INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

Tool for Task Allocation and Scheduling in Hard Real-Time Systems

by

Xiyue Zhang

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of
Master of Science

Department of Systems and Computer Engineering
Carleton University,
Ottawa, Ontario, Canada

April 14, 1998

Copyright 1998, Xiyue Zhang

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-32435-4

The undersigned hereby recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis

Tool for Task Allocation and Scheduling in Hard Real-Time Systems

Submitted by

Xiyue Zhang, B.Sc.

In partial fulfillment of the requirements
for the degree of Master of Science

Thesis Supervisor

Chair, Department of Systems and
Computer Engineering

Carleton University
May 12, 1998

Abstract

Techniques and a tool for allocation and scheduling on hard real-time systems are developed. The hard real-time systems are characterized by a set of independent, strictly periodic transactions with hard end-to-end deadlines. Each transaction is composed of a set of tasks with regular or complex precedence relationships among them. A static allocation technique is used for allocating tasks to processors. A reallocation strategy is developed for re-assigning tasks to processors in case priority assignment for a given allocation fails. A fixed priority scheduling technique for hard real-time systems is introduced. Both preemptive scheduling and non-preemptive scheduling are considered. Instead of using explicit scheduling strategies, the priority computation tool generates a description of the system in terms of constraints that are solved by a CLP-BNR solver to produce a feasible priority set.

Acknowledgments

My deepest thanks go to Professor Shikharesh Majumdar, who made a diligent effort to lead me into the real-time scheduling field, teach me the knowledge, and guide me to successfully complete my degree. His valuable advice and help have kept me confident in times of doubt and led me to success from places of difficulties. By his example of hard work and his pursuit of excellence, he showed me the essentials of being a successful researcher.

A big “Thank you” goes to Alex Hubbard who answered my numberless questions on CLP-BNR, Unix, and other tools.

Lastly, I would like to thank my husband, Hui, who tasted every piece of joy and bitterness that I tasted. Without his love, care, and support my finishing the thesis would never have been possible.

This work was supported by Precise Software Technologies and Telecommunication Research Institute of Ontario (TRIO). I would like to thank Mr. Jeremy James and Mr. Craig Honegger of Precise Software Technologies for their valuable advice.

Table Of Contents

Abstract	iii
Acknowledgments	iv
List of Figures	viii
List of Tables	x
1. Introduction	1
1.1 Background and Motivation	1
1.1.1 Real-Time Systems	1
1.1.2 Allocation and Scheduling for Hard Real-Time Systems	5
1.1.3 Related Work	7
1.2 The Goal of This Research	10
1.3 Problem Description	12
1.3.1 Allocation and Re-Allocation Problem	12
1.3.2 Priority Assignment Problem	13
1.3.3 Relationship Between the Two Problems	14
1.4 Contributions	17
1.5 Organization	17
2. Requirements for the Tool	19
2.1 Scope of This Research	19
2.2 System Model	20
2.2.1 Transaction Model	20
2.2.1.1 Precedence Graph	20

2.2.1.2 Regular Transaction Model	23
2.2.1.3 Complex Transaction Model	25
2.2.2 Communication Model	30
2.2.3 Assumptions	31
3. System Design	34
3.1 Constraints Logic Programming Language: CLP-BNR	34
3.1.1 Introduction to CLP-BNR	34
3.1.2 Why CLP-BNR?	37
3.2 System Architecture: Basic Idea	38
3.2.1 Description of the System Architecture	40
3.2.2 Functions of The Tool	43
4. Tool Implementation	46
4.1 User System Description	46
4.2 System Feasibility Analysis	54
4.3 Allocation and Re-Allocation	57
4.3.1 Allocation	57
4.3.2 Re-Allocation	60
4.4 Priority Computation	62
4.4.1 Time Constraints	62
4.4.1.1 Time Constraints for Preemptive Scheduling	62
4.4.1.2 Time Constraints for Nonpreemptive Scheduling	69
4.4.2 Implementation of Priority Computation	71

4.4.3 An Example of Constraints CLP File of A Simple System	73
4.5 Verification of Existing Systems and Resource Reduction	75
5. Tool Testing	76
5.1 Test Cases Study	76
5.1.1 Test Case 1: A System with Regular Transactions Only	76
5.1.1.1 System Description	76
5.1.1.2 Output Results	79
5.1.1.3 Validation of Tool Output	79
5.1.2 Test Case 2: A System with Regular and Complex Transactions	86
5.1.2.1 Test Case 2: System Description	86
5.1.2.2 Output Results	90
5.1.2.3 Validation of Tool Output	91
5.1.3 Test Case 3: A System with Regular and Complex Transactions	98
5.1.3.1 System Description	99
5.1.3.2 Output Results	101
5.1.3.3 Validation of Test Case 3	103
5.2 Tool Analysis	103
6. Conclusions	106
6.1 Summary	106
6.2 Future Research	107
References	109

List of Figures

Figure 1-1: Transactions in an Aircraft Simulator System	3
Figure 1-2: Relationship Between the Two Problems	16
Figure 2-1: Simple Precedence Graph Examples	21
Figure 2-2: A cyclic Precedence Graph	22
Figure 2-3: A Regular Transaction Example	24
Figure 2-4: A Transaction with two kinds of precedence behavior	26
Figure 2-5: Precedence Graph of Transaction A	28
Figure 2-6: Another Example of Complex Transaction	29
Figure 2-7: An example of multi-level OPR and EPR precedence relationship	30
Figure 2-8: Communication Model for A Four-Processor System	31
Figure 3-1: High Level Description of Tool Operation	40
Figure 3-2: System Architecture	42
Figure 3-3: Flow Chart Diagram for The Tool	45
Figure 4-1: An Input File Example	53
Figure 4-2: An example of inconsistent precedence relationship	55
Figure 4-3: An Example of System with Complex Transactions	66
Figure 4-4: Priority Computation	72
Figure 4-5: Precedence Graph for the Example System	73
Figure 4-6: The Constraints CLP File for the Example System	74
Figure 5-1: Precedence Graph for Test Case 1	77
Figure 5-2: Precedence Graph for Application in Test Case 2	88

Figure 5-3: Precedence Graph for Test Case 3	100
Figure 5-4: Situation of Using Two Processors	102
Figure 5-5: Changes of Tool Response Time When Task Number Increases	104

List of Tables

Table 1: Changes in Tool Response Time When System Complexity Increases	105
---	-----

Chapter 1 Introduction

A real-time system should not only produce logically correct results, but also satisfy explicit time constraints, otherwise it will risk severe consequences including failure. Real-time systems, where a failure to meet time constraints leads to system failure, are often called *Hard Real-Time Systems*. This research focuses on allocation and scheduling problems in hard real-time systems. A tool is developed for allocating tasks and scheduling in multiprocessor-based hard real-time systems. This chapter presents a brief introduction to theories and research concerning the allocation and scheduling in hard real-time systems which are relevant to this thesis. The goal of this research and a description of the problem that this thesis is concerned with are described.

1.1 Background and Motivation

1.1.1 Real-Time Systems

Real-time systems are systems in which the correctness of the system not only depends on the logical results of computation, but also the time at which the results are produced. Depending on the effects of failure to meet certain time constraints to a system, there are three classes of real-time systems [16, 19], each of which are briefly defined as:

- 1) Soft real-time systems, where performance is degraded but not destroyed by failure to meet response time constraints. Examples include GUI systems for which user satisfaction is dependent on the timeliness of response [16, 19].

- 2) Hard real-time systems, where failure to meet response time constraints leads to system failure. Examples are automated factories, flight control systems, space systems, etc. [16, 19].
- 3) Firm real-time systems, that are characterized by a set of firm real-time transactions. Firm real-time transactions are transactions with deadlines where a low probability of missing a deadline can be tolerated. Telephone switches and other communication systems are examples of such systems [19].

This research is concerned with hard real-time systems which are characterized by a set of periodic transactions. Each transaction is composed of a set of communicating tasks with CPU processing requirements, and precedence relationships [5] which can be described by a directed acyclic precedence graph. For example, transactions in an aircraft simulator system described by Precise Software Technologies are shown in Figure 1-1 (see Section 5.1.2.1 for a detailed discussion). Transaction #1, which contains 11 tasks, is a periodic transaction with a period of 25ms. The deadline of transaction #1 is also 25ms. Transaction #2, which consists of only one task: Task User Interface, is a sporadic transaction. It does not have an explicit deadline. However, most initiations of transaction #2 are expected to be processed within 100ms.

Figure 1-1: Transactions in an Aircraft Simulator System

Transactions, which are associated with hard end-to-end deadlines are *hard* transactions. The end-to-end deadline of a transaction is the maximum time allowed for completing all the tasks in the transaction. In Figure 1-1, transaction #1 is a hard transaction with a hard deadline of 25ms. This means every processing request of transaction #1 must finish within 25ms. Hard transactions are the core of hard real-time systems. A *soft* transaction, on the

other hand, is a transaction associated with a soft deadline. This means that failure to meet the deadline of a processing request of a soft transaction will degrade system performance, but not result in a system failure. For example, in Figure 1-1, transaction #2, which contains task User Interface, is a soft transaction. The soft deadline of transaction #2 is 100ms. If a processing request of this transaction fails to meet the soft deadline, this only results in degradation of system performance. Sometimes requirements for soft transactions may be specified in terms of fractions of processing shares (see Section 5.1.2.1 for example).

A periodic single-task transaction, for example Tick in Figure 1-1, is an infinite stream of execution requests with the interval between any two consecutive requests being no shorter than the period of the transaction. Each execution request of a transaction is called an *occurrence* of the transaction. A multi-task transaction (e.g. Transaction #1 in Figure 1-1) is periodic in the sense that its root task(s) (e.g. task Control for Transaction #1 in Figure 1-1) is (are) periodic. This means that the interval between any two consecutive invocations of every root task is not shorter than the period of the transaction. In this research, we are concerned with hard real-time systems which contain hard periodic transactions only. Later, when transaction is mentioned, it means hard periodic transaction.

A critical issue for hard real-time systems is how to allocate tasks to processor(s), and how to schedule tasks on each processor so that all hard transactions can be guaranteed to meet their end-to-end deadlines. This involves allocation and scheduling strategies for hard real-time systems which are discussed next.

1.1.2 Allocation and Scheduling for Hard Real-Time Systems

Computer systems usually have resources that must be shared serially, such as processors, network paths, etc. Scheduling is the definition of an order for contending activities to access a serially shared resource (e.g. a processor). Design of a hard real-time system involves the allocation of resources and time to transactions in such a way that all transactions in the system can meet their deadlines.

In recent years, more and more real-time applications run on multiprocessor or distributed computer systems. Thus, the allocation of tasks to processors in order to increase system schedulability becomes more and more important to real-time systems. There are two classes of allocation strategies. One is dynamic (run-time) allocation which gives more freedom and has potential performance advantages [2]. The other is static allocation (pre-runtime) which is easier to implement and less complex to operate for hard real-time systems (especially embedded systems). The goal of allocation is to allocate resources to transactions such that special objectives are achieved. These objectives may include the minimization of the number of processors used, the minimization of Inter-Processor Communication (IPC), or the minimization of the bottleneck processor utilization. This research provides an allocation and scheduling tool for multiprocessor based real-time systems. The allocator uses static allocation to balance utilization of processors and inter-task communication time.

As in the case of allocation, scheduling may be either static (pre-runtime) or dynamic (run-time). Static scheduling has advantages over dynamic scheduling, such as simplicity in implementation, and predictable behavior during transient overload situations. In hard

real-time systems, static scheduling is most often obtained by either establishing a frame-based sequence of task execution [14, 30], or by assigning fixed priorities to tasks [16]. Among static scheduling, *fixed-priority scheduling* [11, 16] is an effective approach to scheduling periodic transactions in real-time systems. In fixed-priority scheduling, every released but not completed execution request of each task in the system has been assigned a priority. All execution requests of the same task have the same priority. If at any time, there is more than one task competing for a shared resource, the one with the highest priority is executed first.

Scheduling can also be either preemptive or non-preemptive. In preemptive scheduling, an executing task request can be preempted by other task execution request(s) which has (have) higher priority(ies). On the other hand, for non-preemptive scheduling, an executing task will continue its execution as long as needed or until it becomes blocked (e.g. due to a resource conflict).

In this research, a fixed-priority scheduling tool for hard single-processor or multiprocessor real-time systems is developed. Preemptive and non-preemptive scheduling can be specified by the user. This research is concerned with a TRIO industrially specified research project with Precise Software Technologies of Ottawa. Many requirements for the tool developed in this project have been described by Mr. Jeremy James and Mr. Craig Honegger of Precise Software Technologies.

1.1.3 Related Work

In the past few decades, the allocation and scheduling problem in real-time systems has been the focus of a great deal of research. Several real-time allocation algorithms and scheduling theories have been developed.

In the classical paper of fixed-priority *preemptive* scheduling for uni-processor hard real-time systems, Liu and Layland [3] proposed the **Rate Monotonic Algorithm** (RMA) for priority assignment in systems which have a set of independent periodic tasks (single-task transactions) with deadlines equal to their periods. Independent tasks means that the execution of every task does not depend on any other tasks in the system. The basic idea of rate monotonic algorithm is to assign highest priority to the task with the highest rate (or with the shortest period) and assign the priorities of the remaining tasks monotonically in the order of their rates. Later, Leung and Whitehead [10] proposed **Deadline Monotonic Algorithm** (DMA). Deadline monotonic algorithm is a type of priority assignment algorithm for independent tasks with relative deadlines less or equal to their periods. A great deal of research has been done to improve RMA and DMA for various uni-processor preemptive systems [8, 14, 16, 18, 30] and multiprocessor preemptive systems [32]. Kevin Jeffay, Donald Stanat, and Charles Martel [12] give a necessary and sufficient set of conditions for a set of periodic tasks to be schedulable on a uni-processor non-preemptive system.

However, none of this work is suitable for solving the scheduling problem for systems with a set of complex periodic transactions associated with end-to-end deadlines. Here, a complex transaction contains a set of communicating tasks with precedence relationship

among them. If RMA or DMA is used as the priority assignment algorithm for this kind of system, all tasks in the same transaction will have the same priority which is based on the period or deadline of the transaction. This makes it impossible to assign different priorities to different tasks belonging to the same transaction. So for systems which consist of a set of communicating tasks with precedence relationships among them, RMA and DMA is not optimal.

Relatively little work exists on fixed-priority scheduling for transactions with end-to-end deadlines. The most common approach to this problem is to decompose an end-to-end system into a set of uni-processor systems [12, 25], and apply real-time scheduling algorithms for uni-processor systems. Consider for example PERTS [25], a commercial prototyping environment for real-time systems developed by a company called Tri-Pacific Software. PERTS uses the research done at the Software Engineering Institute at Carnegie Mellon University in conjunction with the research done at University of Illinois at Urbana, Champaign. It extends RMA and DMA from uni-processor systems to multiprocessor systems. R. Bettai [20] and Jun Sun [11] developed end-to-end scheduling algorithms for a set of transactions with end-to-end deadlines in a flow-shop system. A flow-shop transaction is a transaction which contains a chain of tasks executing on different processors in order. Flow-shop transactions are simple forms of transaction. In many real systems, transactions are more complicated than flow-shop transactions. They may have fork/join and contain tasks with given precedence relationship, as well as other kind of relationship discussed in Section 2.2.1.3.

Reza Etamadi [21, 22] also studied the flow-shop model. He then extended his study to

the scheduling of periodic transactions with subtransactions and transactions that visit a processor multiple times. Each subtransaction of a transaction is a flow-shop transaction. In this thesis we focus on hard real-time systems with periodic transactions characterized by end-to-end deadlines. Transactions in the system may be very complicated, and defined by special types of precedence graphs. A new approach is used to solve the scheduling problem in hard real-time systems. It does not use any explicit priority assignment algorithms (like PERTS uses RMA and DMA) to assign priorities to tasks. The priority computation module in our tool generates a set of relationships and constraints among task execution times and deadlines. A constraint solver is then used to produce the priority assignment. Because it does not have a specific priority computation algorithm, it is very easy to incorporate additional constraints. As well, it is not limited to some particular transaction model, such as the flow-shop model, the revisiting model of Etemadi [21, 22], etc. It can handle very complex transactions.

