Optimization of Distributed Real-Time Systems with Scenario Deadlines

by

TAO ZHENG

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Engineering

Ottawa-Carleton Institute of Electrical Engineering
Faculty of Engineering
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, Canada, K1S 5B6

August 20, 2002

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

Optimization of Distributed Real-Time Systems with Scenario Deadlines

Submitted by Tao Zheng, B.Sc. (Electronics)

in partial fulfillment of the requirements for the degree of Master of Engineering

_____

Chair, Department of Systems and Computer Engineering

_____

Thesis Supervisor

Carleton University

August 20, 2002

ABSTRACT

This thesis studies optimization of a class of distributed real-time systems which include concurrent scenarios, operations with deterministic or stochastic execution demands, arbitrary precedence between operations and soft or hard deadline requirements. This thesis adopts an optimization approach to configure that class of distributed real-time systems to meet their deadline requirements. The optimization approach uses Layered Queueing Network (LQN) model to describe the systems. According to the LQN simulation results, priority assignment and task reallocation are used to optimize the system to produce a feasible solution. A pre-existing optimization strategy was improved in various ways, re-implemented in a more useful and maintainable form, and evaluated. It was applied to soft real-time systems with stochastic execution demands, which had not been attempted previously. A number of experiments and comparisons indicate the effectiveness and improvement of the new optimization approach.

As SPE (Software Performance Engineering) is used more and more in designing software systems, the optimization could become part of the SPE to evaluate the performance after the performance model is generated.

ACKNOWLEDGMENTS

I would like to express my great thankfulness to my supervisor, Dr. C. M. Woodside for the guidance, advices and encouragement which he has given me. Without his support, I wouldn't have completed this research successfully.

I would also like to thank Hesham El-Sayed and Dorin Petriu for their helps. The models they provided save me a lot of research time.

Financial assistance provided by Ontario Graduate Scholarship in Science and Technology, and by Rational Canada was greatly appreciated.

Finally, special appreciation goes to my parents, to whom this thesis is dedicated.

TABLE OF CONTENTS

LIST OF FIGURES

GLOSSARY

a: An activity in a task or a phase of an entry

a_s*:* The start activity in an entry

activities(e): The set of activities (or phases) in entry *e*

CommOV(a): The total communication overhead incurred by *a*

CommOV(e, T): The communication overhead between entry *e* and task T_2

CommOV(T_1, T_2)*:* The communication overhead between task T_1 and task T_2

CommOV(m): Communication overhead when message *m* is transmitted between different processors

CommOV(m, role): The communication overhead of physical message *m* either on the client side (role = client) or on the server side(role =server)

CommOVGainMetric (T, C): The metric for reallocation of task *T* to processor *C* due to communication overhead

$CommOV_{reply,client}(a)$*:* The client side communication overhead incurred by *a* from the reply messages

$CommOV_{reply,server}(a)$*:* The server side communication overhead incurred by *a* from the reply messages

$CommOV_{req,client}(a)$*:* The client side communication overhead incurred by *a* from the request messages

$CommOV_{req,server}(a)$*:* The server side communication overhead incurred by *a* from the request messages

CPUMetric(T, C): The metric for reallocation of task *T* to processor *C*

$Criticality_S$*:* The criticality metric for scenario *S*

criticalInvokeRate(a): The normalized critical invocation rate of *a* (per second)

criticalInvokeRate(e, S): The normalized invocation rate for entry *e* in scenario *S,* (per second). The asynchronous call is not counted.

criticalInvokeRate(m): The normalized critical invocation rate of sending physical message *m* (per second)

e: An entry in a task

entries(T): The set of entries in task *T*

executionDemand(e): The execution demand of entry *e*

*forCall(e, *):* The set of forwarding calls sent from entry *e*

invokeRate(a): The normalized invocation rate of *a*

invokeRate(m): The normalized sending rate of physical message *m*

m: A physical message which is identified from LQN call

messLen(m): The length of the messages in a physical message *m* (in suitable units)

messNum(m): The number of times the physical message *m* is sent due to one invocation of a particular activity or phase of an entry.

