increased. Parallel and distributed systems that provide high computing power become attractive computer systems for the new generation of real-time applications. One advantage of parallel and distributed systems is that tasks are executed at separate sites to achieve a high performance. The hardware costs of such systems are low in comparison with those of traditional supercomputers that provide the same computing capacity.

The communication subsystems of parallel and distributed real-time systems must support timely message transmission. In parallel and distributed systems, tasks ex-

ecuted at different sites exchange information to achieve a common goal. Efficient communication subsystems are required for parallel and distributed systems to provide the desired performance. The performance measures for general purpose systems and real-time systems are different. Instead of throughput as in general purpose systems, the performance of real-time systems is measured by the ratio of the task arrivals that can be executed before their deadlines. Thus, real-time communication subsystems must deliver messages in a timely fashion. Otherwise, tasks may violate their deadlines, and the system performance degrades drastically. Due to the different performance requirement from general purpose systems, communication support for parallel and distributed real-time systems must be studied as a separate problem.

In the following discussion, we describe the problems of dynamic scheduling for parallel and distributed systems and communication support for wormhole networks. Our approaches for solving the problems are briefly discussed.

2.1 Dynamic Scheduling on Distributed Real-Time Systems

Dynamic scheduling for real-time tasks on systems with more than one processors is a difficult problem. Even with independent tasks of identical execution time, the problem is *NP-hard* [12, 13]. Since we are scheduling tasks with precedence constraints, the problem becomes naturally computational intractable. Thus, to solve the problem, efficient heuristic algorithms with low run-time cost are needed.

One possible approach to reduce the scheduling cost is to partition a task group into subgroups and schedule one subgroup in one stage. In our dynamic scheduling approach, a group of tasks is partitioned into several subgroups such that tasks in the same subgroup are independent. The scheduling of a group occurs in several stages. A subgroup of tasks is scheduled at each stage. The scheduling of a group

occurs in several stages to ensure that the scheduling overhead does not increase dramatically with linear increases in group sizes. By limiting the maximum number of tasks in a subgroup, the scheduling complexity of a subgroup is bounded and the scheduling overhead increases linearly as the number of subgroups grows. Our task group partitioning strategy is different from the strategy described in [14]. In [14], a task group is analyzed and partitioned into task clusters. Tasks in a cluster have precedence relations and are allocated to the same processor for execution. Although the scheduling algorithm in [14] minimizes inter-task communication, the parallelism of the tasks is also minimized. Our scheduling algorithm considers both maximizing the parallelism of task execution and minimizing the communication traffic among tasks. In our algorithm, tasks are the basic scheduling units, and tasks in a subgroup are distributed to different processors for concurrent execution. Our task graph partitioning strategy is simple and the preprocessing overhead needed for scheduling a task graph is minimized.

An important issue for partitioning a group of tasks is the maximum size of each subgroup. In the simplest case, we can limit a subgroup to contain only one task. One scheduling step is required if we only consider distributing the task to one processor. However, if we schedule each task in a group separately, we are not considering the parallelism embedded in the group. When there is more than one independent task in a subgroup, the tasks can be scheduled to execute in parallel. By scheduling as many independent tasks as possible in one stage, we increase the efficiency of distributed systems. The drawback of scheduling a large number of tasks in one substage is that the scheduling overhead for each substage may be so overwhelming that the tasks waiting to be scheduled may be rejected due to a missed deadline even before they are scheduled. Therefore, we study the effect of the size of a subgroup on the performance of the scheduling algorithm.

In our scheduling approach, each node has a scheduling coprocessor to offload the

scheduling overhead from the processor. A coprocessor initially schedules a group of tasks on the local processor where they arrived. The tasks that cannot be locally scheduled are distributed to other processors in the system using a *branch-and-bound* (BB) algorithm. The benefit of the BB algorithm is its ability to search for an optimal schedule for a given set of tasks and available processors. Nevertheless, BB algorithms are generally restricted to static scheduling due to the high computation cost of the algorithm. If we limit the number of tasks and processors in each scheduling stage, then the required scheduling steps for each stage are upper bounded by a constant. As the number of subgroups for a given group increases linearly, the scheduling overhead also increases linearly. Although a BB algorithm is used for describing task distribution in this report, we are not restricted to this particular algorithm for global scheduling. Other optimal static scheduling algorithms exist to solve the same problem.

2.2 Dynamic Scheduling on Mesh Connected Multiprocessors

The problem of task scheduling on parallel multiprocessors is different from that of distributed systems. In parallel systems, the number of I/O ports is limited and each scheduler in the system may take the responsibility of scheduling tasks on more than one processor. Nevertheless, the processors of a distributed system have their own I/O channels, *i.e.*, each processor has a scheduler that schedules the locally invoked tasks.

The other issues that differ for the design of scheduling algorithms for parallel and distributed systems are the cost of the hardware and the extra communication load introduced by the algorithms. Since the cost of a processor is only a small portion of the total cost of a node in a distributed system, the cost of adding one scheduling

coprocessor to each node is relatively inexpensive. However, in a parallel system, processors contribute to a major part of the price to build the system. As the sizes of new parallel systems scaling upward, adding one coprocessor to each node of a parallel system becomes financially infeasible. In addition, the performance of communication subsystems of parallel systems are sensitive to network load. Suppose tasks are scheduled in each individual node of a parallel system, a scheduling coprocessor needs to exchange messages with the other nodes to accurately schedule tasks to remote processors. The extra messages introduced by the scheduling algorithm may increase the network load and prolong message delays. The prolonged message delays can cause tasks to violate their deadline and drastically degrade the performance of the system.

We consider the problem of dynamic scheduling tasks with precedence constraints on mesh connected systems. Our approach for solving the problem is modified from the scheduling algorithm for distributed systems. The approach groups processors into clusters and associates one scheduling coprocessor to each cluster. The scheduling coprocessors first attempt to schedule tasks to their local processors. The BB algorithm is used for local scheduling. Since scheduling coprocessors have the most current states of their local processors, there is no message exchange required for the local scheduling. As the number of scheduling coprocessors required in a system is greatly reduced, hardware cost is also reduced.

The task graph partition scheme employed in the scheduling algorithm is different from the strategy for the distributed systems. The strategy for mesh-connected multicomputers is to select the ready tasks with the shortest scheduling laxity first. A ready task is a task that all of its descendent are successfully scheduled. This strategy guarantees that all the tasks in a subgroup are independent. A subgroup is first locally scheduled by the BB algorithm. If there are tasks that cannot be locally scheduled, a global scheduling process is invoked to distribute the tasks to remote

clusters. The global scheduling algorithm is a simple procedure that chooses remote sites with the lightest loads and the shortest distances for task distribution. This global scheduling algorithm minimizes message exchange and the extra load added to the network.

2.3 Real-Time Flow Control for Wormhole Networks

In parallel and distributed real-time systems, tasks exchange messages. When scheduling cooperating tasks, a scheduling algorithm must consider the communication cost of different task allocations. In order to estimate the communication cost of a task allocation, the communication subsystem should deliver messages in a timely fashion. Otherwise, the unpredictable transmission time of a message may cause tasks to violate their timing constraints. However, the communication subsystems designed for general purpose systems cannot guarantee the timely delivery of messages. Communication subsystems specifically designed for real-time systems are required.

The communication system that we study is the wormhole network. The wormhole network is a popular communication subsystem for parallel systems. The benefits of wormhole networks, which include simple router designs and small buffer sizes, make them attractive for large-scale parallel systems. The use of virtual channels in wormhole networks increases the throughput and bandwidth utilization [15]. The work of Dally [16] showed that increasing the number of virtual channels can increase bandwidth utilization and reduce message delays. However, the conventional flow control scheme, first-come-first-served, is insufficient to support real-time communication. Dally also suggested that a flow control scheme based on the timing properties of messages can improve real-time performance.

Directly carrying timing information with messages in order to make flow control decisions is infeasible for wormhole networks. First, the large buffers required to store the timing information contradict the design principle of wormhole routers. Second, the hardware for making flow control decisions is complicated to implement and increases the cost of building such networks dramatically. The most important reason is that a very complicated mechanism, which requires both software and hardware support, is needed to maintain the correctness of the timing information. For example, the laxity of a message, which is the time before the message deadline, changes and needs to be modified constantly. To keep the laxity carried within the message current, clock synchronization and message exchanges among nodes are involved.

In order to support real-time communication in wormhole networks, a priority based flow control scheme, which include a time encoding technique and a priority adjustment method, is needed. A time encoding technique maps timing information into a priority, which can be represented in a small number of bits. If a priority embedded within a message only requires a small buffer, it will not complicate wormhole router design. A priority adjustment method, which adjust the priority of a message without message exchange, can maintain the correctness of the timing property represented by the priority.

We propose several priority based flow control schemes to support real-time communication and a message dropping method to relief the load contributed by the messages that miss their deadlines from the network. The proposed flow control schemes modifies the implementation of general purpose wormhole routers. With small additional hardware costs, the schemes provide the desired real-time performance. The real-time flow control schemes that we study include: *tightest deadline first* (TDF), *least laxity first* (LLF), *priority climbing* (PC), *enhanced priority climbing* (EPC), *rate monotonic scheduling* (RMS) and *one-bit*. The TDF and LLF schemes map deadline tightness and laxity, respectively, of a message to a priority. The wormhole routers

make flow control decisions based on the priority carried in the message header. The PC and EPC schemes enhance the TDF and LLF schemes by dynamically adjusting the priority of a message. As a message is being transmitted, the timing property of the message changes. The dynamic priority adjustment of PC and EPC schemes improves the performance from the TDF and LLF schemes. The RMS and one-bit schemes are derived from the RMS task scheduling algorithm [17] for static real-time systems. The two schemes require an off-line analysis of the task properties, such as the frequency of the message generation. This enables the schemes to have higher performance than the other schemes. Since tasks arrive to real-time systems can be either static or dynamic, we study the performance of the proposed flow control schemes in both environments. In a static environment, task arrival times are known at the system design time. In contrast, tasks arrive randomly in a dynamic environment. We conduct simulation experiments to compare the performance of the schemes and the conventional *first-come-first-serve* (FCFS) scheme.

A message dropping method can be used in conjunction with both the PC and the EPC schemes. The hardware needed to support the method is a one-bit flag for each virtual channel. The message dropping correctly drops messages that miss their deadlines without message exchanges and a complicated mechanism. Performance studies show that the message dropping method further improves the performance from the PC and the EPC schemes.

CHAPTER 3

Background Review

The objective of a real-time scheduling algorithm is to find a *feasible* schedule for a given set of tasks. A schedule is feasible if the successfully scheduled tasks can be executed and meet their requirements. The requirements for executing a task include a timing constraint (deadline), resource requirements and precedence constraints. In a schedule, some tasks might be rejected due to the fact that one or more of their requirements cannot be satisfied. Therefore, the task rejection ratio is a major metric for measuring the performance of a scheduling algorithm. When a real-time system is operating in a distributed or parallel environment, the scheduling algorithm must estimate the communication overhead between two tasks that are allocated to separate sites. Since tasks exchange information in such an environment to achieve a common goal, a task scheduling algorithm for a distributed or parallel real-time system must consider communication overhead.

Scheduling algorithms for real-time systems can be either static or dynamic. Static scheduling algorithms are designed for those systems that have perfect knowledge of the tasks, and the set of tasks to be executed is fixed. Task scheduling of these systems is done during the system initialization phase, thus the overhead of the scheduling algorithms do not affect the system performance. The results of static scheduling algorithms can be optimal. If there exists a feasible schedule for a given set of tasks, then

an optimal scheduling algorithm can find the solution. However, static scheduling is expensive and inflexible. When adding new tasks to the system, we have to stop the system and redo the scheduling to accommodate the new tasks. In some systems, *e.g.*, nuclear power plants, the cost to shut down the system is extremely expensive such that static scheduling is not suitable. Dynamic scheduling algorithms are designed for those systems that task arrival times are unpredictable. Dynamic scheduling generates schedules when there is a new task arrival. Unlike static scheduling algorithms, dynamic scheduling algorithms do not need perfect knowledge of the tasks before task arrivals. For those applications that operate in a changing environment, dynamic scheduling algorithms are used instead of static algorithms. Nevertheless, the run time cost of dynamic scheduling is high and the scheduling overhead of dynamic scheduling algorithms directly affects the system performance.

Since real-time tasks have deadlines to meet, the message transmissions between any two tasks are time constrained. In other words, the communication subsystem must guarantee that messages are delivered before the estimated transmission time. Real-time researchers have developed real-time protocols and flow control schemes for various types of communication networks. Examples of these networks include *multiple access networks*, and *point-to-point interconnection networks.*

In the remainder of this chapter, we overview the research results for both task scheduling and communication support for real-time systems that operate in parallel and distributed environments.

3.1 Static Scheduling Algorithms

Most of the conventional real-time systems operate in a static environment. Static scheduling algorithms for these systems are designed to serve two task arrival models: *periodic* and *aperiodic.* The set of tasks that executed in both of the two models are

known at design phase. In the periodic model, tasks are executed at fixed intervals. In contrast, the aperiodic model is used for tasks that arrive at irregular intervals. A new task model, which is called imprecise computing, divides a task into two parts and schedules the two parts separately. This model is attracting increasing interest among researchers. In this section we survey the research results of static scheduling algorithms for uniprocessor, parallel and distributed systems.

3.1.1 Static Scheduling on Uniprocessors

Liu and Layland [17] developed a *rate-monotonic* algorithm that assigns fixed priorities to a set of periodic tasks for preemptive execution. The tasks with shorter periods are assigned higher priorities. They showed that this scheme is optimal among all the fixed-priority scheduling algorithms. The feasibility test for a schedule in Liu and Layland's algorithm is a simple function that checks the utilization of the task set. The least upper bound of processor utilization for a given set of fixed priority order tasks is $m(2^{1/m} - 1)$, where m is the number of tasks. Later, Sha *et al.* [18] extended the rate-monotonic algorithms to adjust the priorities of critical tasks. In their algorithm, when critical tasks with long periods are rejected, periods of these critical tasks are transformed into shorter ones so that they can receive higher priorities and be successfully scheduled. Lehoczky *et al.* [19] proposed an analysis scheme, which gives insight to the characteristics of task execution, and the scheme achieves a higher utilization using the rate-monotonic scheduling algorithm. Liu and Layland assumed that the deadlines of the tasks are equal to their period. Leung and Whitehead [20] considered a problem in which the deadlines are shorter than the period, and a *deadline monotonic* algorithm was proposed. This problem was further addressed in [21, 22, 23]. The detail information of the RMS technology is given in Section A.1.

In previous algorithms, the tasks are assumed independent and preemptable. In practice, tasks might require resources for execution. Consequently, high priority

tasks can be blocked by low priority tasks. For example, if a non-preemptable resource is occupied by a low priority task and a high priority task requires that particular resource to execute, the high priority task is blocked. Sha *et al.* developed a *priority ceiling* protocol [24] that synchronizes tasks and their resource requirement to prevent resource conflicts. The proposed priority ceiling protocol can be used in conjunction with the rate monotonic scheduling algorithm.

A new task model, which is called *imprecise computation*, was introduced in [25, 26, 27]. The imprecise computation model assumes that each task consists of two subtasks: mandatory and optional. The mandatory part of a task performs the major computation of the task and delivers the imprecise result. The completion of the optional subtask delivers the precise result of the task. During scheduling, the mandatory subtasks are first scheduled. If a feasible schedule for the mandatory subtasks is found, the optional subtasks are then scheduled. Scheduling algorithms developed for imprecise computation are proposed in [27, 28]. Both of these algorithms are based on the *next-fit* algorithm [29], which is extended from the rate monotonic algorithm.

Horn [30] developed an *earliest-deadline-first* scheduling algorithm for aperiodic tasks on uniprocessors. The tasks with earlier deadlines are executed preemptively before the task with later deadlines. The complexity of Horn's algorithm is $O(n^2)$, where n is the number of tasks.

3.1.2 Static Scheduling on Distributed Systems and Parallel Multiprocessors

Scheduling tasks on parallel and distributed systems is more complicated than that on uniprocessors. In [30], Horn developed an $O(n^3)$ preemptive algorithm, where n is the number of tasks, based on the network flow method. The system of his approach is

modeled as a distributed system with homogeneous processors. Martel [31] extended Horn's algorithm to consider a distributed system with heterogeneous processors. The resulting complexity of the algorithm is $O(m^2n^4 + n^5)$, where m is the number of processors.

One approach of scheduling periodic tasks is to partition the tasks into groups based on the earliest-deadline scheme or rate monotonic scheme. Bannister and Trivedi [32] proposed a *best-fit* algorithm for partitioning the tasks. This best-fit algorithm is simple and can be used together with earliest deadline and rate monotonic schemes. Dhall and Liu [29] modified the rate monotonic algorithm and proposed a *next-fit* algorithm for task partition. Davari and Dhall [33] presented a suboptimal *bin-packing* algorithm, based on the next-fit algorithm, with earliest deadline scheme to partition the tasks into groups. The resulting complexity is $O(n)$.

If the task execution is nonpreemptive, then the problem becomes *NP-hard* [12] even with restricted assumptions. Several optimal nonpreemptive scheduling algorithms, which have polynomial complexity, for tasks with identical execution times are proposed in [34, 35, 36, 37].

Static scheduling algorithms for tasks with precedence relations have been studied in [38, 39, 40, 41, 42, 43, 44]. In [40, 41, 43], *Branch-and-Bound* algorithms are used for allocating tasks to processors. These algorithms provide optimal schedules for a given set of tasks. A task graph modeling and analysis technique is proposed [42] that provides a means of maximizing the concurrency of task executions. A preemptive scheduling algorithm that uses a graph algorithm to efficiently utilize multiprocessors is presented in [38].

3.2 Dynamic Scheduling Algorithms

In static scheduling algorithms, the scheduling steps required to schedule a group of tasks increases dramatically when the size of a group of tasks increases linearly. For dynamic scheduling, a system's performance would degrade severely if the scheduling algorithm demands high overhead. Therefore, existing static scheduling algorithms, which have high computational complexities, cannot be directly applied to the dynamic scheduling problem.

Hong and Leung [45] proved that there is no optimal algorithm for scheduling a set of independent tasks on $m > 1$ identical processors unless the tasks have a common deadline. And an optimal algorithm for dynamic scheduling tasks with a common deadline is proposed. Mok and Dertouzos [13, 46] showed that there can be no optimal scheduler without *apriori* knowledge of the start times of the tasks.

Dertouzos [3] proposed an *earliest deadline* algorithm for dynamic scheduling independent tasks on uniprocessors. However, the scheduling overhead is not addressed. Ramamritham and Stankovic [47] developed a dynamic algorithm based on the earliest deadline scheme and scheduling overhead is accounted in their performance analysis. Zhao *et al.* [48] presented a backtracking scheme for scheduling nonpreemptive tasks with resource constraints. The scheme was extended for scheduling preemptable tasks [49].

Several dynamic scheduling algorithms [50, 51, 52, 53] have been proposed for distributed real-time systems. Stankovic *et al.* proposed a *flexible* algorithm [50] that combines *focus addressing* and *bidding* scheduling algorithms. The flexible algorithm schedules independent tasks for distributed real-time systems. Ramamritham *et al.* presented an *O(n)* algorithm [51] for multiprocessor systems. The algorithm uses the *greedy* method and a heuristic function to evaluate the feasibility of schedules. Shin and Chang [52] used *preferred lists* for task distribution. Preferred lists are or-

dered lists of processors assigned to each processor during the system initialization phase. Blake and Shawn [53] described a *bin packing* algorithm for dynamic scheduling independent tasks on multiprocessor systems. Since these algorithms consider the problem of scheduling individual tasks, they are unsuitable for the problem of scheduling groups of tasks. Cheng *et al.* [14] proposed a dynamic algorithm for scheduling groups of tasks with precedence constraints. The algorithm breaks a task graph into clusters of tasks. Each cluster consists of a group of tasks with precedence relations, and the algorithm distributes the clusters as basic units among the processing nodes in the system. The flexible algorithm is used for distributing the clusters.

3.3 Real-Time Communication Protocols and Flow Control

Communication support for parallel and distributed system has been addressed for *multiple access networks*, and *point-to-point interconnection networks*. In this section, we survey the research results on real-time communication protocols and flow control schemes.

3.3.1 Multiple Access Networks

A multiple access network is an environment that all the sites of the network are communicating over a single communication channel and only one message can be successfully transmitted over the channel at any time [54]. If more than one message is simultaneously transmitted on the channel, these messages collide with each other. In such a case, none of the receivers receive these messages correctly. For the reason that all the sites of the network can monitor the communication channel, a message sent on the channel can be detected by all the sites. The satellite network and the

bus are examples of the multiple access networks. The multiple access network is also known as one of the broadcast networks.

Real-time protocols for multiple access networks can be classified into two categories: *controlled-access* and *contention-based* [54]. Controlled-access protocols are the protocols that provide collision-free channel access. By imposing ordering on the channel access rights to the sites, this type of protocol guarantees that no two sites will transmit messages simultaneously. Unlike controlled-access protocols, contention-based protocols support real-time communication by resolving contentions.

Controlled-access protocol: The controlled-access protocols can be further categorized into *predetermined channel allocation* (PCA) and *demand adaptive* protocols. PCA protocols statically allocate the channel to the sites. In other words, the PCA protocols do not adjust channel assignments according to the changing demands of the sites. Demand adaptive protocols dynamically allocate the channel to the sites that require the service. The most common strategy for PCA protocol that serves in the general purpose network is the time division multiple access (TDMA). TDMA protocols allow each site periodically use the channel for a fixed amount of time for transmission. In a fixed time period, the time is divided into equal length frames. The number of time frames is equal to the number of sites connected to the network. Each site is assigned to one time frame and can transmits messages only in its own time frame. A *slot-switched* TDMA technique [55, 56, 57, 58], which is a variation of the pure TDMA, was proposed to serve real-time communication in a centralized environment. In such an environment, messages are generated by several sources and collected by a central station. The messages are time-multiplexed onto a single outgoing channel by the central station. Maglaris and Lissack [59] investigated a slot-switched TDMA protocol that serves in a distributed environment. The MARS project [60] uses a simple TDMA protocol to support real-time communication.

Demand adaptive protocols, as PCA protocols, offer channel access rights to sites

in some order. The order can be determined off-line or dynamically by the sites that requiring the rights. In demand adaptive protocols, when there is no message to transmit, a site give up the channel access rights to the other sites that have messages waiting in their transmission queues. Two schemes are commonly used to determine the access order: reservation and token passing schemes.

In reservation schemes, there are reservation periods and transmission periods. A reservation period is further divided into a number of slots as many as the number of sites. Each site is associated with one reservation slot. During a reservation period, a site sends a burst of noise in its slot when there is a message to transmit. The sites that reserve the channel are granted the channel access rights in an order determined by their priorities. The sites that have the access rights transmit messages during the transmission periods [61]. Valadier and Powell [62] proposed a *waiting room* protocol that can serve in both centralized and decentralized environments. The waiting room protocol is also studied in [63] for general purpose networks. Lui *et al.* [64] proposed a reservation based protocol for dual-link networks.

In token passing protocols, one token is circulated among sites. A site can transmit message when it possesses the token. Strosnider *et al.* [65] applied the RMS task scheduling algorithm [66] to IEEE 802.5 token ring scheduling. Token passing protocols were also studied in [67, 68, 69]. Sevick and Johnson [70] analyzed the performance of the token passing scheme for the fiber-distributed-data-interface (FDDI) protocol. Shin and Hou [71] presented an analytical analysis of time-constrained CSMA/CD and token based protocols.

Contention based protocol: Unlike controlled-access protocols, contention based protocols allow all sites to transmit messages without channel access rights. When message interferences occur, a collision resolution process is used to determine which site has the channel access rights.

Kurose *et al.* [72, 73] purposed a window protocol for time constrained communica-

tion in multiple access networks. The window protocol resolves collisions by selecting the sites that have messages transmitted in an initial interval (window) for retransmission. If more than one site transmits messages in the initial window, the window size is reduced. This process repeats until only one site is selected. In [72, 73], the laxity before message deadlines is assumed identical. Panwar *et al.* [74] proposed a window protocol that releases the assumption. Zhao *et al.* [75, 76] extended the basic window protocol to improve the performance.

Zhao *et al.* [77, 78] proposed a collision based protocol, which is based on the virtual time CSMA protocol [79], for hard real-time communication. In the virtual time protocol, each site maintains two clocks: a real clock and a virtual clock. A site sends a waiting message when the time of the virtual clock is equal to some parameter of the message. Malcolm and Zhao [80] studied the virtual time protocol to be used in an environment that each message has several versions.

3.3.2 Point-to-Point Interconnection Networks

In point-to-point interconnection networks, sites are connected by individual links among the sites. A link between between a pair of sites is dedicated to the two sites. Point-to-point interconnection networks can be characterized by the switching mechanisms. Example of switching mechanisms are store-and-forward, virtual-cut-through, circuit switching [81] and wormhole routing [15].

Anderson *et al.* [82] proposed a message scheduling algorithm, which employs the RMS task scheduling algorithm, to support real-time communication for continuous media for the DASH project [83]. The switching mechanism of the DASH project is the store-and-forward scheme. Anderson [84] devised a metascheduling algorithm for continuous media. The metascheduling algorithm combines CPU scheduling, network flow control and file system management to guarantee real-time performance. Ferrari and Verma [85] investigated a message scheduling algorithm, which is based on the

earliest-deadline-first task scheduling algorithm [66], for wide-area networks. Kandlur *et al.* [86] combined deadline and fixed priority scheduling for real-time communication in multi-hop networks. Leung *et al.* [87] proposed a dynamic routing algorithm for store-and-forward networks to support real-time communication.

In the HARTS system [88], the system is a hexagonal-mesh-connected parallel real-time system. The switching mechanism for the network is virtual-cut-through. Ramanathan and Shin [89] proposed a multiple copy approach to support real-time communication in an environment such as HARTS. A reliable broadcast algorithm has been presented by Kandlur and Shin [90] for HARTS.

Most of the new parallel systems adopt wormhole routing [15] as their switching mechanisms. However, real-time communication support for wormhole networks has not been widely addressed until recent years. Shukla and Agrawal [91] described a scheduled routing scheme for wormhole networks. The scheduled routing scheme, which schedules and allocates tasks during compile time, was designed to serve periodic real-time applications. Mutka [92] proposed a flow control scheme for wormhole networks that is based on the RMS task scheduling algorithm. The scheme is designed to support systems in which message arrivals can be modeled as a linear bound arrival process [93].

CHAPTER 4

Dynamic Scheduling for Distributed Real-Time Systems

An algorithm for dynamically scheduling tasks with precedence constraints on distributed real-time systems is presented in this chapter. The algorithm partitions the scheduling of a group of precedence related tasks into subgroups. The maximum number of tasks in each subgroup is limited. The subgroups are scheduled independently in stages using a *branch-and-bound* (BB) algorithm. The overhead of the BB algorithm grows exponentially as the number of tasks being scheduled increases linearly. By limiting the number of tasks in a subgroup below a maximum number, which is a constant, the overhead of scheduling one subgroup can be considered a constant. The task group partitioning strategy reduces the overhead of scheduling a group to linear time.

If we partition a group of tasks to subgroups, we do not have a global view of the task relations as we would if we scheduled the entire group simultaneously. In other words, we can construct better schedules by increasing subgroup sizes. The tradeoff of the maximum subgroup size and the quality of schedules is studied. Simulation results are presented to study various parameters affecting the system performance. The proposed scheduling algorithm is a general scheme that can be easily ported to

different types of distributed and parallel systems.

4.1 The System Model

A description of the characteristics of a group of tasks is necessary to illustrate our model. Tasks arrive to the system in groups. Individual task groups arrive independently at each processor as a Poisson process. A deadline is associated with each group of tasks. A task group is modeled as an acyclic graph where directed edges are precedence relations among tasks. A task sends a message or a signal at the end of its execution to subsequent tasks. An example of a group with 10 tasks is given in Figure 4.1.

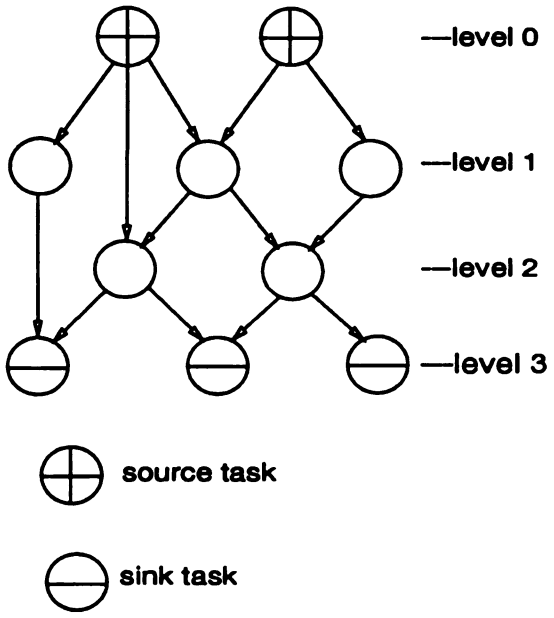


Figure 4.1. An example task graph of a task group.

The level of a task t is determined by the length of the longest path from the *source tasks* to t. A source task is a task without predecessors. Since task graphs

are acyclic, tasks at the same level are independent. The level of a task graph is the length of the longest path in the graph. Figure 4.1 illustrates the levels of tasks in a task graph. The number of tasks in a group is assumed to be exponentially distributed. The computational demands of tasks are assumed to be identical, which is similar to the demands modeled by Kar *et al.* [94]. This assumption can be easily generalized to non-identical demands by using a modified task group partitioning strategy illustrated in Section 5.2. The BB algorithm does not constrain the demands of tasks to be identical. Since we study how to appropriately control the cost of the scheduling algorithm while effectively using its power, the task model enables us to focus on the effects of the searching costs. More specific or complex task models are not needed for this examination but would be needed if well-defined real-time task workloads are given for specific problems.

In order to evaluate the tradeoff between the maximum size of subgroups and the efficiency of the scheduling algorithm, we devise a model of a loosely coupled distributed system that consists of N homogeneous processing nodes. Figure 4.2 illustrates the model. Each node contains a scheduling coprocessor to off-load the scheduling overhead from the task processor. This model corresponds to investigations of other researchers who study architectural issues of distributed real-time systems that have multiple processors in each node [90].

A scheduling coprocessor consists of two major components: the *local scheduler* and the *load distribution scheduler*. Initially, an attempt is made to schedule a newly arrived task group at the local task processor by the local scheduler. Tasks that cannot be executed locally are distributed by the load distribution scheduler. A local scheduling list is maintained by each scheduling coprocessor. Every scheduled task has an entry in the local schedule list of the processor on which it is scheduled. An entry of a local schedule list contains the task identification, group identification, start execution time, worst case execution time of the task, and processor identifications

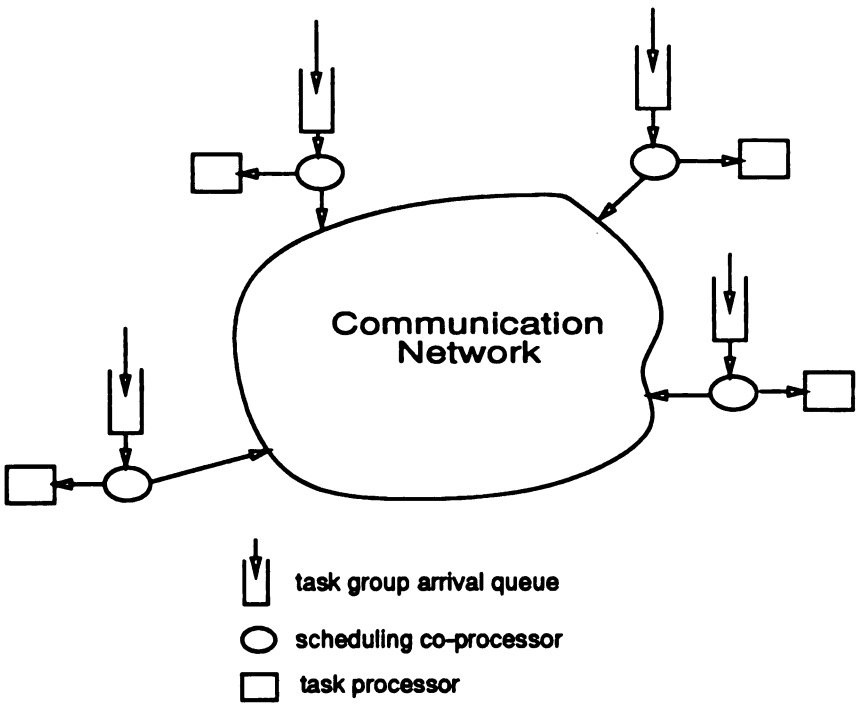


Figure 4.2. The system model.

of the locations of its immediate successors.