Another component of the tool is allocation, which is defined as mapping tasks to processors. There are a variety of strategies for task assignment in real-time systems. But many of them are strategies for allocating a set of independent tasks to multiprocessor or distributed systems [1, 15, 26] based on the schedulability bound for the scheduling algorithm. For example, the schedulability bound for RMA of a set of n independent tasks on a uni-processor system states that a feasible schedule is possible if the utilization of the processor is less than $n \left(2^{\binom{1}{n}} - 1 \right)$ [16]. In [1, 15, 26], new schedulability bounds are derived for multiprocessor and distributed systems. The allocation algorithms in [1, 15, 26] are

based on the corresponding schedulability bounds derived respectively. E. G. Coffman [4] gives a heuristic algorithm called MULTIFIT, which is based on bin-packing techniques, for a set of independent tasks. MULTIFIT is designed for a single application that runs to completion. Later, C. M. Woodside and G. G. Monforton [2] improved MULTIFIT to MULTIFIT.COM to handle a set of communicating tasks executing on multiprocessor systems. In this thesis, a task allocation module is developed which uses the same communication model as the MULTIFIT.COM algorithm. The communication time between any two tasks is considered in case they are allocated on different processors, otherwise the communication time is considered to be *zero*. We consider a processor as a bin. The objective of our allocation algorithm is to balance the utilization of processors and reduce the communication time between processors. The allocation algorithm developed in this thesis is designed to handle a set of periodic communicating tasks with precedence relationship between them. The overall objective of allocation is to increase system schedulability.

1.2 The Goal of This Research

A significant issue for a hard real-time system, which is composed of a set of periodic transactions with hard end-to-end deadlines, is how to schedule these transactions onto shared resource(s) so that every transaction can be guaranteed to meet its hard deadline. The overall goal of this research is to develop techniques and a tool for an integrated approach to allocation and scheduling for hard real-time systems. The research is concerned with a system model defined in Section 2.2. The major objective is to allocate

tasks to processors and determine task priorities in such a way that all *hard* transactions in the system can be guaranteed to meet their deadlines. Tasks in the transactions have complex relationships which to the best of our knowledge are not handled by most other systems.

This research uses fixed-priority scheduling policies. Fixed-priority scheduling is a very efficient approach for scheduling periodic transactions for hard real-time systems, because the run time scheduling overhead is very low compared to dynamic scheduling. Also it is easier to implement and test in comparison to dynamic scheduling. The allocation policy used in this research is static: mapping of tasks to processors does not change during the life time of the system.

RMA and DMA are only optimal for scheduling a set of independent tasks on a single-processor system. They are not optimal for multiprocessor scheduling, especially for the systems described in the next chapter, in which tasks may not be independent of each other. Because of the limitations of RMA and DMA for scheduling communicating tasks on multiprocessor systems, the allocation and scheduling techniques developed in this research are not based on RMA and DMA. A new approach based on Constraint Logic Programming is used for priority computation and is discussed further in Section 3.1. Our techniques and tools are suitable not only for single-processor scheduling, but also for multiprocessor scheduling.

A more detailed description of the objectives of this research is presented in the next section.

1.3 Problem Description

The system this research is concerned with consists of two parts: software and hardware. The software is represented by a set of periodic transactions with hard end-to-end deadlines. Each transaction consists of a set of communicating tasks. The hardware is represented by a set of processors which is connected together by communication links. Now, the problem is how to map tasks to processors, and schedule tasks on each processor, so that every hard transaction in the system can be guaranteed to meet its deadline. This involves two major problems in real-time static priority driven scheduling.

- i) The allocation problem: how to allocate tasks to processors based on task characteristics and other information, so that the assignment increases system schedulability.
- ii) The priority assignment problem: how to assign priority to tasks on each processor so that every transaction can meet its hard end-to-end deadline.

1.3.1 Allocation and Re-Allocation Problem

Static allocation is being used in this research. According to static allocation, if a task is allocated to a processor, every occurrence of the task is executed on the same processor. So, after a static allocation is given for a system, each transaction follows some specific path from processor to processor for every occurrence of the transaction.

Normally, static allocation strategies are designed to improve certain performance matrix of system, such as minimizing Inter-Processor Communication (IPC), maximizing processor utilizations, load balancing, etc.

The periodic task assignment problem is known to be NP-Hard [17]. Since an exhaustive search in the solution space is impractical for all but extremely small systems, we focus on a heuristic solution for the allocation problem.

The goal of our allocation strategies is to increase system schedulability. It is not guaranteed that after an allocation the system will be schedulable. When the allocation strategy fails for a given system, it is necessary to change the allocation through a re-allocation strategy. Also, the information which comes from the failed allocation(s) can be used in re-allocation of a system. This is called the re-allocation problem.

1.3.2 Priority Assignment Problem

It can be said that a priority assignment is *feasible* if the assignment produces a schedulable system. For a hard real-time system, the system is schedulable if and only if all hard transactions in the system can meet their deadlines. Assigning priorities to tasks so that the system is schedulable is called the priority assignment problem.

The fixed-priority assignment problem is also NP-Hard [11, 17]. Most of the research done for the fixed-priority assignment problem is focused on developing heuristic priority assignment algorithms. There are three classes of heuristic priority assignment methods:

- *Rate based methods* [3, 14, 16]. For example, Rate Monotonic (RM) algorithm described in Section 1.1.3.
- *Deadline-based methods* [10, 30, 32]. An example is Deadline Monotonic (DM) algorithm described in Section 1.1.3.
- *Optimization-based methods* [6, 13]. This kind of methods usually start with a

random priority assignment or a priority assignment according to a basic assignment method, and then continually adjust the assignment until a feasible solution is found. An example is the HOPA algorithm in [6].

In this research, a novel approach is used for solving the fixed-priority assignment problem which is based on a constraint solver. The system is described by a set of equalities and inequalities that relate the input and output parameters. The constraint solver computes a set of priorities that satisfy all the constraints imposed by these equalities and inequalities.

1.3.3 Relationship Between the Two Problems

Our tool has two components: the allocation module and the fixed-priority scheduling module. It can be seen that the allocation problem and the priority assignment problem are related. For a system with m periodic transactions and n processors, the allocation problem is to find a strategy to partition all of the tasks in the system into n parts, and map each of the n parts to a processor in such a way that it increases system schedulability. Priority assignment determines the order of execution of tasks in each of the n parts. Allocation and priority assignment must be done in such a way that all of the transactions in the system can meet their hard deadlines. Priority assignment for a system is based on the task allocation for the system. It is possible that a system may be schedulable for one allocation, but not schedulable for another allocation. A good allocation produces a higher probability to find a feasible priority set for a system.

In our allocation and scheduling tool, the allocation module uses system information provided by the user as input to produce a mapping of tasks to processors. The priority

assignment module uses the result from the allocation module plus the system information provided by the user as input to find a feasible priority set for the system. If the priority assignment module cannot find a feasible priority set for the system with the given allocation, the re-allocation module attempts to find another mapping of tasks to processors based on information from the previous failed allocation(s). The results are used as input to the priority assignment module. The iteration continues until a feasible priority set is found for the given system, or a predefined maximum iteration count is reached.

In Figure 1-2, we give a picture of the relationship between these two problems in the context of the tool developed in this research.

Figure 1-2: Relationship Between the Two Problems

1.4 Contributions

One of the major contributions of the thesis is the development of techniques that use Constraint Logic Programming for priority computation. In stead of using explicit priority computation algorithms, a set of constraints that relate task execution times, waiting times, and deadlines are produced. A constraint solver is used to solve these constraints to obtain a feasible priority set for the system. Another novel feature of this research is the use of a complex transaction model that is characterized by special precedence relationships.

A tool for allocation and fixed-priority driven scheduling in hard real-time systems that uses the techniques developed in this research is designed and implemented. This tool can be used for systems with complex transactions. It can map tasks to processors and generate a feasible priority set for the system. It can also be used for verifying the designs of existing systems. It works for preemptive systems as well as non-preemptive systems. As described in the previous section the tool primarily consists of a task allocation module, a priority computation module, and a task re-allocation module.

Another feature of the tool is the resource reduction function. This function advises the user of meeting transaction deadlines on a number of processors that is smaller than the number specified by the user.

1.5 Organization

This thesis is divided into six chapters.

Chapter 2 describes the requirements for the tool. It presents the scope of this research, system definitions, the requirements for the tool, and assumptions made.

Chapter 3 describes the design of the tool and a general introduction to every component of the tool. A brief discussion of CLP-BNR, the Constraint Logic Programming language used in the implementation of the tool is also included.

Chapter 4 presents the implementation of every component of the tool.

Chapter 5 describes the testing of the tool. It includes testing strategies, cases studies and results.

Chapter 6 presents the conclusions of this research and some directions for future research. These include extensions for the tool to handle more complex systems and implementation of the tool in more efficient ways.

Chapter 2 Requirements for the Tool

In this chapter the scope of this research is discussed first. Then the system model that serves as the basis of our work is formally described.

2.1 Scope of This Research

In this research, two traditional problems in hard real-time systems design are considered. One is the allocation problem (described in Section 1.3.1), the other is the fixed-priority assignment problem (described in Section 1.3.2). An allocation and scheduling tool is developed for the group of hard real-time systems.

The hard real-time systems we are concerned with are characterized by a set of *strictly periodic independent* transactions with hard end-to-end deadlines. A transaction is strictly periodic if the interval between any two consecutive occurrences of the transaction is exactly equal to the period of the transaction. Independent transactions means no sharing of task(s) among transactions. Each transaction is composed of a set of tasks with resource requirements, referred to here as processors, and precedence relationships among them. Different tasks belonging to the same transaction can execute on different processors in a multiprocessor environment. The transaction model is described in more details in Section 2.2.1. The transaction model describes the software architecture for a system. The hardware model of a system is represented as a set of processor(s) so that each processor has its own local memory and runs its own kernel. For the multiprocessor real-time systems we are

concerned with, the set of processors in the system are logically connected together, which means that each processor can communicate with the others in the system. This is because a transaction may execute its communicating tasks on different processors.

2.2 System Model

2.2.1 Transaction Model

A transaction is composed of a set of tasks with precedence relationships among them. Task precedence relationships can be described by a directed acyclic precedence graph. Tasks are the basic units of work and the smallest entities that are treated as schedulable units in this research.

2.2.1.1 Precedence Graph

A precedence graph consists of a set of nodes linked together by directed edges. Each node represents a task. The directed edges represent precedence relationship among tasks. There may be at most one such edge between a pair of tasks. The task that the edge is directed from is called the *immediate predecessor* of the task that the edge is directed to. It is also called *source* or *source task* of the edge. The task that the edge is directed to is called the *immediate successor* of the source task of the edge. It is also known as the *destination* or *destination task* of the edge. As well, precedence relationship can be inherited as described by the following rules.

- If task A is an immediate predecessor (successor) of task B, and task B is an immediate predecessor (successor) of task C, then task A is an indirect predecessor (successor) of task C.

- If task A is a predecessor (successor) of task B, and task B is a predecessor (successor) of task C, then task A is a predecessor (successor) of task C. *Note: if we say task A is a predecessor (successor) of task B, it means task A is either an immediate or indirect predecessor (successor) of task B.*

Figure 2-1: Simple Precedence Graph Examples

In Figure 2-1 (a), task A is an immediate predecessor of task B. In Figure 2-1 (b), task A is a predecessor of task B, task C, task D and task E, derived from the two rules given.

A precedence graph of a transaction is used to define precedence relationships among the tasks of the transaction. As has been noted, the precedence relationship is used to describe the synchronization requirements of tasks which belong to the same transaction. It also provides an implicit communication structure of the transaction. Formally, precedence relationships are conditions that have to be satisfied before a task may be

executed. It can be said that the precedence relationship is fulfilled for a task, say T, when the messages of termination of execution from all immediate predecessors of task T are available for T. In other words, task T cannot be ready to run until all immediate predecessors of it finish their executions.

In an acyclic precedence graph, precedence relationship among tasks are always consistent. There are no paths from any task (node) of a directed acyclic precedence graph which directs back to the task. In other words, there are no loops or cycles (as in the graph in Figure 2-2) in the graph. If there is a loop in the precedence graph, the precedence relationship will not be consistent. For example, in Figure 2-2, task A is an immediate predecessor of task B, task B is an immediate predecessor of task C, task C is an immediate predecessor of task A. According to the first rule, task A is a predecessor and a successor of task C at the same time. This gives rise to a conflict. According to the definition of precedence relationships, task A cannot be ready to execute until its immediate predecessor task B finishes its execution. Task B waits for task C to finish execution. Task C waits for task A for the same reason. This produces a deadlock, in which none of the tasks can get executed.

Figure 2-2: A cyclic Precedence Graph

Only those transactions with the communication structure, which can be represented as acyclic directed precedence graphs, are considered, in this research.

2.2.1.2 Regular Transaction Model

A transaction is periodic when the arrival pattern of the transaction is periodic. This means given any occurrence of the transaction at time t , the next occurrence of the transaction will occur at time $t + P$. The P is a constant. For a periodic transaction, P is the period of the transaction.

A regular transaction is a periodic transaction with a hard end-to-end deadline. Formally, the behavior of a regular transaction A with period P and deadline D is given by the following rules for the occurrence and execution of transaction A . If t_k is the time of the k^{th} occurrence of transaction A , then

- The $(k + 1)^{th}$ occurrence of transaction A will occur at time $t_{k+1} = t_k + P$.
- The k^{th} execution of transaction A must begin no earlier than t_k and be completed no later than $t_k + D$. This requires that all tasks of transaction A must finish execution in the interval $[t_k, t_k + D]$.

A regular transaction consists of a set of communicating tasks. The precedence relationship among tasks of a regular transaction can be described by a directed acyclic precedence graph. Each task is defined by the following parameters.

- Worst-case execution time. Worst-case execution time of a task is the maximum amount of time it takes to complete execution.

- Period. All tasks of a regular transaction have the same period, which is period of the transaction.
- Deadline. The transaction deadline is used as the upper bound on the deadline of all tasks which belong to the transaction.
- Precedence relationship with other tasks of the same transaction.
- Resource requirement. Resource is referred to here as processors. Some tasks may need to execute on a special kind of processor in the system.
- Worst-case inter-task communication time. Unless stated otherwise the worst-case inter-task communication time is assumed to be zero.

A simple regular transaction is given in Figure 2-3. The notation in Figure 2-3 is the notation that is used for precedence graphs.

Note: Each task has a label $T(P, D)$, where T = worst-case execution time (ms)
 P = Period of the task (ms)
 D = Deadline of the task (ms)

Figure 2-3: A Regular Transaction Example

For a regular transaction, each task is characterized by the same value of its parameters in every occurrence of the transaction. A very important point here is that tasks of a regular transaction keep the same precedence relationship for every occurrence of the transaction. As will be shown in the next section, this is not applicable for the complex transaction model.

2.2.1.3 Complex Transaction Model

As mentioned in Chapter 1, this project is sponsored by Precise Software Technologies. The problem of handling the kind of transaction which change their behavior periodically was a requirement and suggested by Precise Software Technologies. The complex transaction model is introduced to handle such transactions.

A significant difference between a complex transaction and a regular transaction is that a complex transaction changes its precedence relationship among tasks periodically. For example, transaction A, in Figure 2-4, contains three tasks. Both the period and the deadline of transaction A are equal to 25ms. For every $(2k+1)^{th}$ occurrence (k is a non-negative integer) of task 1, the precedence relationships among its tasks are described by the precedence graph in Figure 2-4 (a). For every $(2k)^{th}$ occurrence (k is a positive integer) of task 1, the precedence relationship among its tasks is described by the precedence graph in Figure 2-4 (b).

Note: Each label for a task represents the worst-case execution time of the task

Figure 2-4: A Transaction with two kinds of precedence behavior

For defining this kind of transactions, three kinds of special precedence relationships are introduced in this thesis. The first is like the precedence relationship between task 1 and task 2 in Figure 2-4 (a). This kind of precedence relationship only exists in every odd numbered occurrence of the *source task* (e.g. task 1) of the precedence relationship. It will be called the **Odd Precedence Relationship** (OPR). The second is like the precedence relationship between task 1 and task 3 in Figure 2-4 (b). This kind of precedence relationship only exists in every even numbered occurrence of the source task of the precedence relationship. It will be called the **Even Precedence Relationship** (EPR). The OPR and EPR relationships are shown by using labels (O) and (E) respectively on the arcs of the precedence graph. We need another type of precedence relationship to describe the situation that Task 2 and Task 3 have a common successor: Task 4 (see Figure 2-5). Task 2

is an immediate predecessor of task 4 only at those occurrences of transaction A when task 2 occurs. Similar is the case of Task 3. This kind of precedence relationship is called the **Tail Precedence relationship (TPR)**. For transaction A, task 4 occurs in every occurrence of transaction A, but it changes its predecessor periodically. In every odd numbered occurrence of transaction A, its immediate predecessor is task 2. In every even numbered occurrence of transaction A, its immediate predecessor is task 3.