messUnitOV: A constant overhead transmitting a message with unit message length (sec per unit)

mc: The mean request frequency for a synchronous call or an asynchronous call

predecessor(a_s): The set of senders that makes calls to the entry that begins with start activity a_s

prob: The forwarding probability of a forwarding call

ProportionalDeadline(T): The proportional deadline of task *T,* in seconds

repeats(a_s, a): The mean repetitions of activity *a* related to its start activity a_s

replyCall(M): The set of all the reply messages in a model

*replyCall(a, *):* The set of reply messages sent by *a*

replyCall(, a):* The set of reply messages sent to *a*

reqCall(M): The set of all the request messages in a model

*reqCall(a, *):* The set of request messages sent by *a*

reqCall(, a):* The set of request messages sent to *a*

S: A scenario

synCall(a, e): A synchronous call from *a* to *e*

synCall(, e):* The set of synchronous calls sent to entry *e*

T: A task

$TaskMetric_T$: The metric, which is used for priority adjustment, for task *T*

TaskSize(T): The task size of task *T*

$TaskSize|_T^S$: The task size of task *T* related to scenario *S*

$TaskWaitMetric_T|_T^S$: The metric that measures waiting time of task *T* in scenario *S*

UtilGainMetric(T, C): The metric estimating the processor utilization decrease if task *T* is reallocated to processor *C*

$Util_C$: The utilization of a target processor *C*

U_T: The utilization of task *T*

$visitRate|_T^S$: The visit rate of task *T* related to scenario *S* (visits per scenario execution)

CHAPTER 1: INTRODUCTION

A system is a real-time system when it can support the execution of applications with time constraints on that execution. Traditionally real-time systems are classified as hard real-time systems and soft real-time systems. In hard real-time systems, the time constraints must be met under all circumstances. The failure to meet the deadlines may cause catastrophe. Aircraft control systems and nuclear reactor control systems are typical hard real-time systems. In soft real-time systems, the results of missing deadlines are not catastrophic. It is acceptable to miss a small percentage of deadlines in soft real-time systems. Typical examples are e-commerce and telecommunication systems. Generally, real-time systems are deployed on more than one processor to make full use of the resources on different sites that are connected by a network. The real-time systems that are deployed on multiple processors are called distributed real-time systems in this thesis.

1.1 Motivation

In order to meet deadlines using allowable resources, the real-time systems must be configured properly. A proper configuration will make full use of the available resources so as to reduce the cost of the systems. Finding a feasible solution under the resource requirements is the objective of the system design.

Much research has been done on hard real-time systems. Priority assignment and task allocation were found as two important issues to ensure that deadline requirements are met in hard real-time systems. Unfortunately, it has been shown that both the problem of assigning priority to end-to-end tasks and the problem of allocating tasks to processors are NP-hard problems [3][23].

Up till now, only a little research is reported on soft real-time systems with stochastic execution demands. Soft deadline and stochastic execution demands are two important characteristics in many real-life applications. For example, in widely used e-commerce systems, the time to search records from a database and the network latency may be

stochastic to some degree. The deadlines are usually soft deadlines because the results of late responses are not catastrophic and it is not necessary to meet the deadlines 100%. Applying hard real-time system algorithms to those systems may cause low efficiency. The optimization approach proposed in this thesis provides a heuristic solution for a general class of real-time systems including soft real-time systems with stochastic execution demands mentioned above.

This thesis considers a class of systems that will be called “distributed real-time systems” with the following characteristics:

- The system has one or more scenarios which run concurrently with specific arrival rates
- The scenarios consist of actions which are distributed on a number of processors connected by a network
- Each scenario has a hard or soft end-to-end deadline requirement
- The execution time of each action is deterministic or stochastic to some degree

To the best of my knowledge, little of the existing work has combined all these aspects together.

Based on heuristic algorithms and simulation results, this thesis provides an optimization approach to configure general distributed real-time systems to meet their deadline requirements. It is especially helpful for the soft real-time systems with stochastic execution demands because the guarantee to meet all the deadlines in such systems is not proven yet.