Every processor logs its own system status information, which is not necessarily the same as the information kept in other processors. To maintain accurate system status information, processors are responsible for broadcasting their load changes to the other processors. This *status change broadcasting* scheme was first proposed by Shin and Chang [52]. A processor can be in three states, *available*, *medium loaded* and *busy*. A processor broadcasts its load change when its status changes from available/busy to busy/available. This scheme can minimize the number of messages needed to maintain system status information kept at the processors. Although the model of the communication subsystem is a point-to-point communication network and no other assumptions need to be made for the topology, we investigate a system with communication delays that models a hypercube topology [95].

4.2 Task Group Preprocessing

Some processing of a task group is needed prior to the scheduling of the tasks. The scheduling of a task group is done in stages that a subgroup is scheduled in each stage. The task graph partitioning strategy extracts independent tasks of a group and separates them into subgroups. The scheduling of a task requires the knowledge of the deadlines of the tasks and the earliest time the tasks can start execution. The task group partitioning and estimating timing properties of tasks are discussed in this section.

4.2.1 Task Graph Partitioning

In order to maximize the parallelism of task execution, independent tasks are scheduled simultaneously for concurrent execution. The task graph partitioning strategy divides a task group into several subgroups such that tasks within the same subgroup are independent. There must not be precedence relations between tasks at the same level. By partitioning tasks into subgroups according to their levels, we are ensured of independence between tasks in the same subgroup.

The number of levels of a random task graph can be any number from one to the size of the task group. We divide each subgroup into smaller subgroups such that the maximum size of each subgroup is a constant S. Selection of the constant S is not a trivial issue since the size of S directly relates to the number of scheduling steps required for a subgroup. If the underlying scheduling algorithm has nonlinear computational complexity, a relatively large S may cause excessive scheduling overhead such that the task rejection ratio becomes unacceptable. We study the relation between S and the time required for each scheduling step.

4.2.2 Estimating Task Deadlines and Earliest Start Execution Time

When a group of tasks arrives, a deadline is associated with the group. A group of tasks is successfully scheduled if all the tasks are scheduled to finish execution to meet the group deadline. During scheduling, each task must have a specific deadline as well as an *earliest start execution time* (ESET). A task deadline is determined such that the subsequent tasks have enough slack time to be executed before their deadlines. The ESET of a task is calculated as the time required for the predecessor tasks can finish execution.

Our scheduling algorithm schedules tasks from the highest level to the lowest level. When scheduling a task, all of its immediate successors are successfully scheduled. A task must be scheduled such that it is guaranteed to finish before any of its successors starts execution. The deadline of a task is chosen as the earliest starting point of all its immediate successors. The deadline of a task t is calculated as

$$D(t) = MIN\{Exectime(s_i), \forall s_i \in Child(t)\},$$

where $D(t)$ is the deadline of t, $Exectime(s_i)$ is the scheduled execution time of task s_i, and $Child(t)$ is the set of all the children of t. Sink tasks have no children, so the deadline of a sink task is the group deadline. We introduce another term, *latest start execution time*(LSET), to simplify later discussions. The LSET of a task specifies the latest time that the task must start execution, so the task can finish execution and sends messages to its children before the task deadline. The LSET of task t is determined as

$$LSET(t) = D(t) - Demand(t) - ComCost(t),$$

where $Demand(t)$ is the computation demand of the task and $ComCost(t)$ is the

worst cast communication time needed to send a signal to its children. The LSET of a task is calculated at the time the task is being scheduled.

The value of the ESET of a task is fixed during the scheduling, and it specifies the earliest time the task can start executing. The ESET of a task t is computed as

$$ESET(t) = clock + MAX\{Demand(P_i), \forall p_i \in PathSet(t)\},$$

where *clock* is the current time, P_i is a path from a source task to the task t, $Demand(P_i)$ is the total demand of the tasks on the path and $PathSet(t)$ is the set of all the paths from source tasks to t. The execution of a task must be scheduled to start at a time within the period from the ESET to the LSET.

4.3 Selecting Processors for Task Allocation

Tasks can be either allocated to their local task processors or distributed to the other remote processors in the system. When tasks are considered for global distribution, the number of destination processors must be equal to or greater than the number of tasks in a subgroup to maximize the parallelism of task execution.

The set of processors selected for task distribution in a scheduling stage is called a *preferred set.* The processors in a preferred set are selected from the available processors in the *preferred list* of the scheduling processor. Each processor has a unique preferred list, which is an ordered list of all the other processors in the system. The order of a processor in a preferred list is determined by the geographic distance from the processor to the owner of the preferred list. The uniqueness of preferred lists minimizes the chance that two or more processors are simultaneously dumping their loads to the same processor. The construction of preferred lists is architecturally dependent as described by the algorithm for k-ary n-cubes in [52]. An example of

preferred lists for a binary 4-cube is given in Table 4.1.

Table 4.1. An example preferred list for a binary 4-cube.

Processor	Preferred List														
0	1	2	4	8	3	5	9	6	10	12	7	11	13	14	15
1	0	3	5	9	2	4	8	7	11	13	6	10	12	15	14
2	3	0	6	10	1	7	11	4	8	14	5	9	15	12	13
3	2	1	7	11	0	6	10	5	9	15	4	8	14	13	12
4	5	6	0	12	7	1	13	2	14	8	3	15	9	10	11
5	4	7	1	13	6	0	12	3	15	9	2	14	8	11	10
6	7	4	2	14	5	3	15	0	12	10	1	13	11	8	9
7	6	5	3	15	4	2	14	1	13	11	0	12	10	9	8
8	9	10	12	0	11	13	1	14	2	4	15	3	5	6	7
9	8	11	13	1	10	12	0	15	3	5	14	2	4	7	6
10	11	8	14	2	9	15	3	12	0	6	13	1	7	4	5
11	10	9	15	3	8	14	2	13	1	7	12	0	6	5	4
12	13	14	8	4	15	9	5	10	6	0	11	7	1	2	3
13	12	15	9	5	14	8	4	11	7	1	10	6	0	3	2
14	15	12	10	6	13	11	7	8	4	2	9	5	3	0	1
15	14	13	11	7	12	10	6	9	5	3	8	4	2	1	0

The size of a preferred set is directly related to the scheduling overhead. Suppose we consider all the processors in the system for task distribution. If the number of processors in the system is large, the scheduling overhead is unacceptable even if the size of the task group is small. However, if the size of a preferred set is smaller than the number of tasks being distributed, we do not explore the possibilities of concurrently executing independent tasks.

We examine four preferred set expansion strategies, *exact*, *plus-4*, *binary* and *Fibonacci* methods. The exact method scans from the head of the preferred list to find available processors to add to the preferred set. The preferred set expansion process terminates when there are exactly the same number of processors in the preferred set as the number of tasks to be distributed. The plus-4 method scans

four processors from the preferred list. If the number of processors included to the preferred set is less than the number of tasks, then 4 more processors are scanned. The process repeats until there are at least as many processors in the preferred set as the number of tasks. The binary method initially scans a number k $(= 2^j,\ j \geq 0)$ processors from the head of the preferred list. The value k is a power of two that is the smallest number equal to or larger than the number of tasks. When the number of available processors in the preferred set is smaller than the number of tasks to be distributed, additional processors in the preferred list are scanned such that the total number of scanned processors is $2k$ $(= 2^{j+1})$. The Fibonacci method increases the number of scanned processors following a Fibonacci sequence.

The exact method includes exactly the same number of processors in the preferred set as the number of tasks, so that the scheduling overhead is minimized. However, due to the inaccuracy of system status information, some processors in the preferred set might be busy and the tasks are rejected. The plus-4, binary and Fibonacci strategies include more processors than the number of tasks and increase the chance of successful scheduling. Nevertheless, these three methods introduce higher scheduling overhead than the exact method. The tradeoff between scheduling overhead and task rejection rate is studied in Section 4.5. Simulation results show that the Fibonacci method has the best performance when the system is heavily loaded.

4.4 The Scheduling Algorithm

The scheduling of a task group is done in stages. Higher level tasks are scheduled in earlier stages than lower level tasks, *i.e.*, task scheduling proceeds from sink tasks to source tasks. If lower level tasks are scheduled before higher level tasks, there is a chance that lower level tasks start execution before all the tasks are scheduled. However, some higher level tasks might be rejected, and the tasks that already have

started execution need to be withdrawn from the system. The process of withdrawing tasks that have started execution complicates the system design. Our scheduling strategy ensures that tasks are executed only if the whole task group is successfully scheduled.

The tasks scheduled in each stage are selected from the remaining unscheduled tasks at the highest level. The tasks in each subgroup must be at the same level to ensure the independence of the tasks. A subgroup of tasks are first considered for local execution by the local scheduler at the site to which they arrived. If some of the tasks in the subgroup cannot be locally scheduled without violating their deadlines, these tasks are globally distributed by the load distribution scheduler. When any of the tasks cannot be scheduled either locally or globally, the group of tasks is rejected. An execution time and a location are associated with each task if they are successfully scheduled.

4.4.1 Local Scheduling

A local scheduler is invoked when new tasks arrive or when other processors are confirming their global task distributions. In the case that there are task arrivals, the local scheduler searches the local schedule list to find free slots for the tasks. The free slot for a task must satisfy the condition that the task can start execution later than its ESETs and earlier than its LSETs. Therefore the task is guaranteed to meet its deadline when executing in the free slot. The tasks that have free slots assigned for them are locally scheduled, and the entries for the tasks are inserted into the local schedule list. If the local scheduler cannot find free slots for some of the tasks in the subgroup, these unscheduled tasks are considered for global distribution by the load distribution scheduler.

When a scheduling coprocessor receives task distribution messages from other processors, the local scheduler processes the messages in the following way. The local

scheduler searches the local schedule list to find free slots as specified by the messages for the globally distributed tasks. If free slots are found for the globally distributed tasks, the slots are reserved for the tasks and an *accept* message that contains the execution time for the task is returned to the sender of the task distribution message. Otherwise, a *reject* message is dispatched to the sender.

4.4.2 Global Scheduling

The load distribution schedulers are invoked when tasks cannot be locally scheduled. Using the number of tasks marked for global distribution, a preferred set of processors is selected from the preferred list. Global distribution lists are generated using a BB algorithm. The details of the BB algorithm is presented in the later discussion of this section. The BB algorithm assigns the tasks to the preferred set of processors such that each task is allocated to a different processor. Then task distribution messages are sent to the destination processors to confirm the global distribution lists. A task distribution message contains the ESET, the LSET and the computational demand of the task. If all of the destination processors respond with an accept message, the tasks are transferred to those destination processors. Otherwise, a second distribution list is selected and confirmed with the destination processors. This confirmation process repeats until either one of the distribution lists is accepted or some of the tasks' deadlines are passed. If some tasks miss their deadlines, the group of tasks is rejected and *withdraw* messages are sent to the processors where tasks of this task group were scheduled. Since the algorithm schedules tasks from the highest level to the lowest level, we are ensured that tasks of a group can start execution only if all the tasks are successfully scheduled. When local schedulers receive withdraw messages, the entries for the scheduled tasks from the group are simply removed from the local schedule lists.

4.4.3 The BB algorithm

A BB algorithm is a depth first search of a tree with bounding conditions. The computational complexity of the search is $O(n^t)$, where n is the number of available processors in the preferred set and t is the number of tasks to be distributed. Figure 4.3 is an example of a task assignment tree. The edge between two vertices represents the assignment of one task to a processor. The label within the square near the edge is the processor to which the task is assigned. For example, if the label of edge (v_1, v_2) is 1 then task t_1 is assigned to processor p_1. The directed arrows with each edge represent the search directions and their labels are the search steps. When the search reaches a leaf of the tree, one complete distribution list is constructed.

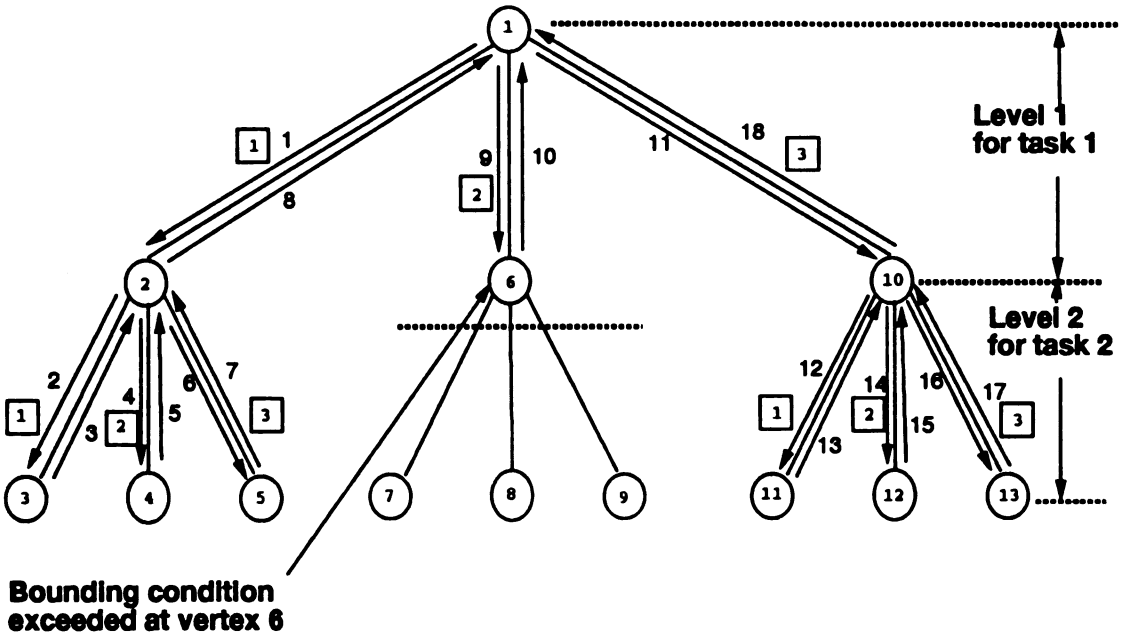


Figure 4.3. A task assignment tree.

When the search reaches a vertex, the LSET of the current distribution list is evaluated and the bounding condition is checked. The LSET of a list l is evaluated

as

$$LSET(l) = MIN\{LSET(t),\ \forall t \in l\},$$

where $LSET(l)$ is the LSET of l and t is a task in l. Intuitively, we want to distribute tasks to processors that provide longer scheduling periods for the tasks. Since the ESETs of tasks are fixed values, the LSETs of tasks determine the periods during which the tasks can be scheduled. The bounding condition is that the LSET of the current list should be later than the previous LSETs of the previously constructed lists. If the bounding condition is not matched, the sub-tree rooted at the vertex is not searched. For example, in Figure 4.3 the sub-tree of vertex v_6 is not searched because an assignment of task t_1 to processor p_2 will produce a list with an earlier LSET than the previously constructed lists.

An outline of the BB algorithm is given in the pseudocode listed in Table 4.2.

Suppose there are n available processors in the preferred set and t tasks to be scheduled. One task is first assigned to an available processor, and the LSET of the task is evaluated. Since the LSET of a task is determined by the execution time of its successors and the communication delays with its successors, we evaluate the LSET of a task whenever it is distributed to a different processor from its successors. If there are other unscheduled tasks, then these tasks are considered for distribution. This procedure continues until all potential distribution lists are constructed. The LSET of the list is updated while assigning tasks to processors. The process of constructing a list is aborted if the LSET of the list is earlier than the earliest LSET among previously saved lists. For simplicity of illustration in Table 4.2, we save the best list and its LSET. The BB algorithm limits the number of lists that it saves.* The procedure of constructing a list repeats until all alternatives of task assignments have been considered.

*We save at most 5 schedules in our experiments.

Table 4.2. The Branch-and-Bound algorithm.

```
procedure Branch-and-Bound( preferred_set, n, dist_set,t)
//preferred_set[1:n] is the array of processors in the preferred set //
//dist_set[1:t] is the set of tasks to be distributed //
var
    LSET[1:t] : real; //array of LSET for each task//
    LLET : real; //the LSET of the current list//
    Best_LLET : real; // the latest LLET among all lists //
    task_indx : integer; // index to the tasks //
    list[1:t] : integer; //array for storing the list //
    best_list[1:t] : integer //array for storing the best list//

subprocedure Search( task_indx, LLET)
var
    preferred_set[1:n] : External;
    dist_set[1:t] : External;
    LSET[1:t]: External;
    temp_LLET : real;
    proc_indx : integer; // index to the processing nodes //
BEGIN
        for proc_indx = 1 to n
        BEGIN
                LSET[task_indx]←MIN("execution time of task_{task_indx}'s successors"
                        − "communication delay") − task demand;
                if(LSET[task_indx] < LLET ) temp_LLET←LSET[task_indx];
                else temp_LLET←LLET;
                if( temp_LLET > Best_LLET ) then
                BEGIN
                        list[task_indx]←proc_indx;
                        if( task_indx < t ) then Search(task_indx+1, temp_LLET);
                        else
                        BEGIN
                                Best_list←list;
                                Best_LLET←temp_LLET;
                        END
                END
        END
END

BEGIN
        Best_LLET←0;
        LLET← ∞
        task_indx←1;
        Search(task_indx, LLET);
END.
```

4.5 Performance Studies

We developed a simulator to evaluate the cost and power of our dynamic scheduling algorithm. We show that significant improvements are obtained by scheduling a subgroup of tasks simultaneously in comparison to a scheme that distributes one task at a time. Nevertheless, the costs of the searching steps of the scheduling algorithm can be significant and can impose limits on the appropriate size of a distribution subset. The results of the simulation experiments show the performance of the algorithm as it is affected by the maximum size of the distribution subset, the scheduling overhead and the preferred set expansion strategies.

4.5.1 The Simulation Model

The simulated system is composed of sixteen homogeneous processing nodes. Each processing node consists of a processor and a scheduling coprocessor. The communication subsystem is a point-to-point network using the store-and-forward switching mechanism. Therefore, the message delay and task transfer delay between each pair of nodes are determined by the path length between the two nodes. The interconnection topology of the network is modeled as a hypercube. Since our study emphasizes the effect of maximum size of the distribution subset and the scheduling overhead, we do not include the communication overhead caused by network contention in the message delay and the task transfer delay. The message delay between two nodes that are separated by a single hop is 5 units of scheduling time.

It is often the case in real-time systems that the task demands are known at the system initialization phase, or at least at the arrival times. In order to reduce the task transfer overheads, it might be feasible to store a code for each task at several nodes (or perhaps all nodes). Therefore, a task transfer implies a message with the identity of the task and the required input for the task. To simplify the simulation,

we assume that the task transfer delay is twice the message delay between the sender and the receiver.

Task groups arrive at each node as a Poisson process, which is independent from the arrivals at the other nodes. The arrival rates for all the nodes are identical. The number of tasks in a group is exponentially distributed with a mean of 6 units, which is a suitable number of tasks for the given size of the system. Task graphs are generated randomly such that they are acyclic random graphs. Computation demands of tasks are chosen to be 100 units. We study the effect of scheduling overhead by varying the cost of each scheduling step from 1-10 units, therefore each scheduling step is 1-10% of a task demand. The task transfer cost to an adjacent node is twice the message delay, such that the task transfer cost per hop is 10% of a task demand. Since the number of scheduling steps grows as the size of a task group increases, the scheduling time can overwhelm the task demands. In addition, the cost of tasks transferred to nodes across several hops can exceed the demands of the tasks.

The deadline tightness of each group affects the performance of a task scheduling algorithm. Deadlines are evaluated by assigning a *deadline tightness factor* to each group. A group deadline is calculated as the computation demand of the group plus the product of the demand and the deadline tightness factor. Suppose the deadline tightness factor of a group is 0.3. The group deadline is the arrival time plus the product of 1.3 and the sum of computation demands of the tasks in the group. Note that it is reasonable to investigate a system with the deadline tightness factor set to 0.0 as long as some tasks in a group can be executed in parallel.

The values of the two thresholds, T_l and T_h, that are used to distinguish the states of a node are 0.3 and 0.7 of the system load, respectively. The system load is the average of the individual node loads kept locally at each node. The time period W for calculating the load of a node is 10,000 time units. The load of a node is computed as the ratio of the number of tasks scheduled at the node to W. The manner of choosing

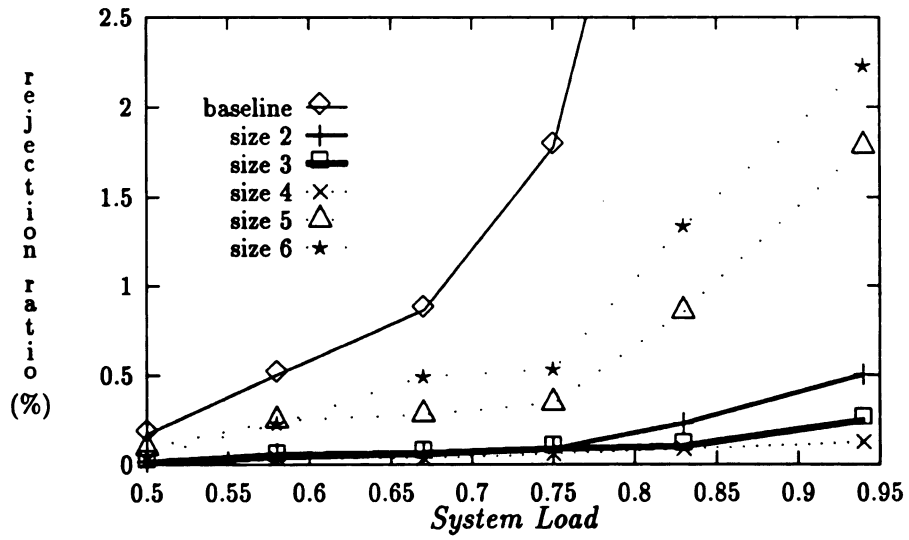
the values of T_l, T_h and W is itself an interesting research topic. The values for our studies are chosen to minimize the message overhead for exchanging state information while keeping accurate state information in each node for load distribution.

4.5.2 Relationship between Maximum Subgroup Size and Rejection Ratio

We compare the performance of our algorithm to a baseline algorithm. The group rejection ratio measures the performance of the task scheduling algorithms. The baseline algorithm distributes a task group along its critical path in the task graph. The scheduling is done in stages. At each stage, one task is selected from the highest level, which has tasks unscheduled, for scheduling. When distributing a task, the baseline algorithm looks for the first available processing node from the beginning of the preferred list and distributes the task to that node. The baseline algorithm appears to be a special case of our algorithm when the maximum size of the distribution subset is equal to 1. The difference is that our algorithm uses a BB algorithm for task distributions and constructs one or more load distribution lists from the preferred set.

Figure 4.4 shows the performance comparison between our algorithm and the baseline algorithm. We varied the maximum size of the distribution subgroups of our algorithm with values from 2 through 6. The unit scheduling time is 1 and the deadline tightness factor is 0.0 in this experiment.

The system load of a well designed real-time system generally does not exceed a specified limit under normal operating conditions. From the fault-tolerance point of view, one advantage of distributed and parallel systems is that the performance of these systems degrades gracefully under node failures. Therefore, the load of a system may exceed the specified limit when there are faulty nodes, and it is worthwhile to examine the stability of the scheduling algorithm subject to very high loads. The



Unit scheduling time = 1, deadline factor = 0.0.

Figure 4.4. Performance of the group scheduling algorithm and the baseline algorithm.

results of our studies show that our version of the scheduling algorithm offers stable performance under very high system loads, as shown in Figure 4.4. The stability is due to the choices made by the BB algorithm for rejecting groups that cannot be scheduled. Note that the x-axis in Figure 4.4 presents the load offered to the system. We can consider very large offered loads since some groups will be rejected. By making good choices for rejections, the overall rejection rate is low and stable. The effective system load will be lower than the offered load.

A primary constraint of the BB algorithm is its computational complexity. The scheduling overhead of the BB algorithm increases exponentially when the number of tasks and the available processors in the preferred set increases. If we constrain the number of tasks in a distribution subgroup to a small number, then the scheduling overhead is limited to a reasonable range. The results presented in Figure 4.4 show that the rejection ratio decreases when the maximum size of distribution subgroups increases from 2 to 4. Nevertheless, the rejection ratio of size 5 and 6 increases and is

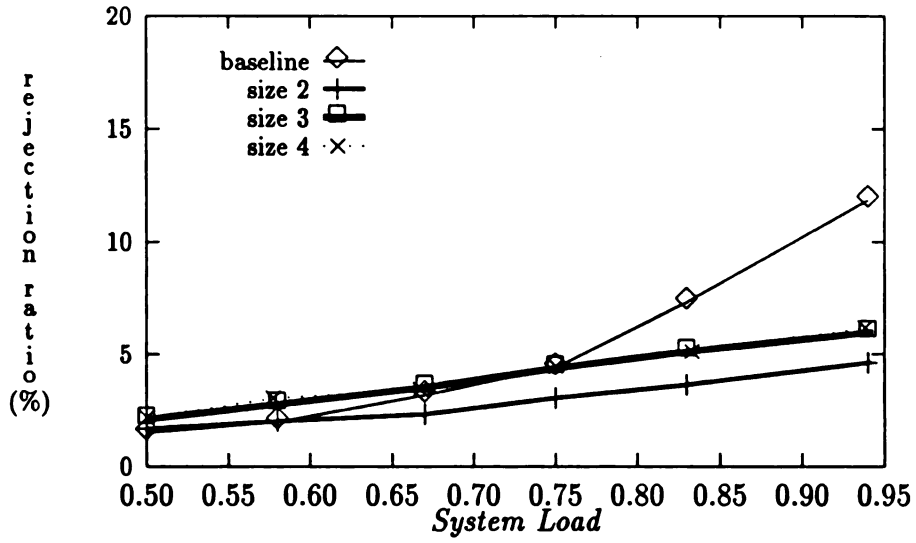
significantly higher than that of sizes 2, 3 and 4. Although larger subgroup sizes can be useful for finding good schedules for off-line scheduling, the larger sizes consume a significantly larger number of scheduling steps. The scheduling time can become the bottleneck of the system causing more rejections.

4.5.3 Effects of Scheduling Overhead

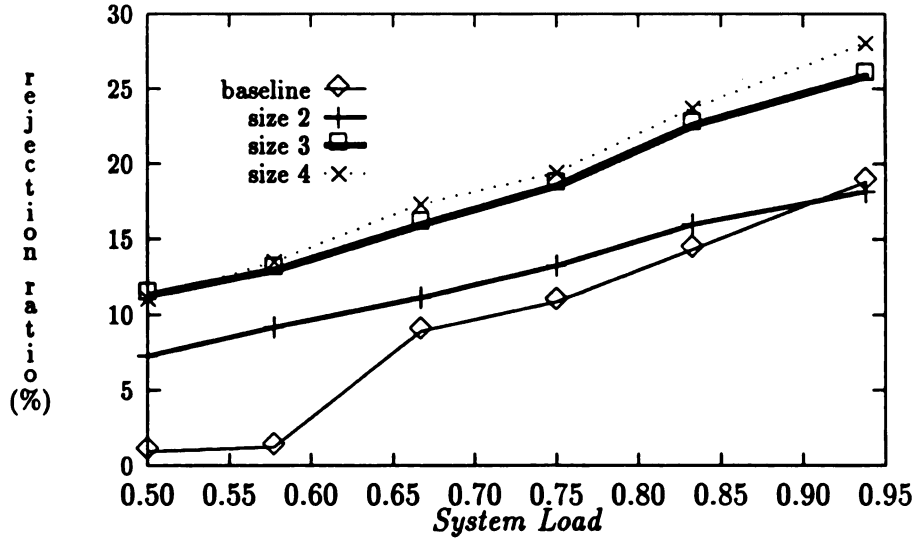
We examined the effect that the scheduling overhead has on the group rejection ratio as we varied the distribution subset size. For example, we looked at the effect when the maximum size of the distribution subgroup ranged from 2 to 4 and the unit scheduling times were 1, 5 and 10 units. The deadline tightness factor for each group was 0.0. Simulation results are shown in Figures 4.4, 4.5(a) and 4.5(b).

These experiments used very large values for the unit scheduling time in comparison to the execution time of a task. The ratio of execution time of a task to the unit scheduling time can be interpreted as either the granularity of tasks or the ratio of the speed of the processor to the speed of the scheduling coprocessor. Suppose the ratio of task execution time to unit scheduling time represents the granularity of tasks. The values of unit scheduling time used in our studies represent very fine grained tasks. In practice, tasks would likely have larger computational demands than the values we studied. It is expected that the unit scheduling time would be smaller than 1% of a task execution time. As advances continue in electronic and VLSI technologies, inexpensive special purpose coprocessors will become available. The speed of the coprocessors will be relatively high compared to the processors they support. The use of scheduling coprocessors for high performance real-time systems becomes a reasonable assumption.

The performance of the scheduling algorithm can be sensitive to the unit scheduling time. The simulation results show that the scheduling algorithm successfully schedules task groups when the unit scheduling time is 1% of the execution time of a



(a)Deadline factor = 0.0, unit scheduling time = 5.



(b)Deadline factor = 0.0, unit scheduling time = 10.

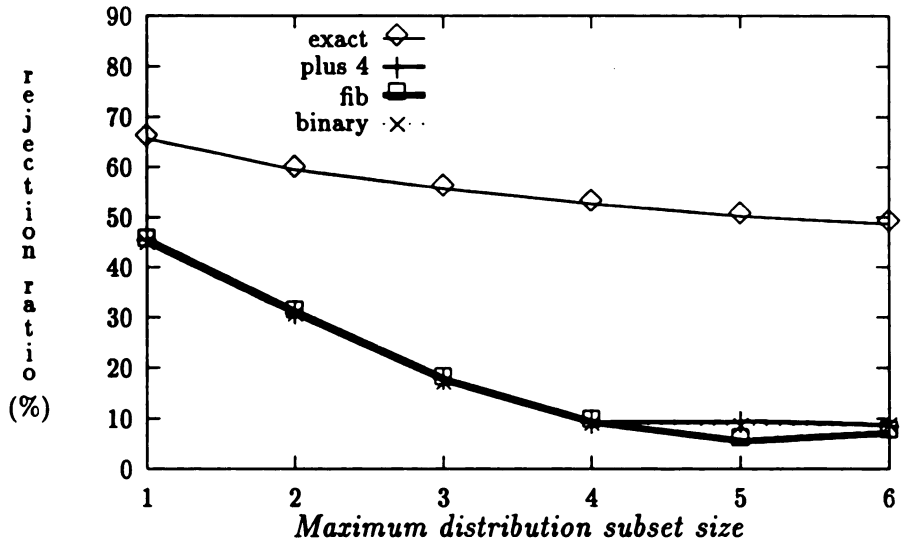
Figure 4.5. Sensitivity of the rejection ratio to the maximum distribution subset size.

task (the results for sizes 2-4 are presented in Figure 4.4). The scheduling algorithm has reasonable performance even if the unit scheduling time is as expensive as 5% of the execution time of a task (see Figure 4.5(a)). However, the algorithm should not be used when each scheduling step is very expensive (*e.g.*, 10% of the execution time of a single task as in Figure 4.5(b)).

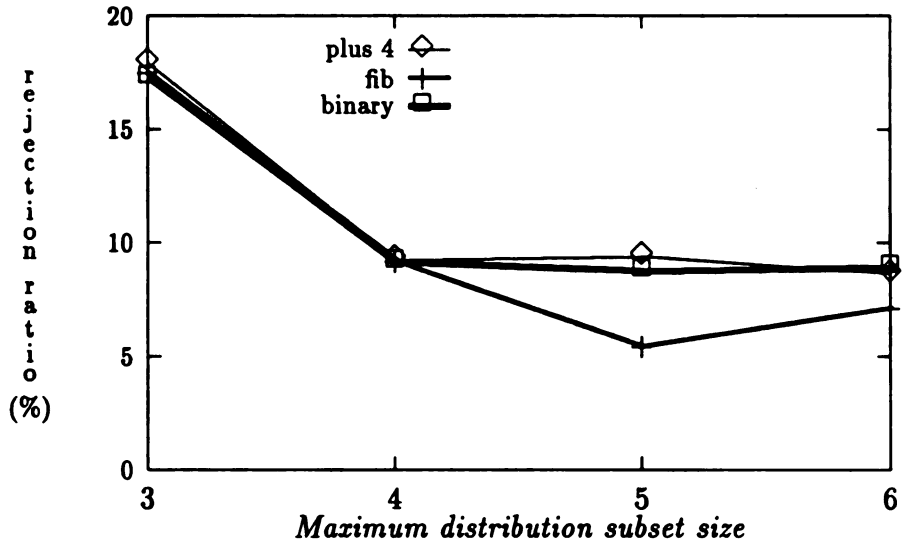
4.5.4 Comparison of Preferred Set Expansion Strategies

The number of processors in a preferred set and the manner in which the set expands has a significant impact on the performance of the BB algorithm. In order to explore the impact, we compared four different preferred set expansion techniques: the binary, Fibonacci, plus-4 and exact methods. The results of the comparison are shown in Figure 4.6(a) and (b).