Complex transactions without TPR relationships are also possible (see Figure 2-6 for example). In Figure 2-6, Task 1 is an immediate OPR predecessor of Task 2, Task 2 is an immediate predecessor of Task 4. So Task 2 and Task 4 only occur at every odd numbered occurrence of the transaction. Transactions with only regular, OPR, and EPR precedence relationships have been observed on real systems such as the aircraft simulator systems (see Section 5.1.2 and 5.1.3). Note that the TPR relationship is not currently handled by the tool. Thus a range of real-time systems that do not use TPR relationships can be handled by the current tool implementation.

$\xrightarrow{(O)}$: notation for Odd Precedence Relationship

$\xrightarrow{(E)}$: notation for Even Precedence Relationship

$\xrightarrow{(T)}$: notation for Tail Precedence Relationship

$\longrightarrow$ notation for regular precedence relationship

Note: Each task with a label $T(P,D)$: T = worst-case execution time
 P = period of the task
 D = the upper bound of the deadline of the task

Figure 2-5: Precedence Graph of Transaction A

Figure 2-6: Another Example of Complex Transaction

A complex transaction is a periodic transaction composed of a set of tasks with not only regular precedence relationship among them, but also Odd Precedence Relationship, or Even Precedence relationship, or Tail Precedence Relationship among them. Parameters of each task of a complex transaction is the same as that of a regular transaction, except that the period and the deadline of a task may be different from the period and the deadline of the complex transaction. Period of the destination task of an OPR or EPR relationship is two times the period of the source task. The deadline of the destination task of an OPR or EPR may be larger than the deadline of the source task.

Note that it is not possible to model transaction A in Figure 2-5 by two different regular transactions, in Figure 2-4 (a) and Figure 2-4 (b), with period 50ms and deadline 50ms. This is because these two transactions, Aa and Ab, are independent of each other. Thus they may all get executed within 25ms. This is not the correct model for system behavior.

If a task has two groups of immediate successors. One group contains immediate OPR successor(s) of this task, the other contains immediate EPR successor(s). We call it a level of EPR and OPR precedence relationship. Our tool is designed for single-level EPR and OPR precedence relationship, not for multi-level EPR and OPR precedence relationship like the one shown in Figure 2-7.

Figure 2-7: An example of multi-level OPR and EPR precedence relationship

2.2.2 Communication Model

In this thesis, the physical details of how processors are connected together are not considered. But, because there are communication and synchronization among tasks of the same transaction, executing on different processors, a logical fully-meshed communication architecture is assumed. This means each processor is connected with all the other processors in the system by logical communication links.

Figure 2-8: Communication Model for A Four-Processor System

An example of the communication model for a four-processor system is shown in Figure 2-8. Lines between processors stand for logical communication links (LCL). These logical communication links may be associated with different communication costs. For example, the communication cost between processor 1 and processor 2 is smaller than that between processor 1 and processor 3.

It is assumed that inter-task communication may occur between a task and its immediate successor. Such communication is assumed to be non-blocking with the data transmission occurring at the end of execution of the predecessor task.

2.2.3 Assumptions

A hard real-time system consists of a set $\{C_i\}$ of processors and a set $\{T_i\}$ of periodic transactions. Each transaction consists of a set $\{T_{i,j}\}$ of tasks with precedence relationship

among them. Each transaction can be described either by the regular transaction model or by the complex transaction model. The following list of assumptions further explains the aspects of our system model.

- Each transaction τ_i is a strictly periodic transaction with period P_i , meaning that the root task(s) of τ_i is (are) strictly periodic task with period P_i . (Note, the root task of a transaction is a task that has no predecessor.) A task that is not a root task of τ_i may or may not be released periodically. But for the sake of convenience, it can be said that every task has a period. For a non-root task, l , the period of task l , P_l , depends on the period of its immediate predecessor, P_{impr} , and the precedence relationship between them:
 - 1) If it is regular precedence relationship, $P_l = P_{impr}$.
 - 2) If it is OPR or EPR relationship, $P_l = 2P_{impr}$.
 - 3) If it is TPR relationship, $P_l = \left(\frac{1}{2}\right)P_{impr}$.
- Each transaction τ_i has a hard end-to-end deadline D_i . In other words, for each occurrence of τ_i at time t , it must complete by time $(t + D_i)$.
- Each task $\tau_{i,j}$ of τ_i has a worst-case execution time $E_{i,j}$. This means that the execution time of every occurrence of $\tau_{i,j}$ is no greater than $E_{i,j}$.
- Deadline for each transaction τ_i is no greater than period of τ_i : $D_i \leq P_i$. This is also true for each task of the transaction. So at any time, there is at most one occurrence of τ_i .

- The precedence relationship among tasks for each transaction is pre-defined, and can be described by a directed acyclic precedence graph.
- Processors are the only shared resources in the system. It is assumed that there is no contention on communication links.
- If communication time among tasks is to be considered, the worst-case data amount inter-changed between each pair of tasks as well as speeds of communication links between processors are to be specified. As well, it is assumed that communication only takes place between tasks with precedence relationship between them. If two communicating tasks are allocated on the same processor, the communication time between them is considered as zero.
- Transactions in the system are independent of each other. There are no shared tasks between any two transactions.

Chapter 3 System Design

In this chapter, a brief introduction to CLP-BNR is given. The system design which is based on the system model defined in Section 2.2 and the problem description defined in Section 1.3 is also discussed. The discussion is based on the following questions:

- What is the basic idea behind the architecture of this tool?
- What is the architecture of the tool?
- What are the major functions of the tool?

3.1 Constraint Logic Programming Language: CLP-BNR

The priority computation module is based on Constraint Logic Programming. A brief description of CLP-BNR the language that was used for implementing the priority computation module is necessary to understand the details of system design. A discussion of CLP-BNR is provided in this section.

3.1.1 Introduction to CLP-BNR

CLP-BNR [27, 28, 29] is a Prolog programming language, and is developed by Nortel Technologies (Known earlier as Bell-Northern Research).

The idea of logic programming is to use symbolic logic. Symbolic logic offers both a declarative method of expressing relations between objects as well as a procedural method of interpreting these relations through the automation of logical reasoning. Prolog is

currently the most successful implementation of a logic programming language. Basically, pure Prolog consists of three elements: resolution, backtracking, and unification. But a practical programming language should introduce language features that may result in non-logical behavior. In CLP-BNR, the mechanisms for handling these non-logical behaviors are the control of the execution flow, communication with input and output devices, efficient arithmetic execution and Prolog programming manipulation.

CLP-BNR is not only a Prolog programming language, but also an interval-based constraint system. CLP-BNR provides constraints not only on real-valued, integer-valued, and boolean-valued variables, but also on intervals. Therefore, it is also called interval based CLP.

Interval is a new data type introduced by CLP-BNR. In contrast to other variable types that have only a single valued at a given point, an interval is a logic variable whose numeric value is constrained to lie in a closed real or integer range. Examples of intervals are $[-2.5, 7.0]$, $[3, 5]$. An interval $[X-, X+]$ has value that lies between an upper bound $X+$ and a lower bound $X-$. An interval also has type information. It can be an integer type, or a floating-point type, or a boolean type interval. The syntax of declaration of an interval in CLP-BNR is as follows:

- 1) Declare a floating-point type interval variable with upper bound UB and lower bound LB: *V: real(LB, UB)*. Here, LB and UB are floating-point constants, for example, -10.2, 7.5, etc.
- 2) Declare an integer type interval variable with upper bound UB and lower

bound LB: *V: integer(LB, UB)*. Here, LB and UB are integer constants, for example, 5, 10, etc. Integer type interval variable V has an integer value which is constrained to lie between LB and UB. For example, V is declared as *V:integer(2, 5)*. So the value of V can be 2, 3, 4, or 5.

- 3) Declare a boolean type interval variable with upper bound 1 and lower bound 0: *V: boolean*. Here, LB and UB are omitted, because boolean type variables can only have value 0 or 1.

CLP-BNR can handle functional and relational arithmetic operations for intervals as well as other types of variables. The interval arithmetic system of CLP-BNR is one of its most advanced features. It allows the expression of arithmetic relationships in a fully logical and relational manner. Also it offers a mechanism for solving sets of linear and nonlinear equations and inequalities.

Two very simple examples of using intervals in CLP-BNR are presented.

- Example I: M and N are two integer type intervals with upper bound 8 and lower bound 0. The value of M is three times the value of N, and M is greater than 3. In CLP-BNR the following query is inputted:

?- *[M, N]: integer(0, 8), {M == 3*N}, {M > 3}.*

The output of CLP-BNR is as follows:

?- *[[6, 2]: integer(0, 8), {6 == 3 * 2}, {6 > 3}].*

This means that the solution for the above query is that M equals to [6, 6] and N equals to [2, 2].

- Example II: X is a floating-point interval with upper bound 3, and lower bound 1. Y is also a floating-point type interval with lower bound 0, and its upper bound is infinity. Y is a non-negative square root of X . Let us try the following query:

*?- X: real(1, 3), Y: real(0, _), {Y**2 == X}, {X == 3}.*

The output of CLP-BNR for this query is as follows:

*?- [{3.0 == _Y ** 2}, {3.0 == 3}] where {_Y: real(1.73205080756888, 1.73205080756888)}.*

This means that the value of Y is a real value interval whose lower bound is the same with its upper bound which is 1.73205080756888.

3.1.2 Why CLP-BNR?

This research focuses on an allocation and scheduling tool for hard real-time systems. CLP-BNR is used as the programming environment of the priority computation section of our tool. CLP-BNR is employed to solve a set of constraints built for transactions in a system, and to obtain the priority assignment for every task in the system.

The reason for using CLP-BNR is: CLP-BNR has an interval arithmetic system, and can handle both functional interval arithmetic operations and relational interval arithmetic operations. In our tool, the constraint solver for priority assignment needs relational interval arithmetic as well as functional interval arithmetic. Therefore, CLP-BNR is a good choice for the implementation language. CLP-BNR has also been used to solve other hard problems such as the Traveling Salesman problem [28], non-linear constrained optimization problems[29], and for solving ordinary differential equations [27].

3.2 System Architecture: Basic Idea

The system model considered in this research is defined in Chapter 2. The tool developed generates a mapping of tasks to processors as well as a feasible priority assignment for all tasks in the system. The computed priorities can then be used by the operating system of the hard real-time system for dispatching the parallel executable tasks during operation.

Each task requires a processor to execute. The functionality of the allocator developed in this research is to assign tasks to processors by using a static allocation policy. The objective of allocation is to increase system schedulability. As noted previously, the priority computation module developed in this research does not use any explicit scheduling algorithms. The hard deadlines for each transaction is expressed as a time constraint. So for a hard real-time system composed of a set of transactions with hard deadlines, a set of constraints, which are based on system information such as task execution times and deadlines, and allocation information as well, is built for the set of transactions respectively. The objective of the priority computation module is to find a feasible priority assignment for the system by solving the set of constraints built. An example of constraints for a simple transaction, A, which consists of only two tasks: task 1 and task 2, is provided here. Task 1 is an immediate regular predecessor of task 2. The deadline of transaction A is D . The worst-case completion time of task 2 is composed of three parts: the time waiting for all of its immediate predecessor(s) to complete, G_2 , the time that task 2 gets delayed by other tasks which are independent with respect to task 2 and allocated on the same processor, W_2 , as well as the execution time of task 2, E_2 . So the completion time of task 2

is expressed by the following equation:

$$C_2 = G_2 + W_2 + E_2$$

Task 2 is the end task of transaction A. The constraint for transaction A is given by following inequality:

$$C_2 \leq D$$

Note that there is no constraint required for task 1 which only contributes to G_2 .

When a priority assignment can ensure all transactions in a hard real-time system meet their deadlines, it can be stated that it is a feasible priority assignment for the system.

The tool takes the user system descriptions, which include transaction description as well as resource description as input in order to generate a mapping of tasks to processors. A feasible priority assignment for the system is obtained by building a set of timing constraints for all transactions in the system and then solving them. A high level description of the operation of the tool is illustrated in Figure 3-1.

Figure 3-1: High Level Description of Tool Operation

3.2.1 Description of the System Architecture

The tool has two major components: the allocation and the priority computation modules. The allocator generates a mapping of tasks to processors for multiprocessor systems. The priority computation module generates a feasible priority set for all tasks in the system in such a way that all transactions in the system can be guaranteed to meet their hard deadlines. To make the tool more powerful, the following additional requirements are considered.

- If a system is found to be “unschedulable” by the tool, it should give some suggestion of changes for making the system schedulable. If requested by the user, the tool should be able to make the change(s): generate a new mapping of tasks to processors and then compute a feasible priority assignment based on the new allocation information for the system.

- The tool should be able to analyze existing systems, and verify the designs of existing systems. Here, an existing system means a system with not only information of transactions and processors, but also the information of allocation (the mapping of tasks to processors) and a priority assignment of the system.
- For a schedulable system, the tool should be able to suggest reduction in processing resources, if a feasible schedule can be produced with such a reduction in system resources.

The two key components of the tool are the allocation and the priority computation modules. These are implemented by using C and CLP-BNR programming languages respectively. The tool runs under the Solaris operating system on Sun workstations. The major modules of our tool are shown in Figure 3-2. The input module manipulates the input system descriptions (resource and transaction descriptions) to generate the inputs to our allocation and priority computation modules. The interface between the input module and the allocation module is a input file which contains the system information provided by the user. The allocation module takes the inputs from the input module to generate a mapping of tasks to processors for multiprocessor systems. The priority computation module contains two parts: a constraint builder and a constraint solver. The constraint builder is used to build time constraints for every transaction in the system based on the inputs from input module and the allocation information generated by the allocator. The constraint solver solves those constraints built by the constraint builder to produce a feasible priority set if the system is schedulable. The system analysis module is used to control the resource reduction and to verify existing system designs.

All modules are written in C except the constraint solver which is written in CLP-BNR. The interfaces between modules which are written in C are through global variables and C procedure calls. The constraints for the constraint solver are generated by the constraint builder. The interface between the constraint builder and the constraint solver is through C function calls for CLP-BNR predicates.

3.2.2 Functions of The Tool

The allocation and scheduling tool consists of four major functions (see Figure 3-2).

- *System Feasibility Analysis Function.* This is the function to process user input data. One of its jobs is to generate the system description which will be used by the allocator and the priority computation module. Also, It checks the system description to see whether or not it is a valid system description that confirms to the system model defined in Section 2.2. It does some feasibility analysis for the system, as well. For example, if scheduling with given resources is impossible, it points it out to user.
- *Allocation Function.* This function allocates tasks to processors for multiprocessor systems.
- *Priority Computation Function.* This function includes two parts: constraints building and constraints solving. It is used to find a feasible priority assignment for a system. The constraint solver is built by using the CLP-BNR programming language.
- *Re-Allocation Function.* It is used to re-allocate tasks to processors in situations where the previous allocation(s) failed to generate a priority set with which all transaction deadlines are met.

A flow chart for the tool in Figure 3-3 shows the flow of control through these functions. The implementation details are provided in Chapter 4 *Tool Implementation*. The way how the tool works is shown in Figure 3-3. First, the input module gets input from the input file

which contains the system descriptions provided by the user. Then the tool does some feasibility analysis. If the system is feasible and if we are only verifying existing system design, it goes to the existing system verification module. Otherwise, it starts the allocation process. If task allocation is specified by the user the allocation step is skipped. Then the tool begins the priority computation to find a feasible priority assignment for the system. If it cannot find any feasible priority assignment for the system, it starts the re-allocator to generate a new mapping of tasks to processors and then goes back to the priority computation step. A maximum number of re-allocation iteration is specified by the tool designer. If the maximum count is reached and still no feasible priority set found, the tool will terminate itself. If priority computation succeeds, it will do resource reduction if it is requested by user. Otherwise the tool will output the results and terminate itself.

AI: Allocation Information

Figure 3-3: Flow Chart Diagram for The Tool

Chapter 4 Tool Implementation

The allocation and scheduling tool is composed of six functions: system feasibility analysis, allocation, re-allocation, priority computation, existing system verifying, and resource reduction for schedulable systems. The priority computation function is implemented in CLP-BNR; other functions are implemented in C. In this chapter, the detailed implementation of these six functions is described.