Recently, software designers get benefits from applying the Software Performance Engineering (SPE) [30] technique which implies predicting the performance at early stages of software lifecycle. The optimization could become part of SPE to provide a feasible solution after the performance model is generated.

1.2 A General Optimization Approach

This thesis provides a general approach to optimizing the distributed real-time systems with the help of LQN simulation. The optimization approach is suitable for both hard real-time systems and soft real-time systems. A hard real-time system is considered as a

special case of a general real-time system where 100% of the requests must meet their deadlines instead of a specific percentage of successful requests (e.g. 95%), and the execution demands are deterministic instead of stochastic. The percentages of successful requests are collected from the simulation results, and the stochastic execution demands could be simulated by specific distribution generators.

Two factors are considered in this optimization approach:

Priority assignment: which is concerned with how to assign priorities to the tasks (also called processes in many papers). In real-time systems, the tasks must be scheduled properly to meet their deadlines. Task priorities could be fixed or dynamic. The fixed priority approach is simple to implement and more predictable under overload conditions. Due to these reasons, this thesis focuses on the fixed priority approach. It means that when a priority is assigned to a task, the priority will always be the same, and each task has a different priority level. In this thesis, a larger number means a higher priority, e.g. a task with priority 2 has higher priority than a task with priority 1. This priority notation is different from many other papers.

Task allocation: which is concerned with how to assign tasks to processors. The task allocation could be static or dynamic. Because most real-time systems built today are static to be simpler and more predictable, we focus on the static task allocation in this thesis.

Layered Queueing Network (LQN) model is used as the performance model in this thesis. Most of the system behaviors in distributed systems (e.g. hardware devices, software tasks, nested services, precedence constraints and multithreads) could be easily expressed in the LQN model. Deadline requirements are also included in the LQN model. A distributed real-time system can be easily described by a proper LQN model.

An LQN simulation tool called “parasrvn” is used to analyze the performance of the LQN model. The performance results of the simulation provide the fundamentals for system optimization.

The optimization approach provided in this thesis was first proposed by H.M. El-Sayed, D. Cameron, and C.M. Woodside in [10][11]. The optimization strategies are improved and modified based on experiment results, heuristic reasons and evaluation for soft real-time systems with stochastic execution demands.

Figure 1.1 is a high level flow chart of the new optimization approach proposed in this thesis. The blocks with bold lines are the new steps introduced in this thesis. The tasks are first allocated to processors by a heuristic allocation algorithm, and the priorities are assigned to tasks according to a heuristic priority assignment algorithm as initialization. The communication cost will be added to model before model evaluation if applicable. If the solution is not feasible, then the priorities are adjusted according to metrics based on the simulation results. If priority adjustment is not able to meet the deadline requirements, one task will be reallocated to a different processor. The communication cost will be removed/added before/after the task reallocation if applicable. The priority adjustment and task reallocation will be repeated until a feasible solution is found or the termination conditions are met. The final optimized result will be produced after removing the communication cost if applicable.