To highlight the impact of the methods for expanding preferred sets, we experimented with a system of 64 nodes. The average number of tasks in a group is 16. The unit scheduling time was 5 units, the deadline tightness factor was 0.5, and the system load was 0.8. The results in Figure 4.6(a) show that the binary, Fibonacci and plus-4 methods have significantly lowered rejection ratios than the exact method when the system is heavily loaded. The performance of the three methods improved more rapidly than the exact method when we increased the maximum distribution subset size from 1 to 6. Since the three methods may have more processors in the preferred set than the number of tasks in the distribution set, the BB algorithm can produce better distribution lists from the enlarged search space. When the system is heavily loaded, it is likely that the local information in the node is inaccurate because some of the processors marked available in the preferred set have become busy. Therefore, a larger number of processors in the preferred set in comparison to the number of tasks in the distribution subgroup can help reduce the rejection ratio of the system.



(a)Comparisons of the four expansion methods; maximum subgroup size 1–6



(b)Comparisons of binary, Fibonacci and exact methods; maximum distribution subgroup size 3–6

System load = 0.8, Unit scheduling time = 5 units, Deadline tightness factor = 0.5

Figure 4.6. Sensitivity of the scheduling algorithm to the preferred set expansion method.

Figure 4.6(b) shows a detailed comparison of the binary, Fibonacci and plus-4 techniques. The maximum distribution subgroup size shown in this figure varies from 3 to 6. The figure indicates that the Fibonacci method performs better than the binary and plus-4 methods when the maximum distribution subset size is larger than 4. The Fibonacci method does a better job of balancing the tradeoff between producing good quality schedules and minimizing the scheduling overhead in comparison to the binary and plus-4 methods. The rejection ratio obtained when using the Fibonacci method is nearly half of the ratio obtained when using the binary and plus-4 methods. The binary method, in general, includes more processors in the preferred set than Fibonacci method. Therefore, the search space of the binary method is larger than Fibonacci method, *i.e.*, the binary method provides better quality schedules. However, the resulting scheduling overhead of the binary method is larger, which itself can cause task groups to miss their deadlines. The plus-4 method includes fewer processors in the preferred set. As a result, the scheduling overhead is less than of the Fibonacci method. Nevertheless, the plus-4 method has a higher rejection ratio because the quality of the generated schedules is poorer than the schedules generated by the Fibonacci method.

4.6 Summary

An algorithm for dynamic scheduling of cooperating tasks on distributed systems is presented in this chapter. The majority of previous on-line algorithms schedule independent tasks to meet their deadlines. However, high-level real-time jobs usually consist of tasks with precedence constraints. An efficient algorithm that schedules cooperating tasks must consider the order of task execution. Otherwise, the computational capacity of processors can be underutilized.

The scheduling algorithm dynamically schedules a task group. When a task group

arrives at a node, the tasks are first scheduled to the local processor. If there are tasks that cannot be locally scheduled without violating their deadlines, a global scheduling algorithm is invoked to distribute the tasks to remote processors. The task allocation scheme for the global scheduling algorithm is the BB algorithm.

By limiting the search space of the algorithm, the scheduling overhead is reasonable for on-line scheduling. The search space is limited in two ways. First, scheduling is partitioned into stages, and only a subset of independent tasks is scheduled at each stage. A scheduling subset is further divided into smaller distribution subsets. The maximum number of tasks of a distribution subset is a constant. Second, the distribution of a subset is limited to a preferred processor set. The preferred processor set is a subset of all the processing nodes. Therefore, the scheduling overhead of each stage is upper bounded by a constant. Since the complexity of scheduling one stage can be considered as a constant, the number of steps required to schedule a task group grows as the same rate as the size of the group increases. The overhead of the scheduling algorithm is suitable for on-line scheduling.

We developed a simulator to examine the power and cost of the proposed algorithm. Simulation studies showed that the BB algorithm performs very well even when the cost of unit scheduling overhead is relatively high. One interesting observation is that the BB algorithm provides good performance when scheduling task groups with tight deadlines. This is due to the precision of the load distribution schedules generated by the BB algorithm. Since we consider both the locations and the execution times for tasks distribution, we are ensured that the execution of the tasks of a task group follow the precedence order. Therefore, the computational capacity of the processors is efficiently utilized.

CHAPTER 5

Scheduling Mesh Connected Multiprocessors

As the availability of inexpensive and powerful microprocessors, the design and development of parallel multiprocessors with large number of processors has gathered increasing attention from computer researchers. Several topologies have been proposed to interconnect these processors, which include the tree, the hypercube and the mesh. The mesh is a very popular topology for the new generation of multiprocessors since it has low connectivity and high scalability. Mesh connected multiprocessors provide high reliability and efficient communication mechanisms that make them ideal candidates for real-time applications. We developed a distributed scheduling algorithm for mesh connected multiprocessors. The scheduling algorithm dynamically schedules cooperating tasks to the processor mesh.

The scheduling algorithm presented in Chapter 4 can be directly ported to parallel systems. However, the cost of associating one scheduling coprocessor to each node is unacceptable when the size of a system becomes large. The other issue is that the messages required for scheduling increase the load of the network and thus cause contention and message delays. The scheduling algorithm described in this chapter is specifically designed for parallel multiprocessors. The scheduling algorithm groups

processors in a system into clusters and associates one scheduling coprocessor to each processor cluster. The number of scheduling coprocessors in the system is greatly reduced. In the algorithm, a scheduling coprocessor does not require any message exchange to schedule tasks to the processors in its local cluster. The network contention is reduced. Furthermore, the scheduling algorithm employs a revised task graph partitioning strategy from the previous algorithm. The task graph partitioning strategy enables the scheduling algorithm to serve a non-identical task demand model.

5.1 The System Model

The system is modeled as a mesh with $m \times m$ processors. The processors are further grouped into clusters, and each cluster is a $a \times a$ submesh, where $a \leq m$. In addition to the processors, each cluster has a scheduling coprocessor to offload the scheduling overhead from processors. Each scheduling coprocessor has two major components: a local scheduler and a global scheduler. The local scheduler maintains the status of the processors in its cluster and schedules tasks to these processors. The global scheduler generates global distribution lists for those tasks which cannot be locally scheduled. The global distribution lists are confirmed by the global scheduler with remote scheduling coprocessors. Scheduling coprocessors are interconnected by another layer of a mesh network. Figure 5.1 describes the model of the system.

The messages for scheduling and exchanging system status are transmitted through the coprocessor network to reduce communication traffic from the processor network. The processor and coprocessor meshes are *wormhole* routed. The time for transmitting messages between any pair of nodes is assumed identical. This assumption is particle, if the communication subsystem can transmit messages to meet their timing constraints. The communication support to achieve the required system performance is an important issue. However, it is not within the scope of this chap-

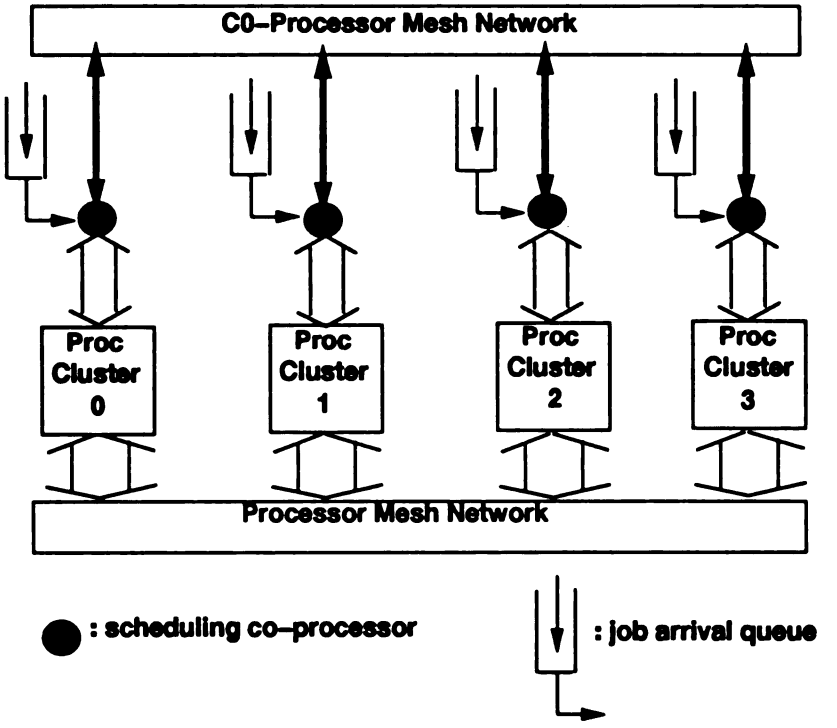


Figure 5.1. The system model of the mesh connected multiprocessor

ter. Communication support for real-time wormhole networks to deliver messages in a timely fashion is addressed in Chapter 6 and 7. Sensors, which collect inputs, are attached to the scheduling coprocessors, *i.e.*, task groups arrive to scheduling coprocessors. Task groups are scheduled by the scheduling coprocessors to which they arrived.

Task groups arrive to scheduling coprocessors as independent Poisson processes. Each task group consists of a group of tasks with precedence relations. The precedence relations of the tasks can be represented as an acyclic graph as described in Section 4.1. The execution demands of tasks are normally distributed with mean μ and variance σ^2. The number of tasks in a task groups is drawn from an exponential distribution with parameter t. Each task group has a deadline, and the tightness of deadlines are decided by the deadline tightness factor (DTF). The deadline of a task group j is

computed as

$$deadline(j) = clock + demand(j) * (DTF(j) + 1),$$

where $deadline(j)$, $demand(j)$ and $DTF(j)$ are the deadline, total computation demand and DTF of task group j, respectively, and *clock* is the arrival time of task group j. Task deadlines, ESETs and LSETs of tasks are calculated as described in Section 4.1.

5.2 The Scheduling Algorithm

A new task group arrives at a scheduling coprocessor is scheduled in stages. In each stage, a scheduling subset is selected from the *ready list* for scheduling. The ready list of a task group, which is maintained by the scheduling coprocessor, contains the set of all the *ready* tasks. A task is ready if all of its successors are scheduled. Initially, the ready list of a task group consists of the sink tasks. A ready task is removed from the ready list if it is successfully scheduled. When a task is successfully scheduled, the predecessors of the tasks are notified. If all the successors of a task are scheduled, the task is ready and inserted to the ready list. The number of tasks in a scheduling subset must be less than or equal to the maximum scheduling subset size S. When selecting a scheduling subset, the task with the shortest *scheduling window* is chosen first. The scheduling window of a task is the period from the current time to its LSET, *i.e.*, the task must be scheduled and transferred to a processor for execution in its scheduling window. The rest of the scheduling subset are selected in the same manner until there are S tasks in the subset or all the tasks in the ready list are selected. The subset is first considered for local scheduling. If there are tasks in the subset which cannot be scheduled, these tasks are globally distributed to the other

clusters.

5.2.1 Local Scheduling

The scheduling subset is distributed to a preferred set of processors using the BB algorithm described in Section 4.4.3. The preferred set is selected from the local processors which have the lightest loads in the cluster. The number of processors in the preferred set is p $(= 2^k)$, which is the least number greater than or equal to the size of the scheduling subset. Because the processors in the preferred set are local to the scheduling coprocessor, the coprocessor has the perfect information of these processors. Therefore, the schedules generated by the BB algorithm have the exact execution times for the tasks rather than the approximated times as in Section 4.4.2.

5.2.2 Global Scheduling

The global scheduler of a cluster is invoked when some of the tasks in the scheduling subset cannot be locally scheduled. During the global scheduling phase, the global scheduler first selects a remote cluster from the *preferred cluster list.* The construction of the preferred cluster lists is the same as the method described for the preferred lists in Section 4.3. The only difference between preferred lists and preferred cluster lists is that preferred cluster lists are ordered lists of clusters rather than processors as in preferred lists. The preferred cluster list of a cluster is ordered by the distances from the cluster to remote clusters. When selecting a remote cluster for task distribution, the clusters with distance 1 are first scanned. If there are available clusters within 1 hop, then the cluster with the lightest load among these clusters is selected. If there is no available cluster within distance 1, the clusters with distance 2 are scanned. This procedure repeats until one available cluster is found, or all the clusters in the list are scanned. In the case that there is an available cluster, a message is sent to that cluster

with all the information regarding the unscheduled tasks in the scheduling subset. When receiving such a message, the remote cluster tries to schedule these tasks to its local processors. If the tasks can be scheduled, then an *accepted* message containing the execution times for the tasks is sent to the cluster which globally distributes the tasks. Otherwise, a *rejected* message is sent to that coprocessor. Suppose there is no available cluster in the preferred cluster list, the cluster with the lightest load is chosen for task distribution. The same procedure of confirming the distribution is performed. If the tasks are successfully scheduled at a remote site, then the ready list is updated and the tasks are transferred to the remote site. Otherwise, a second cluster is selected from the preferred cluster list for global distribution. The process of global distribution terminates if either one remote site accepts these tasks or the task group is rejected. A task group can be rejected under two conditions: all the remote clusters reject the globally distributed tasks, or one of the LSETs of the tasks is reached. If a task group is rejected, then the already scheduled tasks are withdrawn from the scheduling lists of the processors at which they are scheduled.

5.3 Performance Analysis

A simulator is developed to study the performance of the scheduling algorithm. In addition to the task group rejection ratio, we study the relationship between the *local rejection ratio* and scheduling overhead. A task group is locally rejected if one of the tasks' LSETs is reached. We discuss the parameters that affect the local rejection ratio and the tradeoff of hardware costs and system performance.

5.3.1 The Simulation Model

The model of the simulated system is a 16x16 mesh connected multiprocessor with homogeneous processors. Each 4x4 submesh forms a cluster, and a scheduling co-

processor is associated with each cluster. The scheduling coprocessor and the 16 processors communicate through a local bus. A layer of the mesh network interconnects the scheduling coprocessors. The communication delay for transmitting one message is 1 unit time. The task transfer time is 2 units. It is reasonable to assume that all tasks submitted to a real-time system are known before their arrivals, so that we can store codes representing the tasks in each cluster. Thus, a task transfer means a message is sent with the task identification and the required input to initiate the task execution. Task groups arrive at scheduling coprocessor cp_i as a Poisson process with parameter λ_i.

The number of tasks in a task group is exponentially distributed with an average of 6 tasks. The task graph of a task group is randomly generated such that it is an acyclic graph. The computation demands of tasks are normally distributed with the mean of 100 units and the standard deviation as 1. The load of a processor is calculated as the ratio of total demands in the load window W to the length of W. The load window W is the period from current time to the next 10,000 unit times, which is a fixed window length. The two load thresholds T_l and T_h, which distinguish the state of a cluster, are 0.3 and 0.7, respectively. The maximum scheduling subset size is fixed to 4, which is a suitable number for the simulation model.

5.3.2 Relationship between The Rejection Ratio and Scheduling Overhead

The scheduling overhead of the algorithm directly relates to the unit scheduling time. The unit scheduling time directly relates to the scheduling overhead of a task group. The unit scheduling time is the overhead for assigning one task to one processor. We vary the unit scheduling time from 0 to 7 units and compare the performance of the system. When the unit scheduling time is zero, the scheduling overhead of

the algorithm is zero. As we increase the unit scheduling time, the overhead of scheduling one task group becomes higher as does the rejection ratio. Figure 5.2 is the simulation result. The average deadline tightness factor (DTF) of the experiment is 5. The rejection ratio of unit scheduling time 0 is the lowest among all the results, as

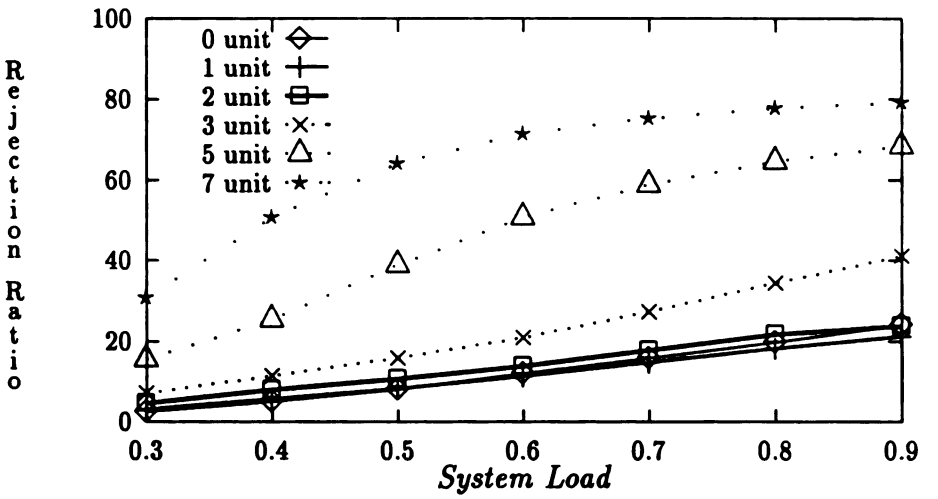


Figure 5.2. Task group Rejection Ratio for Different Unit Scheduling Time; DTF=5.

expected. Although unit scheduling time 1 and 2 units are rather large in comparison with the task processing demands, the scheduling algorithm can achieve a similar performance compared to the case of no scheduling overhead. The main reason for achieving this low rejection ratio is that the scheduling overhead is overlapped with the task processing. However, as the unit scheduling time increases linearly, the rejection ratio changes dramatically. If the unit scheduling time is greater than 2 units, the performance of the system becomes unacceptable.

5.3.3 Local Rejections

A task group is rejected if one of its tasks cannot be scheduled to any processor or one of the tasks' LSET is reached before its is successfully scheduled. In the latter case, the task group is locally rejected. We now focus on the question of the location of the scheduling bottleneck. Does it occur due to local rejections or remote rejections?

The local task group rejection ratios for unit scheduling times from 0–7 units are presented in Figure 5.3. The results show that when the unit scheduling time is greater than 1 unit, the scheduling overhead greatly increases the local rejection ratios. When the unit scheduling time is large, the scheduling overhead incurred

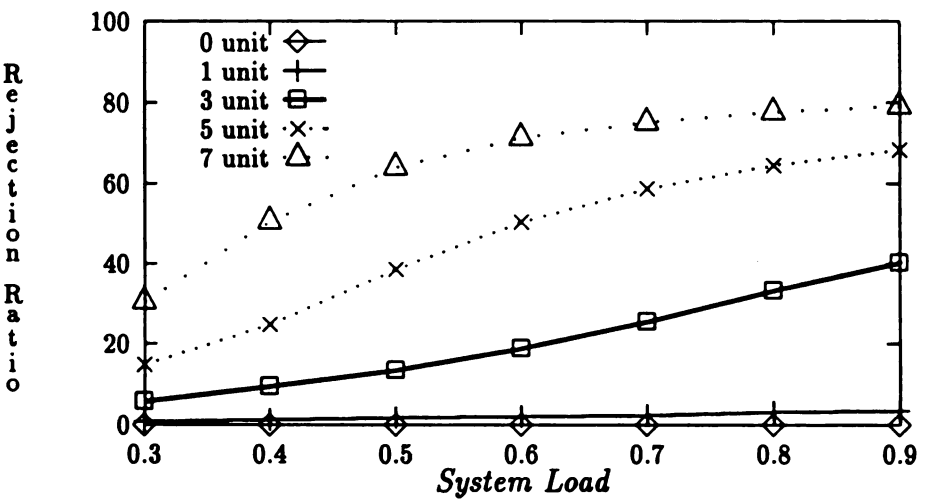


Figure 5.3. Comparisons of local rejection ratios with different unit scheduling time; DTF=5.

for scheduling a task group with large number of tasks becomes unacceptable. In the experiment, the number of tasks in a task group is drawn from an exponential distribution such that there is a certain portion of the task groups with a large number of tasks. These large task groups are rejected with very high probabilities. Also, the

effect of the scheduling overhead encountered by one large task group may propagate to the schedulability of the task groups that arrive later.

Rejection ratios increase as system load becomes higher; however, system load does not affect the local rejection ratios. Figure 5.4 shows the percentage of local rejections in all the rejected task groups. It is clear that system load is not a factor for local rejections. Instead, large unit scheduling time is the cause of the high local rejection ratios. In other words, when unit scheduling time is large, a large portion of the rejected task groups are rejected before they are completely scheduled. The

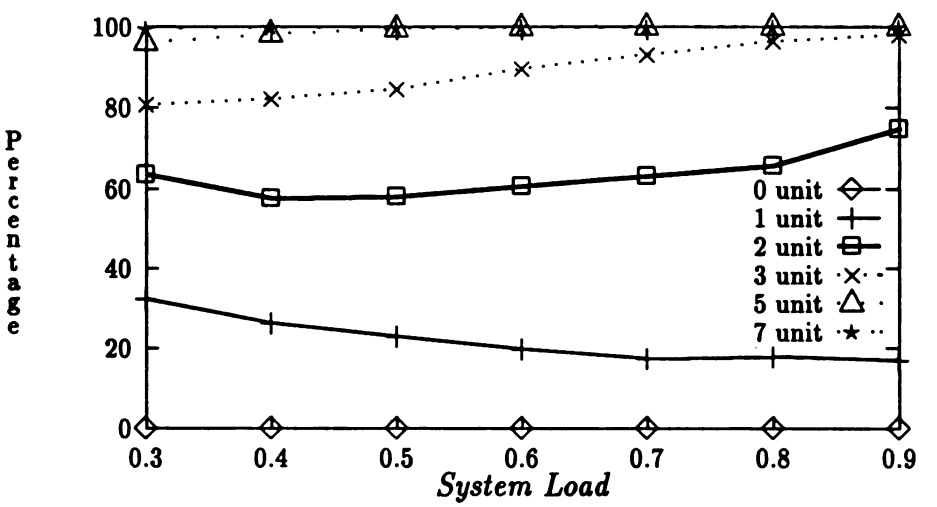


Figure 5.4. Percentage of local rejections; DTF=5.

unit scheduling time can be interpreted as the processing capacity of the scheduling coprocessor. One may choose to reduce hardware costs by using scheduling coprocessors that have less computing capacity than the task processors. Nevertheless, the coprocessors may become the bottleneck of the system if their computing capacity is too much smaller than the task processors. The overhead of our scheduling increases

linearly as the number of tasks in a task group increases. Unless the overhead of a scheduling algorithm increases less than this linear rate, the speed of the scheduling coprocessor is an important factor when designing a system.

Another parameter affects the efficiency of the algorithm is the tightness of the task group deadlines. In Figure 5.5, we compare the local rejection ratios for DTF of 2, 5, 10 and 100 with system load fixed at 0.8. If the task groups submitted to the system have loose deadlines, the scheduling window for each task is larger. As a result, the impact of scheduling overhead on the system performance is not so obvious when the task groups have loose deadlines. In addition, using fast scheduling

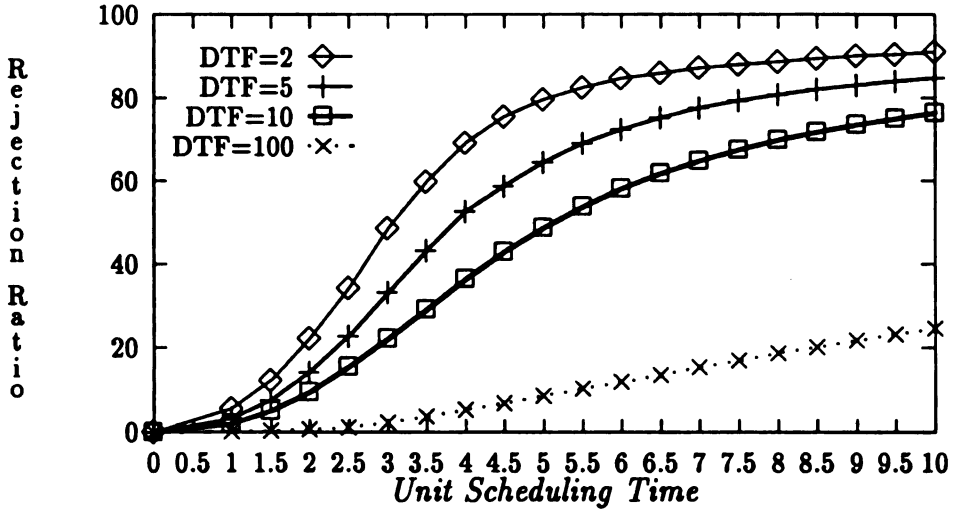


Figure 5.5. Local rejections ratios for DTF=2, 5, 10, 100; system load=0.8.

coprocessors can reduce the impact of the deadline tightness on the performance. As shown in Figure 5.5, when unit scheduling time is 1 unit, the rejection ratios for tight and loose deadlines are similar.

5.4 Summary

A dynamic scheduling algorithm for tasks with precedence constraints on mesh-connected multiprocessors is presented in this chapter. The scheduling algorithm partitions a group of precedence related tasks into subgroups of independent tasks and schedule each subgroup individually. Instead of dividing a task group by levels, the task group partitioning strategy selects tasks from the ready list to form the scheduling subgroups. This partitioning strategy enables the scheduling algorithm to serve models in which tasks have non-uniform demands.

The scheduling algorithm is different from the algorithm for distributed system described in the previous chapter. The processors are divided into clusters such that each cluster is a physically connected submesh. A scheduling coprocessor is responsible of scheduling tasks groups on the cluster. The BB algorithm is used for the local scheduling instead of the global scheduling. Tasks which cannot be locally scheduled are globally distributed using a simple task distribution scheme. The task distribution scheme selects remote clusters with lightest loads and shorter distances from the preferred cluster list.

The performance of the proposed scheduling algorithm is analyzed through simulation results. The results show that the algorithm performs very well even if the scheduling coprocessors are moderately slow. The reasons of the task rejections and the parameters that affect the system performance are discussed.

CHAPTER 6

Virtual Channel Flow Control for Static Traffic

A dynamic scheduling algorithm for parallel systems is described in Chapter 5. One problem that arises for parallel systems is the communication subsystem support for real-time requirements. A real-time communication subsystem must deliver messages in a timely fashion to guarantee the deadlines of task executions. This chapter addresses the issue of communication support for wormhole networks.

Most modern large-scale parallel multicomputers adopt direct networks as their communication subsystems. A direct network is a network with point-to-point interconnections among the processing nodes. Direct networks can be characterized by their interconnection topologies and switching mechanisms. In addition, the total communication bandwidth of a direct network increases as the number of nodes increases. The mesh, the hypercube and the tree are examples of topologies for interconnecting direct networks. The switching mechanism of a network determines the means of message buffering and transmission. Virtual-cut-through, circuit switching [81] and wormhole routing [15] are examples of switching mechanisms for parallel systems. The benefits of wormhole networks, which include simple router designs and small buffer sizes, make them attractive for large-scale parallel systems.

Virtual channels were proposed to provide efficient communication and deadlock free routing for multiprocessors [96]. The use of virtual channels in wormhole networks increases the throughput and utilization of network bandwidth [16]. Flow control in wormhole networks manages two types of resources: virtual channels and bandwidth of the physical links. In general purpose wormhole networks, first-come-first-serve and round-robin are the most common service strategies for virtual channel assignment and bandwidth allocation, respectively. Nevertheless, real-time systems require messages to be transmitted before their deadlines. The conventional flow control schemes cannot directly support real-time requirements.

Dally [16] suggested that decisions for flow control that are based on the timing properties of messages can improve the real-time performance. The performance of soft real-time systems is measured in terms of the deadline miss ratio. However, the capability to carry the timing property of a message in the message header requires large buffers that greatly increase the hardware complexity and the cost of a wormhole router. To provide real-time performance without complicating the wormhole router design, we need an efficient method to map the timing property of a message to a priority, which can be represented with a small number of digits, and a scheme to modify the priority to reflect the current timing property of the message.

Several real-time virtual channel flow control schemes for wormhole networks are presented in this chapter. The schemes differ in their *priority mapping strategies, arbitration functions* and *priority adjustment methods.* A priority mapping strategy assigns initial priorities to messages according to their timing properties. The priority of a message can be represented with a small number of digits and can be carried within the message header. An arbitration function determines the allocation of network bandwidth to virtual channels. A priority adjustment method modifies the initial priority of a message according to the current timing property of the message and increases the probability for meeting the message deadline. Therefore, an analysis

of the schemes indicates the relative value of the arbitration functions, the priority mapping strategies, and the priority adjustment schemes to the performance of real-time flow control schemes.

Since an appropriate model for communication traffic depends on the real-time application, we choose two communication models, static and dynamic, in order to evaluate a wide range of workloads for the proposed schemes. A large number of the existing real-time systems are operating in static environment, a model of static communication traffic is necessary to study the performance of the proposed flow control schemes. The static communication model for our study is the *linear bounded arrival process* [82, 86, 93]. In the linear bounded arrival process model, messages are periodically generated by the message sources and the message sources are statically allocated to the nodes. The linear bounded arrival process serves as the traffic model of an environment where task executions or data sampling are periodic. The performance of the flow control schemes for serving dynamic communication traffic is discussed in the next chapter.

Although analytical modeling is important for providing insight to the performance of new flow control schemes, it is very difficult to model the system that we study analytically. It is even difficult to construct an analytical model for a general purpose wormhole network with virtual channels that does not consider real-time. Simulation is necessary to analyze the performance of the flow control schemes. We implemented a simulator, which simulates the network at the flit level to study the detailed behavior of the messages, for analyzing the performance of the flow control schemes and the message dropping method. The evaluation of a flow control scheme must consider both the performance and the cost of the hardware implementation to support the scheme. Simulation experiments were designed to compare and to evaluate the relative importance of the arbitration function, the priority mapping scheme, and the method for priority adjustment.

As expected, simulation results show that a priority mapping strategy is crucial to the system performance. A good priority mapping strategy must assign priorities to messages according to their timing properties, *e.g.*, deadline tightness and laxity before deadline. Nevertheless, a priority based arbitration function can greatly improve the message deadline guarantee ratio in comparison to a conventional round-robin arbitration scheme that is normally implemented in a wormhole network arbitrator. In addition, we find that a priority adjustment method can improve the manner that the flow control scheme of the network adapts to the state of the load of the network. There is a chance that a low priority message can be blocked by high priority messages for an indefinite period. A priority adjustment method modifies the priorities as messages are blocked. Therefore, a low priority message will not be inhibited inappropriately from meeting its deadline in case there is a large burst of high priority traffic.

6.1 Wormhole Networks and Virtual Channels

An increasing number of real-time applications are developed on parallel systems, and most new parallel systems adopt wormhole networks as their communication subsystems. In a wormhole network, processing nodes are interconnected by a point-to-point network and communicate by transmitting messages. Real-time tasks executed on parallel systems exchange information and share data to achieve a common goal. Due to the timing constraints of task executions in a real-time system, messages must be delivered before their deadlines. Otherwise, tasks may violate their timing constraints and cause severe consequences. A wormhole network with virtual channels, which has low network latency and can support priority based flow control, is an attractive system for real-time applications.

The flow control of a wormhole network operates at the flit level in which flits are

the basic units for buffering. A flit is the smallest unit of information that a queue or a channel can handle. The size of a flit is usually one or two bytes. Only a few flits are buffered at each node; therefore, the sizes of buffers required in a node are small. Wormhole routing transmits messages in a pipelined fashion. A message is divided into flits. When the header flits of a message are received by a node, the node decides the next node the message should be routed based on the information contained in the message header. As soon as the receiving buffer of the next node is available, the flits are forwarded to that node. As header flits are forwarded, the subsequent flits follow one hop from the downstream nodes. In other words, the message uses the buffers along the path between the source node and the destination node. The network latency of a message can be calculated as $(L_h/Bw)D + L/Bw$ [97], where L_h, Bw, D and L are the length of a message header, the physical link bandwidth, the distance between the source and the destination nodes, and the message length, respectively. When $L_h \ll L$, the network latency is mostly determined by L/Bw, which is insensitive to the distance between the source node and the destination node. The advantages of wormhole routing, which include small buffer sizes and low network latency, make it the most promising switching technique for massively parallel systems.

Physical links in direct networks are precious resources that are usually under utilized. Virtual channels have been proposed to reduce network contention and to improve physical link utilization. The virtual channels of a physical link require extra flit buffers and an arbitration scheme to share the link bandwidth among the buffers. The cost of virtual channels in a wormhole routed network is low since the number and the size of buffers are small. In contrast, virtual channels in other networks are expensive since large buffers are required. Virtual channels provide a different view in comparison to the physical network since several logical networks can be extracted from the physical network. Hence, adaptive deadlock-free routing algorithms that

require multiple channels between two neighboring nodes are feasible to implement on virtual channels [96, 98]. In addition, virtual channels increase the degree of fault tolerance and connectivity of a network. For example, in the case of channel or node failures, a message can be rerouted around the faulty region, which is difficult in a network without virtual channels. However, network latency is slightly increased by sharing bandwidth among the virtual channels.