4.1 User System Description

As shown in Figure 3-2, the input for the tool is the system description provided by the user, which consists of the transaction description (software) and the resource description (hardware). Based on the system model defined, and assumptions made in Chapter 2, the transaction description of a system requires the following five kinds of information:

- Worst-case execution time of every task
- period and deadline of each transaction
- precedence relationships among tasks
- worst-case communication time between tasks
- allocation restriction (if any) for each task (e.g. a task can only execute on certain type of processors)

The resource description contains the following three kinds of data:

- Total number of processors provided by the system
- task allocation restriction (if any) for processors. If processors are not same, a task may be restricted to execute on certain type(s) of processor(s).
- The data transmission speed of every communication link, if task communication is to be considered.

The user system description is described by a file in text-only format, which is called the input file of the tool. The input file contains up to ten kinds of data describing the characteristics of transactions and resources in the system.

- *Total number of tasks (n), and total number of processors (m), in the system.* They should be given at the beginning of the input file, because they provide the sizes of memory blocks which should be allocated for user system description data. They are integer type parameters.
- *Worst-case execution time of each task (WCET).* An array, E , containing n positive real numbers is used. It should be noted that tasks are named distinctly by integer numbers from 1 to n . The first element of WCET array, $E[0]$, stands for the worst-case execution time of task 1, and so on. The naming convention used for processors is similar to that of tasks. The processors are named distinctly by integer numbers from 1 to m . So every task in a system has a distinct name, so does each processor in the system. Determining the worst-case task execution times requires a significant amount of effort (see [25] for example) and is beyond the scope of this thesis.

- *Period of each task.* An array, P , containing n positive real numbers. $P[i]$ (i is an integer from 0 to $(n-1)$) is used to specify the period of task $(i+1)$. As noted before, the period of every task, belonging to a regular transaction, should be the same as the period of the transaction. The period of every task belonging to a complex transaction, may be different from the period of the transaction. The relation between the period of task and the period of the transaction that it belongs to is described in Section 2.2.1.3.
- *Deadline of each task.* An array, D , containing n positive real numbers, is used to specify the upper bound on the deadline of task 1 to n respectively. The deadline of each task, belonging to a regular transaction, should be no greater than the deadline of the transaction. The deadline of a task, which has OPR or EPR precedence relationship (defined in Section 2.2.1.3) with its predecessor, may be greater than the deadline of the transaction. But the deadline of each task should be no greater than the period of the task, which is represented by following inequality:

$$D[i] \leq P[i]$$

where $0 \leq i \leq (n-1)$.

Note: User can specify the accurate deadline for each task. But for most of the cases, it is not easy to do that. For our tool, user can use the transaction deadline or period as the upper bound on the deadline of the tasks which belong to the transaction, and put it in the entry for the task in the task deadline array. In the priority computation for the system, the

time constraints are built for each transaction not each task. The reason for this is to simplify the design of the tool.

- *Precedence relationship among tasks.* A two dimensional array, $R[i][j]$, is used to contain precedence relationship information of a system with n tasks. It is called the **Precedence Matrix** of the system. The value of each element in Precedence Matrix is an integer from -5 to 5 . The meaning of these values are described:

- 1) $R[i][j] = 0$: task $(i+1)$ is not an immediate (regular, OPR, EPR and TPR) predecessor of task $(j+1)$.

Note: this is true for the input file. After the precedence matrix is processed by the initial feasibility analysis function, $R[i][j] = 0$ means task $(i+1)$ and task $(j+1)$ are independent of each other, which means there are no precedence relationship (immediate or indirect) between them.

- 2) $R[i][j] = 1$: task $(i+1)$ is an immediate regular predecessor of task $(j+1)$
- 3) $R[i][j] = -1$: task $(i+1)$ is an immediate regular successor of task $(j+1)$.

Note: the negative values from -1 to -5 for $R[i][j]$ are used by the tool, not by the user input file. In the input Precedence Matrix, user just needs to indicate the immediate predecessor relationship (regular, OPR, EPR, and TPR precedence relationship).

- 4) $R[i][j] = 2$: task $(i+1)$ is an immediate OPR predecessor of task $(j+1)$
- 5) $R[i][j] = -2$: task $(i+1)$ is an immediate OPR successor of task $(j+1)$
- 6) $R[i][j] = 3$: task $(i+1)$ is an immediate EPR predecessor of task $(j+1)$
- 7) $R[i][j] = -3$: task $(i+1)$ is an immediate EPR successor of task $(j+1)$

8) $R[i][j] = 4$: task $(i+1)$ is an immediate TPR predecessor of task $(j+1)$

9) $R[i][j] = -4$: task $(i+1)$ is an immediate TPR successor of task $(j+1)$

10) $R[i][j] = 5$: task $(i+1)$ is an indirect predecessor of task $(j+1)$.

Note: this value is used by the tool, not by the user input file.

11) $R[i][j] = -5$: task $(i+1)$ is an indirect successor of task $(j+1)$.

A simple Precedence Matrix example is given. There are three tasks, task 1, task 2, and task 3, in the example system. Task 1 is an immediate regular predecessor of task 2, task 2 is an immediate OPR predecessor of task 3. The user input Precedence Matrix is following

Precedence Matrix: $\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \end{bmatrix}$. The Precedence Matrix is changed by the tool as: $\begin{bmatrix} 0 & 1 & 5 \\ -1 & 0 & 2 \\ -5 & -2 & 0 \end{bmatrix}$.

- *Data transmission speed of each communication link.* If the data transmission speed is the same between every processor pair, a float type variable, dis , is used to contain the data transmission speed of the system. Otherwise, a two dimensional array, $Dis[m][m]$, is used to contain this information. $Dis[i][j]$ stands for the data transmission speed of the communication link between processor $(i+1)$ and processor $(j+1)$. $Dis[i][i]$ should be set to 0, and other entries should be positive real numbers.
- *Worst-case data length sent between tasks.* This is stored as an array, $Com[n][n]$. $Com[i][j]$ represents the worst-case amount of data transmitted from task $(i+1)$ to task $(j+1)$. $Com[i][i]$ should be 0, and other entries are non-negative float numbers.

Note: the unit of this information should be consistent with the unit of data transmission speed for communication links. For example, if the unit of worst-case data amount sent between tasks is mega bits (Mbit), the unit of data transmission speed should be Mbit/second.

- *Resource restriction information for each task, if it is necessary. An array, $RR[m][n]$, is used for resource restriction information. $RR[i][j] = 0$ means that task $(j+1)$ cannot execute on processor $(i+1)$. $RR[i][j] = 1$ means that task $(j+1)$ can execute on processor $(i+1)$.*

Note: if any task in the system can execute on any processor, the resource restriction matrix can be omitted in the input file.

- *Allocation information. This is optional. It is for systems in which the user provides the mapping of tasks to processors. An array, $Allc$, is used to represent this kind of information. $Allc[i] = j$ means that task $(i+1)$ is allocated on processor $(j+1)$.*

Note: if user wants to provide partial allocation information, the resource restriction matrix can be used for this purpose. For example, if task i is allocated on processor j , then the resource restriction matrix can indicate that task i can be allocated only on processor j .

- *Priority assignment information for each task. This is also optional data. It is for verifying existing system. It is specified by an array Pra . Equation $Pra[i] = j$ means that priority of task $(i+1)$ is j , where $1 \leq j \leq l$. Here, l is the total number of tasks allocated on the processor on which task $(i+1)$ is allocated.*

Note: priorities of tasks allocated on the same processor should be distinct from each other. The larger the number, the higher is the priority level. It is assumed that the number of priority levels

that can be handled by the operating system is greater than or equal to the maximum number of tasks allocated on a processor.

A simple input file example is shown in Figure 4-1. The left section shows the input file of the system. The right section is the explanation of each field of the input file.

Input File	
6 2	: The system has 6 tasks and 2 processors.
1 0.6 1.4 4 2.1 3.2	: WCET of task 1 to 6 respectively.
10 10 8 14.6 29.2 14.6	: Period of each task.
10 10 6 12.8 25 12.8	: The deadline of a transaction is used as the upper bound on deadlines of tasks that belong to the transaction.
0 1 0 4 1 0 0 0 0 0 0 0 0 0 0 0 0	: The system contains three transactions. The precedence graph is shown as follows:
<pre> graph TD task1((task 1)) --> task2((task 2)) task4((task 4)) --> task5((task 5)) task4 --> task6((task 6)) task3((task 3)) </pre>	
0 1 0 1 0 0	: Data transmission speed matrix. For this system, it is not necessary to include it in input file. Because it can be specified by one value. Here we just show it as an example.
0 0.2 0.1 0.02 0 0 0 0 0 0 0 0 0 0 0 0	: Worst-case amount of data transmitted among tasks.
1 1 1 1 1 0 1 0 1 0 1 1	: Resource restriction matrix. Task 2 and task 4 can not execute on processor 2. Task 6 can not run on processor 1.
1 1 2 2 1 2	: Allocation Information. Task 1, task 2 and task 5 are allocated on processor 1. The other tasks are on processor 2.
1 3 3 1 2 2	: Priority Information. On processor 1, task 1 has the highest priority level (1). Then it is task 5 (2). Then it is task 2 (3). On processor 2, the priority order is task 4, task 6, then task 3.

Figure 4-1: An Input File Example

4.2 System Feasibility Analysis

After the tool obtains the system description data from the input file, the system feasibility analysis function is called. One of its jobs is checking the input system description to see whether the user system is a valid system or not. A valid system is a system that conforms to the system model defined in Section 2.2. Another job of the system feasibility analysis function is undertaking some basic schedulability analysis for the input system. This is a necessary protection for the tool to prevent invalid and some unschedulable systems from becoming further processed by other functions of the tool. For example, A loop in the precedence graph will cause infinite loop in the program used for constraints building.

The system validation check-up includes the following features:

- **Period check:**

- 1) (Rule 1). If task i is an immediate regular successor of task j , their periods should be the same: $P[i] = P[j]$.
- 2) (Rule 2). If task i is an immediate OPR or EPR successor of task j , the period of task i should be twice the period of task j : $P[i] = 2P[j]$.
- 3) (Rule 3). If task i is an immediate TPR successor of task j , the period of task i should be twice the period of task j : $P[i] = \left(\frac{1}{2}\right)P[j]$.

- **Deadline check:**

- 1) (Rule 1). Deadline of each task, say i , should be no greater than the period of the task: $D[i] \leq P[i]$.

- 2) (Rule 2). If task i is an immediate EPR successor of task j , the deadline of task i should be no less than the sum of the period of task j and the worst-case execution time of both tasks: $P[j] + E[i] + E[j] \leq D[i] \leq P[i]$.

• **Precedence relationship check:**

- 1) (Rule 1). Every transaction in the system can be represented by an acyclic precedence graph. In other words, there is no loop precedence relationship in any transaction in the system.
- 2) (Rule 2). There should be no inconsistent precedence relationship among tasks. An example of inconsistent precedence relationship is shown in Figure 4-2. In the example, task 4 is an immediate OPR successor of task 2, so it only occurs in every odd number of occurrences of the transaction. But from the relationship between task 4 and task 3, task 4 can only occur in every even number of occurrences of the transaction. They conflict with each other.

Figure 4-2: An example of inconsistent precedence relationship

Also, some other checking features are provided in order to check other data provided in the system description. These include checking the values of the input data to be in the range they should be, e.g. the periods, deadlines of tasks should be positive real numbers; checking the resource restriction matrix to see if every task has some processor(s) on which it can be allocated.

The basic schedulability analysis includes the following features:

- **Utilization check:** Utilization of a task, say i , is given by: $UT[i] = \frac{E[i]}{P[i]}$ ($UT[i]$ is the utilization of task i . $E[i]$ is the worst-case execution time of task i , $P[i]$ is the period of task i). The utilization of a processor, say j , is the sum of utilization of all tasks allocated on the processor: $UP[j] = \sum_{i \in \Phi} UT[i]$ ($UP[j]$ is the utilization of processor j . Φ represents the set of tasks allocated on processor j)
- 1) (Rule 1). For systems with allocation information, utilization of each processor should be no greater than 1.

Note: If utilization of a processor is larger than 1, the system is unschedulable for sure.

- 2) (Rule 2). For system without allocation information, the sum of utilization of all tasks in the system should be no greater than the number of processor(s).
- **Best-case finish time check:** Based on the allocation information, this function will compute the best-case finish time for all tasks in the system, which includes communication time between tasks. For system without allocation information, this check will be done after the allocation function is called. Note that the best-case finish time is the finish time of a task that results with no contention on the

shared resource; the task can execute whenever it is ready. The best-case finish time of each task should be no greater than deadline of the task.

The system feasibility analysis function also performs some data processing. For example, it generates a new precedence matrix, which includes not only immediate precedence relationship information, but also indirect precedence relationship information, based on the precedence matrix given. Also, it computes the best-case finish time for each task, which will be used by the re-allocation function and the priority computation function later on. It will also compute the worst-case communication time between tasks based on allocation information.

4.3 Allocation and Re-Allocation

4.3.1 Allocation

The allocation function is used to map tasks to processors for multiprocessor systems. Some idea from Coffman's MULTIFIT (bin-packing) allocation algorithm [4] and from Woodside & Monforton's MULTIFIT-COM allocation algorithm [2] are used. But the allocation algorithm nevertheless has lots of differences with them, because the system model handled by the allocator used in this project is more complicated. MULTIFIT and MULTIFIT-COM allocation algorithms consider the problem of allocating n non-periodic tasks to m identical processors. MULTIFIT-COM also handles communication between tasks. The allocation problem considered here is to allocate n tasks that belong to periodic transactions to m processors. Processors may not be identical, and some tasks may not execute on some processors. Communication link speed can also vary from one link to

another. Tasks may exhibit complex precedence relationships.

The allocation program is implemented in C. It is based on utilization of each processor and communication time between tasks. If two tasks are allocated on different processors, the worst-case communication time between them is computed from the worst-case amount of data sent between them and the communication link speed between the two processors. Communication time between tasks allocated on the same processor is considered as 0. Note that contention for a link by multiple tasks is ignored. Future versions of the tool should consider handling link contention.

The objectives of the allocation algorithm are as follows:

- For systems in which communication is not considered, tasks are allocated to processors in such a way that utilization of each processor is approximately the same.
- For systems in which communication between tasks is considered, the objective of allocation algorithm is to find an effective trade-off between utilization of each processor and communication time among tasks.

The algorithm used by our allocator is presented next.

Function Allocation {

Initialize the utilization of each processor to 0;

Determine initial task sizes; *(i)*****

sort the tasks in non-increasing order;

for every task (say i) in the sorted order set {

find the processor set, PI, that task i can be allocated on; *(ii)*****

if communication is considered {

find the lowest utilization, Umin, of all processors in set PI;

```

    find a subset of processors of set PL, PIsub, such that utilization of each processor in
    PIsub, is less than Umin*(1+R); ***(iii)***
    for each processor in set PIsub, compute the sum of transmitted data amount
    between each task on the processor and task i;
    find the processor, say j, with the highest value of this sum of transmitted data
    amount in set PIsub; if there is a tie, choose the processor with lowest utilization;
    allocate task i to processor j;
  } else {
    find a processor in set PL, say j, with the lowest utilization;
    allocate task i on processor j;
  }
}
}

```

i) Utilization of each task is used as the initial task size. There are three possible ways of computing this initial task size:

1) (Option 1). Utilization of each task is computed by equation: $UT[i] = \frac{E[i]}{P[i]}$.
 $P[i]$ is the actual period of task i . This can be used for all kinds of systems.

2) (Option 2). Utilization of each task is computed by equation: $UT[i] = \frac{E[i]}{MinP[i]}$.
 $MinP[i]$ is the smallest value of periods of all predecessors of task i . This can be used for systems with complex transaction(s). For some complex transactions it is possible that utilization is very high for some period of time. If option 1 is used, it may lead to a mapping of tasks to processors which results in an unschedulable system. An example will be shown in Section 5.1.3.

3) (Option 3). Utilization of each task is computed by equation: $UT[i] = \frac{E[i]}{AveP[i]}$.
 $AveP[i]$ is the average of the periods of all predecessors of task i . This can be used by systems with complex transactions.

User can choose one of the options for allocating tasks to processors.

- ii) There are two issues to be considered here. One is that allocating the task on the processor will not result the utilization of the processor to be larger than 1. The other is that the task can execute on the processor, according to the resource restriction information provided for the task.
- iii) There is a trade-off between utilization of each processor and communication among tasks. $R = 0$ means that only processor utilizations are being considered in the allocation process, where as $R = \infty$ means that only inter-task communication is being considered. We have chosen $R = \min\left(0.1, \frac{1}{U_{min}} - 1\right)$. This was observed to produce effective allocation decisions.

The current implementation of the tool does not check whether the utilization of a communication link is always below 1. Since the reduction of inter-processor communication is one of the objectives of the allocation algorithm, the possibility of saturation of such a communication link is quiet remote. The incorporation of such a check in the tool is straight forward and should be considered in the future versions.

4.3.2 Re-Allocation

The re-allocation function is used to allocate tasks to processors when the previous allocation has failed. The algorithm for re-allocation is different from that of allocation. It is based on the previous allocation as well as the input system description. The objectives of re-allocation are as follow:

- To minimize the communication time between tasks belonging to the critical

transaction. A critical transaction is a transaction with the minimum difference between its deadline and best-case finish time of the longest path of the transaction. The difference between deadline and the best-case finish time of a task is called the *critical value* of the task.

- To reduce the utilization of heavily loaded processors.

The algorithm of re-allocation is as follows:

Function Re-Allocation{

if communication is considered {

find the transaction with minimum critical value;

loop for each task in the transaction{

find a processor set P_i , that task i can be allocated on;

compute the sum of the communication time between tasks on the processor and task i for each processor in P_i ;

re-allocate task i on the processor with maximum sum of communication time with tasks allocated on the processor;

}

else {/*communication between tasks are considered as zero.*/

find a processor P_j with the maximum utilization;

sort tasks allocated on processor P_j in the non-decreasing order of task utilizations;

for each task in the sorted set{

find the processor P_i with the minimum utilization in the processor set that the task can be allocated on;

if the sum of utilization of processor P_i and the utilization of the task is not greater than the utilization of processor P_j , re-allocate the task to processor P_i ;

}

}

}

The number of times a system can be re-allocated is upper bounded by a value specified by the designer. A system can be re-allocated until a feasible priority assignment is found for the system, or the specified number of iterations is reached.

4.4 Priority Computation

Priority computation is the core function of our tool. Based on the system description and allocation information, it builds a set of constraints for all transactions in the system, and solves the set of time constraints to obtain a feasible priority assignment for the system. The priority computation is composed of two parts: constraints building and constraints solving.

4.4.1 Time Constraints

The time constraints for a transaction with preemptive scheduling is different from time constraints when non-preemptive scheduling is used. For preemptive scheduling, a lower priority task can be preempted by higher priority tasks(s) on the same processor. For non-preemptive scheduling, a task will continue running until completion after it starts executing. Higher priority tasks cannot preempt a lower priority task which might be running. In the following analysis, time is measured from the beginning of a transaction period. The period of a regular transaction is equal to the period of any task in the transaction. In a complex transaction, different tasks may have different periods. The period of a complex transaction is defined to be equal to the largest task period.

4.4.1.1 Time Constraints for Preemptive Scheduling

For preemptive scheduling, the worst-case finish time of a task, say i , is composed of two parts: The first is the time period from the point at which an occurrence of the transaction occurs to the point at which task i becomes ready to run. In the other words, it is the period of time task i waits for all its predecessor(s) to finish. It is called *wait-to-ready*

time of task i . The other is the period of time measured from task i becoming ready to the completion of its execution. It is called **ready-to-completion time** of task i . The worst-case finish time of task i is thus given by the following equation:

$$F_i = \max(G_j, \dots, G_k) + W_i \quad (1)$$

Where

F_i : the worst-case completion time of task i

W_i : the worst-case ready-to-completion time of task i

$\max(G_j, \dots, G_k)$: the wait-to-ready time of task i . $(G_j, \dots, G_k)$ represents the worst-case

completion times of all the predecessors of task i . The definition of G_l is as follows:

- If task l is an immediate regular, or OPR, or TPR predecessor of task i ,
 $G_l = F_l + Com_{li}$. Com_{li} is the worst-case communication time between task l and task i . If task l and task i are allocated on the same processor, Com_{li} is 0.
- If task l is an immediate EPR predecessor of task i , $G_l = P_l + F_l + Com_{li}$.

For preemptive scheduling, task i can only be preempted by tasks on the same processor that have higher priority than task i , and are independent with respect to task i . A task is said to be independent with respect to task i when it is neither a predecessor nor a successor of task i . Lehoczky et al. [9] showed that W_i , which is the worst-case ready-to-completion time of task i , is the smallest non-negative root of the following equation:

$$x = g(x) = \left(\sum_{j \in \Phi} \left\lceil \frac{x}{P_j} \right\rceil \right) E_j + E_i \quad (2)$$

Φ represents the set of tasks allocated on the same processor with task i which have higher priority than task i , and are independent with respect to task i .

According to Equation (2), w_i can be computed by using following equation:

$$W_i = \left(\sum_{j \in \Phi} \left\lceil \frac{W_i}{P_j} \right\rceil \right) E_j + E_i \quad (3)$$

Note that $\left\lceil \frac{W_i}{P_j} \right\rceil$ is an upper bound on the number of times task i can be delayed by task j .

The definition of Φ in Equation (3) is the same as in Equation (2). An important point here is that the priority of each task is unknown and is the output of our tool. So how can we find the Φ for each task? The solution is in Equation (4);

$$W_i = \left(\sum_{j \in \Gamma} \left(\min \left(1, \max \left(0, PR_j - PR_i \right) \right) \right) \left\lceil \frac{W_i}{P_j} \right\rceil \right) E_j + E_i \quad (4)$$

Γ represents the set of tasks that are allocated on the same processor on which task i are allocated, and are independent with respect to task i . PR_j represents the priority of task j , and priority value of each task should be a non-negative integer. As we mentioned before, if $PR_j > PR_i$, it means that task j has higher priority than task i . In case that task j has higher priority than task i , which means $PR_j > PR_i$, $(\min(1, \max(0, PR_j - PR_i)))$ is 1. In case that task j has lower priority than task i , $(\min(1, \max(0, PR_j - PR_i)))$ is 0. In Equation (4), only those

tasks with higher priority than task i in task set τ contribute to w_i . So we can say that Equation (4) is the same as Equation (3). Equation (4) is used by the priority computation module to build constraints for systems with only regular transactions. Later, when a task set is used to represent tasks with higher priority than a task, say i , it means that expression $\min(1, \max(0, PR_j - PR_i))$ is used to select tasks that have higher priority than task i .

If there are complex transactions in the system, the solution is different. Because for a system containing complex transactions, some task in Φ may not be able to preempt task i . For example, in Figure 4-3, all tasks are allocated on the same processor. Task 2 and task 3 are independent with respect to each other. Assume that task 2 has higher priority than task 3; however, task 2 cannot preempt task 3, because in every period of 50ms, task 2 must complete before 20ms (which is the deadline of task 2), and the best-case start time of task 3 is 26ms (The best case of task 1 finishing its second execution is 26ms). Assume that both task 2 and task 3 have higher priority than task 4. Task 2 is an immediate OPR successor of task 1, and task 3 is an immediate EPR successor of task 1. This means that task 2 and task 3 will not compete with each other at any time. So we can say that at least within period of 6ms (26ms-20ms), it is not possible for both task 2 and task 3 to preempt task 4.

Figure 4-3: An Example of System with Complex Transactions

OPR and EPR precedence relationships group tasks into different groups. Tasks with the same immediate OPR (EPR) predecessor are in the same group. If two groups have the same immediate OPR and EPR predecessor, they are a pair of non-competing groups, which means that tasks in one group do not compete with the tasks in the other group. First consider that there is just one pair of non-competing groups on a processor. If task i does not belong to any of the non-competing groups. w_i is computed by the following equation:

$$w_i = \left(\sum_{j \in \Gamma} \left(\left(\min \left(1, \max \left(0, PR_j - PR_i \right) \right) \right) \left\lceil \frac{w_i}{P_j} \right\rceil \right) \right) E_j + \max(EO_1, EO_2) + E_i \quad (5)$$

Γ represents a task set which contains all tasks allocated on the same processor with task i , are independent with respect to task i , and do not belong to any non-competing

groups. PR_j represents the priority of task j as in Equation (4).

$\max(EO_1, EO_2)$ represents the time period that task i waits for tasks in the two non-competing groups on the same processor which have higher priority than task i , and independent with respect to task i . The two groups are represented as $group_{opr}$ and $group_{epr}$ in Equation (6) and (7). EO_1 and EO_2 are computed by the following equations:

$$EO_1 = \left(\sum_{l \in group_{opr}} \left\lceil \frac{\max(0, W_i - D)}{P_l} \right\rceil \right) E_l + \left(\sum_{n \in group_{epr}} \left\lceil \frac{W_i}{P_n} \right\rceil \right) E_n \quad (6)$$

$$EO_2 = \left(\sum_{l \in group_{epr}} \left\lceil \frac{\max(0, W_i - D)}{P_l} \right\rceil \right) E_l + \left(\sum_{n \in group_{opr}} \left\lceil \frac{W_i}{P_n} \right\rceil \right) E_n \quad (7)$$

Where $D = \max(0, BBT_{epr} - DL_{opr})$. BBT_{epr} is the best-begin time for the task in EPR group that executes first. The best-begin time of a task corresponds to the situation in which there is no resource contention. As soon as the task becomes ready, it can begin execution. DL_{opr} is the largest of the task deadlines in the OPR group. D is the shortest period between tasks in OPR group finishing executions and tasks in EPR group starting executions. An example is the 6ms time period for the system in Figure 4-3. D is the shortest period of time in which both groups of tasks cannot delay tasks not belonging to any non-competing group.

In Equation (6), EO_1 represents the waiting time for the case that in the period of duration D , tasks in the OPR group do not delay task i . In Equation (7), EO_2 represents the waiting time for the case in which the period of duration D , tasks in the EPR group do not delay task i . The worst-case waiting time that task i waits for tasks in the two non-

competing groups, which are independent with respect to task i and have higher priority than task i , is $\max(EO_1, EO_2)$, which is used in Equation (5).

If task i belongs to one of the non-competing groups, w_i is computed by the following equation:

$$w_i = \left(\sum_{j \in \Phi} \left\lceil \frac{w_i}{P_j} \right\rceil \right) E_j + E_i \quad (8)$$

Where Φ is a task set which contains all tasks allocated on the same processor that have higher priority than task i , are independent with respect to task i , and do not belong to the other non-competing group.

These equations shows how to compute w_i if there is only one pair of non-competing groups on the processor. If there are more than one pair of non-competing groups, say M pairs, the $\max(EO_1, EO_2)$ part in Equation (5) comes to be $\sum_{k=1}^M \max(EO_{1k}, EO_{2k})$. Here, EO_{1k} represents the EO_1 value for k^{th} non-competing group pair. EO_{2k} represents the EO_2 value for k^{th} non-competing group. Other terms in the right hand side of Equation (5) will be the same. Equation (9) is applicable to tasks that belong to one of the non-competing groups. If task i belongs to one group of the k^{th} pair of non-competing groups, then w_i is computed by the following equation:

$$W_i = \left(\sum_{j \in \Phi} \left\lceil \frac{W_i}{P_j} \right\rceil \right) E_j + \sum_{r \neq k} \max(EO_{1r}, EO_{2r}) + E_i \quad (9)$$

Where Φ is a task set which contains all tasks allocated on the same processor that have higher priority than task i , are independent with respect to task i , and do not belong to any non-competing groups OR belong to the same non-competing group that task i belongs to.

The above equations show how to develop an expression for the worst-case completion time of each task in the system. The time constraint that we need to satisfy is the worst-case completion time of the last task(s) of every transaction should be no greater than the deadline of the transaction.

4.4.1.2 Time Constraints for Nonpreemptive Scheduling

For nonpreemptive scheduling, a task will not be preempted by a higher priority task when it is running. The worst-case finish time of task i , is composed of three parts: the first part is the wait-to-ready time. The second is the time period from task i becoming ready to the start of its execution. It is called *ready-to-start time* of task i . The third is the worst-case execution time of task i . The worst-case finish time of task i is given by the following equation:

$$F_i = \max(G_j, \dots, G_k) + W_i + E_i \quad (10)$$

Where F_i : the worst-case completion time of task i

W_i : the worst-case ready-to-start time of task i . Note that W_i for nonpreemptive scheduling

does not include the execution time of task i , and includes the time that task i waits for a lower

priority task, which is running when task i becomes ready, to complete execution. W_i for preemptive scheduling includes the execution time of task i , but does not include the time for waiting any lower priority task, because higher priority task can always preempt lower priority task in preemptive scheduling.

$\max(G_j, \dots, G_k)$: the wait-to-ready time of task i . This is as same as in Equation (1).

Other notations are the same as those in Equation (1).

For nonpreemptive scheduling, if there is no non-competing groups on the processor which task i is allocated on, W_i can be computed by using following equation:

$$W_i = \left(\sum_{j \in \Phi} \left\lceil \frac{W_i}{P_j} \right\rceil \right) E_j + \max(E_k, \dots, E_m) \quad (11)$$

Where $\max(E_k, \dots, E_m)$ is the maximum worst-case execution time of tasks which are allocated on the same processor with task i , and independent with respect to task i , and have lower priority than task i . When a higher priority task becomes ready, a lower priority task may be in execution. For non-preemptive scheduling, higher priority task cannot preempt the lower priority task which starts running. The worst-case is when a higher priority task becomes ready, the lower priority task, which has the maximum worst-case execution time among the lower priority tasks, has just started running. $\max(E_k, \dots, E_m)$ presents the waiting time for this situation.

Note that the other terms in Equation (11) are the same as those in Equation (3).

For a system containing complex transactions, the w_i for a task, i , that belongs to any non-competing group is computed by following equation:

$$W_i = \left(\sum_{j \in \Phi} \left\lceil \frac{W_i}{P_j} \right\rceil \right) E_j + \sum_{r \neq l} \max(EO_{1r}, EO_{2r}) + \max(E_k, \dots, E_m) \quad (12)$$

Where l is the non-competing group pair that contains task i . $\max(E_k, \dots, E_m)$ is the maximum worst-case execution times of tasks which are allocated on the same processor with task i , and independent with respect to task i , and have lower priority than task i . These tasks do not belong to the other non-competing group, which task i does not belong to, in the non-competing group pair l . When a higher priority task becomes ready, a lower priority task may be in execution. The other terms used in Equation (12) are the same as those used in Equation (9).

4.4.2 Implementation of Priority Computation

The priority computation is implemented in two parts. One is the constraints building function, which is implemented in C. The other is the constraints solving function, which is implemented in CLP-BNR. A constraint builder is used to generate a CLP-BNR program based on the input system description and allocation information. This is shown in Figure 4-4.

Figure 4-4: Priority Computation

The reason why priority computation is implemented in C and CLP-BNR, is because CLP-BNR provides a relational interval arithmetic based constraint solver. This is needed for solving the constraints created by the constraint builder. The constraint solver engine computes a set of priorities that satisfies all the constraints generated by the constraint builder. If such a set of priorities is not feasible, failure is reported by the engine. The allocation information is either from user input or from the computation of our allocator. The constraint builder, written in C, is responsible for generating time constraints for all transactions in the system, based on the user system description and allocation information. Its output is a constraints CLP file. The interface function is used to call constraint solver, which is written in CLP-BNR. The constraint solver CLP library contains some predicates

which are needed by the constraint solver. The constraint solver is used for solving those constraints. Its output is in a file, CLP output file, which contains the feasible priority assignment for the input system if the system is schedulable according to our tool.

4.4.3 An Example of Constraints CLP File of A Simple System

In this section, the constraints CLP file generated by the constraint builder for a simple system is given. There are two identical processors in the system. The precedence graph of this system is given in Figure 4-5.