6.2 Flow Control

Messages in a wormhole routed network compete for two types of resources: virtual channels and the bandwidth of physical links. The two components of a flow control scheme that manage these two types of resources are the virtual channel assignment strategy and the arbitration function. A virtual channel assignment strategy decides which arrival flits can use the virtual channels. Wormhole routing transmits flits in a pipelined fashion and only the header flits of a message carry the routing information. Thus, after the header flits of a message are transmitted to the next out-going virtual channel, the subsequent flits must follow on the same channel. In other words, once a message is assigned to a virtual channel, it occupies the virtual channel until the last flit of the message leaves the node. The arbitration function of a physical link determines the bandwidth sharing strategy, *i.e.*, an arbitration function selects the next flits to be sent on the physical channels. The relationship of the flow control components and the resources of a physical link is illustrated in Figure 6.1.

We study a range of flow control schemes that differ in their means of arbitration, priority mapping and priority adjustment. We will evaluate the effect of each component towards the overall performance. Also, we will discuss the hardware complexities required to provide such schemes. The priority mapping scheme decides the initial priority of a message. The priority of a message can be used for both virtual channel

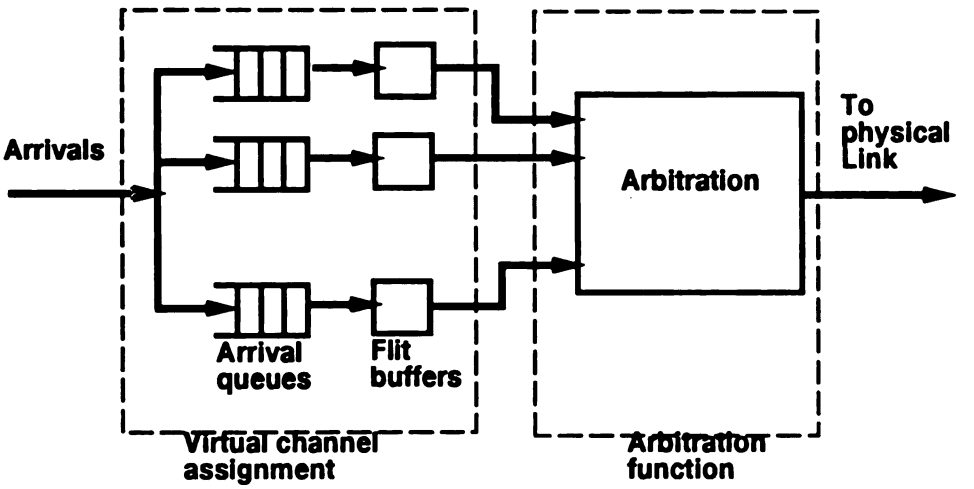


Figure 6.1. Flow control components and the resources of a physical link.

assignment and arbitration. A priority adjustment scheme varies the initial priority according to the current status of a message. Table 6.1 is a brief summary of the seven schemes discussed in this chapter. A *first-come-first-served* scheme (FCFS) is the conventional flow control scheme implemented in general purpose wormhole networks. A *tightest-deadline-first* scheme (TDF) and a *least-laxity-first* scheme (LLF) map different timing properties of messages to their priorities. The TDF and LLF requires n priority bits in a message header to encode the priority.* The strategy for priority assignment in the TDF scheme is *earliest-deadline-first* (EDF). The study in [99] showed that the EDF strategy is a good strategy for dynamic real-time task scheduling. Mok and Dertouzos [13] showed the effectiveness of the LLF strategy for dynamic scheduling of real-time tasks. A *priority-climbing* scheme (PC) adjusts the priority of a message by counting the number of times the message header is blocked from accessing an outgoing link. An *enhanced priority climbing* scheme (EPC) en-

*Suppose we have N ($= 2^n$) virtual channels, n is the number of bits that required to represent all the virtual channels

hances the PC scheme by means of a priority based arbitration function. In addition to the n priority bits, the PC and EPC scheme use the remaining bits of a flit for counting the number of times the message is blocked. A *rate-monotonic-scheduling* scheme (RMS) is a variation of the static task scheduling algorithm [17], and an *one-bit* scheme is a modification of the RMS scheme. The RMS scheme requires g priority bits to represent the global priority.[†] The one-bit scheme needs one bit to indicate if the message is normal or early. Although it is expected that schemes for real-time virtual channel flow control would be based on scheduling algorithms such as earliest deadline first, least laxity first, and rate monotonic scheduling, the challenge is to provide flow control schemes that operate efficiently with little modification to existing wormhole routing technologies. The following discussion presents details of each scheme.

Table 6.1. A summary of the flow control schemes.

Scheme	Priority Mapping	Arbitration	Priority Adjustment	Priority bits
FCFS	None	Round-robin	None	None
TDF	Deadline tightness	Round-robin	None	n
LLF	Laxity	Round-robin	None	n
PC	TDF or LLF	Round-robin	Monotonically increased as a function of time	one flit
EPC	TDF or LLF	Priority based	Monotonically increased as a function of time	one flit
RMS	Message frequency	Priority based	Global adjustment for burst arrivals	g
One-Bit	Message frequency	Priority based	Local adjustment for burst arrivals	1

[†]The number g is the number of bits required to represent all the message sources.

6.2.1 Conventional Wormhole Network Flow Control — First Come First Served

The FCFS scheme assigns a virtual channel to the message with the earliest arrival time in the waiting queue. The arbitrator of a physical link polls its virtual channels in a round-robin fashion. The physical link can be allocated to a virtual channel if there is a flit in the buffer and there are free flit buffers in the direction of next node. The FCFS scheme is used in general purpose wormhole networks. Therefore, no extra hardware is needed for serving real-time traffic. Dally's investigation [16] showed that increasing the number of virtual channels using the FCFS scheme in a conventional wormhole network can reduce the message contention problem and increase bandwidth utilization. He also showed that an *oldest-packet-first* arbitration function further improves the performance from the round-robin scheme.[‡]

6.2.2 Priority Mapping Schemes Based on Message Generation Times

If flow control decisions are based on the time that messages are generated, the header of a message must carry several extra bytes that encode the message generation time. Therefore, the buffer size of virtual channels becomes large. On the other hand, the hardware for maintaining correct timing information in the header flits may complicate the router design. Since a requirement to carry the message generation time in the header flits may introduce substantial hardware cost, an uncomplicated yet powerful time encoding technique is needed. We propose two priority assignment schemes (TDF and LLF) that embed the timing property of a message in a few priority bits.

[‡]An oldest-packet-first scheme allocates a physical link to the virtual channel that holds the oldest flit in the node.

Tightest Deadline First

The TDF algorithm assigns priorities to messages according to their deadline tightness factors (DTF). The DTF of a message $m_i(k)$, which is generated by message source k, is calculated as

$$DTF(m_i(k)) \;=\; Deadline(m_i(k)) \;/\; Latency(m_i(k)),$$

where $DTF(m_i(k))$, $Deadline(m_i(k))$ and $Latency(m_i(k))$ are the DTF, time before the message deadline and network latency of $m_i(k)$, respectively. The network latency of a message is calculated as

$$m_i(k) \;=\; 3 * C * D \;+\; C * L,$$

where C is the time needed to transmit one flit from a node to a neighboring node. The value of C depends on the router implementation, and a typical value for C is 20ns [100]. D is the distance from the source node to the destination node. L is the number of body flits. Note that a message header consists of three flits, x distance, y distance and the message priority. Thus the first term on the right hand side of the formula counts the end-to-end delay of three flits. For a link with 2^n virtual channels, the DTF of a message is mapped to a priority p, where $1 \leq p \leq 2^n$. The function for mapping the DTF of message to the priority p is as follows:

$$p = \begin{cases} NumVC + 1, & \text{if } DTF(m_i(k)) < 1 \\ NumVC + 1 - DTF(m_i(k)), & \text{if } 1 \leq DTF(m_i(k)) \leq NumVC + 1 \\ 1, & \text{if } DTF(m_i(k)) > NumVC + 1, \end{cases}$$

where $NumVC$ is the number of virtual channel of a physical link. The priority of a message is encoded into n bits and carried by the message header. Based on the

priority, a message can request to be allocated any virtual channel number that is lower than its priority. For example, if the priority of a message is 2 and there are 4 virtual channels, then the message can request to be allocated either virtual channel 0 or 1. Note that the virtual channels of a physical link are assumed to be numbered from 0 to 3. The higher the priority of a message, the larger number of virtual channels from which a message can request an allocation. Thus, the probability of meeting deadlines for messages with tight deadlines is increased. The arbitration function is the round-robin scheme. The TDF scheme needs hardware support for the priority based channel assignment.

Least Laxity First

The LLF scheme takes a different priority mapping approach than that of the TDF scheme. The priority assignment of a message is based on its laxity rather than the DTF as in the TDF scheme. The laxity of a message $m_i(k)$ is the delivery time remaining for the message, and it can be estimated as

$$Lx(m_i(k)) \;=\; Deadline(m_i(k)) \;-\; Latency(m_i(k)),$$

where $Lx(m_i(k)$ is the laxity of the message. We assign higher priorities to the messages with less laxity. The function of the priority mapping is the following,

$$p = \begin{cases} NumVC + 1, & \text{if } lx(m_i(k)) < 1 \\ NumVC + 1 - lx(m_i(k)), & \text{if } 1 \le lx(m_i(k)) \le NumVC + 1 \\ 1, & \text{if } lx(m_i(k)) > NumVC + 1, \end{cases}$$

where

$$lx(m_i(k)) = Lx(m_i(k))/W \tag{6.1}$$

and W is a system dependent constant. The choice of W determines the number of messages in each priority level. A larger W assigns more messages at each priority level. Suppose the number of messages that have short laxity is large in a system. A large W should be used to assign high priorities to these messages. Note that the number of priority levels is the same as the number of virtual channels associated with a physical link, and the priority of a message is encoded into n bits to be carried by the message header. The virtual channel assignment scheme and the arbitration function for the LLF scheme are the same as those of the TDF scheme. The hardware support for the LLF scheme is also the same as the TDF scheme.

6.2.3 Dynamic Priority Adjustment Schemes — Priority Climbing

In a priority based flow control system, there is a chance that a burst of high priority messages will block low priority messages for an indefinite long time. As a result, a large number of low priority messages may miss their deadlines. In the LLF and TDF schemes, priorities are assigned to messages according to their initial timing properties. When messages are inserted to the network, their timing properties change constantly. For example, the laxity of a message becomes shorter. A priority adjustment scheme is needed to maintain fairness among messages and modify the priorities to reflect the current timing properties.

A priority adjustment scheme, which requires nodes exchanging messages to modify message priorities, increases hardware complexity. The PC scheme dynamically adjust priorities by utilizing the local information available to messages at each node. The scheme increases the priority of a message in relation to the number of times the header flit of a message is blocked. Both the TDF and the LLF schemes can be used to assign the initial priority of a message. Suppose 2^n virtual channels are associated

with

the

and

are

The

the

of t

the

the

in t

it ca

0. V

is in

the

ered

Sinc

a me

[§]A

that a

with each physical link, we need n bits to represent all priority levels. To implement the PC scheme, each message header carries a one-byte (or one-flit) priority value, and the n most significant (leftmost) bits represent the priority. Since only a few bits are required to represent all the priority values, one byte is sufficient for the scheme. The $8-n$ bits, called the *blocked count* (BC), are used for counting the blocking time the message encountered. The initial value of BC is zero. Figure 6.2 is an example of the priority and the BC carried by a message header. The arbitration method for

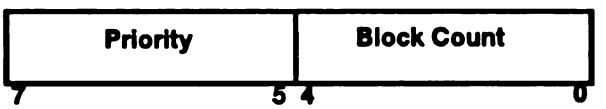


Figure 6.2. An 8-bit priority and block count carried by a message header.

the scheme is the round-robin method. When all the virtual channels are polled once, the arbitrator of a link increments the BCs of the message headers that are blocked in the flit buffers of the link by one. If the value of a BC exceeds the maximum value it can represent, the priority of the message is increased by one and the BC is reset to 0. When transmitting a message, as the laxity of the message is reduced, its priority is increased. The virtual channel assignment strategy and the arbitration function of the PC are the same as those of TDF and LLF.

The rate of updating the BC of a message is a design issue that should be considered. Suppose each physical link has 2^n virtual channels, the size of a BC is $8-n$. Since the polling of all the virtual channels can be done in one cycle, the priority of a message is incremented by one after the message is blocked for 2^{8-n} cycles.[§] For a

[§]A cycle is the time needed to transmit one flit from one node to a neighboring node provided that all the resources are available.

message with the lowest priority, the priority of the message becomes the highest one after 2^7 blocking cycles. Suppose a wormhole network employs fast routers that the network cycle is short, the priority updating rate might be too fast such that a large number of messages have the highest priority when the network is loaded. Thus, the performance of the PC scheme degrades to that of the FCFS scheme. In order to prevent the performance degradation, a counter can be used to decrease the priority updating rate. For example, a counter counts 4 cycles before incrementing the BCs and the priority updating rate becomes a quarter of the original rate without using the counter.

6.2.4 Priority Based Arbitration — Enhanced Priority Climbing

In the PC scheme, the priority of a message determines the number of virtual channels from which one can be allocated to a message in a node, and the arbitration function is round-robin. To improve the PC scheme, the EPC scheme uses the priority of a message differently to make virtual channel assignments and bandwidth allocation decisions. When more than one message competes for an available virtual channel, the virtual channel is assigned to the message with the highest priority. If there is a tie, the channel is assigned to the message with the earliest arrival time. The arbitrator of a physical link also uses the priorities of messages assigned to the virtual channels to make the bandwidth allocation decision. The bandwidth of the link is allocated to the highest priority message. In the FCFS, TDF, LLF and PC schemes, the arbitration function is the round-robin scheme and deadlock is prevented by the underlying routing algorithm. Although the EPC scheme uses a priority based arbitration, as long as the routing algorithm is deadlock free that the EPC scheme will not cause deadlock. Figure 6.3 is an example of the virtual channel assignment

and the bandwidth allocation for the EPC scheme. In the example, three header flits are competing to be allocated to an outgoing virtual channel VC2. Since Msg 3 has the highest priority 4, VC2 is assigned to Msg 3. Among all the virtual channels, VC0, VC1 and VC2 have available flits in their buffers. The priority, 4, of VC2 is the highest among the three virtual channels; therefore, the bandwidth is allocated to VC2 and Msg 3 is transmitted on the physical link. The EPC scheme further reduces

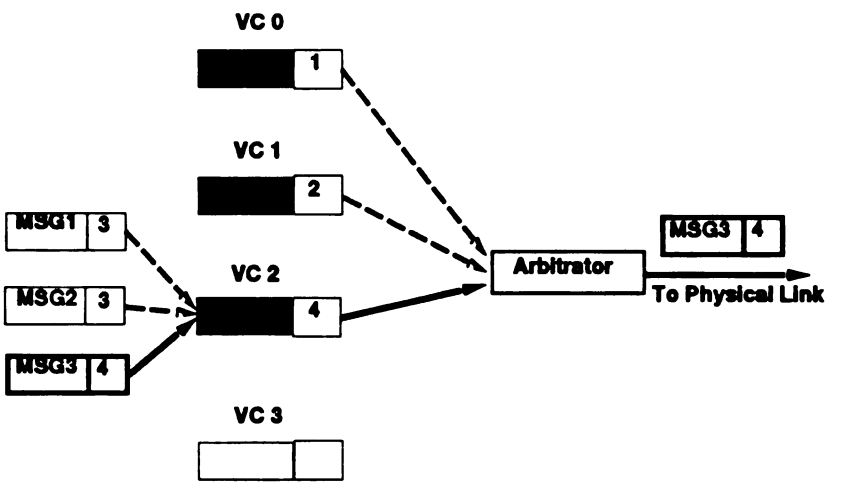


Figure 6.3. An example of the virtual channel assignment and arbitration decisions in the EPC scheme.

the chance of being blocked in the PC scheme for high priority messages. The cost of the EPC scheme is the hardware to support the priority based virtual channel assignment and arbitration function.

6.2.5 Frequency-Based Priority Mapping and Arbitration — RMS

Since a large number of real-time applications are executed at regular intervals, it is suitable to consider that the traffic generated by these types of applications to be periodic. The RMS algorithm was proposed by Liu and Layland [17] for scheduling periodic tasks. The algorithm assigns higher priorities to higher execution frequency tasks. In the RMS algorithm, the priorities of tasks are fixed during their execution. The deadline of a task is assumed to be at the end of its period. The RMS algorithm has been proven to be an optimal algorithm for a restricted class of tasks that are preemptively executed on a uniprocessor. Mutka [92] proposed a real-time message scheduling algorithm for wormhole networks that uses the RMS algorithm to schedule periodic messages with burst arrivals.

In the RMS algorithm, a set of message sources periodically generates messages. The algorithm first allocates the message sources to the processing nodes in the system. The RMS algorithm then assigns global priorities to message sources according to their periods. The virtual channels on the path of a message source are statically assigned to the message source. For a given physical link in the network, a number of different message sources may be routed over the same physical link. The RMS algorithm analyzes the feasibility of the message source allocations, and assigns higher numbered virtual channels to higher priority sources. The arbitrator of the physical link allocates the bandwidth to the virtual channel that has a flit waiting in the buffer with the highest priority. The messages generated by a message source inherit the priority of the source. The priority of the message is encoded into g bits, where g is the number of bits required to represent all the global priority levels, and carried by the message header. The deadline of a message is at the end of its period. If a message is generated earlier than its period, the deadline of the message is calculated

as the normal period length plus the time before its expected generation time. For example, suppose the expected message generation time is t and the deadline of the message is $t + d$, where d is the length of the period. If a burst message is generated at t' ($t' < t$), the deadline for the burst message is $t + d$ rather than $t' + d$. The priority of the message is re-assigned to the priority of the message source that shares the same path with the message and has the shortest deadline that is larger than the burst message deadline. The global priority adjustment scheme ensures that a burst message will not block the normal messages generated by lower priority message sources. In order to support the RMS scheme, each virtual channel in the network needs a priority register to store the priority carried by a message header. The arbitrator, which is designed using simple combination logic, must include support for the highest-priority-first scheme. Also, each node needs a static table to save the deadlines of all the message sources routed through the node.

6.2.6 Reducing the Cost of the RMS Scheme — The One Bit Approach

The costs for supporting the RMS scheme include several priority bits in the header of a message and a table at each node that is used for determining the priority adjustment of burst messages. These costs caused us to design a simplified version of the scheme. The one-bit scheme is the same as the RMS scheme with the exceptions of the priority assignment and the adjustment scheme. Instead of carrying a multiple-bit priority as in the RMS algorithm, the one-bit approach carries one bit to indicate whether it is generated earlier than its period. The priority of a message at a node is the outgoing virtual channel number to which the message is assigned. The arbitrator of a link allocates the bandwidth to the highest numbered virtual channel with a flit waiting in the buffer. The priority of an early message is locally adjusted at each

node on the path. When an early message requests a virtual channel in a node, the message is assigned to an available virtual channel with a lower number than the channel number that is assigned to its message source. Since the priority of a burst message is locally adjusted at each node, the one-bit scheme does not need a static table in each node to store the deadlines of all the message sources routed through the node. Also, the size of the priority of each message is reduced to one bit.

We have described each of the schemes for managing real-time communication in a wormhole network. We must evaluate the schemes to determine the benefits of the different priority mapping strategies, priority adjustment methods and arbitration functions. The next section presents an evaluation of the schemes on the linear bounded arrival process traffic model.

6.3 The Simulation Model

We implemented a simulator in CSIM [101] to study the performance of different flow control schemes. Since constructing an analytic model of the system that we study is very difficult and complicated, even for a general purpose system that does not consider real-time, a simulation study is necessary to analyze the performance of the proposed flow control schemes. The simulator was implemented such that message sources concurrently generate messages. The simulation was performed at the flit level to study the detailed behavior of individual flits. Flits are individual processes in the simulation that compete for resources in the network.

The simulated system is modeled as a 2-D mesh, which is similar to the topology of an iWarp machine [102], and wormhole routing is the switching mechanism. The dimension of the mesh is 10x10. Each node of the mesh has a pair of in-coming and out-going physical links in each direction, and each physical link can support 2^n virtual channels, where $1 \leq n \leq 3$. The bandwidth of a physical link is 8 *flits/time unit*. The

flit

for

rou

dea

coo

the

6.3

Sinc

it is

add

that

type

prop

in t

of r

term

used

of t

mess

of so

mess

mess

The

that

by a

averag

flit buffer of a virtual channel can hold 8 flits. Each physical link has an arbitrator for multiplexing among the virtual channels associated with the link. Messages are routed using the *X-Y* routing (or dimension routing), which is a deterministic and deadlock-free routing scheme. The X-Y routing first routes a message to the same coordinate on the X-dimension as the destination, and then the message is routed on the Y-dimension to the destination.

6.3.1 The Model of Communication Traffic

Since a large number of real-time applications have tasks executed at regular intervals, it is appropriate to model the messages generated by the tasks as periodic traffic. In addition to the periodic traffic, a portion of the messages may be generated in bursts that violate the periods. A model that has often been useful for describing this type of real-time communications, called the *linear bounded arrival process*, was first proposed by Cruz [93]. Anderson *et al.* [82, 84] used this model for continuous media in the DASH system [83] and Kandlur *et al.* [86] used this model to solve problems of real-time communication in multi-hop networks. With some modifications, the terminology we use for the model is taken from Anderson *et al.* [82] and has also been used by Kandlur *et al.* [86] and Mutka [92]. Table 6.2 is a description of the parameters of the communication model. We assume there are M different sources of real-time messages that can be placed at any processing node in the system. The arrival process of source k has three important parameters: maximum message size ($S(k)$), maximum message rate ($R(k)$), and maximum burst rate ($B(k)$), where $1 \leq k \leq M$. From the message rate, we define the minimum period between messages to be $I(k) = 1/R(k)$. The burst parameter $B(k)$ allows processes to generate short-term bursts of traffic that violate the long-term average data rate. The number of messages generated by a source cannot exceed $(B(k) + R(k)) * t$ for any time t, where $t > 0$. The average data rate is $S(k) * R(k)$. The burst parameter is significant for networks that

Table 6.2. Parameters of the linear bounded arrival process.

Parameter	Description
M	Number of message sources
$R(k)$	Maximum message rate of message source k
$S(k)$	Maximum message size of message source k
$B(k)$	Maximum burst rate of message source k
$I(K)$	Minimum period of message source k
Bw	The bandwidth of a physical link
$l(m_i(k))$	The logical generation time of the ith message generated by message source k
$d_i(k)$	The deadline of $m_i(k)$

perform internal message buffering since it specifies the number of buffers needed at intermediate nodes along a message path. The maximum size of the burst parameter is not significant for wormhole networks since they do not buffer entire messages internally, but rather a small fraction of a message at any time.

The ith message generated by a source k is labeled $m_i(k)$. This message is physically generated at time t_i, but has associated with it a *logical generation time,* $l(m_i(k))$. The logical generation time of the $0th$ message is defined as $l(m_0(k)) = t_0$, the actual generation time. The logical generation times of all subsequent messages occur at

$$l(m_i(k)) = max(\ l(m_{i-1}(k)) + I(k),\ t_i\ ),\quad i > 0. \tag{6.2}$$

The communication system must guarantee that any message $m_i(k)$ with logical generation time $l(m_i(k))$ will be delivered to its destination by $l(m_i(k)) + d_i(k)$, where $d_i(k)$ is the deadline for the message. The deadline $d_i(k)$ will be less than or equal to the period of the communication pattern between the source and the destination.

6.3.2 Message Source Generation

The message sources are generated off-line. A message source k is generated by first randomly selecting source and destination nodes. Next, to generate a range of short and long messages, a message size $S(k)$ is randomly selected from four sizes, 96, 200, 296 and 400 flits. A message rate $R(k)$ is uniformly drawn from the range $(0, Mr(NumVC))$, where $Mr(NumVC)$ is the maximum message rates for $NumVC$ virtual channels. The maximum message rate $Mr(NumVC)$ is calculated as

$$Mr(NumVC) = 2 * Bw/(S(k) * NumVC),$$

where Bw is the bandwidth of the link. After a message source is generated, the feasibility of the message source is tested using a RMS test at each node of its path. Suppose there are k message sources that have their paths overlapped on a physical link with N virtual channels, where $k \leq N$. The RMS test is the following:

$$\sum_{i=1}^{k}(R(i) * S(i))/Bw) \quad \leq \quad N(2^{1/N} - 1). \tag{6.3}$$

Since a virtual channel can only be assigned to one message source, the number of message sources that have their paths overlapped on a physical link is limited by the number of virtual channels. The RMS inequality was provided by Liu and Layland [17]. They showed that $N(2^{1/N} - 1)$ is the least upper bound of the utilization for N periodic tasks scheduled by the RMS algorithm. As N becomes large, $N(2^{1/N} - 1) \rightarrow \ln 2 = 0.69$. If the inequality is satisfied, the message source is included in the current set. Otherwise, it is rejected. The message source generation is terminated if the current set can utilize the network bandwidth for at least 95% of the RMS bound or if 98% of the physical links have all their virtual channels assigned. Appendix A describes a tool, DRMS, that can perform feasibility tests and message

source allocation for periodic communication traffic such as the linear bounded arrival process described in this section.

6.4 Performance Analysis

We conducted experiments to study the three key components that relate to the performance of real-time flow control: arbitration, priority assignment and priority adjustment. The performance of a flow control scheme was measured by the message delayed ratio. The message delayed ratio is the percentage of number of messages that miss their deadlines to the total number of messages. The simulation results presented in this chapter are from a network with 8 virtual channels per physical link unless specified. The simulation results were collected such that the system load was varied from 0.7 to 0.875, where the load for a fully utilized network was 1.0. When system load was 0.7, the traffic was generated periodically by the message sources without burst arrivals and it was close to the RMS bound, which is computed from Equation 6.3 to be $8(2^{1/8} - 1) = 0.72$ for 8 virtual channels. The system load was increased by injecting burst arrivals. At a system load of 0.875, the burst rate ($B(k)$) of a message source k is 25% of the maximum message rate ($R(k)$).

6.4.1 Number of Virtual Channels

Dally [16] showed that increasing the number of virtual channels per physical link can improve the network utilization and reduce the contention problem in a conventional wormhole network using the FCFS scheme. Our study verifies that the effect of adding more virtual channels to a physical link is essential to the real-time performance measure. Figure 6.4 compares the performance of the FCFS scheme using 2 to 8 virtual channels. The results indicate that increasing the number of virtual channels per physical link can significantly reduce the delayed ratio. The main reason for the

effect is that the number of virtual channels is directly related to the probability of blocking a message. As the number of virtual channels of a link increases, the probability of a message being blocked is reduced.

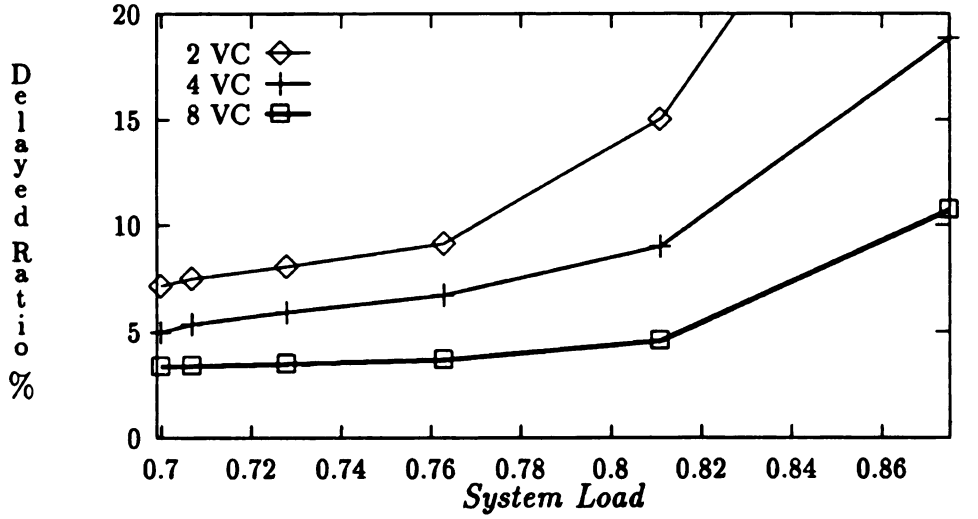


Figure 6.4. Performance of FCFS scheme with different number of virtual channels.

6.4.2 Priority Mapping

Although increasing the number of virtual channels can reduce the delayed ratio, assigning priorities to messages according to their timing properties can further improve the performance. Figure 6.5 shows the performance comparison of the FCFS, the RMS, the one-bit, the TDF and the LLF schemes. As expected, the RMS scheme has the best performance and the FCFS scheme has the worst performance among the five schemes. The RMS scheme proposed by Mutka [92] guarantees all the periodic messages will meet their deadlines when there are no burst arrivals. In order to

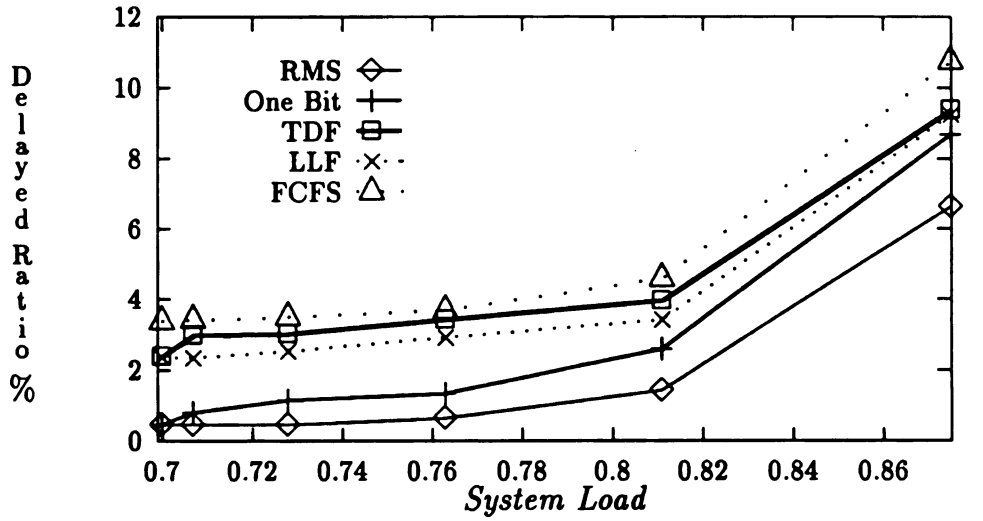


Figure 6.5. Comparison of priority mapping schemes.

simplify our discussion, the blocking factor[¶] is neglected in our RMS test for allocating the message sources as is done in [92]. Nevertheless, we do simulate the message blocking factor for all the schemes that we study. Therefore, the delayed ratio of the RMS scheme at system load 0.7 was caused by the blocking factor. The one-bit scheme has similar performance to the RMS scheme when the burst rate is small. As burst rate increases, the performance of the scheme declines. The large amount of burst traffic saturates the lower numbered virtual channels and cause low priority and burst messages to miss their deadlines. Both the TDF and LLF schemes improve the performance of the FCFS scheme. The LLF scheme outperforms the TDF scheme by a small margin, which is due to the choice of the parameter W that is used in the LLF scheme to determine the number of message sources at each priority level. W $(= NumVC/2)$ is a function of the number of virtual channels for the results we

[¶]The blocking factor is the blocking time due to the preemption of low priority messages when high priority messages are ready for transmission.

present in this chapter. Therefore, the laxity of a message used in the LLF scheme is a better timing property for priority mapping than the TDF. By limiting the number of virtual channels a message can request, the two schemes reduce the delayed ratio without the substantial hardware cost of the RMS scheme.

6.4.3 Priority Adjustment

A priority mapping scheme assigns a priority to a message according to the initial timing property of the message. However, the initial timing property may change such that the initial priority cannot represent the current status of the message. In the RMS scheme, a message inherits the priority of the message source that generates the message. In the case that a message arrives earlier than the period, the priority is globally adjusted and lowered to prevent the unnecessary blocking of normal low priority traffic. The initial priority of a message in the TDF and the LLF schemes reflects the time before the message deadline. As a message is transmitted in the network, the timing property changes. The PC scheme adjusts the initial priority by counting the number of times a message is blocked in the network. Figure 6.6 shows the performance improvement using the PC scheme in comparison with the RMS scheme. The global priority adjustment of RMS scheme provides stable performance until the system load is close to the saturation point. The PC scheme increases the probability of meeting deadlines for low priority messages and reduces the delayed ratio for both the TDF and the LLF schemes. Although the RMS scheme performs well, the global priority adjustment needs a static deadline table in each node for storing the deadlines of all the message sources routed through the node. Since buffers are the most expensive resources, a system designer should consider the hardware cost to support the RMS scheme. In contrast, the PC scheme only requires a block count in each message header and an adder in each arbitrator to increment the block counts. Since the TDF and the LLF schemes require that a message header carries a priority

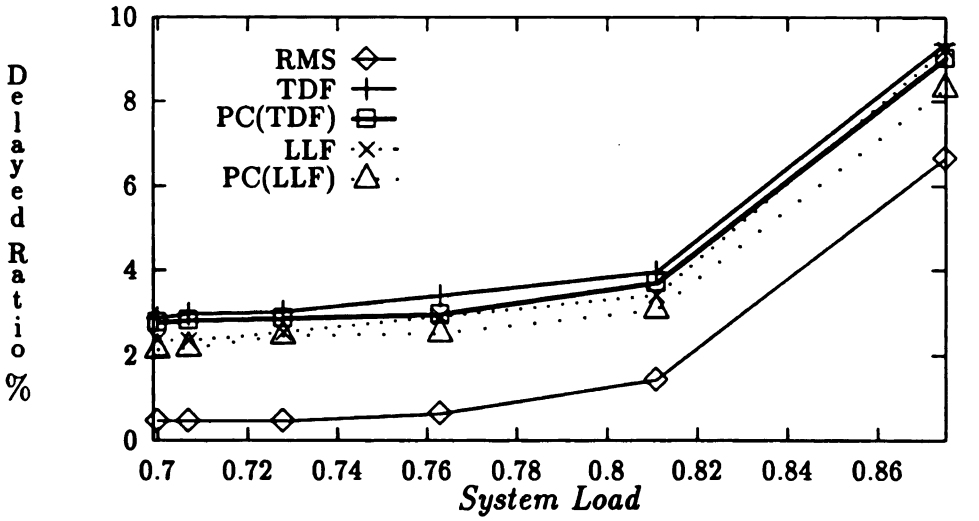


Figure 6.6. Comparison of priority adjustment schemes.

value, we can integrate the priority and the block count into one byte. Also, an additional adder in each arbitrator will not complicate the hardware design.

6.4.4 Prioritized Virtual Channel Assignment and Arbitration Function

In the previous performance discussion of the TDF, the LLF and the PC schemes, the priority of a message is used to determine the number of virtual channels it can request. The EPC scheme uses the priority of a message for both virtual channel assignment and arbitration. Figure 6.7 is the comparison of the EPC scheme with the other schemes. From the results, we can see that the EPC scheme further improves the performance of the PC scheme. In addition, the EPC scheme has better performance than the one-bit approach when the system load is greater than 0.8.

The priority adjustment of the PC and the EPC scheme only increases the priority

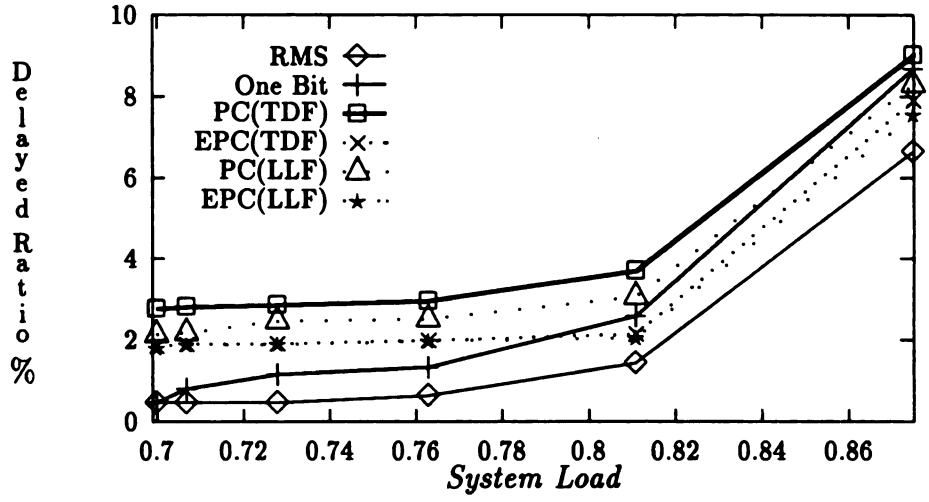


Figure 6.7. Comparison of the prioritized virtual channel assignment and arbitration with the other schemes.

carried by the header of a message. When a body flit arrives at an intermediate node of its path, it inherits the priority of the message header when the header left the node. Thus, the priority of the message is higher in the nodes that are closer to the destination. We can adjust the priorities of a message in all the nodes on its path by increasing the priorities of all the priority registers that have flits blocked in the virtual channels of a physical link. The performance of the EPC scheme is enhanced by adjusting priorities of nodes on the path. The percentage decrease of the delay ratio by the approach from the EPC scheme versus the system load is presented in Figure 6.8.

6.4.5 Latency

The latency of a message is measured from the time it is generated to the time the destination receives the last flit of the message. Although the latency of a message

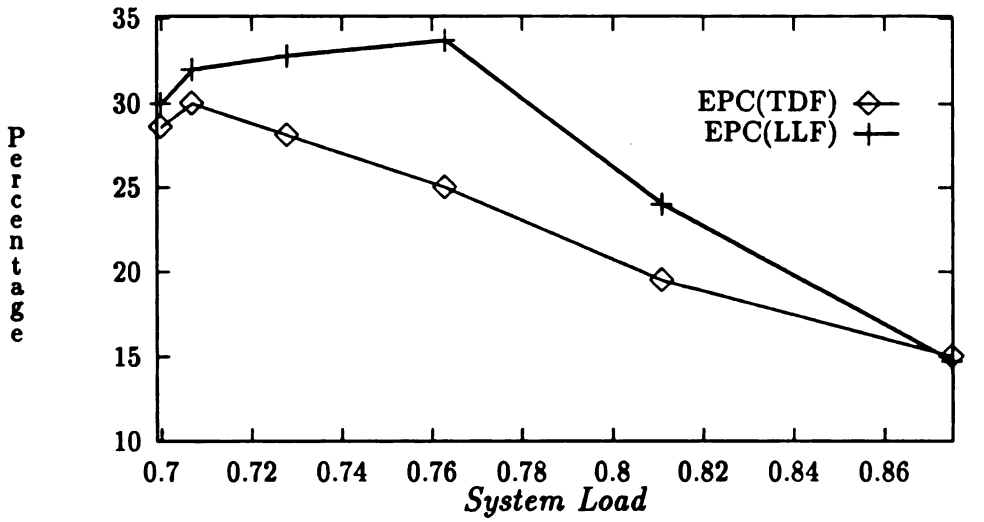


Figure 6.8. Percentage of performance improved by adjusting priorities of nodes on the path.

is not the major performance measure for real-time communication systems, it is an implication of the performance of a flow control scheme. Figure 6.9 compares the average latency of flow control schemes using different priority mapping schemes. As expected, the RMS scheme has the smallest latency and the FCFS scheme has the largest latency. Both the TDF and the LLF schemes reduce the latency from the FCFS scheme. The one-bit scheme has small latency when burst rate is low. However, the contention in the lower numbered virtual channels affects the latency of the scheme when burst rate becomes high.

6.5 Summary

As an increasing number of real-time applications are developed on large scale parallel multicomputers, real-time communication support has become an important issue.

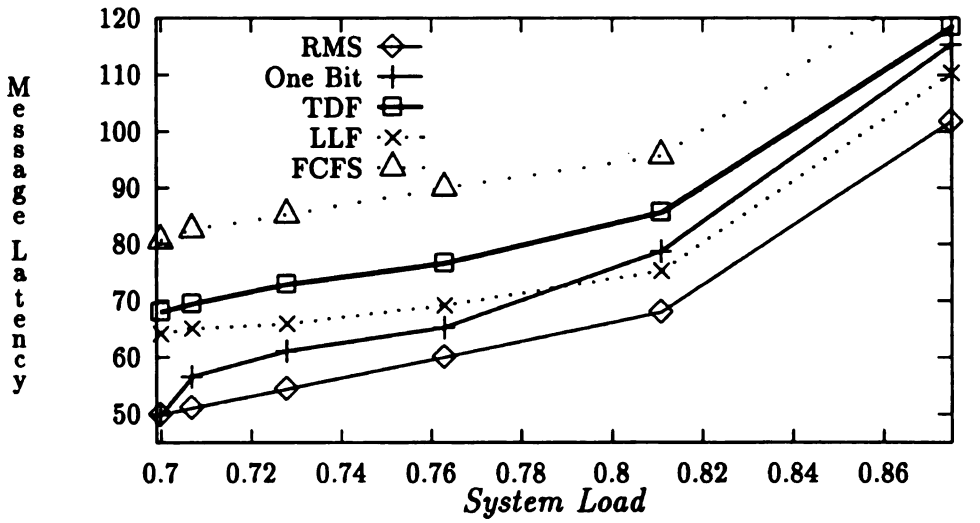


Figure 6.9. Average latencies of the different flow control schemes.

We presented seven real-time flow control schemes for wormhole networks, which are the most popular communication subsystems for large scale parallel multiprocessors. The flow control schemes differ in their priority mapping strategy, priority adjustment methods and arbitration functions. The tradeoff between the hardware costs and the performance has been addressed in the discussion.

We implemented a simulator to study the performance of the schemes and designed experiments to compare the importance of priority mapping, priority adjustment and arbitration. The linear bounded arrival process, which models static traffic, was used to evaluate the performance of the schemes. The simulation results have shown that priority mapping is critical to the performance. An appropriate priority mapping strategy should embed the timing properties (*e.g.*, the deadline tightness or the laxity before the message deadline) of a message in the priority such that flow control can make decisions based on the properties. The timing properties of a message may change during its transmission. Therefore, a priority adjustment

method that modifies the priority of a message when the timing properties change can improve the performance of a flow control scheme. In addition, a priority based arbitration function reduces the chance that low priority messages are blocked too long by high priority messages. Since the priority of a message indicates the time before the message deadline, a priority based arbitration scheme further improves the performance of a flow control scheme. Figure 6.10 is a summary of the performance versus the extra hardware costs of the flow control schemes. It is clear that a better performing scheme introduces extra hardware costs compared to the other schemes. The one-bit scheme is the only exception since its performance degrades when a large portion of the messages is generated in bursts.

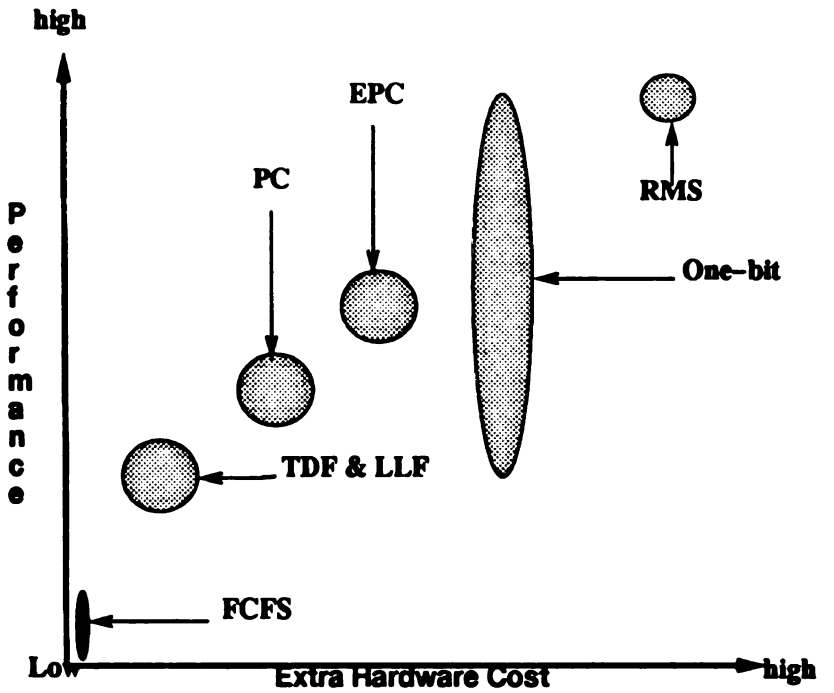


Figure 6.10. The tradeoff between performance and extra hardware costs.

The FCFS scheme is a conventional flow control algorithm found in most general

purpose wormhole routers. The virtual channel assignment is based on the arrival times of the messages, and the arbitration function is either round-robin or random allocation. By increasing the number of virtual channels per physical link, the delayed ratio, which is the real-time performance measure, can be reduced. Since the FCFS scheme does not need extra hardware to support real-time communications, it does not introduce any extra cost to conventional wormhole networks. However, the performance of the FCFS scheme is the worst among all the schemes presented in the chapter.

The RMS scheme has the best performance among all the flow control schemes. The scheme globally assigns priorities to messages sources, and the priority of a source is determined by the period of the source. When a message is generated earlier than its period, the priority is globally adjusted to a lower value to prevent the unnecessary blocking of lower priority messages. However, the hardware cost to support the scheme is the most expensive. The one-bit approach is a modification of the RMS scheme. Instead of carrying a priority in each message header, the one-bit indicator in a header distinguishes the message as a normal message or a burst message. The scheme uses the number of a scheduled virtual channel to represent the priority of a normal message, and a burst message is assigned to a lower numbered virtual channel that is available. The one-bit scheme has similar performance to the RMS scheme when the burst rate is small. When the burst rate increases, lower priority messages compete with burst messages for the virtual channels. As a result, the performance of the scheme degrades with large burst traffic. Nevertheless, the hardware cost of the one-bit scheme is not as expensive as that of the RMS scheme.

The TDF and the LLF schemes map, respectively, the deadline tightness factor and the laxity before the deadline of a message to its priority. The priority of a message determines the number of virtual channels from which a header flit of a message can request an assignment. Both schemes increase the probability of meeting dead-

lines for the messages that have shorter time before the deadlines, and the schemes perform better than the FCFS scheme. The TDF and the LLF schemes require simple changes to the conventional router so that the hardware cost is minimized. The PC scheme is a priority adjustment method that improves the performance of priority based flow control schemes such as the TDF and the LLF schemes. In addition to the priority, each message header carries a BC for the purpose of counting the number of times the message is blocked. When the value of the BC exceeds the maximum value it can represent, the priority of the message is incremented by one. The EPC scheme is based on the PC scheme, and the priority of a message in the scheme is used for making the virtual channel assignment and physical link allocation decisions. Simulation results showed that the EPC scheme enhances the performance compared to the PC scheme and performs nearly as well as the RMS scheme. The hardware support for the EPC scheme increases the complexity of the router design. However, the increased complexity is for combinatorial logic rather than the buffers, so it is not as expensive as the RMS scheme.

CHAPTER 7

Virtual Channel Flow Control for Dynamic Traffic

As the number of dynamic parallel real-time applications increases, communication support for these systems becomes an important issue. In this chapter, we use the Poisson arrival process as a communication traffic model for dynamic real-time applications to evaluate the performance of the LLF, PC and EPC schemes on wormhole networks. In the Poisson arrival process model, the traffic is modeled as dynamic arrivals that may be generated as a Poisson process at each node. This traffic model is suitable for dynamic real-time applications. The possibility for congestion increases as wormhole networks scale to large sizes. We study the effect of the contention delays imposed onto messages as the network size increases. Our simulation results indicate that the performance of the conventional FCFS scheme degrades dramatically as the network size scales upward. The proposed PC and EPC schemes provide the desired performance as the network size becomes large.

Real-time messages lose their value if they are not transmitted and received before their deadlines. If messages become worthless when they are tardy,* then the network resources consumed by the tardy messages should be released. Otherwise,

*A tardy message is one that misses its deadline.

the worthless tardy messages contribute to the load of the network and cause resource contention. As wormhole networks increase in size, the effect of tardy messages on the load of the network resources may increase. It is possible that a large number of tardy messages will consume resources that would better serve other messages. The message dropping method that we present utilizes the priority of a message and the local information in each node to drop tardy messages efficiently. The performance study verifies that the message dropping scheme improves the performance of the PC and EPC scheme.

7.1 Flow Control

The flow control schemes that we study for dynamic communication traffic are the LLF, PC and EPC schemes discussed in Chapter 6. Our previous study in Section 6.4 showed that the laxity before the message deadline is an appropriate timing property for priority mapping. Messages with less laxity should be delivered earlier to meet the deadline. In a priority based flow control system, there is a chance that high priority messages block low priority messages for an indefinite period. As a result, a large number of low priority messages miss their deadlines, and starvation of low priority messages can occur. For the PC and EPC schemes, the value of the priority byte is used to estimate the laxity before the message deadline. When transmitting a message, as the laxity of the message is reduced, its priority is increased.

In Chapter 6, the W in Equation 6.1 of the priority mapping function for the LLF scheme was assigned a value suitable for the static communication study. The value was determined by the off-line analysis of the message sources such that communication traffic can utilize all the virtual channels. However, unlike static models, dynamic communication traffic cannot be analyzed off-line. The W is assigned an estimated blocking time that causes the priority of a message to be incremented by

one. By using the estimated blocking time for priority assignment, the priority of a message can provide accurate estimation of the laxity before deadline. For example, assume that there are 4 virtual channels. A message with initial priority one means that the message can be blocked for a total time of $3W$ ($= (4 - 1) * W$) or less before it misses the deadline. The blocking time W can be estimated as

$$W = 2^{(8-n)} * T_d,$$

where T_d is the estimated period after which a BC is incremented by one. The exact time used by the PC scheme that causes a BC to be incremented depends on the number of virtual channels that have flits in the buffers and the number of flits that are available for transmission. The choice of T_d is a system design issue that requires extensive study.

7.2 Performance Analysis

We analyze the performance of the flow control schemes in this section. The simulation experiments are designed to compare the priority based schemes with the conventional FCFS scheme. We study the effects of upward scaling the network size on the performance of the flow control schemes.

7.2.1 The Model of Performance Study

The communication traffic is modeled as Poisson arrival process. In this dynamic traffic model, message arrivals are dynamically generated as an independent Poisson process at each node. Unless specified, the size of the mesh is 8x8 nodes in this following discussion. The number of virtual channels associated with each physical link is 8. The studies in [16] and Section 6.4 showed that 8 virtual channels are

suitable for wormhole networks to achieve a reasonable performance. In order to generate a mixture of messages range from short to long, the length of a message is drawn from the exponential distribution with an average length of 400 flits. The laxities of messages are exponentially distributed with the average period of five times the estimated message latencies as described in Section 6.1. It is common in a real-time system that the majority of the messages have loose deadlines and only a small number of critical and urgent messages have tight deadlines. The average laxity is selected such that a large portion of the messages have long laxities, *i.e.*, low priorities.

7.2.2 Priority Based *vs.* FCFS Flow Control

Figure 7.1 compares the performance of the flow control schemes. The results show that the LLF scheme, which maps the laxity of a message to a priority, reduces the deadline missed ratio from the conventional FCFS scheme. Although increasing the

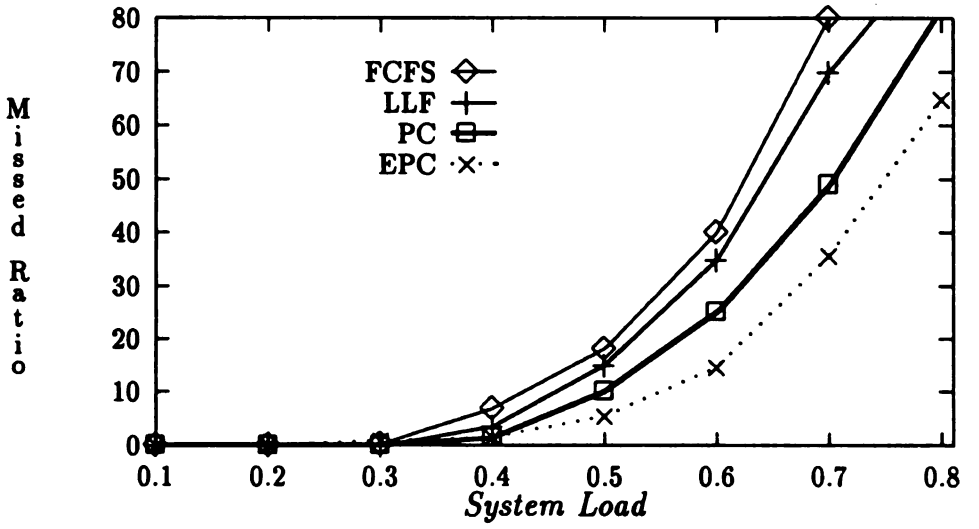


Figure 7.1. Performance comparison of the flow control schemes.

nur

the

the

the

the

me

the

blo

inc

ove

wit

Th

ba

sin

the

an

car

loa

7.

The

wor

incre

wher

the F

demo

number of virtual channels increases the network throughput for the FCFS scheme, the increased throughput does not offer a lower deadline missed ratio. By limiting the number of virtual channels a message can request, the LLF scheme increases the chance that a message with smaller laxity can meet its deadline. However, in the LLF scheme, the messages with low priorities can be blocked by higher priority messages for an indefinite period. To prevent starvation of low priority messages, the PC priority adjustment scheme increments the priority of a message when it is blocked for a period that causes the blocked count to cycle through all its values. An increment in priority increases the probability that a message may meet its deadline.

The simulation results show that the PC scheme improves the system performance over the LLF scheme. The EPC scheme further increases the chance that messages with small laxities can be successfully transmitted in comparison to the PC scheme. The additional use of the priority of a message for virtual channel assignment and bandwidth allocation can reduce the deadline missed ratio for the EPC scheme. The simulation results validate that the EPC scheme has the best performance among all the flow control schemes. Although, in Figure 7.1, the performance gain from the PC and EPC schemes at system load lower than 0.3 is small, we expect that the schemes can significantly improve the performance from the FCFS scheme even at low system load when the number of nodes in the system becomes large.

7.2.3 Scalable Real-Time Network

The study in [103] presented an analytic representation of a simplified wormhole network in which the authors showed that the contention delay imposed onto a message increases as the size of a wormhole network increases. It follows in our study that when the number of nodes in a system becomes large, the real-time performance of the FCFS scheme drops dramatically due to the long contention delays. Figure 7.2 demonstrates the effect of upward scaling of the network size to the deadline missed

ratio for the FCFS and the EPC schemes.

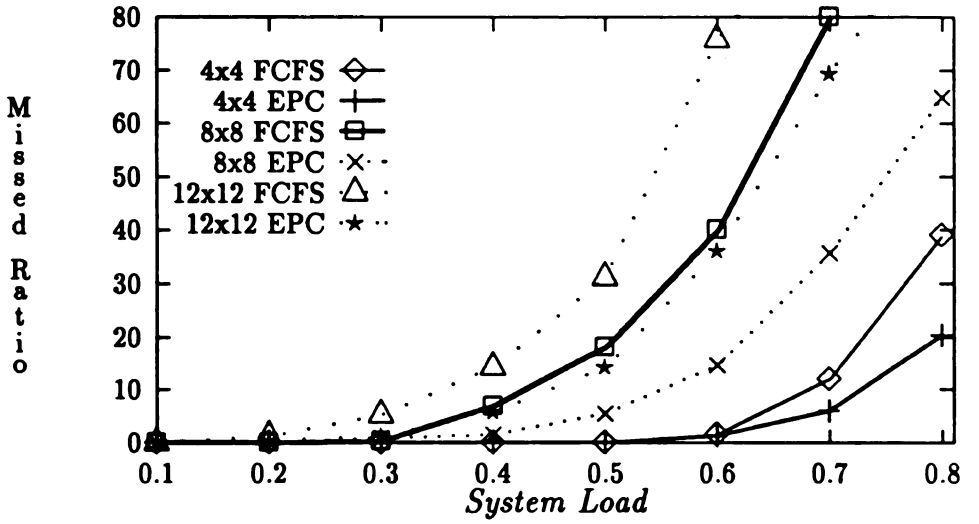


Figure 7.2. Deadline missed ratio increases as the network size scales upward.

The EPC scheme is designed to reduce the contention delays for messages with smaller laxities. When the network size is small, the differences between the performance of the EPC and the FCFS schemes are small. However, the differences increase as the network size increases. When the system size is 12x12, the performance of the FCFS scheme is unacceptable when the system load is higher than 0.3. However, the EPC scheme can provide reasonable performance at higher system loads. The detrimental effects of increasing the sizes of scalable multiprocessors lead us to develop a scheme to reduce network contentions as the sizes of systems increase. In the next section we present a message dropping scheme to reduce network load.

7.3 Message Dropping

Unlike a general purpose system, a real-time message loses its value if it misses the deadline. If a message in the network becomes worthless when it misses its deadline, it is unnecessary to continue delivering the message. If a tardy message is not removed from the network, the message contributes to the load of the network and causes contention with other messages. A message dropping method can reduce the contention delays caused by a tardy message and can improve the system performance. However, dropping a message in a wormhole network is not as straight forward as in a store-and-forward network. Since the flits of a message are spread along its path, we need a distributed scheme that can independently drop a flit without generating additional traffic (*e.g.*, signaling all the flits). In addition, the decision to drop a flit should be based on the local information stored in the node where the flit is located. There are four issues for dropping a message: when to drop a flit, which node is responsible for dropping the flit, when to release the reserved virtual channel and which flit is the tail of the dropped message.

7.3.1 The Message Dropping Scheme

In the EPC scheme, the priority register of each virtual channel contains the priority and the BC of a flit. The contents within a priority register provides a means for estimating the laxity remaining for the message to meet its deadline. If the value of the priority register exceeds the maximum value it can represent, the transmission of the message is considered late, *i.e.*, the flit has missed its deadline. The flit in the virtual channel should be dropped.

Suppose a flit is dropped at the node where it is found to be late, and the virtual channel reserved for the message is released. There is a possibility that a portion of the message has been successfully transmitted to the receiver. Consequently, some

virtual channels reserved on the message path will not be released. The other problem with this scheme is that the receiver of a partially dropped message cannot determine which flit is the last flit of the message. Since a portion of a message is dropped in the network, the normal tail flit of the message will not reach the receiver. Only the on-time flits can be successfully transmitted such that the receiver cannot distinguish a partially dropped message from a normal message. The message dropping scheme should generate a new tail flit to signal the receiver that the message is dropped. If the message dropping scheme does not release the virtual channel when dropping a flit, the normal tail flit should not be dropped and is responsible for releasing all the virtual channels reserved for the message. The drawback of not dropping the normal tail flit is that a reserved virtual channel cannot be used by other messages while waiting for the tail flit. In order to achieve an improved virtual channel utilization, we need a message dropping scheme in which individual flits can release the virtual channels when they are late.

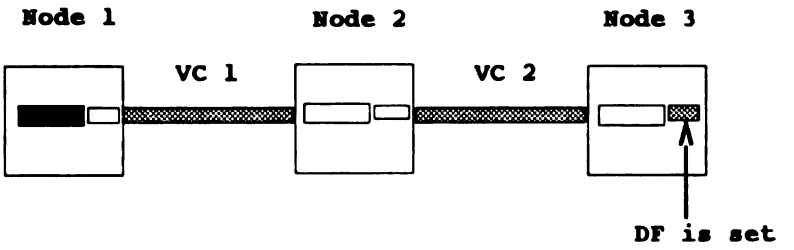
In our message dropping scheme, each virtual channel has a one-bit drop-flag (DF). When the value of a priority register exceeds the maximum number it can represent, an overflow signal is generated to set the DF associated with the virtual channel. The flit buffered in the virtual channel is considered to be late. When the late flit leaves the node, a tail flit signal is generated by raising the tail control line from low to high. In other words, the late flit becomes a tail flit. However, a header flit cannot be a tail flit that a router distinguishes header and tail flits from body flits by detect a raise and a drop of voltage, respectively, from the tail control line [100]. When implementing a router to support the message dropping scheme, a header flit cannot be treated as a late flit. A late flit continues its transmission and is dropped at the next node on the path that has a DF set. When entering a node, a late flit checks the DF of the virtual channel. If the DF is set, then the flit is dropped and the virtual channel is released. If the DF is not set, the flit is transmitted to the next

node. When the late flit leaves the node, the virtual channel is released. Note that an on-time flit is dropped at a node where the DF is set by a previous late flit, and the virtual channel is not released in this case. When a node receives a partial message, the message is discarded.

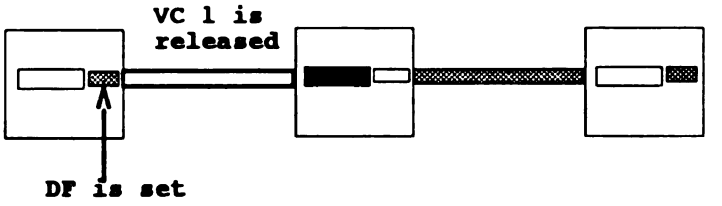
One advantage of using the EPC scheme with the message dropping scheme is that the trailing flits of a late flit are found to be late and dropped with high probability. Suppose one flit is blocked at one node, the trailing flits are also blocked. Thus, the value of the priority registers that buffer the trailing flits are incrementing as the flit is blocked. In other words, when one flit is found to be late, the trailing flits might have been found to be late or found to be late in a very short period. Consequently, the network resources that are consumed by the late flits are efficiently released.

In order to drop a tardy message instantaneously, we can drop messages with additional hardware support in conjunction with the concept of detecting late flits. When the DF of a virtual channel is set, we can generate a *drop* signal propagate on both directions of the message path until it reaches the header and tail flits. When receiving such a signal, virtual channels drop flits stored in their flit buffers and release the virtual channels. The hardware requirement to support this scheme is a drop signal line for each virtual channel and logics to generate the drop signal.

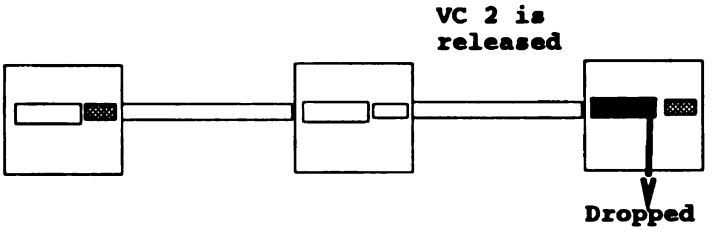
An example of dropping a late flit is illustrated in Figure 7.3. In Figure 7.3(a), the DFs of the virtual channels reserved for the message are not set, except at Node 3. A flit of the message is found to be late at Node 1. The late flit should be dropped at the next node where the DF is set. Since the DF in Node 1 is not set, the flit should be transmitted to Node 2. After the flit is transmitted to Node 2, in Figure 7.3(b), the DF in Node 1 is set and the virtual channel VC1 is released. Since the DF in Node 2 is not set, the flit must be sent to Node 3. In Figure 7.3(c), the flit is transmitted to Node 3 and virtual channel VC2 is released. Since the DF in Node 3 is set, the flit is dropped at Node 3.



(a) A flit is found to be late in Node 1, and the DF in Node 3 is set.



(b) The late flit is sent to the next node, and VC1 reserved for the message is released.



(c) The late flit is sent to Node 3, and VC 2 is released. Since the DF is set, the flit is dropped at the node.

Figure 7.3. An example of dropping a flit.

Our message dropping scheme drops messages correctly. The conditions for correctly dropping a message include the following:

1. The scheme guarantees that the receiver of a dropped message receives one and only one tail flit.

2. All the virtual channels reserved for a dropped message are released.

3. The subsequent flits of the first late flit are dropped.

In our scheme, a late flit is dropped at the next node where the DF is set. Suppose flit f_l is the first late flit of the message and f_l is found to be late at node n_k. Since the preceding flits of f_l are all on-time flits, the DFs of the nodes between n_k and the destination node are not set. Flit f_l will be received by the receiving node as the tail flit of the message. The virtual channels between n_k and the destination nodes are all released by f_l. As the DF in n_k is set by f_l, all the subsequent flits of f_l that reach n_k are dropped. Thus, f_l is the only tail flit received by the receiver. Condition (1) is satisfied.

Suppose all succeeding flits of the new tail flit f_l are on-time; these flits are dropped at n_l and the normal tail flit releases the virtual channels between the sender and n_l such that all the virtual channels reserved for the message are correctly released. In the case that some succeeding flits of f_l are found to be late, the virtual channels between the two nodes where two adjacent late flits are found will be released. Suppose flits f_i and f_j are adjacent late flits, where $i > j$, and flits between f_i and f_j are on-time. Further suppose that nodes n_x and n_y are the nodes where the flits f_i and f_j, respectively, are found to be late. The virtual channels between n_x and n_y are released by f_j, and f_j is dropped at node n_x. The normal tail flit will release the virtual channels between the sender and the node where the last flit is found to be late. Therefore, the message dropping scheme correctly releases all the virtual channels reserved for a dropped message and condition (2) is met.

If the succeeding flits of a late flit are transmitted to the node where the late flit found to be late, the succeeding flits are dropped at the node. Thus, the subsequent flits of the new tail flit are all dropped, which satisfies condition (3).