Figure 4-5: Precedence Graph for the Example System

The allocation generated by the allocator is; task 1, 2, 5 are allocated on one processor, task 3 and 4 are allocated on the other processor. The constraints CLP file for preemptive scheduling of this system generated by the constraint builder is shown in Figure 4-6.

```

ltest:- load_context(mylib),
[X1, W1]:real(0.0, 50.000000),
[X2, W2]:real(0.0, 50.000000),
[X3, W3]:real(0.0, 33.500000),
[X4, W4]:real(0.0, 12.500000),
[X5, W5]:real(0.0, 50.000000),
[P3,P4]:integer(1,2),
distinct([P3,P4]),
[P1,P2,P5]:integer(1,3),
distinct([P1,P2,P5]),
{P1 > P2},
{P2 > P5},

test1(X1, W1, P1, X2, W2, P2, X3, W3, P3, X4, W4, P4, X5, W5, P5),
firstfail([P1,P2,P3,P4,P5]),
!,
open(Stream, result, read_write, 0),
swrite(Stream, P1," ", P2," ", P3," ", P4," ", P5),
close(Stream, 0).

test:- ltest -> true;
[open(Stream, result, read_write, 0),
swrite(Stream, "-1"),
close(Stream, 0)].

test1(X1,W1,P1,X2,W2,P2,X3,W3,P3,X4,W4,P4,X5,W5,P5):-
{X1 == 10}, ← :The worst-case finish time of task 1 is its worst-case
               execution time. Because task 2 and 3 are its successors
               Task 1 cannot be preempted by task 2 and task 3.

{X2 == 10+ X1}, ← :The worst-case finish time of task 2 is its worst-case
                  execution time + the worst-case finish time of its
                  immediate regular predecessor, task 1

[N3_1]:integer(0, _),
{N3_1 < W3/12.5 + 1.0, N3_1 >= W3/12.5},
{W3 ==N3_1 * min(1, max(0, P4-P3))*5+ 5},
{X3 ==W3},
For task 3, the worst-case
execution time is the waiting
period of task 4 (if task 4 has
higher priority than task 3) +
its worst-case execution time

[N4_1]:integer(0, _),
{N4_1 < W4/33.5 + 1.0, N4_1 >= W4/33.5},
{W4 ==N4_1 * min(1, max(0, P3-P4))*5+ 5},
{X4 ==W4},
The case of task 4 is
similar as task 3

{X5 == 5+ X2}, ← : The worst-case finish time of task 2 is the sum of its
true.             worst-case execution time and the worst-case finish
                  time of its immediate regular predecessor, task 2.

```

Figure 4-6: The Constraints CLP File for the Example System

4.5 Verification of Existing Systems and Resource Reduction

The tool can also be used to verify existing design of hard real-time systems. If in the system description, a mapping of tasks to processors and a priority assignment are given, the existing system verification function will call the constraints building function to build a set of time constraints based on all the information obtained. The constraint solver in CLP-BNR will then verify whether or not each of these constraints is satisfied with the given priority set.

For those systems, which are found to be schedulable by the tool, a question comes to mind: are the resources of the system under-used? The resource reduction function is used to find out whether or not the processors in the system are appropriately utilized. It is implemented in the way that when a system is found to be schedulable by the tool, the resource reduction function will keep on decreasing the number of processors used in the system, computing the priorities each time, until it cannot find a feasible priority set for the system.

Chapter 5 Tool Testing

In this chapter testing of the allocation and scheduling tool is presented. First, three case studies are examined. Case 1 is a regular system which contains regular transactions only. Case 2 and Case 3 are systems which contain not only regular transactions but also complex transactions. Non-preemptive scheduling is used in Case 1. whereas preemptive scheduling is used in Case2 and Case 3.

5.1 Test Cases Study

For each test case, the system description is given, and the outputs of the tool for the system are presented. Also, the output result for the system is manually verified by using the input system description.

5.1.1 Test Case 1: A System with Regular Transactions Only

Test case 1 is a simple regular system which consists of regular transactions only. So there are no OPR, EPR and TPR precedence relationships between tasks in the system. The communication between tasks is considered. The system uses non-preemptive scheduling.

5.1.1.1 System Description

There are two regular periodic hard transactions in the system. One transaction contains seven tasks: task 1 to task 7. The other transaction contains 5 tasks: task 8 to task 12. The precedence graph of these two transactions is presented in Figure 5-1.

Note: each task is associated with a label $T(P, D)$
 T = the worst-case execution time of the task
 P = period of the task
 D = upper bound deadline of the task

Figure 5-1: Precedence Graph for Test Case 1

Here, the period of a task is equal to the period of the transaction that the task belongs to. The deadline of the transaction that a task belongs to is used as the upper bound on the deadline of the task.

The period of transaction #1 is 50ms, the deadline of transaction #1 is 50ms. The period of transaction #2 is 25ms, the deadline of transaction #2 is also 25ms. There are three processors available in the system. The data transfer speed of each communication link between processors is 10 kbit/second. The data transfer amount between tasks in transaction #1 is very large, and cannot be omitted. The data transfer amount between tasks in transaction #2 is very small, and can be considered as 0. Note that the data transfer amount for transaction #1 is specified in kbit. The input file is as following:

12 3 (Task number, Processor number)

5 4 3 6 5 1 1 2 2 4 5 6 (Worst-case execution time of each task)

50 50 50 50 50 50 50 25 25 25 25 25 (period of each task)

50 50 50 50 50 50 50 25 25 25 25 25 (upper bound deadline of each task)

0 1 1 0 0 0 0 0 0 0 0 0

0 0 0 1 1 0 1 0 0 0 0 0

0 0 0 0 0 1 1 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 (Precedence Matrix)

0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 1 1 0 0

0 0 0 0 0 0 0 0 0 0 0 1

0 0 0 0 0 0 0 0 0 0 1 0

0 0 0 0 0 0 0 0 0 0 0 1

0 0 0 0 0 0 0 0 0 0 0 0

10 (Communication link speed)

0 4 2 0 0 0 0 0 0 0 0 0

0 0 0 1 5 0 2 0 0 0 0 0

0 0 0 0 0 2 1 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 (Communication Data Size Matrix)

0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 (Communication Data Size Matrix (continue))

0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0

```

1 1 1 0 1 0 0 1 1 0 1 1
1 0 1 1 1 1 1 0 0 1 1 0 (Resource Restriction Matrix)
0 1 1 1 1 1 1 1 1 1 0 1

```

5.1.1.2 Output Results

For the system described above, the tool outputs a mapping of these twelve tasks to the three processors, and computes a feasible priority set for non-preemptive scheduling on the system. The following shows the output of the tool for this system:

The allocation is as follows:

processor 1: task1 task2 task3 task5 task8
processor 2: task6 task7 task10 task11
processor 3: task4 task9 task12

A feasible priority set is as follows:

processor 1: 4(Task1) 2(Task2) 3(Task3) 1(Task5) 5(Task8)
processor 2: 1(Task6) 2(Task7) 4(Task10) 3(Task11)
processor 3: 1(Task4) 3(Task9) 2(Task12)

The utilization of each processor is as follows:

processor 1: 0.42
processor 2: 0.40
processor 3: 0.44

Note that higher priority number implies higher priority.

5.1.1.3 Validation of Tool Output

The time constraint equations for non-preemptive scheduling are given in Section 4.4.1.2. The testing is based on Equation (10) and (11) in that section. The worst-case finish time of each task, F_i , is computed according to Equation (10), and the ready-to-start time of each task, w_i , is computed according to Equation (11).

There are five tasks allocated on processor #1. Task 8 has the highest priority 5, and it is the root task of transaction #2. According to Equation (10), its wait-to-ready time, $\max(G_j, \dots, G_k)$, is 0. Because task 8 is the highest priority task on this processor, W_8 is computed by following equation, which is obtained from Equation (11):

$$W_8 = \left(\sum_{j \in \Phi} \left\lceil \frac{W_8}{P_j} \right\rceil \right) E_j + \max(E_k, \dots, E_m) = 0 + \max(E_k, \dots, E_m) = \max(E_1, E_2, E_3, E_5) = 5ms$$

When task 8 becomes ready, another lower priority task may be in execution. Since non-preemptive scheduling is used, task 8 has to wait for the completion of this task. So the worst-case finish time of task 8 is the sum of the maximum time that it waits for a lower priority task to finish execution, and the worst-case execution time of itself.

$$F_8 = \max(G_j, \dots, G_k) + W_8 + E_8 = 0 + 5 + 2 = 7ms$$

Task 1 has the second highest priority 4 on processor #1. It is the root task of transaction #1. So it does not have any predecessors, and all lower priority tasks on processor #1 are successors of task 1. According to Equation (10), its wait-to-ready time, $\max(G_j, \dots, G_k)$, is 0. W_1 is computed by following equation, which is obtained from Equation (11):

$$W_1 = \left(\sum_{j \in \Phi} \left\lceil \frac{W_1}{P_j} \right\rceil \right) E_j + \max(E_k, \dots, E_m) = \left\lceil \frac{W_1}{P_8} \right\rceil E_8 + 0 = \left\lceil \frac{W_1}{P_8} \right\rceil E_8$$

The following iterative method is used here for solving equation

$$W_i = \left(\sum_{j \in \Phi} \left\lceil \frac{W_i}{P_j} \right\rceil \right) E_j + \max(E_k, \dots, E_m) \text{ to get } W_i:$$

- 1) The initial value of w_i is computed by equation: $w_{i0} = \sum_{j \in \Phi} E_j + \max(E_k, \dots, E_m)$.
- 2) Use the initial value of w_i , w_{i0} , to compute the first iteration value of w_{i1} . If w_{i1} equals to w_{i0} , stop the iteration, the value of w_i is w_{i1} .
- 3) use w_{in} computed from n^{th} iteration to compute $w_{i(n+1)}$. If $w_{i(n+1)}$ is equal to w_{in} , stop the iteration, and the value of w_i is $w_{i(n+1)}$.
- 4) Keep on iterating until we get w_i .

Note that this method is only used in our validation of the tool outputs. In the constraints solver, a CLP-BNR solver is used for solving the equations to get w_i .

By using the iterative method, the value of w_1 is E_8 which is 2ms. So the worst-case finish time of task 1 is the waiting time for task 8 to finish execution and the worst-case execution time of itself.

$$F_1 = E_8 + E_1 = 2 + 5 = 7ms$$

The priority of task 3 is 3. It is an immediate successor of task 1. According to Equation (10) and (11), the worst-case finish time of task 3 is composed of three parts: (1) waiting for task 1 to finish, (2) worst-case ready-to-start time, w_3 , and (3) worst-case execution time of itself, E_3 . Because task 1 and task 3 are allocated on the same processor, the communication time between them is 0.

$$w_3 = \max(E_2, E_5) + \left\lceil \frac{w_3}{P_8} \right\rceil E_8$$

w_3 is computed as 7ms by using the iterative method. The worst-case finish time of task

3 is 17ms.

$$F_3 = F_1 + W_3 + E_3 = 17ms$$

Task 2 has a priority, 2, on processor #1. It is an immediate successor of task 1. The worst-case finish time of task 2 is composed of three parts: (1) waiting for task 1 to finish, (2) worst-case ready-to-start time, w_2 , and (3) worst-case execution time of task 2. According to Equation (11), w_2 is computed by the following equation;

$$w_2 = E_3 + \left\lceil \frac{W_2}{P_8} \right\rceil E_8$$

w_2 is computed as 5ms by using the iterative method. The worst-case finish time of task 3 is 16ms.

$$F_2 = F_1 + W_2 + E_2 = 16ms$$

The lowest priority task on processor #1 is task 5. It is an immediate successor of task 2. w_5 is computed by equation:

$$w_5 = \left\lceil \frac{W_5}{P_3} \right\rceil E_3 + \left\lceil \frac{W_3}{P_8} \right\rceil E_8$$

By using the iterative method, w_5 is computed as 5ms. The worst-case finish time is given by following equation:

$$F_5 = F_2 + W_5 + E_5 = 26ms$$

Task 5 is one of the four end tasks, which are task 4 to task 7, of transaction #1. The worst-case finish time of task 5 is 26ms, which is less than the deadline of transaction #1,

50ms. Note that the transaction deadline is used as the upper bound of the deadline of tasks belonging to that transaction.

For processor #2, there are 4 tasks allocated on it. Task 10 has the highest priority which is 4. It is an immediate successor of task 8. Because non-preemptive scheduling is used, task 10 can be delayed by at most one lower priority task that was in execution when task 10 became ready. w_{10} is thus given by

$$w_{10} = \max(E_6, E_7) = 1ms$$

The worst-case finish time of task 10 is

$$F_{10} = (F_8 + Com_{8,10}) + w_{10} + E_{10} = (7 + 0) + 1 + 4 = 12ms$$

Task 11 has the second highest priority value, 3. It is an immediate successor of task 10. w_{11} is computed in a way similar to the computation of w_{10} .

$$w_{11} = \max(E_6, E_7) = 1ms$$

The worst-case finish time of task 11 is given by

$$F_{11} = F_{10} + w_{11} + E_{11} = 12 + 1 + 5 = 17ms$$

Task 11 is one of the two end tasks, which are task 11 and task 12, of transaction #2. The worst-case finish time of task 11, which is 17ms, is less than the deadline of transaction #2 25ms, which is the upper bound deadline of task 11.

Task 7 has priority 2. It is an immediate successor of both task 2 and task 3. w_7 is computed as:

$$W_7 = E_6 + \left\lceil \frac{W_7}{P_{10}} \right\rceil E_{10} + \left\lceil \frac{W_7}{P_{11}} \right\rceil E_{11}$$

By using the iterative method, w_7 is computed as 10ms. According to Equation (10), the worst-case finish time of task 7 is computed as:

$$F_7 = \max((F_2 + Com_{2,7}), (F_3 + Com_{3,7})) + W_7 + E_7 = \max((16 + 0.2), (17 + 0.1)) + 10 + 1 = 28.1ms$$

The worst-case communication time between two tasks, say task i and task j , which are allocated to different processors, is computed by

$$com_{i,j} = \frac{cds_{i,j}}{DLS}$$

$com_{i,j}$ is the worst-case communication time between task i and task j . $cds_{i,j}$ is the maximum communication data size transferred between task i and task j . DLS is the communication link speed between the two processors that task i and task j are allocated

on. So $com_{2,7} = \frac{cds_{2,7}}{DLS} = \frac{2}{10} = 0.2ms$, and $com_{3,7} = \frac{cds_{3,7}}{DLS} = \frac{1}{10} = 0.1ms$

Task 7 is one of the end tasks of transaction #1. Its worst-case finish time is 28.1ms, which is less than the deadline of transaction #1, 50ms (the upper bound deadline of task 7).

Task 6 has the lowest priority, 1, on processor #2. It is an immediate successor of task 3. According to Equation (11), w_6 is computed as: (by using the iterative method)

$$W_6 = \left\lceil \frac{W_6}{P_7} \right\rceil E_7 + \left\lceil \frac{W_7}{P_{10}} \right\rceil E_{10} + \left\lceil \frac{W_7}{P_{11}} \right\rceil E_{11} = 10ms$$

The worst-case finish time of task 6 is given by

$$F_6 = (F_3 + Com_{3,6}) + W_6 + E_6 = \left(17 + \frac{1}{10}\right) + 10 + 1 = 28.1ms$$

Task 6 is one of the end tasks of transaction #1. Its worst-case finish time is 28.1ms, which is less than its upper bound deadline 50ms.

There are three tasks allocated on processor #3. Task 9 has the highest priority which is 3. It is an immediate successor of task 8. Since task 12 is a successor of task 9, it cannot contribute to the waiting time of task 9. Thus the worst-case waiting time of task 9, W_9 , is computed by

$$W_9 = E_4 = 6ms$$

The worst-case finish time of task 9 is given by

$$F_9 = (F_8 + Com_{8,9}) + W_9 + E_9 = (7 + 0) + 6 + 2 = 15ms$$

Task 12 has the second highest priority value, 2, and it is an immediate successor of task 9. W_{12} is computed by

$$W_{12} = E_4 = 6ms$$

The worst-case finish time of task 12 is given by

$$F_{12} = F_9 + W_{12} + E_{12} = 15 + 6 + 6 = 27ms$$

Task 12 is an end task of transaction #2. Its worst-case finish time is less than its upper bound deadline, 25ms. Note that for some cases, the worst-case finish time of a task may be over counted, like F_{12} , the waiting time for lower priority task 4 to finish execution is counted twice, one in F_9 , the other is in W_{12} . So for some cases, the worst-case finish time is more pessimistic than the exact value.

Task 4 has the lowest priority, 1, on processor #3. It is an immediate successor of task

2. W_4 is computed as

$$W_2 = \left\lceil \frac{W_2}{P_9} \right\rceil E_9 + \left\lceil \frac{W_2}{P_{12}} \right\rceil E_{12} = 8ms$$

The worst-case finish time of task 4 is given by

$$F_4 = (F_2 + Com_{2,4}) + W_4 + E_4 = (16 + 0.1) + 8 + 6 = 30.1ms$$

Task 4 is an end task of transaction #1. The worst-case finish time of task 4 is less than the upper bound deadline of task 4, which is the deadline of transaction #1.

So all transactions can meet their deadlines. The output of the tool is correct.