Since the dropping scheme satisfies the correctness conditions, the scheme is correct. Figure 7.4 is an example of dropping a tardy message. In the example, flits w,

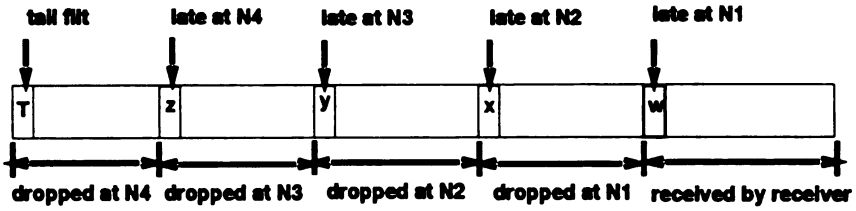


Figure 7.4. Dropping a tardy message.

x, y and z are found to be late at nodes $N1$, $N2$, $N3$ and $N4$, respectively. Since w is the first late flit of the message, it will be received by the receiver as the tail flit of the message. The virtual channels between node $N1$ and the destination node are released by w. The flits between two adjacent late flits, including the latter late flit, are dropped at the node where the former flit is found to be late. For example, the flits succeeding w to the next late flit x are dropped at $N1$, where w is found to be late. The virtual channels between $N1$ and $N2$ are released by x. The subsequent flits of the last late flit z are dropped at $N4$, and the virtual channels between the sender and $N4$ are released by the tail flit T. The message dropping scheme provides a distributed means of dropping a flit without any global information. The idle time of the virtual channels reserved for a tardy message is also reduced.

7.3.2 The Effect of Message Dropping

The message dropping scheme provides an efficient means for reducing the network congestion produced by tardy messages. This issue increases in importance as wormhole networks scale to large sizes. Figure 7.5 is a comparison of the performance of the EPC scheme with and without message dropping. Note that the deadline missed ratio for the message dropping scheme includes both the tardy messages that are completely received and the dropped messages. The results indicate that the cases with

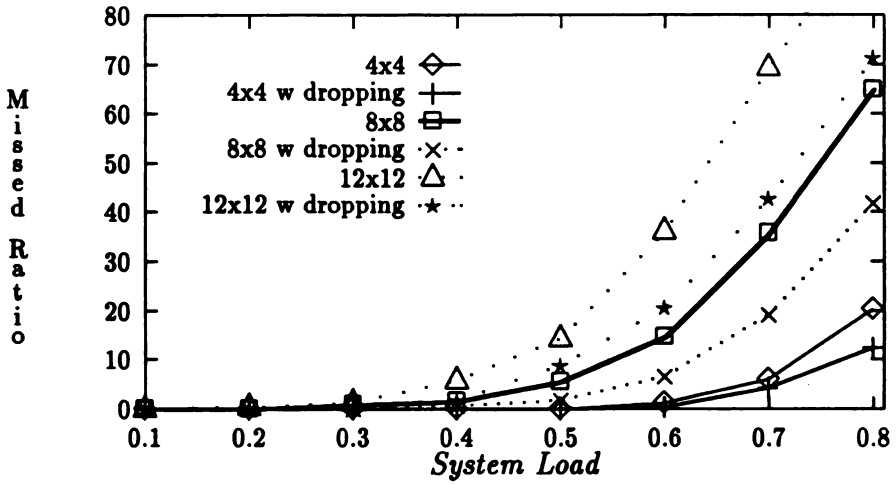


Figure 7.5. The effect of the message dropping on the performance.

message dropping have significantly reduced the deadline missed ratio from that of the EPC scheme. This is especially true when the networks increase in size. In addition, by dropping tardy messages, the system can provide an acceptable performance at a higher load than the EPC scheme without message dropping.

7.3.3 The Size of BC

The number of bits in a BC determines the largest number it can represent and the rate that the priority of a message is incremented. The priority of a message increments slower with a larger BC size and faster with a smaller BC size. In the message dropping method, the BC size also determines the rate for dropping a message. A large BC size drops messages slower such that the message may have missed its deadline before it is dropped. In contrast, a small BC size may drop messages too fast such that the dropped message could meet its deadline by continuing to be transmitted.

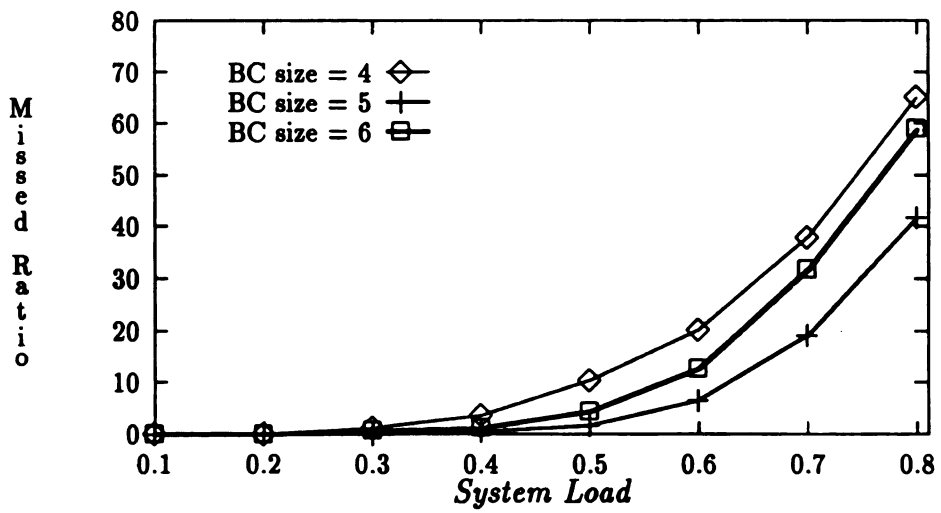


Figure 7.6. The size of BCs affects the rate of dropping messages and the performance.

Figure 7.6 shows the performance when BC sizes are equal to 4, 5, and 6 bits. The detailed statistics indicate that the percentages of the dropped messages that contribute to the total deadline missed ratio are 99%, 80% and 35% for the 4-, 5-, and 6-bit BCs, respectively. In other words, the BC size of 4 bits discards messages too fast so that some dropped messages could have met their deadlines by continuing to

be transmitted. The 6-bit BC drops a small portion of the tardy messages; therefore, the performance improvement is less significant than that of the 5-bit BC.

7.4 A Router Design to Support Real-Time Flow Control

The proposed real-time flow control schemes require special features to support priority based virtual channel assignment and arbitration. We describe a router design that can be used for the EPC scheme and the message dropping method.

Figure 7.4 describes the router design for a 2D mesh that each physical link has 2 virtual channels. In the EPC scheme, the priority of a message determines the number

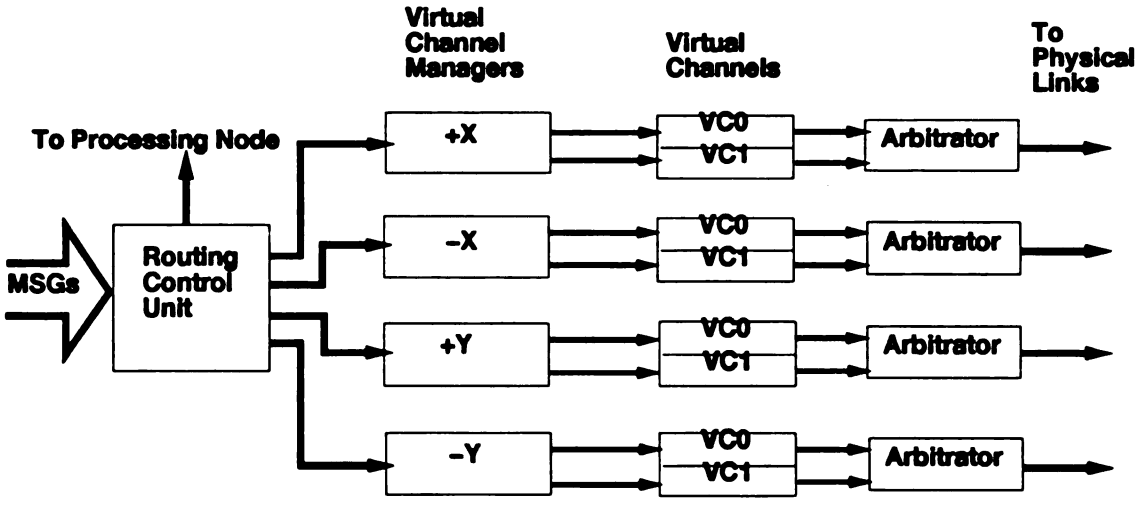


Figure 7.7. A router design to support priority based flow control and message dropping.

of virtual channels it can request. When more than one message is competing for an available virtual channel, the virtual channel is assigned to the message with the highest priority. A message header consists of three flits. The first two flits contain

the distance on X and Y directions. The third flit is the priority and BC of the message. When a message header arrives at a router, the first two flits are used to make a routing decision. After the routing decision is made, the virtual channel manager assigns a virtual channel to the message based on the priority contained in the third flit and the availability of the virtual channels.

Figure 7.4 illustrates the components of two virtual channels associated with a physical link. The size of the flit buffer for each virtual channel is 3 flits. When a

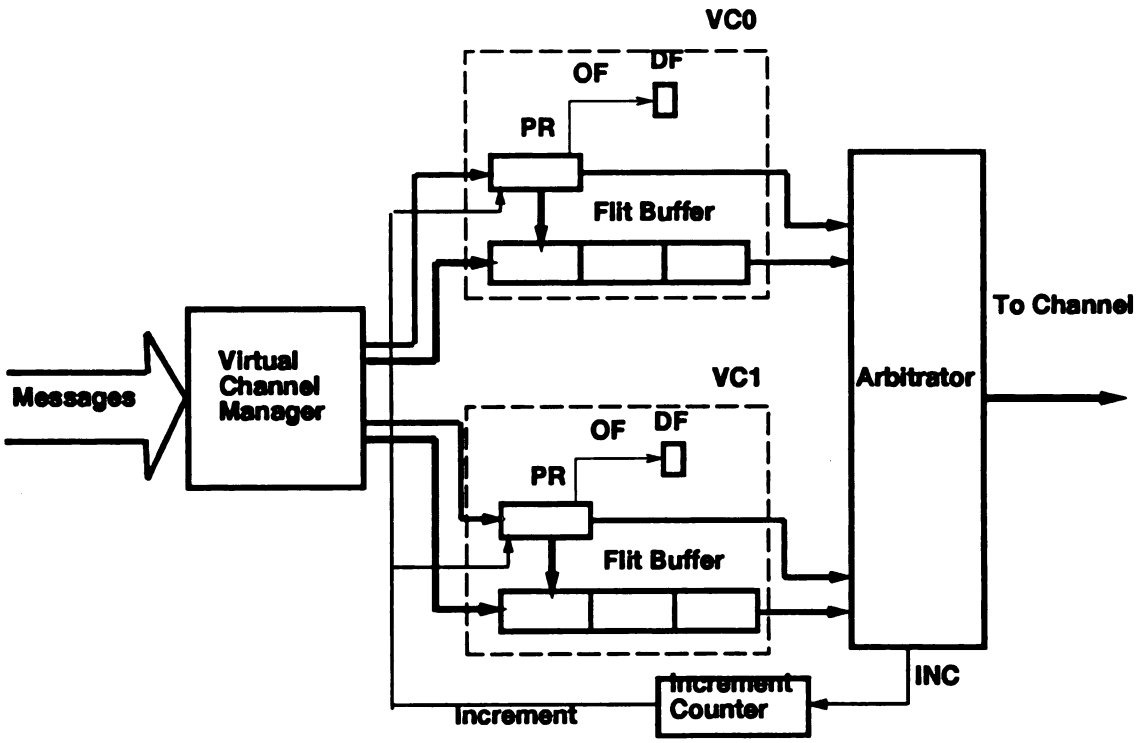


Figure 7.8. The components of virtual channels.

message is assigned to a virtual channel, the value of the priority byte, which is the third flit of the message header, is stored in the priority register (PR) of the virtual channel. The value of a priority register retains when the message header leaves the node, and the trailing flits of the message inherit the priority saved in the PR.

The arbitrator polls the virtual channels for flits to be transmitted on the physical channel. The polling of all virtual channels can be done within one cycle. As all the virtual channels are polled once, the arbitrator increments the value of the increment counter by one. The counter is designed to adjust the priority updating rate such that low priority messages will not have the highest priority after been blocked for a short time. After the counter reaches its full count, an increment signal is triggered to increase the value of the PRs which have flits blocked in their flit buffers. Before a message header leaves the node, the value of the third flit is updated with the priority stored in the PR.

In order to support the message dropping method, each virtual channel has a one-bit delay flag (DF). As the value of a PR exceeds the maximum value it can represent, the overflow (OF) signal sets the DF associated with the virtual channel. When the last flit currently buffered in the virtual channel is transmitting to the next node, a tail signal is generated such that the flit becomes a tail flit. When flits arrive at a virtual channel that has its DF set, the flits are dropped. If a tail flit is detected, the DF is cleared.

By adding one control line to each virtual channel, we can drop a late message completely and fast. Figure 7.4 illustrates an example of using the drop signal (D/S) lines. In Figure 7.4(a), the D/S lines associated with the the virtual channels are initially low and the DF of router 2 is set (flits in the buffers are found to be late). In Figure 7.4(b), router 2 sends a drop signal to each direction of the message path by raising the D/S lines to high. The flits stored in router 2 are dropped. When a drop signal is detected, in Figure 7.4(c), router 1 and 3 empty their flit buffers and release all the resources reserved for the message. The DFs in router 1 and 3 are set. Then, in Figure 7.4(d) router 1 and 3 reset the D/S lines to low. When router 1 detects the D/S lines become low again, the DF is cleared. After a router receives a drop signal, if the flits buffered in the router are not tail or header flits, the router generates a

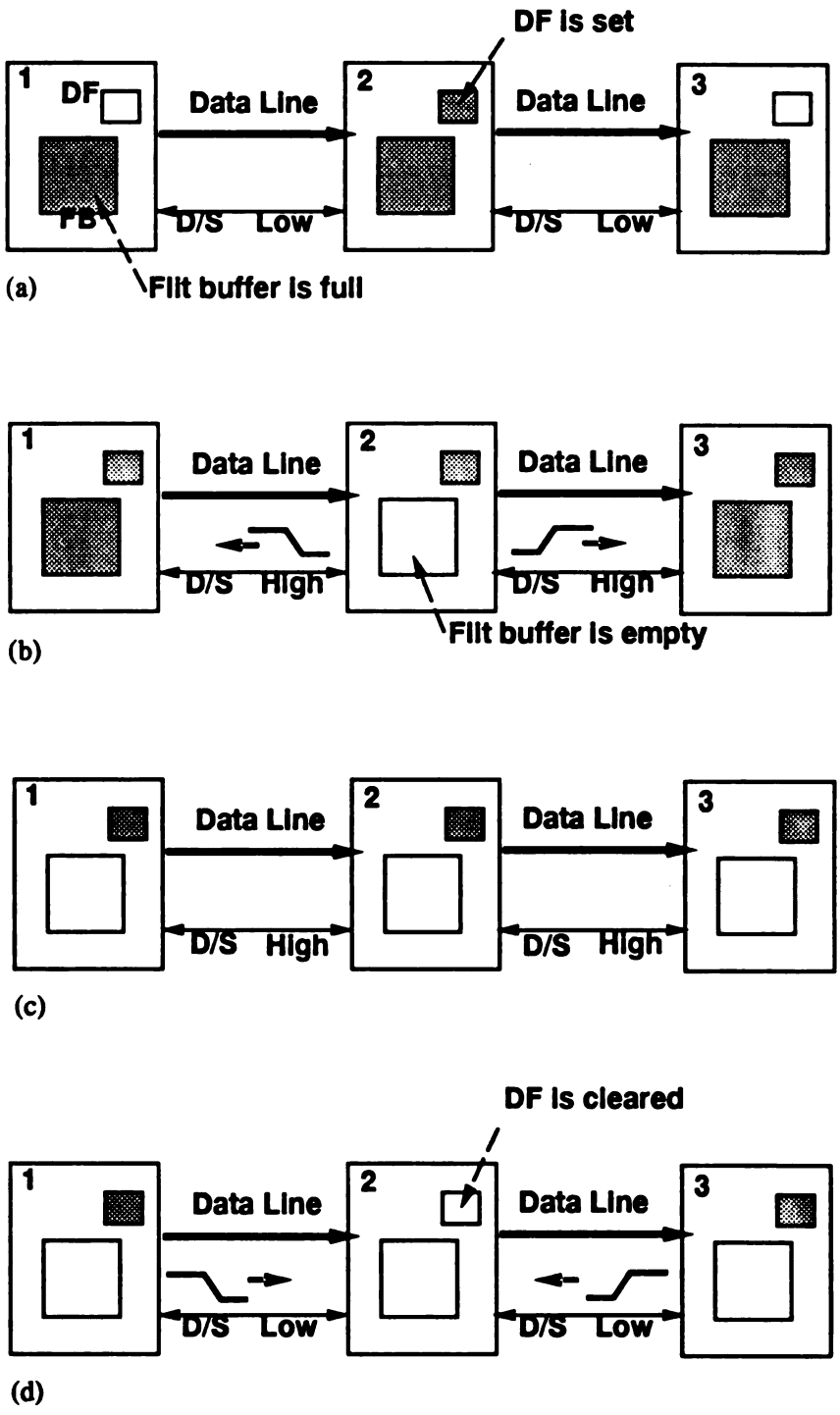


Figure 7.9. Dropping a message using an additional drop signal line with each virtual channel.

drop signal to the downstream or upstream routers on the path. Otherwise, the drop signal is not generated and the message is completely dropped.

7.5 Summary

We studied the performance of the LLF, PC and EPC flow control schemes, which were described in Chapter 6, for dynamic communication traffic in wormhole networks. In addition, a message dropping method, which can be used in conjunction with the PC and EPC schemes, was presented to reduce the network load that are contributed by tardy messages.

Simulation experiments were designed to compare: the performance of the FCFS conventional scheme and the proposed schemes and the effects of upward scaling the network size on the performance. Simulation studies verified that our flow control schemes outperform the conventional FCFS scheme implemented in general purpose wormhole networks. The LLF priority mapping scheme effectively maps the transmission laxity of a message to a priority. The use of priorities for flow control decisions reduces the deadline missed ratio from the conventional FCFS strategy. The PC scheme dynamically adjusts the priority of a message to reflect the current state of the message such that it can further improve the performance of the system. Furthermore, the use of priorities for virtual channel assignment and bandwidth allocation in the EPC scheme increases the chance of meeting deadlines for the messages with smaller transmission laxities. As a result, the EPC scheme improves the performance in comparison to the PC scheme. When the network sizes increases, the possibility for congestion is increased. Thus, the performance of the FCFS scheme degrades dramatically such that a large number of the messages miss their deadlines. The EPC scheme, which delivers the messages with the shortest laxity first, provides the desired real-time performance as the network size becomes large.

Real-time messages lose their value when they miss the message deadlines. However, tardy messages consume valuable network resources that the messages must be removed from the network. The proposed message dropping method effectively releases network resources occupied by the tardy messages, which is very important as the size of the wormhole network increases. The simulation results indicated that the system improves its performance by dropping tardy messages. The flow control schemes and the message dropping method deliver required performance for large-scale real-time wormhole networks.

CHAPTER 8

Conclusion and Directions for Future Research

This chapter summarizes the major contributions of this dissertation and sketches the plan for future work.

8.1 Summary of Major Contributions

A need for new techniques to support parallel and distributed real-time system design [4] motivates this work. Parallel and distributed systems deliver cost-efficient computing power and provide reliable operating environments for an increasing number of real-time applications that are developed on these systems. Although fruitful research results have been produced for conventional uniprocessor systems, the design of parallel and distributed real-time applications are still based on ad hoc approaches [8]. This dissertation addressed two closely related design issues of parallel and distributed real-time systems: dynamic scheduling of precedent-constrained tasks and communication support in wormhole networks.

The real-time applications that demand high performance parallel and distributed computing systems often control complicated jobs and operate in changing environ-

ments. These applications usually consist of groups of tasks that can be concurrently executed on separate processors. The key component of a parallel and distributed system that maximizes the parallelism embedded in a task group and the performance of a system is task scheduling. However, dynamic task scheduling on parallel and distributed real-time systems has not been widely studied. The task scheduling algorithm presented in this dissertation provides a new approach to solve the problem. The algorithm combines task graph partitioning, least-laxity-first scheduling and branch-and-bound task allocation techniques such that the approach is efficient and can be applied to both parallel and distributed systems. By partitioning a task group into smaller subgroups, which consist of independent tasks, the cost of scheduling a subgroup is kept within a constant limit. Consequently, the overhead of scheduling a task group is increased linearly with the growth of the group size. The least-laxity-first scheduling optimizes the chance that a task can meet the deadline in a processor. For a given task subgroup and processor set, the branch-and-bound algorithm ensures that the tasks can be successfully allocated if there is a feasible schedule. The parameters that affects the performance of the scheduling algorithms are addressed to supply real-time designers with the means of fine tuning the algorithm for different design considerations.

The success of a task scheduling algorithm for parallel and distributed real-time systems requires the support from the communication subsystems. Tasks executed in a parallel and distributed system exchange messages to achieve a common goal. Suppose the transmission time of messages is unpredictable, scheduled tasks might wait for messages and miss their deadlines. Since wormhole routing is a common switching mechanism for parallel multiprocessors, the issue of real-time communication in wormhole networks is anticipated to attract growing attentions. Dally [16] suggested that flow control decisions that are based on the timing properties of messages can improve deadline guarantee ratio. The flow control schemes described in this disser-

tation are proposed for wormhole networks with virtual channels to support real-time communication. The flow control schemes utilize timing properties of messages to allocate network resources to messages. The schemes employ several priority mapping, priority adjustment, arbitration and message dropping techniques. A priority mapping scheme encodes the timing property of a message into a priority which can be represented with a small number of digits. A priority adjustment method modifies the priority of a message to reflect the current status of the message. An arbitration function determines how the network bandwidth is allocated to messages. Tardy messages, which consume valuable network resources, are removed from the network by a message dropping method. The performance of the flow control schemes are studied in both static and dynamic environments. The tradeoff between hardware cost and performance of the schemes are analyzed. With small additional costs and simple modifications, the flow control schemes outperform the FCFS flow control scheme implemented in most wormhole routers.

The task scheduling algorithm and flow control schemes are shown to be valuable for the design of parallel and distributed real-time systems. The directions for future research are presented in the next sections.

8.2 Priority Based Task Scheduling

A real-time system usually consists of critical tasks and essential tasks [9]. Critical tasks have *hard* deadlines such that missing the deadlines causes severe consequences. Unlike critical tasks, essential tasks, which are less critical, have soft deadlines such that real-time systems are usually designed to tolerate a small number of essential tasks to miss their deadlines. A priority based scheduling algorithm, which schedules critical tasks with higher priorities, can ensure that all critical tasks will meet their deadlines.

The concept of imprecise computation [25] can be useful for scheduling a task group that consists of both critical and essential tasks. The scheduling of a task group can be done in two phases: mandatory and secondary. Critical tasks are scheduled in the mandatory phase, and essential are scheduled in secondary phase. Since tasks in a group have precedence relations, the task graph partitioning scheme of a scheduling algorithm must consider both the criticalness of tasks and precedence constraints. Meanwhile, the scheduling overhead should be kept minimal for dynamic scheduling. By combining our task scheduling algorithm with the imprecise computation task model, we anticipate that a new priority based scheduling algorithm can solve the problem.

8.3 Task Scheduling with Resource Requirement

Real-time tasks require resources for their execution. Resource scheduling is equally important as task scheduling in a scheduling algorithm. However, different types of resources require different management principles that complicates the scheduling problem. The scheduling algorithm presented in this dissertation can easily be integrated with resource scheduling by adding resource requirements to the bounding condition of the BB algorithm. When searching for a feasible schedule, the BB algorithm considers both the computation demands and resource requirements. Thus, the scheduling algorithm produces feasible schedules without increasing the scheduling overhead.

8.4 Real-Time Flow Control in ATM Networks

Broadband ISDN (B-ISDN) systems are built on a technology base of *asynchronous transfer mode* (ATM) [104]. The B-ISDN networks provide a means for integrated

transport for a variety of services, which include voice, video telephony, low and high speed data. Congestion and flow control in ATM networks has been considered as one of the fundamental challenges. Since continuous media transport, *i.e.*, voice and video images, is a service in ATM networks, the real-time requirements of continuous media must be supported [105].

The flow control mechanisms of ATM and wormhole networks have the same characteristics that they both require simple schemes and limited-sized buffering [106]. In ATM networks, a cell, which is the basic unit of buffering, can carry a multiple bit priority for flow control decisions. As real-time cells must be delivered such that the end-to-end delays are bounded, the concepts of our flow control schemes can be utilized for ATM flow control schemes. However, the different services have unique performance requirements that the design of a flow control scheme must guarantee the "quality of service" promised by B-ISDN systems.

APPENDICES

APPENDIX A

The DRMS Tool

The number of dynamic real-time applications is increasing; however, some real-time applications are inherently static. For example, most data acquisition systems acquire data at fixed intervals [107]. As parallel and distributed systems are emerging as the computer systems for real-time systems, methods to apply existing static design techniques to the new systems is an critical issue. In this appendix, we describe a tool, DRMS, which combines several static scheduling algorithms and the BB task allocation scheme, which is presented in Chapter 4, to assist real-time system designers schedule periodic tasks to distributed systems.

Researchers of real-time scheduling have had great success in producing static algorithms that are practical, widely used, simple to implement and in some cases optimal. An example of such a result is the real-time scheduling algorithm known as the *rate monotonic scheduling* (RMS) algorithm. The RMS algorithm was shown by Liu and Layland [66] to be an optimal fixed priority real-time scheduling algorithm for a restricted class of tasks that are preemptively executed on a uniprocessor. The term "optimal" means that if there exists a feasible schedule* for which a set of fixed priority tasks can meet their deadlines, the RMS algorithm will provide such a schedule. RMS assigns priorities to tasks according to the frequencies that they execute. Accordingly,

*A schedule is feasible if it is a schedule for which all the deadlines will be met.

the task that executes at the highest frequency (and therefore has the shortest period) is assigned the highest priority. All other tasks are assigned priorities relative to the frequency they execute. Typically the tasks served by the RMS algorithm are periodic, although researchers have investigated how to extend RMS technology for serving aperiodic tasks by creating a new periodic task that guarantees computing cycles for aperiodic tasks [108].

The success with the development of the technology associated with RMS has motivated designers to build systems that apply the RMS algorithm to their applications [109]. A system designer may define a set of periodic tasks and then must determine whether the deadlines of all tasks can be met. Simple tests can be conducted to determine whether a set of tasks can be feasibly scheduled on a single processor. However, the designer may be developing applications on a distributed system in which tasks may execute on several processors in the system. The situation becomes complicated when the designer must determine how to partition the set of tasks so that all tasks are feasibly scheduled on all processors in the system. Further complications arise when some tasks are more important for the application being served than other tasks. Some tasks may be characterized as critical tasks, while other tasks are considered to be less critical but have timing constraints. Critical tasks are those tasks that must be executed before their deadlines or a catastrophe may occur. These tasks are often described to have *hard* deadlines. Less critical tasks (those tasks with *soft* deadlines) normally should be scheduled to meet their deadlines, but it may be acceptable for these tasks to miss their deadlines on a few occasions. The tasks assigned to a processor normally have deterministic computing demands and periods. However, on some occasions the previously determined demands or periods may be violated, such that a transient overloaded condition occurs. During transient overloaded conditions it is more important that the critical tasks finish their executions before their deadlines than for the non-critical tasks to meet

their deadlines. In addition, it is unlikely that a designer simply wants to know if one particular allocation is feasible. The designer would likely ask questions such as:

- Suppose I have a set of critical tasks that must meet their deadlines, and a set of tasks that are less critical but have timing constraints. How do I partition the tasks to the processors so that in the presence of transient overloads the less critical tasks are primarily affected? What if the priorities of my critical tasks do not match the priority assignment of the RMS algorithm?

- What happens to a feasible set of tasks if I shorten the deadline of one task in the task set?

- What happens if one task is moved to a different processor? Will the task set remain feasible?

- Will the set of tasks be scheduled feasibly if I change the amount of memory allocated to a processor in the system? How will the system change as I substitute a processor with one that has a different processing capacity? What happens when the context switching time of tasks change?

Researchers have developed solutions to the problems posed by these questions, but it may be very difficult and time-consuming for designers to obtain the answers to their questions. Since designers are interested in asking questions about many different scenarios, we built a tool for analyzing the feasibility of scheduling sets of tasks that execute on multiple processors. Our tool, called DRMS,[†] answers the "what-if" questions that a designer may ask, such as those listed above. To use DRMS, a user specifies a set of processors with different processing and memory capacities, and a set of tasks with different parameters such as periods, processing demands, and context

[†] DRMS gets its name because it serves a *D*istributed environment composed of processors that use *RMS* for scheduling local tasks.

switching times. The tool searches for feasible allocations of a set of tasks to a set of processors. If there exists a feasible allocation, the tool can find it. If no task allocations are feasible, the tool can provide a list of the best infeasible schedules. The best infeasible solution is a feasible solution for the largest proper subset of the set of tasks.[‡] In order to make the tool more convenient to use, the user is supplied with a graphical user interface to specify resources and tasks and to examine the results of an analysis of the system. For a given allocation of tasks to processors, a graphical time-line can be generated with important scheduling points labeled so that a user can examine when critical operations occur on all the processors in the specified system.

Practical tools have been developed for real-time software development. For example, Scheduler 1-2-3 [110] is a X-window based interactive tool that can verify whether a set of hard real-time tasks will complete by their deadlines. A schedulability analyzer component of the tool can verify deadline constraints for rate monotonic scheduling algorithm. This tool conducts similar tests that DRMS conducts for determining scheduling feasibility. With different goals, DRMS pursues the activity of determining how to allocate tasks to processors. If there are no feasible allocations, DRMS finds the best of the infeasible tasks, or performs transformations on the tasks to attempt to find feasible allocations. DRMS is suitable for making task allocations in real-time environments that have incorporated RMS technology, such as the case study of an avionic system reported in [109]. Scheduler 1-2-3 has components not included in DRMS, which include a synthetic workload generator and a real-time monitor/debugger. However, Scheduler-1-2-3 does not have the task allocator of DRMS, nor the features for support of a heterogeneous processor environment.

This appendix provides details about the design and the development of DRMS. The features of DRMS are described and examples are given of its usage. Section A.1

[‡]A proper subset strictly contains fewer elements than the total number of elements of the set.

provides background information of real-time scheduling technology, and in particular the development of the RMS technology. The requirements for an RMS analysis tool are described in Section A.2. Section A.3 presents the design of our strategy for allocating tasks to processors in order to generate feasible schedules. An example of the usage of DRMS with the user interface is presented in Section A.4. A example of using DRMS for scheduling periodic tasks and their communication traffic is described in Section A.5. Section A.6 summarizes the discussion.

A.1 Background of the Rate Monotonic Scheduling Algorithm

We surveyed the development of the RMS techniques in Section 3.1.1. The detail background information of the RMS technology, which is relevant to the design of DRMS, is presented in this section. We describe DRMS in later sections.

A.1.1 The Worst-Case Bound of the Rate Monotonic Scheduling Algorithm

Liu and Layland [66] were the first to address the problem of scheduling periodic tasks. They considered both fixed priority and dynamic priority scheduling algorithms. They assumed that real-time tasks were periodic such that a task was ready for execution at the beginning of its period and had a deadline at the end of its period. Tasks could be preempted, but the costs for context switching were ignored. In addition, there was no synchronization between tasks. Liu and Layland determined that the RMS algorithm is an optimal real-time scheduling algorithm for task sets with fixed priorities. In the RMS algorithm, the priority of a task is related to the frequency that a task is executed, such that the highest frequency task has the highest priority. This

means that the highest priority task has the shortest period. An important result is that the worst-case utilization bound for the rate monotonic algorithm is $n*(2^{1/n}-1)$, where n is the number of tasks, which decreases monotonically from 0.83 when $n = 2$ to $\ln 2$ when $n \rightarrow \infty$. In addition, they found that the *deadline driven scheduling algorithm* is optimal for the case in which priorities vary. This algorithm assigns the highest priority to the task that has the earliest deadline.

A.1.2 Pre-Period Deadlines and Utilizations Larger than the Worst-Case Bound

Leung and Whitehead [111] considered the case when the deadline occurs earlier than at the end of its period. They introduced the *deadline monotonic* algorithm, which uses fixed priorities and the priority of a task is assigned to be the inverse of its deadline. Liu and Layland's result [66] of the worst-case utilization bound for the rate monotonic algorithm has been generalized by Lehoczky and Sha [112]and Lehoczky [113] for the case that deadlines of tasks occur earlier or at the end of the periods.