5.1.2 Test Case 2: A System with Regular and Complex Transactions

This test case is a system which contains a complex transaction which is transaction #1 and regular transactions from transaction #2 to transaction #5. It also contains a soft transaction. The communication between tasks is not considered. System has 2 processors. Preemptive scheduling is used. In this test case the allocation information is given by the user.

5.1.2.1 Test Case 2: System Description

This test case system comes from Precise Software Technologies. It was described to us through meetings, emails, and fax transmission [7]. It is an aircraft simulator system. In this system, there are 16 tasks: Control, Malfunction, Flight Parameters #1, Flight Parameters #2, Visual Display, Visual Output, VOR #1, VOR #2, ILS #1, ILS #2, Marker,

Navigation Aids #1, Navigation Aids #2, Tone Generator, Tick and User Interface. The user Interface task is treated as a soft real-time task; we just need to guarantee that it receives 5% of a CPU. So the system has 15 periodic hard real-time tasks. The deadline for each task is specified by the user. The precedence graph for this system is shown in Figure 5-2.

Note: Each task has a label $T(P, D)$: T = task execution time
 P = period of task
 D = deadline of task

Figure 5-2: Precedence Graph for Application in Test Case 2

There are three kinds of relationship among tasks:

- 1) Precedence relationship, which includes regular, OPR, EPR and TPR precedence relationship.
- 2) Data flow only relationship. Example: the relationship between Marker & Tone Generator. There is only data flow between these two tasks. But Tone Generator runs EXACTLY every 67.5ms, and Marker's period is 100ms. When there is new data from Marker, Tone Generator will buffer the data and use the new data in the next execution. So they are treated as independent tasks. This kind of relationship also exists between Flight Parameters #1 & Visual Display, Visual Display & Visual Output.

The input file of this test case is as following:

15 2 (task number, processor number)

2 2 2 5 8 8 8 2 6 6 4 8 3 0.5 (worst-case execution time of tasks)
 25 25 50 50 100 100 100 100 100 100 100 25 33.3 67.5 12.5 (period)
 25 25 50 50 100 75 100 75 100 100 75 25 33.3 67.5 12.5 (deadline)

```

0 1 4 4 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 4 3 4 3 4 4 3 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (precedence matrix)
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```

1 2 1 1 2 2 1 1 2 1 1 2 2 2 1 (allocation information)

5.1.2.2 Output Results

The result shows that this system needs at least 2 processors to meet deadlines of the 16 tasks. Single processor cannot be used because the total utilization of the processor will be 1.29.

For the two processor case, we give task execution time, period and deadline, relationship among tasks and allocation information, and the tool computes a feasible priority set for system. In the three processor case, the tool can generate a successful allocation for the system as well as a feasible priority set. In the two processor case, automatic allocation does not work so well. So a task & processor allocation was provided for the two-processor case.

Following is the output of the tool (the allocation information is given) for two processors:

*processor 1: Control, Flight Parameters #1, FLight Parameters #2, ILS #1, ILS #2,
Navigation Aids #1, Navigation Aids #2 and Tick*
*processor 2: Malfunction, VOR #1, VOR #2, Marker, Visual Display, Visual
Output and Tone Generator*

The following feasible priority set is computed by the tool:

processor 1: 7(Control), 6(FP #1), 3(FP #2), 1(ILS #1), 2(ILS #2), 4(NA #1),
5(NA #2), 8(Tick)
processor 2: 3(Malfunction), 4(VOR #1), 1(VOR #2), 5(Marker), 6(VD),
2(VO), 7(TG)

The utilization of each processor is as following:

processor 1: 0.54
processor 2: 0.70

The utilization of every processor is less than 95%, so User Interface can run on any of the two processors. Note that the priority of User Interface Task is less than that of any other task allocated on the same processor.

5.1.2.3 Validation of Tool Output

The technique for generating time constraints for preemptive scheduling is described in Section 4.4.1.1. The worst-case finish time of each task, F_i , is computed according to Equation (1). Because there are a pair of non-competing groups in the system, the ready-to-completion time of each task, w_i , which does not belong to any of these two non-competing groups, is computed by using Equation (5), (6) and (7). The ready-to-completion time of each task, w_i , which belong to one of these two non-competing groups, is computed according to Equation (8).

Now examine the system described in Figure 5-2.

There are eight tasks allocated on Processor #1: Task Control, Flight Parameters #1 (FP #1), FLight Parameters #2 (FP #1), ILS #1, ILS #2, Navigation Aids #1 (NA #1), Navigation Aids #2 (NA #2) and Tick. A feasible priority set for tasks allocated on processor #1 computed by the tool is 7(Control), 6(FP #1), 3(FP #2), 1(ILS #1), 2(ILS #2), 4(NA #1), 5(NA #2), 8(Tick).

Task Tick has the highest priority(8), and it has no predecessor. Since preemptive scheduling is used, w_{Tick} is 0 according to Equation (5). So it can execute whenever it is ready to run. According to Equation (1), the worst-case finish time of task Tick is computed as:

$$F_{Tick} = 0 + E_{Tick} = 0.5ms$$

Task Tick actually is a single-task transaction. The worst-case finish time of task Tick, 0.5ms, is much less than the deadline of task Tick, 25ms.

Task Control has the second highest priority, 7, and it has no predecessor. According to Equation (5), $W_{Control}$ is computed by

$$W_{Control} = \sum_{j \in \Phi} \left(\left\lceil \frac{W_{Control}}{P_j} \right\rceil \right) E_j + \max(EO_1, EO_2) + E_{Control} = \left\lceil \frac{W_{Control}}{P_{Tick}} \right\rceil E_{Tick} + 0 + E_{Control}$$

By using the iterative method described in test case 1, $W_{Control}$ is computed as 2.5ms. Only task Tick can preempt task Control. The worst-case finish time of task Control is computed according to Equation (1).

$$F_{control} = \max(F_j, \dots, F_k) + W_{control} = 0 + 2.5 = 2.5ms$$

Flight Parameters #1 has priority 6. It has EPR precedence relationship with Control. It does not belong to any of the non-competing groups. According to Equation (5), W_{FP1} is computed by

$$W_{FP1} = \sum_{j \in \Phi} \left(\left\lceil \frac{W_{FP1}}{P_j} \right\rceil \right) E_j + \max(EO_1, EO_2) + E_{ControlFP1} = \left\lceil \frac{W_{FP1}}{P_{Tick}} \right\rceil E_{Tick} + 0 + E_{FP1}$$

By using the iterative method, W_{FP1} is computed as 2.5ms. According to Equation (1), the worst-case finish time of task Flight Parameter #1 is computed as:

$$F_{FP1} = (F_{Control} + P_{Control}) + W_{FP1} = (2.5 + 25) + 2.5 = 30ms$$

Therefore it will take 27.5ms to wait for Control to finish the second time execution, then it is ready to run. It may need to wait for Tick (because Tick has the higher priority). So in the worst-case it will take 30ms to finish.

Navigation Aids #2 has priority 5. It also has OPR precedence relationship with Flight

Parameters #1. As well, NA #1 & ILS #1 have EPR precedence relationship with FP #1. So it belongs to the OPR non-competing group. According to Equation (8), w_{NA2} is computed as:

$$w_{NA2} = \left\lceil \frac{w_{NA2}}{P_{Tick}} \right\rceil E_{Tick} + E_{NA2}$$

By using the iterative method, w_{NA2} is computed as 6.5ms. According to Equation (1), the worst-case finish time of task Navigation Aids #2 is given by

$$F_{NA2} = (F_{FP1} + 0 \times P_{FP1}) + w_{NA2} = (30 + 0) + 6.5 = 36.5ms$$

Task Navigation Aids #2 is an end task of transaction #1. its worst-case finish time, 36.5ms, is less than its deadline 75ms.

Navigation Aids #1 has priority 4. It also has EPR precedence relationship with Flight Parameters #1. As well, NA #2 & ILS #2 have OPR precedence relationship with FP #1. So it belongs to the EPR non-competing group. According to Equation (8), w_{NA1} is computed as:

$$w_{NA1} = \left\lceil \frac{w_{NA1}}{P_{Tick}} \right\rceil E_{Tick} + E_{NA1}$$

By using the iterative method, w_{NA1} is computed as 6.5ms. According to Equation (1), the worst-case finish time of task Navigation Aids #1 is given by

$$F_{NA1} = (F_{FP1} + P_{FP1}) + w_{NA1} = (30 + 50) + 6.5 = 86.5ms$$

Task Navigation Aids #1 is an end task of transaction #1. its worst-case finish time, 86.5ms, is less than its deadline 100ms.

Flight Parameter #2 has priority 3. It has EPR precedence relationship with Control. There are two non-competing groups: NA #1 and NA #2, which have higher priority than task Flight Parameter #2. According to Equation (5), W_{FP2} is computed as:

$$W_{FP2} = \left\lceil \frac{W_{FP2}}{P_{Tick}} \right\rceil E_{Tick} + \left\lceil \frac{W_{FP2}}{P_{FP1}} \right\rceil E_{FP1} + \max(EO_1, EO_2) + E_{FP2}$$

According to Equation (6), EO_1 is computed by

$$EO_1 = \left\lceil \frac{\max(0, W_{FP2} - D)}{P_{NA2}} \right\rceil E_{NA2} + \left\lceil \frac{W_{FP2}}{P_{NA1}} \right\rceil E_{NA1}$$

D is the difference between the best start time of task Navigation Aids #1 and the deadline of task Navigation Aids #2, which is $(79 - 75) = 4ms$. According to Equation (7), EO_2 is computed by

$$EO_2 = \left\lceil \frac{\max(0, W_{FP2} - D)}{P_{NA1}} \right\rceil E_{NA1} + \left\lceil \frac{W_{FP2}}{P_{NA2}} \right\rceil E_{NA2}$$

D is the same. By using the iterative method, W_{FP2} is computed to be 14ms. According to Equation (1), the worst-case finish time of task Flight Parameter #2 is computed by

$$F_{FP2} = (F_{Control} + P_{Control}) + W_{FP2} = (2.5 + 25) + 14 = 41.5ms$$

It is an end task of transaction #1. its worst-case finish time, 41.5ms, is less than its deadline 50ms.

ILS #2 has priority 2. It has OPR precedence relationship with FP #1. And even though NA #1 has higher priority than ILS #2, they belong to different non-competing groups; NA

#1 cannot preempt ILS #2. According to Equation (8), W_{ILS2} is computed as:

$$W_{ILS2} = \left\lceil \frac{W_{ILS2}}{P_{Tick}} \right\rceil E_{Tick} + \left\lceil \frac{W_{ILS2}}{P_{NA2}} \right\rceil E_{NA2} + \left\lceil \frac{W_{ILS2}}{P_{FP2}} \right\rceil E_{FP2} + E_{ILS2}$$

By using the iterative method, W_{ILS2} is computed as 20ms. The worst-case finish time of task ILS #2 is given by

$$F_{ILS2} = (F_{FP1} + 0 \times P_{FP1}) + W_{ILS2} = (30 + 0) + 20 = 50ms$$

It is an end task of transaction #1. Its worst-case finish time, 50ms, is less than its deadline 75ms.

ILS #1 has lowest priority 1. It has EPR precedence relationship with FP #1. And even though NA #2 & ILS #2 have higher priorities than ILS #1, they belong to different non-competing groups. They cannot preempt ILS #1. According to Equation (8), W_{ILS1} is computed as:

$$W_{ILS1} = \left\lceil \frac{W_{ILS1}}{P_{Tick}} \right\rceil E_{Tick} + \left\lceil \frac{W_{ILS1}}{P_{NA1}} \right\rceil E_{NA1} + \left\lceil \frac{W_{ILS1}}{P_{FP2}} \right\rceil E_{FP2} + E_{ILS1}$$

W_{ILS1} is computed as 20ms by using the iterative method. The worst-case finish time of task ILS #1 is given by

$$F_{ILS1} = (F_{FP1} + P_{FP1}) + W_{ILS1} = (30 + 50) + 20 = 100ms$$

It is an end task of transaction #1. Its worst-case finish time, 100ms, is no greater than its deadline 100ms.

There are seven tasks allocated on processor #2. They are: task Malfunction, VOR #1, VOR #2, Marker, Visual Display (VD), Visual Output (VO) and Tone Generator (TG). A

feasible priority set for the tasks allocated on processor #2 computed by the tool is: 3(Malfunction), 4(VOR #1), 1(VOR #2), 5(Marker), 6(VD), 2(VO), 7(TG).

Task Tone Generator has the highest priority(7), and it has no predecessor. So it can execute whenever it is ready to run. According to Equation (1), the worst-case finish time of task Tone Generator is given by

$$F_{TG} = 0 + E_{TG} = 3ms$$

It is a single-task transaction. Its worst-case finish time, 3ms, is far less than its deadline, 67.5ms.

Task Visual Display has the second highest priority(6), and it has no predecessor. So it only Tone Generator can preemptive it. According to Equation (5), W_{VD} is computed by

$$W_{VD} = \left\lceil \frac{W_{VD}}{P_{TG}} \right\rceil E_{TG} + E_{VD}$$

By using the iterative method, W_{VD} is computed as 7ms. According to Equation (1), the worst-case finish time of task Visual Display is given by

$$F_{VD} = \max(F_j, \dots, F_k) + W_{VD} = 0 + 7 = 7ms$$

It is a single-task transaction. Thus Visual Display can always meets its deadline (25ms).

Task Marker has priority 5. It has EPR precedence relationship with FP #1. It belongs to EPR non-competing group. According to Equation (8), W_{Marker} is computed by

$$W_{Marker} = \left\lceil \frac{W_{Marker}}{P_{TG}} \right\rceil E_{TG} + \left\lceil \frac{W_{Marker}}{P_{VD}} \right\rceil E_{VD} + E_{Marker}$$

It is computed as 9ms by using the iterative method. According to Equation (1), the worst-case finish time of task Marker is given as

$$F_{Marker} = (F_{FP1} + P_{FP1}) + W_{Marker} = (30 + 50) + 9 = 89ms$$

89ms is less than its deadline 100ms.

VOR #1 has priority 4. It has EPR precedence relationship with FP #1. It belongs to EPR non-competing group. According to Equation (8), W_{VOR1} is computed as:

$$W_{VOR1} = \left\lceil \frac{W_{VOR1}}{P_{TG}} \right\rceil E_{TG} + \left\lceil \frac{W_{VOR1}}{P_{VD}} \right\rceil E_{VD} + \left\lceil \frac{W_{VOR1}}{P_{Marker}} \right\rceil E_{Marker} + E_{VOR1}$$

W_{VOR1} is computed as 17ms. The worst-case finish time of task VOR1 is given by

$$F_{VOR1} = (F_{FP1} + P_{FP1}) + W_{VOR1} = (30 + 50) + 17 = 97ms$$

97ms is less than its deadline 100ms.

Malfunction has priority 3. It has a regular precedence relationship with Control. According to Equation (5), W_{Malf} is computed as

$$W_{Malf} = \left\lceil \frac{W_{Malf}}{P_{TG}} \right\rceil E_{TG} + \left\lceil \frac{W_{Malf}}{P_{VD}} \right\rceil E_{VD} + \left\lceil \frac{W_{Malf}}{P_{Marker}} \right\rceil E_{Marker} + \left\lceil \frac{W_{Malf}}{P_{VOR1}} \right\rceil E_{VOR1} + E_{Malf}$$

It is computed as 19ms. The worst-case finish time of task Malfunction is then given by

$$F_{Malf} = F_{Control} + W_{Malf} = 2.5 + 19 = 21.5ms$$

It is an end task of transaction #1. Its worst-case finish time, 21.5ms, is less than its deadline 25ms.

Visual Output has priority 2. It has no predecessor. According to Equation (5), W_{VO} is

computed as:

$$W_{VO} = \left\lceil \frac{W_{VO}}{P_{TG}} \right\rceil E_{TG} + \left\lceil \frac{W_{VO}}{P_{VD}} \right\rceil E_{VD} + \left\lceil \frac{W_{VO}}{P_{Marker}} \right\rceil E_{Marker} + \left\lceil \frac{W_{VO}}{P_{VOR1}} \right\rceil E_{VOR1} + \left\lceil \frac{W_{VO}}{P_{Malf}} \right\rceil E_{Malf} + E_{VO}$$

It is computed as 31ms. So the worst-case finish time of task Visual Output is given by

$$F_{VO} = W_{VO} = 31ms$$

31ms is less than its deadline 33.3ms.

VOR #2 has the lowest priority 1. It has OPR precedence relationship with FP #1. And VOR #1 & Marker belongs to the different non-competing group. According to Equation (8), W_{VOR2} is computed as:

$$W_{VOR2} = \left\lceil \frac{W_{VOR2}}{P_{TG}} \right\rceil E_{TG} + \left\lceil \frac{W_{VOR2}}{P_{VD}} \right\rceil E_{VD} + \left\lceil \frac{W_{VOR2}}{P_{VO}} \right\rceil E_{VO} + \left\lceil \frac{W_{VOR2}}{P_{Malf}} \right\rceil E_{Malf} + E_{VOR2}$$

It is computed as 25ms. According to Equation (1), the worst-case finish time of task VOR #2 is given by

$$F_{VOR2} = (F_{FP1} + 0 \times P_{FP1}) + W_{VOR2} = (30 + 0) + 25 = 55ms$$

55ms is less than its deadline 75ms.

So all transactions can meet their deadlines. The output result of the tool for this test case is also correct.

5.1.3 Test Case 3: A System with Regular and Complex Transactions

This test case is concerned with a system which contains a complex transaction and regular transactions. It is generated from test case 2. The difference between this test case and test case 2 is that all OPR and EPR precedence relationships in test case 2 are changed

to EPR precedence relationship in this test case. The communication between tasks is not considered. The system has 3 processors. The scheduling policy is preemptive scheduling. In this test case the allocation information is not given. This is an example that utilization of each task is computed by option (2) (explained in Section 4.3.1), for allocating tasks to processors.

5.1.3.1 System Description

There are the same 16 tasks in the system as in test case 2: Control, Malfunction, Flight Parameters #1, Flight Parameters #2, Visual Display, Visual Output, VOR #1, VOR #2, ILS #1, ILS #2, Marker, Navigation Aids #1, Navigation Aids #2, Tone Generator, Tick and User Interface. We treat User Interface task as a soft real-time task; it just uses 5% of a CPU. So the system has 15 periodic hard real-time tasks. In Figure 5-3, the precedence relationship between tasks is given by the precedence graph of this system.

- - - ➔ : Data Flow Only Relationship

Note: Each task has a label T(P, D): T = task execution time

P = period of task

D = deadline of task

Figure 5-3: Precedence Graph for Test Case 3

The input file of this test case is as follows:

15 2 (task number, processor number)

2 2 2 5 8 8 8 8 2 6 6 4 8 3 0.5 (worst-case execution time of tasks)

25 25 50 50 100 100 100 100 100 100 100 25 33.3 67.5 12.5 (period)

25 25 50 50 100 100 100 100 100 100 100 25 33.3 67.5 12.5 (deadline)