It is common in practice that task sets can be feasibly scheduled with the RMS algorithm even if they have utilizations greater than the worst-case bound. This has been described by Lehoczky, Sha, and Ding [114]. An analysis technique to determine if a task set can be feasibly scheduled is summarized as follows. Consider a set of n periodic tasks $\tau_1, \ldots, \tau_n$, with periods $T_1, ..., T_n$ respectively, where $T_i \leq T_{i+1}$ and $1 \leq i < n$. Each task τ_i has a computation requirement C_i and a deadline $D_i \leq T_i$. The analysis assumes that the tasks are ready to run at their initiation times and may be preempted instantly. Overhead such as swapping times is ignored. For the worst case conditions, the cumulative demand $W_i(t)$ made by the tasks $\tau_1, \ldots, \tau_i$ as

a function of time over the interval [0,t] is

$$W_i(t) = \sum_{j=1}^{i} C_j * \lceil t/T_j \rceil. \tag{A.1}$$

The following are defined for notational convenience, as was described in [114].

$$L_i(t)a = aW_i(t)/t, \tag{A.2}$$

$$L_i a = amin_{0<t\leq D_i} L_i(t), \tag{A.3}$$

$$La = amax_{1\leq i\leq n} L_i. \tag{A.4}$$

$L_i(t)$ is the utilization of the processor as a function of time during the interval [0, t]. Since priorities are assigned to tasks by the RMS algorithm, τ_i will be preempted only by tasks $\tau_1, \ldots, \tau_{i-1}$, and τ_i will only preempt tasks τ_j with periods $T_j > T_i$. A processor will be busy serving tasks $\tau_1, \ldots, \tau_i$ during the entire time between [0,T_i] and will not be idle during the period until either τ_i completes or misses its deadline, which ever is first. Higher priority tasks may execute several times during the interval [0, T_i] since their periods are less than or equal to T_i.

The criterion for a task set to be feasible is then given by the following two conditions, which were proved in [114].

1. τ_i can be feasibly scheduled for all task phasings using the RMS algorithm if and only if $L_i \leq 1$.

   L_i is the least utilization of the processor during the period [0,D_i]. As indicated by Equation A.1, the cumulative demand changes only at multiples of the task periods. To evaluate L_i, we only need to consider t at values equal to $T_1, T_2, \ldots, T_{i-1}$, and D_i and multiples of $T_1, T_2, \ldots, T_{i-1}$ that are less than D_i. As long as the evaluation of the utilization within the period [0, D_i] is less than or equal to one, τ_i finishes before or at the time that L_i occurs, so τ_i can be

feasibly scheduled. One only needs to accumulate the demands of the higher priority tasks during the period until D_i, in addition to the demand of task τ_i.

2. The entire task set can be scheduled for all task phasings using the RMS algorithm if and only if $L \leq 1$.

   This criteria naturally follows the previous criteria by considering all values of $L_i, 1 \leq i \leq n$. The entire task set is schedulable if each task is schedulable.

A.1.3 Period Transformations for Critical Tasks

Some practical problems arise when a designer applies RMS to an application. Consider a critical task in the task set that does not have the shortest period, and therefore does not have the highest priority. If a transient overload in the system occurs, the critical task might not meet its deadline. However, tasks with shorter periods that are not critical may meet their periodic constraints. Several situations may cause transient overloaded conditions. It may occur because a task must briefly violate its previously defined period or computing demand. Likewise, it may occur because a new task is introduced to a processor in the system that causes the original schedule to be invalidated. We need a method to identify an arbitrary task as critical and to elevate its priority for times of transient overload conditions. The method for enabling a critical task to have its priority elevated is period transformation [115]. This technique transforms the period of a critical task so that its deadline will be met in spite of periods of transient overload where some non-critical tasks will miss their deadlines. For example, consider two tasks: τ_1 and τ_2. Task τ_1 has a period $T_1 = 13$ and a computing demand $C_1 = 4$. Task τ_2 has a period $T_2 = 22$ and a computing demand $C_2 = 10$. The RMS algorithm elevates the priority of τ_1 over τ_2 since $T_1 < T_2$. Let us assume that τ_2 is more critical than τ_1. In order to increase the priority of τ_2, we can transform τ_2 into two periods of length 11 with demand 5, and introduce a

precedence on the two periods of τ_2. With the transformation, τ_2 will execute with higher priority than τ_1.

A.1.4 The Priority Inversion Problem

A higher priority task may be blocked by a lower priority task if the lower priority task is executing in a critical region or consuming a server's resources that are requested by a higher priority task. The condition when a higher priority task is blocked by a lower priority task is called priority inversion [116]. To deal with this problem, Sha, Rajkumar and Lehoczky described a priority ceiling protocol in [116] and proved a theorem that states that a set of n tasks using the priority ceiling protocol can be scheduled with RMS if the following conditions are satisfied:

$$\forall i, \quad 1 \leq i \leq n, \quad C_1/T_1 + C_2/T_2 + \ldots + C_i/T_i + B_i/T_i \leq \quad i * (2^{1/i} - 1), \qquad \text{(A.5)}$$

where C_i is the service demand for task i, T_i is the period of task i, and B_i is the worst case blocking time that task i experiences due to lower priority tasks. Notice that Equation A.5 is the worst case utilization bound given by Liu and Layland [66] with the addition of the blocking time due to priority inversion experienced by task τ_i, which is the task with the longest period.

A.2 Requirements for the DRMS Tool

The components and features to be incorporated within a tool that aids real-time system designers is probably best specified by the designers who have the experience building real-time systems. We have been working with real-time system designers located at the IBM Federal Sector Division in Owego, New York in order to define the components of DRMS. By means of tapping into their experience, we have iden-

tified and defined the important features of DRMS, which we have implemented in a practical tool. A case study of an application that uses RMS technology and requires the feasibility analysis that DRMS can provide is given in [109].

A.2.1 Task Specifications

A user of DRMS provides the specifications of the tasks and processors for which DRMS finds feasible allocations. A list of the input parameters for task specifications is given in Table A.1. The parameters that a user would naturally expect to be supported include C_i, which is the processing demand of task i required within one period, and T_i, which is the length of the period associated with task i. D_i, which is the deadline of task i, should be specified if it is less than the period of a task. In order to model cases of synchronization and requests to remote servers in a distributed system, the blocking time, B_i, for task i must be specified. The blocking time is the worst-case time that a task may have to wait due to lower priority tasks. The blocking time may be due to the fact that a lower priority task is in a critical section, or a lower priority task is using a server that is requested by a higher priority task. We assume that the priority ceiling protocol is used to define a worst-case bound for the blocking time.

Many other parameters have been identified to be important components of DRMS. These include the following task specification parameters:

- The amount of memory that task i consumes, MEM_i. A practical assumption is that all tasks are loaded within memory, and the memory of a processor is shared by all tasks assigned to the processor. It is possible that a processor may have enough processing capacity to execute all tasks feasibly, but not have the memory that the tasks require.

- The time required for the operating system to perform a context switch for a

Table A.1. Parameters for Task Specification.

Parameter	Definition
$NumTask$	Number of tasks to be allocated.
C_i	Processing demand of task i.
TST	Common constant time for context switching of tasks.
$ITST_i$	Context switch time of task i.
T_i	Period of task i.
D_i	Deadline of task i.
B_i	Task blocking time.
J_i	Jitter of task i.
GID	Group identification of a task group.
$Pbind$	Processor identification to which a task is bound for execution
TI_i	Importance of task i. $TI_i = critical$ indicates task i is a critical hard real-time task. If TI_i is an integer, it indicates the number of periods that task i should be subdivided.
P_G	Priority granularity.
$Priority$	User specification of task priority.
MEM_i	Amount of memory demanded by task i.

preempted task. There may be a constant context switch time for all tasks, TST, and separate context switch times for individual tasks, $ITST_i$.

- A task has a hard deadline or a soft deadline. A hard deadline implies that a task is critical, and must meet its deadline. A soft deadline implies that the deadline is important, but it is possible due to transient overload conditions that a task at times will miss its deadline. If task i is specified to have a hard deadline, that is $TI_i = critical$, then if any non-critical task is guaranteed to meet its deadline, task i must be guaranteed to meet its deadline. Given an allocation where task i is not guaranteed to meet its deadline but another non-critical task will be guaranteed to meet its deadline, the tool should perform period transformation to elevate the priority of the critical task. The tool will automatically perform period transformation only if it is necessary. In addition,

a user may control the amount of period transformation that is performed on a task. A user can manually specify the number of components to transform the period of task i by means of TI_i. For example, if $TI_i = 3$, then the period of task i should be transformed into three subperiods.

- Task groups, *GID*. Some tasks may be required to execute as a group on the same processor. For these tasks there are no restrictions onto which processor to place the group, but all tasks in the group must be placed on the identical processor.

- A task must execute on a specific processor, *Pbind*. This requirement allows a user to explicitly allocate a task to a processor. In some cases the user knows that a particular task must always execute on a special processor.

- Jitter of task i, J_i. A feasible task is guaranteed to be completed by the end of its period. However, the exact completion time of a task may vary from period to period. Therefore, if a task sends a message at the end of its period, the time that messages are generated from the task will not be exactly periodic, but will have some irregularity of time from period to period, which is known as jitter. In order to overcome jitter, a completed task may buffer its message until its latest possible completion time.

- The granularity of the priorities assigned to tasks, P_G. The RMS algorithm orders the priorities of tasks. The granularity states the number of priority levels between the order determined by the RMS algorithm. If the granularity is 2, then the RMS algorithm assigns priorities to tasks with values of 2, 4, 6, $\ldots n$, where n is the highest priority task. A user has the flexibility to provide a manual assignment of a priority to a particular task by means of the *priority* parameter. The specified priority may violate the priority assigned by the RMS

algorithm. Regardless, DRMS will analyze the feasibility of allocations with user specified priorities.

A.2.2 Processor Specifications

A few parameters are specified for the processors serving the real-time system. The parameters include the number of processors in the system, the amount of memory available for tasks on each processor and the processing capacity of each processor. The processing capacity PS_k for processor k is the capacity of the processor normalized according to the demand C_i specified for tasks where $0 \leq i \leq NumTasks$. For example, a task with a computational requirement of 1 unit will execute for 1 unit on processor i if $PS_i = 1$. The task will execute for 0.5 unit if $PS_i = 2$. If a value of PS_i is not explicitly specified, it is assumed to be 1. If $PS_k = 2$ and $PS_l = 1$, then processor k can serve tasks at twice the rate of processor l. These parameters are identified in Table A.2.

Table A.2. Parameters for Processor Specification.

Parameter	Definition
$NumProc$	Number of processors in the system
PS_i	Computing capacity of processor i
MS_i	Amount of memory on processor i

A.2.3 Methods of Analysis

We implemented three schedulability tests in DRMS to determine if a task allocation can meet its deadlines. The tests are described below and are ordered from the most pessimistic assumptions for determining feasibility to the least pessimistic assump-

tions for determining feasibility. However, for all tests, if an allocation is determined to be feasible, then the deadlines of all tasks are guaranteed. The most pessimistic test is the *single inequality schedulability test.* The single inequality test for n tasks allocated to processor k, where $n \leq NumTask$, is given by

$$\sum_{i=1}^{n}\left(\frac{C_i + 2 * C_{si}}{PS_k * T_i}\right) + max_{i=1}^{n-1}\left((1 - \frac{D_i}{T_i}) + \frac{B_i}{T_i} + \frac{J_i}{T_i}\right) \leq n * (2^{1/n} - 1), \quad \text{(A.6)}$$

$$and \quad \sum_{i=1}^{n} MEM_i \leq MS_k, \quad \text{(A.7)}$$

where C_{si} is the task switching overhead for task i. The single inequality test is an extension of Liu and Layland's [66] worst-case analysis for the RMS algorithm. C_{si} is $ITST_i$ if task i has specified an individual switch time. Otherwise C_{si} is TST. The equation shows C_{si} multiplied by two, since one context switch is represented at the beginning of the task execution and one context switch is represented at the end of the task execution. The denominator of the first term in Equation A.6 contains PS_k, which indicates that the processing demand required by tasks on processor k is divided appropriately by the rate that processor k can serve tasks. This enables the feasibility test to be used for distributed environments composed of processors with heterogeneous processing capacities. For every processor in the system, the memory requirements of the tasks assigned to the processor must be satisfied, which is represented in Equation A.7.

We have modified the analysis described in Equation A.5 to incorporate deadlines that are less than the period. If the deadline is equal to the period, then the term (1-D_i/T_i) in Equation A.6 is reduced to zero. Otherwise, the early deadline can be considered an additional blocking time. The test is pessimistic since it assumes that the task with the worst blocking time and the worst effect due to a pre-period deadline is the task that must always be used for the cumulative utilization. Note that we only

need to accumulate the blocking time, the jitter and the effect due to early deadlines of one task, since other tasks can execute during the blocking time and the interval after a pre-period deadline.

A less pessimistic test for RMS feasibility is the *multiple inequality test.* The multiple inequality test for n tasks allocated to processor k is given by

$$\sum_{i=1}^{1}\left(\frac{C_i + 2 * C_{si}}{PS_k * T_i}\right) + \left(1 - \frac{D_1}{T_1} + \frac{B_1}{T_1} + \frac{J_1}{T_1}\right) \leq 1 * (2^{1/1} - 1), \tag{A.8}$$

$$\sum_{i=1}^{2}\left(\frac{C_i + 2 * C_{si}}{PS_k * T_i}\right) + \left(1 - \frac{D_2}{T_2} + \frac{B_2}{T_2} + \frac{J_2}{T_2}\right) \leq 2 * (2^{1/2} - 1), \tag{A.9}$$

$$\cdots \quad \cdots \quad \cdots$$

$$\sum_{i=1}^{n}\left(\frac{C_i + 2 * C_{si}}{PS_k * T_i}\right) + \left(1 - \frac{D_n}{T_n} + \frac{B_n}{T_n} + \frac{J_n}{T_n}\right) \leq n * (2^{1/n} - 1), \tag{A.10}$$

$$\text{and} \quad \sum_{i=1}^{n} MEM_i \leq MS_k. \tag{A.11}$$

All inequalities must be satisfied for the task sets to be schedulable. This test is very similar to the single inequality test but the blocking times, jitter and early deadlines are considered individually. Only the lowest priority task for each inequality must consider the blocking time, jitter and an early deadline. These conditions are sufficient for a set of tasks to have a feasible schedule.

The *numerical test* is the least pessimistic test for determining feasibility of allocations that may have utilizations greater than the worst case bound given by Liu and Layland [66]. This is the test for exact schedulability described by Lehoczky, Sha and Ding [114] and cited in Equations A.1–A.4 in Section A.1, with modifications to include task switching times, memory requirements and processing capacities. The numerical test may show feasible schedules that are missed by the single and multiple inequality tests. The numerical test considers scheduling points, where the end of

a task's period is a scheduling point. A higher priority task may have several periods repeated within the period of a lower priority task and therefore have scheduling points inserted in the list of scheduling points for a lower priority task. There is an inequality associated with each scheduling point. For a given task, the numerical test uses only those scheduling points that are less than or equal to the task's period. For each task, each scheduling point (less than or equal to the task's period) is examined to determine if the execution time, overhead time and blocking associated with the task can be satisfied before the scheduling point. For example, suppose task τ_1 has $T_1 = 7$, τ_2 has $T_2 = 10$ and τ_3 has $T_3 = 22$. The set of scheduling points considered when evaluating the feasibility of task τ_1 is $\{7\}$. The set is $\{7, 10\}$ when evaluating the feasibility of τ_2 and is $\{7, 10, 14, 20, 21, 22\}$ when evaluating the feasibility of τ_3.

Since many feasible allocations may be produced when RMS analysis is performed on a set of tasks allocated to a set of processors, the user should be able to order the feasible schedules according to different criteria. A user can examine the allocation on the basis of the lowest mean utilization of the processors, the lowest variance of the utilizations of the processors, or the smallest difference between the least and the most utilized processors.

A.3 The Branch-and-Bound Algorithm for Task Allocation

We implemented a method to allocate a set of tasks to a set of processors in order to analyze the feasibility of the allocation. Our method for finding allocations will discover any existing feasible allocation. If there are no feasible means of allocating the set of tasks to the set of processors, then our method will find allocations that are the best of the infeasible solutions.

In order to find feasible allocations, we use the *Branch-and-Bound (BB)* algorithm,

which is described in Section 4.4.3. The BB algorithm has been used extensively by researchers for the processor allocation and scheduling problem. Ma, Lee and Tsuchiya [117] described its use for task allocation in a distributed system that serves real-time tasks that were scheduled before execution. The use of the BB algorithm for precedence-constrained periodic tasks has been explored in [38]. These researchers did not target their methods for allocating tasks to processors with the basic assumption that each processor would serve the tasks assigned to it using the RMS algorithm, as we require from our tool.

A basic BB algorithm uses a task assignment tree to generate feasible allocations. An example of a task assignment tree is given in Figure 4.3. The BB algorithm traverses the task assignment tree in a depth-first fashion to allocate the tasks to the processors and to determine if the given allocation can meet the periodic deadlines. If the BB algorithm determines that a task assignment exceeds the bounding condition, then it is known that the current task assignment does not lead to a feasible schedule. The algorithm records an infeasible allocation if it is better than any of the current allocations stored, up to a user defined limit of the number of infeasible allocations that will be stored. Then, the BB algorithm backtracks up the tree one level and searches depth-first with a new untested branch. If it is determined at one level that the bounding conditions are not exceeded, the search continues down the current path. When the search has successfully traversed to a leaf of the tree, a feasible task assignment is found.

Unless a user explicitly specifies which feasibility test to use, the algorithm begins searching for feasible schedules using the single inequality test as the bounding condition at each node in the task assignment tree. If no feasible schedules are found in the BB search, then the BB algorithm uses the multiple inequality test as the bounding condition. Likewise, if no feasible schedules are found, then the BB algorithm attempts to find feasible schedules using the numerical test as the bounding

condition.

It is possible that a large number of feasible schedules can be found. Since information for every allocation can consume valuable disk space, the user may limit the number of feasible schedules stored. For a large number of tasks and processors, the search could consume an extremely large amount of time in order to find all possible allocations since the complexity of the BB algorithm is $O(NumProc^{NumTask})$ for $NumProc$ processors and $NumTask$ tasks. For example, we tested an example of 16 processors and 40 tasks that required 2 days on a Sun Sparcstation 2 to find all the 1.46^{48} feasible allocations. Since it is not expected that a user wants to know all feasible allocations nor is it expected that a user wants to wait two days to obtain the results, a user can specify time limits of how long the search may continue before it is terminated. For example, the user can specify that DRMS should provide all feasible allocations it can find within two minutes. In addition, a user can specify a limit on the number feasible allocations to find. This means a user can specify that the search should stop as soon as the first feasible allocation is found. For most practical purposes, the user wants to obtain only a few feasible allocations and does not need all possible allocations. When only a few feasible allocations exist, the branch-and-bound algorithm quickly eliminates the majority of the branches in the search tree. Although the time complexity of the branch-and-bound algorithm may appear distressing for a user, the limits that may be placed on the search time and the number of allocations needed makes the algorithm very practical for use in DRMS. Most of the examples we have considered had less than 30 tasks and 8 processors. On a Sun Sparcstation 2 workstation, the tool completed its search within a minute for all feasible allocations for the majority of these examples.

The order that untested branches are searched may affect the amount of elapsed time for the tool to find a feasible schedule. Each branch represents a different processor to place a task. At each level in the search tree the next untested branch

that is selected for searching is a branch that has the lowest accumulated processor utilization. This means that the initial strategy for allocating tasks to processors balances the utilizations of all processors. The search algorithm will consider all branches if the user wants the tool to find all possible feasible allocations. Tasks are inserted into the BB task assignment tree starting from the highest priority task to the lowest priority task. Priorities are determined by the RMS algorithm unless the user has specified the priority of a task that might deviate from the RMS priority order. If a user manually specifies that a task should have its period transformed, the period transformation will be done before the BB algorithm starts building the task assignment tree.

Users can specify that period transformations can be performed on critical tasks while they do not manually specify how the transformations should be performed. The BB algorithm must perform an extra check when a task assignment tree is built for which all tasks in a task set cannot be feasibly scheduled. If no feasible allocations can be found, the algorithm must examine whether any of the non-feasible tasks are critical. For all critical tasks that are not feasible, the algorithm performs period transformation on the critical tasks in order to elevate their priorities above less critical tasks that have been determined to be feasible. Next, the algorithm performs the RMS analysis to determine if all critical tasks are feasible. If again there are some non-critical tasks that are feasible while critical tasks are not, the algorithm performs period transformation on all critical tasks, including those that are feasible and are lower in the RMS priority than some non-critical task. If no allocation can be found such that all critical tasks are feasible, it is determined that there are no successful allocations.

A.4 User Interface for DRMS

The user interface helps the user define the real-time system parameters for processors and tasks and display the results of RMS feasibility analysis. The user interface consists of two parts: the input user interface and the output user interface. We built the interface for the X window system using routines provided in Xlib and by the Athena toolkit [118]. These routines enable portability between systems that support the X window system, regardless of the window toolkit built on the X window system supported by a workstation. This section briefly introduces the interface of the DRMS tool.

A.4.1 Input User Interface

The input user interface enables the user to specify real-time system parameters for which feasible allocations of tasks to processors will be found. For a description of how the tool is used, suppose the file that is loaded into the tool describes three processors as presented in Table A.3. Each processor has an associated name, memory size and processing speed.

Table A.3. Processor Parameters.

Procesor Name	Memory Size	Speed
display processor	7000	0.65
signal processor	8192	1.0
mission processor	4384	1.5

A similar interface is provided for the user to load task descriptions. Suppose the task descriptions of 16 tasks are loaded by the tool as given in Table A.4. Notice that each task has specified computational demand (C), period (T), size (MEM),

individual context switching time (ITST), pre-period deadline (PPD), blocking time (B), jitter (J), task importance (TI), group identification (GID), processor binding (PBind) and priority. The entries that do not have a value specified means that the particular parameter is ignored for the task. The default value for task switching time in this example is 0.623 units of time and the default size of a task is 10 units of memory. Notice that only the tasks "Timer_Intrpt" and "Radar_Trgt_Upd" have context switch times different from the default value. The tasks "Camera_Snapshot", "Radar_Trcking_Fltr" and "Dsply_Stores_Upd" have pre-period deadlines. All other tasks have deadlines at the end of their periods.

Table A.4. Task Parameters.

Task Specifications (Default task context switch time, TST is 0.623)											
Task Name	C	T	MEM	ITST	PPD	B	J	TI	GID	PBind	Pri
Timer_Intrpt	0.051	10	3	0.1		0.000				1	
Camera_Snapshot	10.546	600	50		580	55.32		0	1	2	5
Radar_Trcking_Fltr	17.846	75	40		70	0.15					
RWR_Cntct_MGM	27.146	75				0.15					
Bus_Poll_Dvc	35.596	120				0.15	0.5				
Bit_E_Stat_Upd	254.546	3000	110			0.00	0.1	0			
Camera_Aim	21.596	150				0.15		3	1		
Radar_Trgt_Upd	66.996	150		1		0.15		0			
Nav_Upd	35.996	177	90			0.15					
Dsply_Graphic	49.596	240				0.15			2	0	3
Dsply_Hook_Upd	37.696	240				0.15			2		
Trck_trgt_upd	87.296	300				0.15					
Nav_String_CMDS	190.696	600	55			0.15	0.11				
Dsply_Stores_Upd	54.696	600	20		570	0.15			2		
Dsply_Keyset	24.696	600	30			0.15					
Dsply_Stat_Upd	21.596	600				0.15		6	2		

In order to specify new definitions for tasks and processors within the graphical user interface or change the value of existing parameters, the options "Global," "Processor," and "Task" are available for the user. The "Global" option is used to change the number of processors and tasks in the system, as well as to specify default parameters for the processors such as speed and memory size, and parameters for

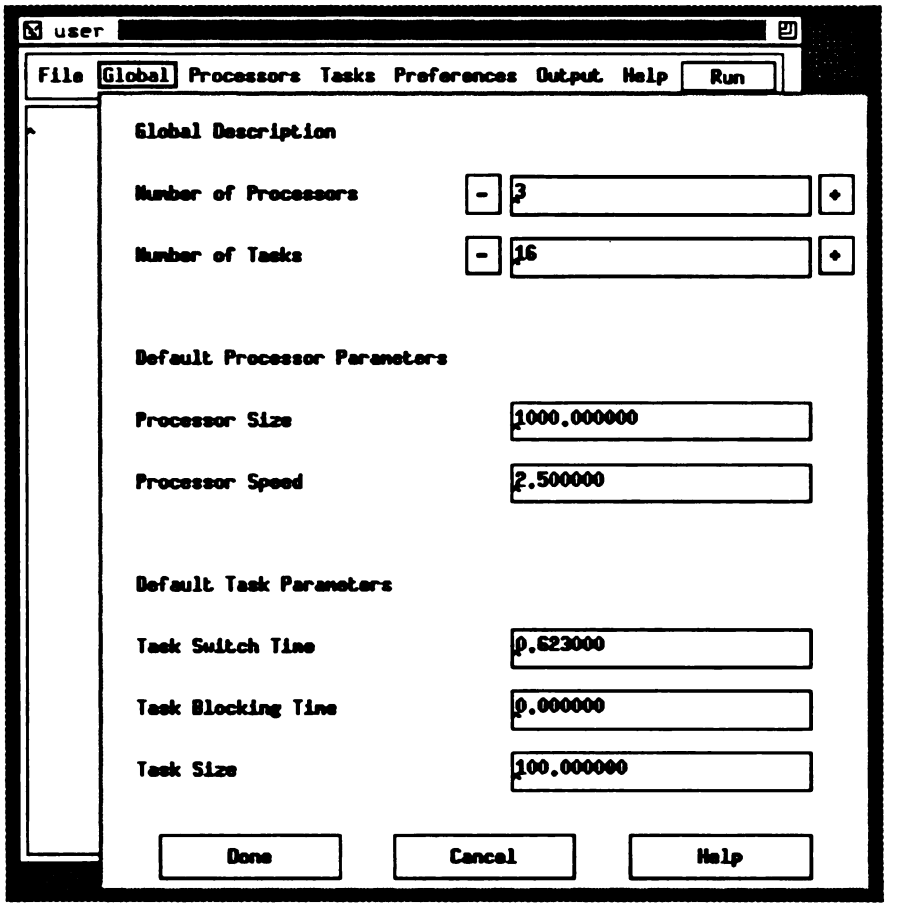


Figure A.1. Specifying Global Parameters.

tasks such as context switch times and blocking times. Figure A.1 shows the use of the "Global" option. Parameters of individual processors can be changed by selecting the "Processor" option in the main menu. Similarly, individual task parameters can be changed by selecting the "Task" option in the main menu.

The BB algorithm begins searching for feasible allocations of tasks to processors after the user selects the "Run" button. Before running the BB algorithm, the user may want to specify command line options such as the specific test to use, whether to use period transformation, or the time limit for searching for allocations. These options are specified when the "Preferences" option is selected on the main menu, as illustrated in Figure A.2. The "Run" button changes to a "Stop" button when the

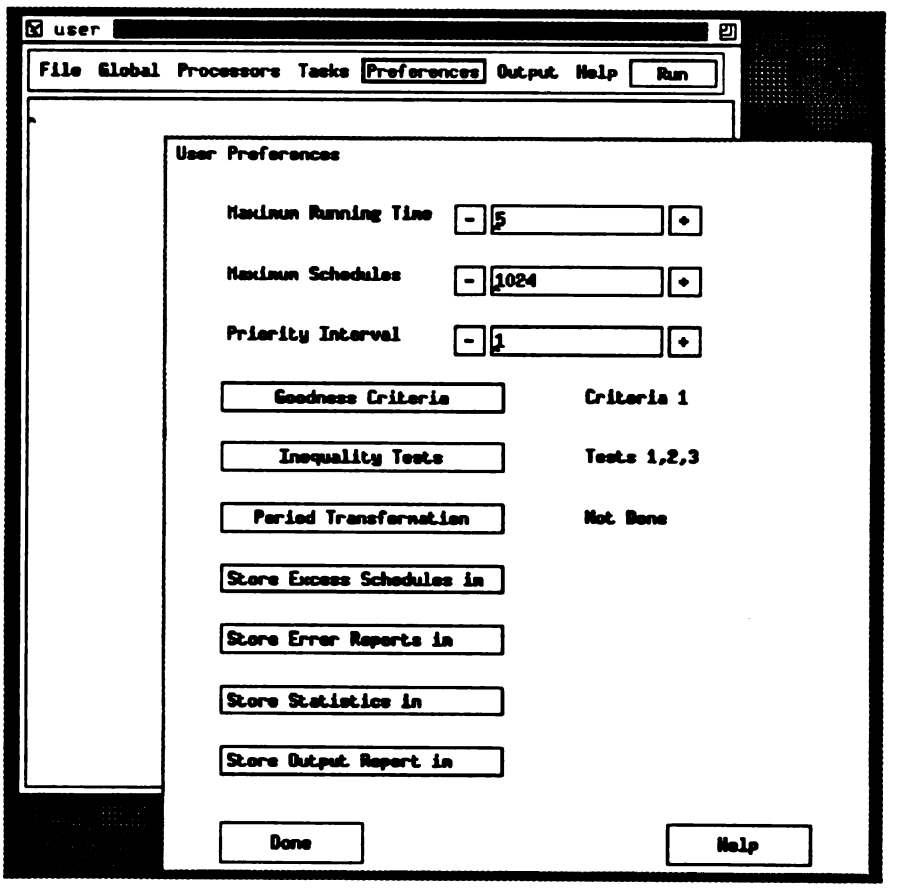


Figure A.2. Specifying Execution Preferences.

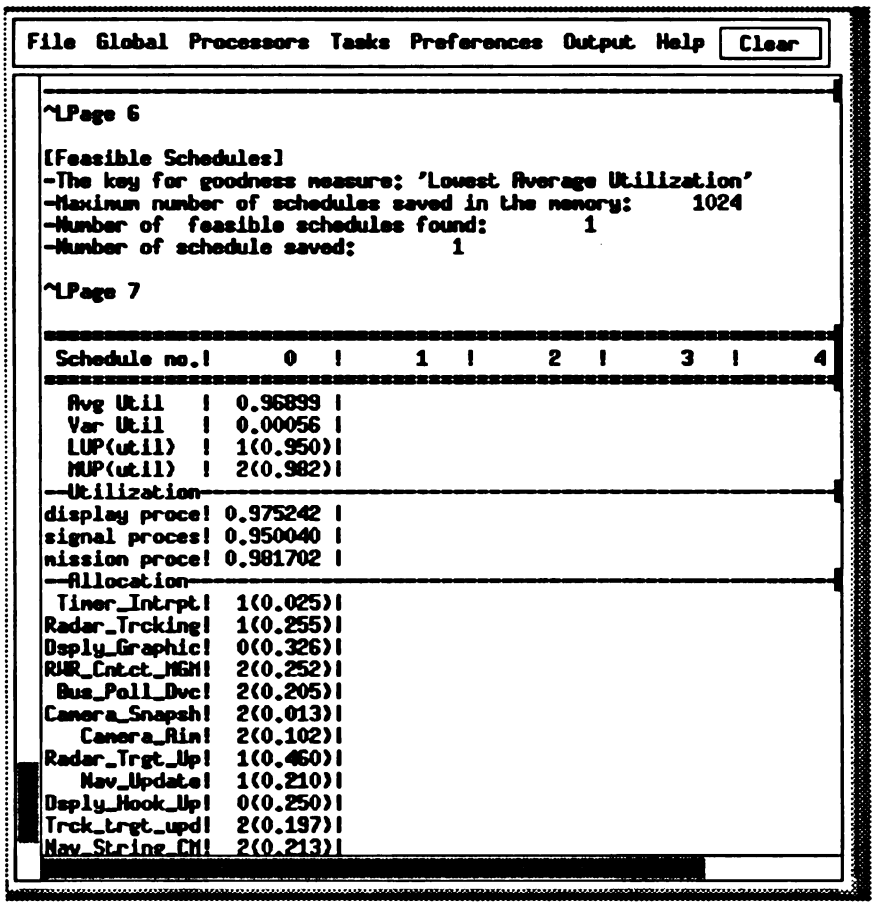


Figure A.3. Execution Results.

BB algorithm starts running. This enables a user to abort the tool in its search for feasible allocations.

When the DRMS tool completes execution and the allocations are generated, a report of the feasible allocations is listed in the window as shown in Figure A.3. This example required less than 30 seconds on a Sun Sparcstation 2 workstation. Notice that the tool has found that only one feasible allocation exists for the currently defined set of tasks and processors. The display shows statistics about the allocation, which include the utilization of each processor and how much each tasks contributes to the utilization of a processor. Notice in the example that the utilization of each processor is greater than 95%, which means the single and multiple inequality tests failed to

find a feasible allocation. Only the numerical test could find a feasible allocation. When the user specifies that a graphical display is needed, the tool presents the user with the output user interface.