```

0 1 4 4 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 4 4 4 4 4 4 4 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (precedence matrix)
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```

5.1.3.2 Output Results

We need 3 processors to meet the deadlines of the 16 tasks. Now let us examine the reason why it cannot be allocated on 1 or 2 processor(s). If all 16 tasks are allocated on one processor, the total utilization of the processor will be 1.29; which is impossible to achieve. So it is a failure case. For the two processors case, there are 7 successors of Flight Parameters #1: they are VOR #1 (execution time 8 ms), VOR #2 (8ms), ILS #1 (8ms), ILS #2 (8ms), Marker (2ms), Navigation Aids #1 (6ms) and Navigation Aids #2 (6ms). The rate of each is 100ms. The total execution time of all 7 tasks is 46ms. It is important to note when they can start execution in a 100ms period. They can be ready to execute only after

Flight Parameters #1 finishes the second time execution within this 100ms period. Flight Parameters #1 can start only after Control finishes the second time execution within a 50ms period. So in the best case, Control will not be delayed by other tasks, and Flight Parameters #1 is merely delayed by Control. So it can be shown that the finish time of the second execution of Flight Parameters #1 within a 100ms period is 79ms. So the 7 successors of Flight parameters #1 only have 21ms to complete their execution (in the best case), otherwise they will miss their deadlines. If we use two processors, there are only two 21ms slots for the 7 tasks in the best case: a total of 42ms of CPU time. This is still less than the sum of the worst-case execution times for the tasks. So some task(s) will miss its(their) deadlines. This following figure illustrates an idea of this situation.

Figure 5-4: Situation of Using Two Processors

For this test case, option (2) (explained in Section 4.3.1) is used by the allocation module. The output result is as following:

The allocation computed by the tool is as follows:

processor 1: Flight Parameters #1, ILS #1, Navigation Aids #1, Navigation Aids #2, Tick
 processor 2: Control, Flight Parameters #2, VOR #2, Marker, Visual Output
 processor 3: Malfunction, VOR #1, ILS #2, Visual Display, Tone Generator

The following feasible priority set is computed by the tool:

processor 1: 5(FP #1), 1(ILS #1), 2(NA #1), 3(NA #2), 4(Tick)

processor 2: 5(Control), 1(FP #2), 2(VOR #2), 3(Marker), 4(VO)

processor 3: 2(Mailfunction), 3(VOR #1), 4(ILS #2), 1(VD), 5(TG)

The utilization of each processor is as following:

processor 1: 0.28

processor 2: 0.52

processor 3: 0.44

As in test case 2, User Interface can run on any of the three processors.

5.1.3.3 Validation of Test Case 3

Since the description of the manual testing of test case 3 is similar to test case 2, it is not describe here. The results were verified manually and the tool output was found to be correct.

5.2 Tool Analysis

In this section a number of observations that have resulted from experiments with the tool are presented.

The priority computation problem is a NP-hard problem. In this research, a new constraint-based approach is used to solve this problem. Our constraints solver is built by using CLP-BNR. An investigation of scalability of the priority computation component was carried out. Through some test cases, we find that CPU time spent on priority computation for a system depends not only on the number of transactions or tasks in the system, but also on the complexity of precedence relationship among tasks.

The hardware environment for the testing of our tool is a SPARC ULTRA 1 Solaris workstation with 96 MB of main memory and 268 MB of virtual memory. In one

experiment with our tool, we increase the number of tasks in the system by adding a transaction with the same precedence graph. We find that when the number of tasks in the system increases, the CPU time for priority computation increases. The result is shown in Figure 5-5. The rate of increase in CPU time is found to be reasonable. Increasing the number of tasks by a factor of 8 increased the CPU time consumed by the tool by a factor of 4.4 approximately.

Figure 5-5: Changes of Tool Response Time When Task Number Increases

We also find that when a system becomes more and more complicated, the CPU time of priority computation for the system does not increase very much. Here, an increase in system complexity means that the difference between the deadline of each transaction and the worst-case finish time of the transaction decreases. The results are shown in Table 1. In

the tests, the system complexity increases by decreasing the deadline of each transaction in the system. In Table 1, the first row shows the percentage of decrease of deadline of each transaction. The second row shows the tool response time when the deadline of each transaction decreases.

Table 1: Changes in Tool Response Time When System Complexity Increases

<u>percentage of decrease in deadline of each transaction</u>	<u>2%</u>	<u>4%</u>	<u>6%</u>	<u>8%</u>	<u>10%</u>
<i>tool response time (second)</i>	<i>0.88</i>	<i>0.90</i>	<i>0.91</i>	<i>0.93</i>	<i>0.94</i>

In other tests conducted with our tool, we also find that when the transaction structure becomes more and more complicated (as we go from regular transaction module to complex transaction, for example), the CPU time needed for priority computation increases.

Chapter 6 Conclusions

In previous chapters we have presented the research that has been completed for the hard real-time system allocation and scheduling tool. In this chapter a summary of the work described in the thesis is presented first. Possible future areas of research are discussed next. Some preliminary results from this thesis are presented in [31].

6.1 Summary

We are concerned with multiprocessor systems on which periodic transactions with hard deadlines are run. An important aspect of designing such a system is the effective allocation of tasks in the transactions to processors and the computation of priorities for these tasks such that all the transactions meet their deadlines.

This research has introduced a complex transaction model that describes tasks with complex precedence relationships. Techniques and a tool for allocation and static priority driven scheduling in hard real-time systems consisting of regular and complex transactions are developed. A novel method based on Constraint Logic Programming for computing task priorities is described. This tool consists of two major parts. One is an allocation module for allocating tasks to processors in multiprocessor real-time systems. The allocation algorithm uses some ideas in the MULTIFIT.COM allocation algorithm [2]. Our allocation algorithm is designed and implemented to handle periodic communicating tasks with complex precedence relationship among them. To the best of our knowledge no other

allocation algorithm handles the complex precedence relationships present in the complex transaction model. The other part of the tool is a priority computation module which can be used to compute a feasible priority set for hard real-time system. A new approach is used for solving the priority assignment problem in hard real-time systems. Our priority computation module has no explicit priority assignment algorithms to assign priority to tasks. From the allocation information and a description of the system parameters provided by the user, the tool generates a set of constraints on the completion times of the tasks in the system. A constraint solver, which is built by using CLP-BNR, is used to solve these constraints, in order to obtain a feasible priority set for the tasks in the system. The regular transaction model as well as the complex transaction model can be handled by the tool. The performance of our tool meets our expectations.

6.2 Future Research

In this thesis, an allocation and scheduling tool for hard real-time systems is developed. One of the directions for future works is the incorporation of the ability to handle TPR relationships in the tool. Currently user specifications are described in an input file. Another direction for future work is to develop a friendly and efficient graphic user interface (GUI) for the tool.

There are three kinds of real-time transactions: hard real-time transactions, firm real-time transactions, and soft real-time transactions. In this research we have focused on hard real-time transactions. Another direction to be explored is to extend this tool to handle all of these three kinds of transactions. This will make the tool more powerful. A simple

extension will be to handle transactions with the deadlines specified as ranges. The current tool implementation can handle this with the deadline of a soft transaction being specified as the upper end point of the range. However, firm real-time transactions which need deadlines to be met for a specified fraction of the time, for example, are worthy of research.

In this research only periodic transactions are considered. In the real world, there are real-time applications which contain sporadic transactions. To extend this research to handle periodic and sporadic transactions at the same time is one direction for future research.

The contention for communication links was assumed to be negligible. Research for introducing the ability to handle contention for communication links is warranted.

References

- [1] A. Burchard, J. Liebeherr, Y. F. Oh, and S. H. Son, "Assigning Real-Time Tasks to Homogeneous Multiprocessor Systems", *IEEE Transactions on Computers*, 44(12), December 1995, pp. 1429-1442.
- [2] C. M. Woodside and G. G. Monforton, "Fast Allocation of Processes in Distributed and Parallel Systems", *IEEE Trans. on Parallel and Distributed Systems*, 4(2), February 1993 pp. 164-174.
- [3] C. W. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment", *Journal of the ACM*, 20(1), January, 1973, pp. 46-61.
- [4] E. G. Coffman, *Computer and Job Shop Scheduling Theory*, John Wiley and Sons Inc., New York, 1976.
- [5] G. J. Nutt, *Operating Systems: a Modern Perspective*, Addison-Wesley, Mass., 1997.
- [6] J. Garcia and M. Harbour, "Optimized Priority Assignment for Tasks and Messages in Distributed Hard Real-Time Systems", in *Proceedings of the 3rd Workshop on Parallel and Distributed Real-Time Systems*, Sofia (Bulgaria), April 1995, pp. 124-132.
- [7] J. James, C. Honegger, *Personal Communications*, Precise Software Technologies, Ottawa, 1997.

-
- [8] J. Lehoczky, "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines", in *Proceedings of 11th IEEE Real-Time Systems Symposium*, Lake Buena Vista (Florida), December 1990, pp. 201-209.
 - [9] J. Lehoczky, L. Sha, and Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior", in *Proceedings of IEEE Real-Time Systems Symposium*, Santa Monica (California), December 1989, pp. 166-171.
 - [10] J. Leung and J. Whitehead, "On the Complexity of Fixed-priority Scheduling of Periodic, Real-Time Tasks", *Performance Evaluation*, 2(4), December 1982, pp. 237-250.
 - [11] J. Sun, R. Bettati, and J. W. S. Liu, "An End-to-End Approach to Schedule Tasks with Shared Resources in Multiprocessor Systems", in *Proceedings of the 11th IEEE Workshop on Real-Time Operating Systems and Software*, 1994, pp. 18-22.
 - [12] K. Jeffay, D. F. Stanat, and C. U. Martel, "On Non-Preemptive Scheduling of Periodic and Sporadic Tasks", in *Proceedings of the 20th IEEE Real-Time Systems Symposium*, San Antonio (Texas), December 1991, pp. 129-139.
 - [13] K. Tindell, A. Burns, and A. Wellings, "Allocating Real-Time Tasks: An NP-Hard Problem Made Easy", *Real-Time System Journal*, 4(2), May 1992, pp. 145-165.
 - [14] L. Sha, J. P. Lehoczky, and R. Rajkumar, "Solutions for Some Practical Problems in Prioritized Preemptive Scheduling", in *Proceedings of IEEE Real-Time Systems Symposium*, New Orleans (Louisiana), December 1986, pp. 181-191.
 - [15] M. F. Storch and J. W. S. Liu, "Heuristic Algorithms for Periodic Job Assignment", in *Proceedings of the Workshop on Parallel and Distributed Real-Time Systems*, Newport

- Beach (California) April 1993, pp. 245-251.
- [16] M. H. Klein, T. Ralya, B. Pollak, R. Obenza, and M. G. Harbour, *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1993.
- [17] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.
- [18] N. Audsley, A. Burns, K. Tindell, M. Richardson, and A. Wellings, "Applying New Scheduling Theory to Static Priority Preemptive Scheduling", *Software Engineering Journal*, 8(5), May 1993, pp. 284-292.
- [19] P. A. Laplante, *Real-Time Systems Design and Analysis: An Engineer's Handbook*, IEEE Computer Society Press, 1992.
- [20] R. Bettai, "End-to-End Scheduling to Meet Deadlines in Distributed Systems", PhD thesis, University of Illinois at Urbana-Champaign, 1994. <http://pertsserver.cs.uiuc.edu/papers/>.
- [21] R. Etemadi, "End-to-End Scheduling in Hard Real-Time Multiprocessor Systems", PhD thesis, Department of Systems and Computer Engineering, Carleton University, August, 1996.
- [22] R. Etemadi, G. K. Karam, and S. Majumdar, "Heuristic Algorithms for Priority Assignment in Flowshops", in *Proceedings of IEEE Performance Computer and Communications (IPCCC 98)*, Tempe (Arizona), February 1998, pp. 15-22.
- [23] S. Chatterjee and J. Strosnider, "Distributed Pipeline Scheduling: End-to-End Anal-

- ysis of Heterogeneous, Multi-resource Real-Time Systems”, in *Proceedings of the 15th International Conference on Distributed Computing Systems*, Vancouver (British Columbia), May 1995, pp. 204-211.
- [24] S. -T. Levi and A. K. Agrawala, *Real Time System Design*, McGraw-Hill Publishing Company, New York, 1990.
- [25] Tri-Pacific Software, “PERTS: A Prototyping Environment for Real-Time Systems” (a commercial software package), <http://www.tripac.com/perts/index.html>.
- [26] T. -S. Tia and J. W. S. Liu, “Assigning Real-Time Tasks and Resources to Distributed Systems”, in *Proceedings of the 2nd Workshop on Parallel and Distributed Real-Time Systems*, Cancun (Mexico) April 1994, pp. 157-167.
- [27] W. J. Older, “Application of Relational Interval Arithmetic to Ordinary Differential Equations”, BNR Computing Research Laboratory, Ottawa, March 1995.
- [28] W. J. Older, “CLP(BNR) Algorithms for Traveling Salesman”, BNR Computing Research Laboratory, Ottawa, August 1993.
- [29] W. J. Older, “Using Interval Arithmetic for Non-Linear Constrained Optimization”, BNR Computing Research Laboratory, Ottawa, March 1993.
- [30] W. K. Shih, J. W. S. Liu, and C. L. Liu, “Modified Rate-Monotonic Algorithm for Scheduling Periodic Jobs with Deferred Deadlines”, *IEEE Transactions on Software Engineering*, 19(12), December 1993, pp. 1171-1179.
- [31] X. Zhang, “Allocation and Scheduling Tool for Hard Real-Time Systems”, Poster paper, *TRIO Final Research Results Conference*, Ottawa (Ontario), April 1998, 12

pages.

- [32] Y. F. Oh, "Scheduling Periodic Tasks in a Hard Real-Time Environment", Department of Computer Science, University of Virginia. <http://www.cs.virginia.edu/~yo5u/publication.html>

IMAGE EVALUATION TEST TARGET (QA-3)

APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989

© 1993, Applied Image, Inc.. All Rights Reserved