A.4.2 Output User Interface

The output interface displays the periods during which various tasks are allocated on various processors along a horizontal time-line. The user may select any point on the time-line and determine the task that has priority for execution at that point according to the feasibility analysis. When the user expands the vertical scale sufficiently such that the allocation of only one processor remains visible on the screen, the output interface marks important points on the time-line, such as points of task execution, context switching and blocking times.

The user can obtain additional information, such as the parameters of a specific processor and tasks assigned to the processor, by "clicking" on the corresponding processor box on the screen. Figure A.4 shows the parameters of the "mission processor" and the parameters of the six tasks allocated to the "mission processor." The task parameters shown in Figure A.4 are the task execution time (TET), the task period (TP), the blocking time (TBT) and the context switching time (TST).

The output user interface provides facilities for the user to see the effect of moving a task from one processor to another. When the user presses the "Move" button on the main menu, the list of processors and the tasks allocated to them are displayed on the screen.The user can select a task for moving and then select a processor for the task. If the user selects the "Redisplay" or the "Analyze" key, the time-line corresponding to the modified allocation are displayed on the screen.

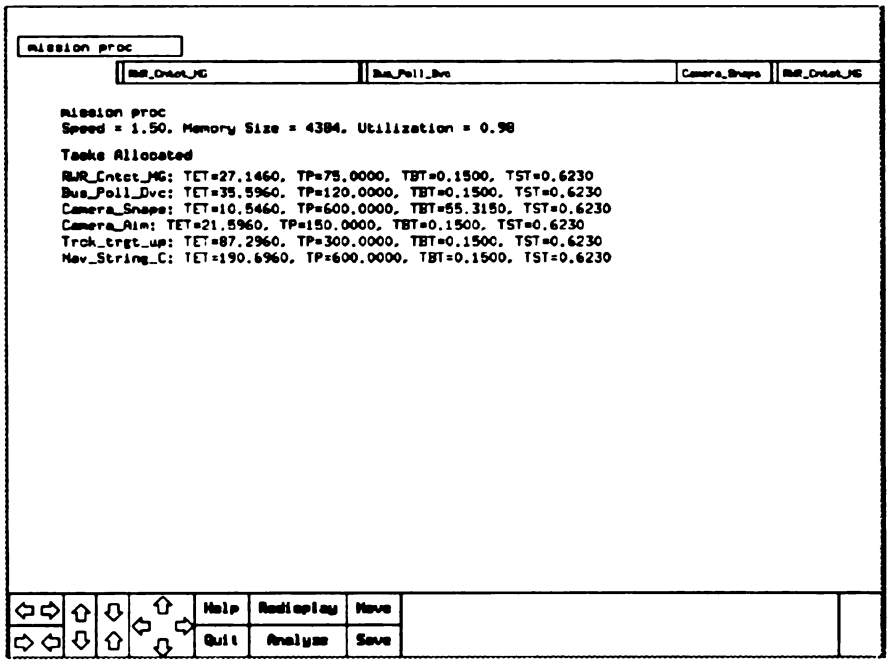


Figure A.4. RMS analysis on the Display Processor.

A.5 An Example Usage of DRMS for Scheduling Periodic Tasks with Communication Requirements

The usage of DRMS is not limited to schedule independent periodic tasks. Periodic tasks that execute on parallel and distributed systems generate communication traffic at regular intervals. This type of communication traffic can be scheduled by the RMS scheduling algorithm. The RMS scheduling algorithm has been applied to IEEE 802.5 token rings [65], multi-hop networks [86] and wormhole networks [92]. In this section, an example of using DRMS for scheduling periodic tasks and their communication traffic on parallel and distributed real-time systems is illustrated.

Suppose a set of periodic tasks with communication requirements is to be scheduled to a parallel or distributed system. DRMS can find feasible allocations of the

tasks in two phases. In the first phase, DRMS produces all the possible allocations that the tasks can be successfully executed without communication overhead. As tasks are allocated to processors, in the second phase DRMS is used to analyzed the communication traffic generated by the tasks. If the communication traffic of a possible allocation passes the RMS tests, a feasible schedule that satisfies both the tasks' computational demands and communication requirements is found.

Figure A.5 describes the usage of DRMS for scheduling a set of periodic tasks with communication requirements. A system designer supplies DRMS with a network description and a task set description. A network description consists of the network topology and the information of processors connected by the network. A task description includes the task description and the communication requirements of each task. DRMS produces possible schedules without considering communication requirements in the first phase. In a possible schedule, tasks, which generate messages, become message sources. A message source description is generated according to the communication requirements of tasks and message source allocations. According to the message source description and the network description, DRMS analyzes the feasibility of the message source allocations and finds all the feasible schedules which satisfy both the tasks' computational demands and communication requirements. A network simulator, such as the wormhole network simulator described in Section 6.3, can also be used to verify the feasibility of message source allocations.

A.6 Summary

Rate monotonic scheduling technology has evolved such that researchers have found ways to use the technology in a wide variety of practical real-time system applications. DRMS is a powerful tool that was built after we consulted with engineers who develop real-time applications. Our work with real-time engineers enabled DRMS to become

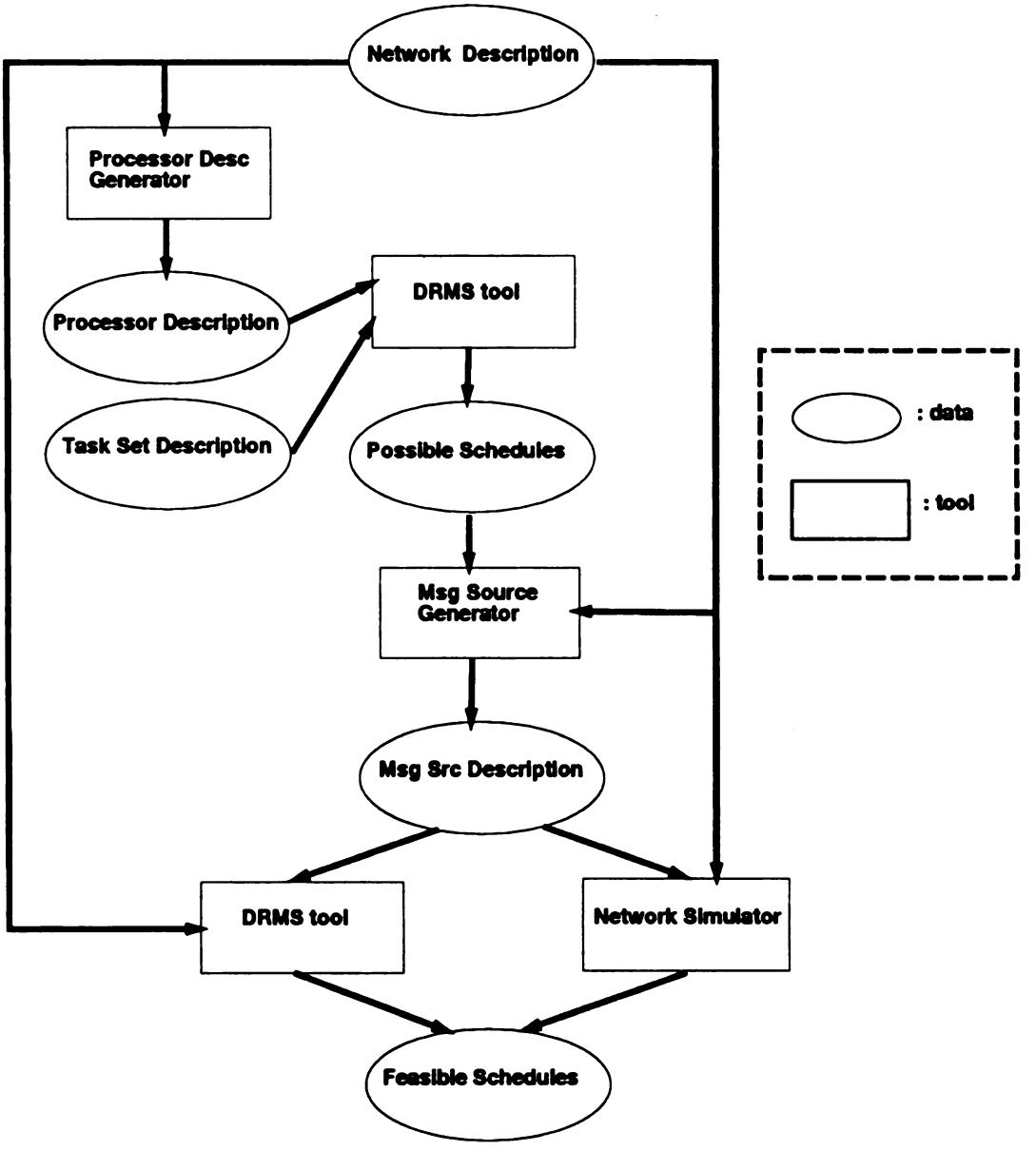


Figure A.5. Scheduling periodic tasks and their communication traffic using DRMS.

a practical tool for designers and engineers who apply RMS technology. For a set of tasks and processors, DRMS finds feasible allocations of the tasks to processors such that real-time constraints are guaranteed. If there exists any feasible allocation, DRMS will find the allocation. In addition, DRMS allows users to try different scenarios of task allocations for which DRMS determines the feasibility of the scenario. Users can adjust practical parameters such as memory and processing capacities, priority assignments, and many other parameters so that the users can apply rate monotonic scheduling in a distributed environment that serves real-time applications.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] S.-T. Levi and A. K. Agrawala, *Real-Time System Design.* McGraw-Hill Publishing Company, 1990.

[2] A. Burns and A. Wellings, *Real-Time Systems and Their Programming Languages.* Addison-Wesley Publishing Company, 1989.

[3] M. Dertouzos, "Control Robotics: The Procedural Control of Physical Processes," in *Proc. of the IFIP congress*, pp. 807–813, 1974.

[4] J. A. Stankovic, "Misconceptions About Real-Time Computing: A Serious Problem for Next-Generation Systems," *IEEE Computer*, vol. 21, pp. 10–19, 8 1988.

[5] P. A. Laplante, *Real-Time Systems Design and Analysis.* IEEE Computer Society Press, 1993.

[6] J. Y.-T. Leung, "Research in Real-Time Scheduling," in *Foundations of Real-Time Computing: Scheduling and Resource Management*, Kluwer Academic Publishers, 1991.

[7] G. Conte and D. D. Corso, *Multi-Microprocessor Systems for Real-Time Applications.* D. Reidel Publishing Company, 1985.

[8] J. A. Stankovic, "Real-Time Computing Systems: The Next Generation," in *Hard Real-Time Systems*, IEEE Computer Society Press, 1988.

[9] J. A. Stankovic and K. Ramamritham, "The Spring Kernel: A New Paradigm for Real-Time Systems," *IEEE Software*, vol. 8, pp. 62–72, May 1991.

[10] S. Dutt and J. P. Hayes, "Some Pratical Issues in the Design of Fault-Tolerant Multiprocessors," in *IEEE Proc. of Intl. Symp. on Fault-Tolerant Computing*, pp. 292–299, 1991.

[11] B. E. Aupperle and J. F. Meyer, "State Space Generation for Degradable Multiprocessor Systems," in *IEEE Proc. of Intl. Symp. on Fault-Tolerant Computing*, pp. 308–315, 1991.

[12] J. D. Ullman, "Complexity of Sequence Problem," in *Computer and Job-Shop Scheduling Theory* (E. G. Coffman, ed.), J. Wiley, 1976.

[13] A. K. Mok and M. L. Dertouzos, "Multiprocessor Scheduling in a Hard Real-Time Environment," in *Proc. 7th Texas Conf. on Computing Systems*, 1978.

[14] S. Cheng, J. A. Stankovic, and K. Ramamritham, "Dynamic Scheduling of Groups of Tasks with Precedence Constraints in Distributed Hard Real-Time Systems," in *Proc. of Real-Time Symp.*, pp. 166–174, IEEE, 1986.

[15] W. J. Dally and C. L. Seitz, "The Torus Routing Chip," *Journal of Distributed Computing*, vol. 1, no. 3, pp. 187–196, 1986.

[16] W. J. Dally, "Virtual-Channel Flow Control," *IEEE Trans. on Parallel and Distributed Systems*, vol. 3, pp. 194–205, March 1992.

[17] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of The ACM*, vol. 20, pp. 46–61, Jan. 1973.

[18] L. Sha, J. P. Lehoczky, and R. Rajkumar, "Solutions for Some Practical Problems in Prioritized Preemptive Scheduling," in *Proc. of Real-Time System Symp.*, pp. 181–191, IEEE, 1986.

[19] J. Lehoczky, L. Sha, and Y. Ding, "The Rate Monotonic Scheduling Algorithm Exact Characterization and Average Case Behavior," in *Proc. of Real-Time System Symp.*, pp. 166–171, IEEE, 1989.

[20] J. Leung and J. Whitehead, "On Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks," in *Performance Evaluation,2*, pp. 237–250, 1982.

[21] J. Lehoczky and L. Sha, "Performance of Real-Time Bus Scheduling Algorithms," in *Performance Evaluation Review, 14*, ACM, 1986.

[22] D.-T. Peng and K. Shing, "A New Performance Measure for Scheduling Independent Real-Time Tasks," technical report, Real-Time Computing Laboratory, University of Michigan, 1989.

[23] J. Lehoczky, "Fixed Priority Scheduling of Jobs with Arbitrary Deadlines," in *Proc. of Real-Time System Symp.*, pp. 201–209, IEEE, 1990.

[24] L. Sha, R. Rajkumar, and J. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Trans. on Computers*, vol. 39, pp. 1175–1185, Sept. 1990.

[25] K. Lin, S. Natarajan, J. W. S. Liu, and T. Krauskopf, "Concord: A System of Imprecise Computation," in *Proc. of Compsac*, pp. 75–81, IEEE, 1987.

[26] K. Lin, S. Natarajan, and J. W. S. Liu, "Imprecise Results: Utilizing Partial Computations in Real-Time Systems," in *Proc. of Real-Time System Symp.*, IEEE, 1987.

[27] J. W. S. Liu, K. J. Lin, and S. Natarajan, "Scheeduling Real-Time, Periodic Jobs Using Imprecies Results," in *Proc. of Real-Time System Symp.*, pp. 252–260, IEEE, 1987.

[28] J. Y. Chung, J. W. S. Liu, and K. J. Lin, "Scheduling Periodic Jobs that Allow Imprecise Results," *IEEE Trans. on Computers*, vol. 19, pp. 1156–1173, Sept. 1990.

[29] S. K. Dhall and C. L. Liu, "On A Real-Time Scheduling Problem," *Oper. Res*, vol. 26, pp. 127–140, Jan. 1978.

[30] W. A. Horn, "Some Simple Scheduling Algorithms," in *Naval Research Logistics Quarterly, 21*, pp. 177–185, 1974.

[31] C. Martel, "Preemptive Scheduling with Release Time, Deadlines and Due Times," *Journal of The ACM*, vol. 29, pp. 812–829, Jul. 1982.

[32] J. A. Bannister and K. S. Trivedi, "Task Allocation in Fault-Tolerant Distributed Systems," in *Acta Informatica*, Springer-Verlag, 1983.

[33] S. Davari and S. K. Dhall, "An On-Line Algorithm for Real-Time Task Allocation," in *Proc. of Real-Time System Symp.*, IEEE, 1986.

[34] T. Lang and E. B. Fernandez, "Scheduling of Unit-Length Independent Tasks with with Execution Constraints," *Information Processing Letters*, vol. 4, no. 4, 1976.

[35] B. Simons, "A Fast Algorithm for Multiprocessor Scheduling," in *Proc. 21st Annual Symp. on Foundation of Computer Science*, 1980.

[36] B. Simons, "Multiprocessor Scheduling of Unit-Time Jobs with Arbitrary Release Times and Deadlines," *SIAM Journal for Computing*, vol. 12, no. 2, 1983.

[37] B. Simons and M. Sipser, "On Scheduling Unit-Length Jobs with Multiple Release Time/Deadline Intervals," *Operations Research*, vol. 32, no. 1, 1984.

[38] K. Ramamritham, "Allocation and Scheduling of Complex Periodic Tasks," in *Proc. of 1990 Int. Conference on Distributed Computing Systems*, pp. 108–115, IEEE, 1990.

[39] R. R. Muntz and J. Coffman, E. G., "Preemptive Scheduling of Real-Time Tasks on Multiprocessor Systems," *Journal of ACM*, vol. 17, pp. 324–338, 4 1970.

[40] G.-H. Chen and J.-S. Yur, "A Branch-and-Bound-with-Underestimates Algorithm for the Task Assignment Problem with Precedence Constraint," in *Proc. of Intl. Conf. on Distributed Computing Systems*, pp. 494–501, IEEE, 1990.

[41] P.-Y. R. Ma, E. Y. S. Lee, and M. Tsuchiya, "A Task Allocation Model for Distributed Computing Systems," *IEEE Trans. on Computers*, vol. c-31, pp. 41–47, 1 1982.

[42] D.-T. Peng and K. G. Shin, "Modeling of Concurrent Tasks Execution in a Distributed System for Real-Time Control," *IEEE Trans. on Computers*, vol. c-36, pp. 500–516, 4 1987.

[43] D.-T. Peng and K. G. Shin, "Static Allocation of Periodic Tasks with Precedence Constraints in Distributed Real-Time Systems," in *Proc. of Intl. Conf. on Distributed Computing Systems*, pp. 190–198, IEEE, 1989.

[44] K. Ramamritham, "Allocation and Scheduling of Complex Periodic Tasks," in *Proc. of Intl. Conf. on Distributed Computing Systems*, pp. 108–115, IEEE, 1990.

[45] K. S. Hong and J. Y.-T. Leung, "On-Line Scheduling of Real-Time Tasks," in *Proc. of Real-Time System Symp.*, pp. 244–250, IEEE, 1988.

[46] M. L. Dertouzos and A. K. Mok, "Multiprocessor On-Line Scheduling of Hard-Real-Time Tasks," *IEEE Trans. on Software Engineering*, vol. 15, pp. 1497–1506, Dec. 1989.

[47] K. Ramamritham and J. A. Stankovic, "Dynamic Task Scheduling in Distributed Hard Real-Time Systems," *IEEE Software*, vol. 1, pp. 65–75, Jul. 1984.

[48] W. Zhao, K. Ramamritham, and J. A. Stankovic, "Scheduling Tasks with Resource Requirements in Hard Real-Time Systems," *IEEE Trans. on Software Engineering*, vol. 12, pp. 564–577, May 1987.

[49] W. Zhao, K. Ramamritham, and J. A. Stankovic, "Preemptive Scheduling Under Time and Resource Constraints," *IEEE Trans. on Computers*, vol. 36, pp. 949–960, Aug. 1987.

[50] J. A. Stankovic, K. Ramamritham, and S. Cheng, "Evaluation of a Flexible Task Scheduling Algorithm for Distributed Hard Real-Time Systems," *IEEE Trans. on Computers*, vol. c-34, 12 1985.

[51] K. Ramamritham, J. A. Stankovic, and P.-F. Shiah, "*O(n)* Scheduling Algorithms for Real-Time Multiprocessor Systems," in *Proc. of Intl. Conf. on Parallel Processing*, pp. III–143–151, IEEE, 1989.

[52] K. G. Shin and Y.-C. Chang, "Load Sharing in Distributed Real-Time Systems with State-Change Broadcasts," *IEEE Trans. on Computers*, vol. c-38, pp. 1124–1142, 8 1989.

[53] B. A. Blake and K. Schwan, "Experimental Evaluation of a Real-Time Scheduler for a Multiprocessor System," *IEEE Trans. on Software Engineering*, vol. 17, pp. 34–44, Jan. 1991.

[54] J. F. Kurose, M. Schwartz, and Y. Yemini, "Multiple-Access Protocols and Time-Constrained Communication," *Computing Surveys*, vol. 16, pp. 43–70, Mar. 1984.

[55] J. G. Gruber, "Delay Relatd Issues in Intergated Voice and Data Networks," *IEEE Trans. on Communication*, vol. COM-29, pp. 786–800, Jun. 1981.

[56] E. Arthurs and B. W. Stuck, "A Theoretical Traffic Performance Analysis of an Intergrated Voice-Data Virtual Circuit Packet Switch," *IEEE Trans. on Communication*, vol. COM-27, pp. 1104–1111, Jul. 1979.

[57] O. A. Mowafi and W. J. Kelly, "Intergated Voice/Data Packet Switching Techniques for Future Military Networks," *IEEE Trans. on Communication*, vol. COM-28, pp. 1655–1662, Sept. 1980.

[58] M. J. Fischer and T. C. Harris, "A Model for Evaluating the Performance of an Intergraged Circuit and Packet Switched Multiplex Structure," *IEEE Trans. on Communication*, vol. COM-24, pp. 195–202, Feb. 1976.

[59] B. Maglaris and T. Lissack, "A Priority TDMA Protocol for Satellite Data Communication," in *IEEE International Communication Conference*, pp. 73.3.1–73.3.5, 1981.

[60] A. Damm, J. Reisinger, W. Schwabl, and H. Kopetz, "The Real-Time Operating System of MARS," *ACM Operating Systems Review*, vol. 23, no. 3, pp. 141–151, 1989.

[61] L. Kleinrock and M. O. Scholl, "Packet Switching in Radio Channels: New Conflict-Free Multiple Access Schemes," *IEEE Trans. on Communication*, vol. COM-28, pp. 1015–1029, Jul. 1980.

[62] J. C. Valadier and P. D. R., "On CSMA Protocols Allowing Bounded Channel Access Times," in *Proc. of Intl. Conf. on Distributed Computing Systems*, IEEE, 1984.

[63] A. Moura and J. Field, "Collision Ccontrol Algorithms in CSMA-CD Networks," *Computer Communication*, vol. 4, Feb. 1981.

[64] L. Sha, S. S. Sathaye, and J. K. Strosnider, "Scheduling Real-Time Communication on Dual-Link Networks," in *Proc. of Real-Time System Symp.*, pp. 188–197, IEEE, 1992.

[65] J. K. Strosnider, T. E. Marchok, and J. Lehoczky, "Advanced Real Time Scheduling Using the IEEE 802.5 Token Ring," in *Proc. of the IEEE Real-Time Systems Symposium*, pp. 42–52, 1988.

[66] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[67] J. K. Strosnider and T. E. Marchok, "Responsive, Deterministic IEEE 802.5 Token Ring Scheduling," *Real-Time Systems Journal*, Sep 1989.

[68] Y.-H. Lee and L.-T. Shen, "Real-Time Communication in Multiple Token Ring Networks," in *Proc. of the IEEE Real-Time Systems Symposium*, pp. 146–154, 1990.

[69] B. Chan, G. Agrawal, and W. Zhao, "Optimal Synchronous Capacity Allocation for Hard Real-Time Communications with the Timed Token protocol," in *Proc. of Real-Time System Symp.*, pp. 198–207, IEEE, 1992.

[70] K. C. Sevcik and M. J. Johnson, "Cycle Time Properties of The FDDI Token Ring Protocol," *IEEE Trans. on Software Engineering*, vol. 13, pp. 376–385, Mar. 1987.

[71] Y.-H. Lee and L.-T. Shen, "Real-Time Communication in Multiple Token Ring Networks," in *Proc. of the IEEE Real-Time Systems Symposium*, pp. 146–154, 1990.

[72] J. F. Kurose and M. Schwartz, "A Family of Window Protocols for Time-Constrained Application in CSMA Networks," in *INFOCOM 83*, pp. 405–413, 1983.

[73] J. F. Kurose, M. Schwartz, and Y. Yemini, "Controlling Time Window Protocols for Time-Constrained Communication in A Multiple Access Environment," in *8th International Data Communication Symposium*, pp. 75–84, 1983.

[74] S. Panwar, D. Towsley, and J. Wolf, "Optimal Scheduling Policies for A Class of Qeues with Customer Deadlines to The Beginning of Service," *Journal of The ACM*, vol. 35, Oct. 1988.

[75] W. Zhao, J. A. Stankovic, and K. Ramamritham, "A Multiaccess Window Protocol for Time Constrained Communications," in *Proceedings of the 8th International Conference on Distributed Computing Systems*, IEEE, Jun 1991.

[76] W. Zhao, J. A. Stankovic, and K. Ramamritham, "A Window Protocol for Time Constrained Messages," *IEEE Transactions on Computers*, vol. 39, pp. 1186–1203, Sep 1990.

[77] W. Zhao and K. Ramamritham, "A Virtual Time CSMA Protocol for Hard Real-Time Communications," in *Proc. of the IEEE Real-Time Systems Symposium*, pp. 120–127, 1986.

[78] W. Zhao and K. Ramamritham, "Virtual Time CSMA Protocol for Hard Real-Time Communications," *IEEE Transactions on Software Engineering*, vol. SE-13, pp. 938–952, Aug 1987.

[79] M. L. Molle and L. Kleinrock, "Virtual Time MSMA: Why Two Clocks Are Better Than One," *IEEE Trans. on Communication*, vol. COM-33, Sept. 1985.

[80] N. Malcolm and W. Zhao, "Version Selection Schemes for Hard Real-Time Communications," in *Proc. of the IEEE Real-Time Systems Symposium*, pp. 12–21, 1991.

[81] P. Kermani and L. Kleinrock, "Virtual Cut-through: A New Computer Communication switching Technique," *Computer Network*, vol. 3, no. 4, pp. 267–286, 1979.

[82] D. P. Anderson, S. Y. Tzou, R. Wahbe, R. Govindan, and M. Andrews, "Support for Continuous Media in the DASH System," in *Proc. of the 10th International Conference on Distributed Computing Systems*, pp. 54–61, 1990.

[83] D. P. Anderson and D. Ferrari, "An Overview of the DASH Project," Tech. Rep. UCB/CSD 88/406, Unvi. California, Berkeley, Feb. 1988.

[84] D. P. Anderson, "Metascheduling for Contiunous Media," *ACM Trans. on Computer Systems*, vol. 11, pp. 226–252, Aug. 1993.

[85] D. Ferrari and D. C. Verma, "A Scheme for Real-Time Channel Establishment in Wide-Area Networks," *IEEE Transactions on Selected Area in Communications*, vol. 8, no. 3, pp. 368–379, 1990.

[86] D. D. Kandlur, K. G. Shin, and D. Ferrari, "Real-time Communication in Multihop Networks," in *Proc. of the 11th International Conference on Distributed Computing Systems*, pp. 300–307, 1991.

[87] J. Y.-T. Leung, T. W. Tam, and G. H. Young, "On-Line Routing of Real-Time Messages," in *Proc. of the IEEE Real-Time Systems Symposium*, pp. 126–135, 1990.

[88] K. G. Shin, "HARTS: A Distributed Real-Time Architecture," *IEEE Computer*, vol. 24, pp. 25–35, May 1991.

[89] P. Ramanathan and K. G. Shin, "Delivery of Time-Critical Messages Using A Multiple Copy Approach," *ACM Trans. on Computer Systems*, vol. 10, pp. 144–166, May 1992.

[90] D. D. Kandlur and K. G. Shin, "Reliable Broadcast Algorithms for HARTS," *ACM Trans. on Computer Systems*, vol. 9, pp. 374–398, Nov. 1991.

[91] S. B. Shukla and D. P. Agrawal, "Allocation and Communication in Distributed Memory Multiprocessors for Periodic Real-Time Application," in *Proc. of Intl. Conf. on Parallel Processing*, pp. I–212–219, IEEE, 1991.

[92] M. W. Mutka, "Guaranteeing Deadline Constraints of Real-Time Traffic in a Wormhole Network," Technical Report CPS-92-16, Department of Computer Science, Michigan State University, 1992.

[93] R. L. Cruz, "A Calculus for Network Delay and a Note on Toplogies of Interconnection Networks," Tech. Rep. UILU-ENG-87-2246, University of Illinois at Urbana-Champaign, 1987.

[94] G. Kar, C. N. Nikolaou, and J. Reif, "Assigning Processes to Processors: a Fault-Tolerant Approach," in *Proc. of Fault-Tolerance Computing Symp.*, pp. 306–309, IEEE, 1984.

[95] A. L. DeCegama, *The Technology of Parallel Processing:Parallel Processing Architectures and VLSI Hardware.* Prentice Hall, Inc., 1989.

[96] W. J. Dally and C. L. Seitz, "Deadlock-Free Multicast Wormhole Routing in Multicomputer Networks," *IEEE Trans. on Computers*, vol. 36, pp. 547–553, May 1987.

[97] L. M. Ni and P. K. McKinley, "A Survey of Wormhole Routing Technique in Direct Networks," *Computer*, vol. 26, no. 2, pp. 62–76, 1993.

[98] D. H. Linder and J. C. Harden, "An Adaptive and Fault-tolerant Wormhole Routing Strategy for k-ary n-cubes," *IEEE Trans. on Computers*, vol. 40, pp. 2–12, Jan. 1991.

[99] C. D. Locke, H. Tokuda, and E. D. Jensen, "A Time-Driven Scheduling Model for Real-Time Operating Systems," technical report, Carnegie-Mellon University, 1985.

[100] P. Y. Song, "Design of A Network for Concurrent Message Passing Systems," Master's thesis, Department of Electrical Engineering and Computer Science, MIT, 1988.

[101] H. Schwetman, *CSIM Reference Manual(Revision 16).* Microelectronics and Computer Technology Corporation, 1992.

[102] S. Borkar, R. Cohn, G. Cox, S. Gleason, T. Gross, H. T. Kung, M. Lam, B. Moore, C. Peterson, J. Pieper, L. Rankin, P. S. Tseng, J. Sutton, J. Urbanski, and J. Webb, "iWARP: An Integrated Solution to High-Speed Parallel Computing," in *IEEE Proc. Supercomputer Conf.*, pp. 330–338, 1988.

[103] W.-J. Guan, W. K. Tsai, and D. Blough, "An Analytical Model for Wormhole Routing in Multicomputer Interconnection Networks," in *Proc. of Intl. Parallel Processing Symp.*, pp. 650–654, IEEE, 1993.

[104] M. D. Prycker, *Asynchronous Transfer Mode–Solution for Broadband ISDN.* Ellis Horwood, second ed., 1993.

[105] L. Trajkovic and S. J. Golestani, "Congestion Control for Multimedia Services," *IEEE Network*, vol. 6, pp. 20–26, Sept. 1992.

[106] A. E. Eckberg, D. T. Luan, and D. M. Lucantoni, "An Approach to Controlling Congestion in ATM Networks," *Int'l J. of Digital and Analogue Comm. Sys.*, vol. 3, pp. 123–135, 1990.

[107] M. Bunnell and M. Bunnell, "Real-Time Data Acquisition," *Dr. Dobb's Journal*, pp. 36–44, Jun. 1989.

[108] J. P. Lehoczky, L. Sha, and J. K. Strosnider, "Enhanced Aperiodic Responsiveness in Hard Real-Time Environments," in *Proc. of the 8th IEEE Real-Time Systems Symposium*, pp. 261–270, 1987.

[109] C. D. Locke, D. R. Vogel, and T. J. Mesler, "Building a Predictable Avionics Platform in Ada: A Case Study," in *Proc. of the IEEE Real-Time Systems Symposium*, pp. 181–189, 1991.

[110] H. Tokuda and M. Kotera, "Scheduler 1-2-3: An Interactive Schedulability Analyzer for Real-Time Systems," in *Proc. of CompSAC 88, Computer Software and Applications Conference*, pp. 211–219, 1988.

[111] J. Leung and J. Whitehead, "On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks," *Performance Evaluation*, vol. 2, 1982.

[112] J. P. Lehoczky and L. Sha, "Performance of Real-Time Bus Scheduling Algorithms," *ACM Performance Evaluation Review*, vol. 14, 1986.

[113] J. P. Lehoczky, "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines," in *Proc. of the 11th IEEE Real-Time Systems Symposium*, pp. 201–209, 1990.

[114] J. Lehoczky, L. Sha, and Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," in *Proc. of the 10th IEEE Real-Time Symposium*, pp. 166–171, 1989.

[115] L. Sha, J. Lehoczky, and R. Rajkumar, "Solutions For Some Practical Problems in Prioritized Preemptive Scheduling," in *Proc. of the IEEE Real-Time Symposium*, pp. 181–191, 1986.

[116] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Transactions on Computers*, vol. 39, Sep 1990.

[117] P.-Y. R. Ma, E. Y. S. Lee, and M. Tsuchiya, "A Task Allocation Model for Distributed Computing Systems," *IEEE Transactions on Computers*, vol. 31, no. 1, pp. 41–47, 1982.

[118] A. Rye, ed., *Xlib Programming Manual*, vol. One. O'Reilly & Associates, Inc., second ed., September 1991.