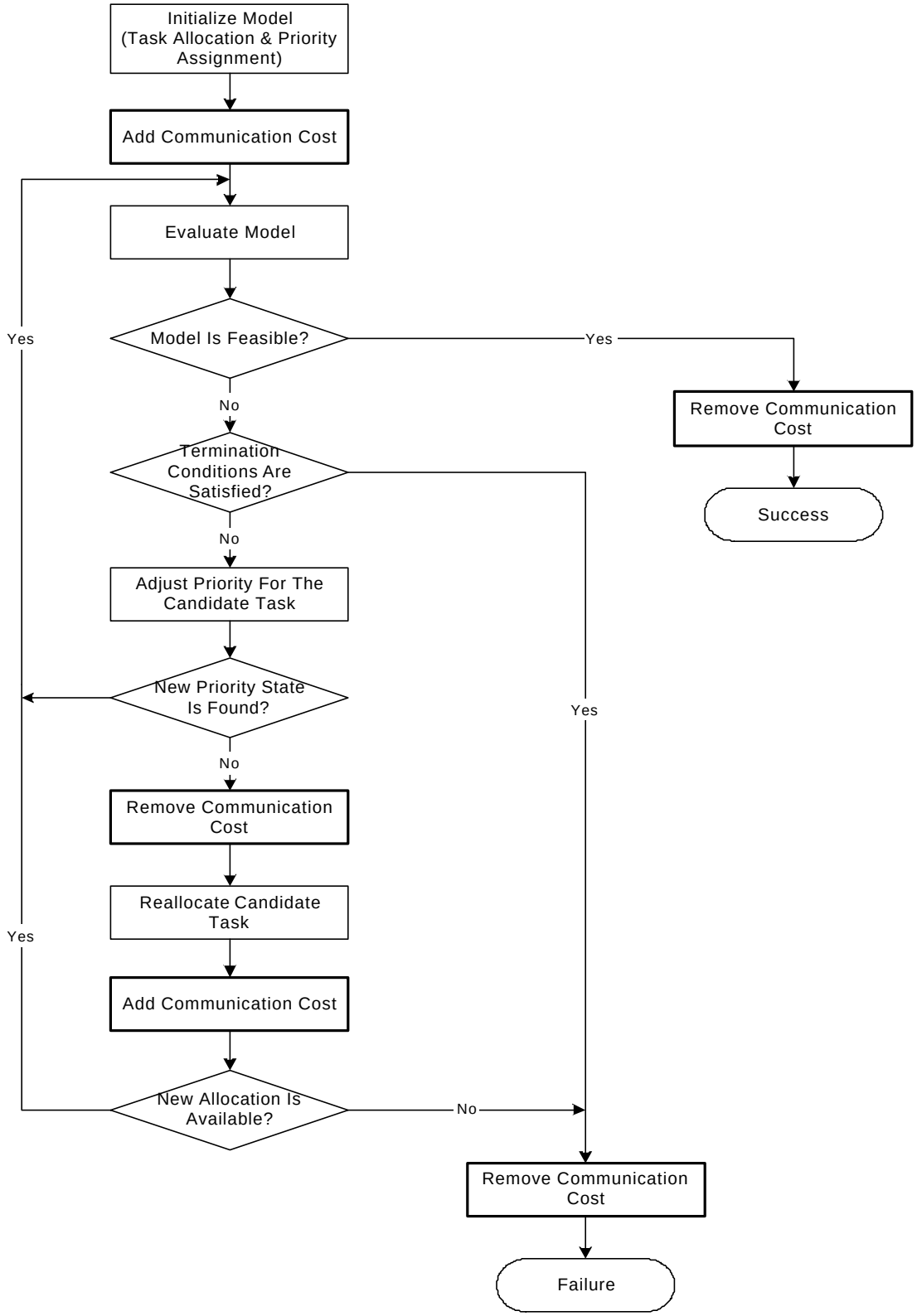


Figure 1.1 High Level Flow Chart of Optimization Approach

1.3 Research Contributions

The following research contributions are made in this thesis:

1. Evaluate the optimization approach on distributed soft real-time systems with stochastic execution demands, which were not done in the previous work

   Based on the optimization approach proposed by H. El-Sayed et al., this thesis extends the optimization approach to soft real-time systems with stochastic execution demands. Algorithms are provided to handle communication cost (communication overhead and network delay) in the distributed systems. To the best of author's knowledge, it is the first time to use optimization approach for distributed soft real-time systems with stochastic execution demands and scenario deadlines. It shows that the optimization approach really improves performance on soft real-time systems as well as on hard real-time systems.

2. Improve and modify the optimization strategies in several aspects

   The "fixed" priority approach, which is defined to assign different priority levels for different tasks in this thesis, is used instead of the "static" priority approach, which is defined to allow equal priority levels for different tasks in this thesis. H. El-Sayed's original priority adjustment algorithm uses a "static" priority approach that allows different tasks to have the same priority level. However, hard real-time system theories require a "fixed" priority approach that forces different priority levels for different tasks. The static priority approach allowing equal priority level with First In First Out (FIFO) scheduling discipline is considered as mixed priority approach because FIFO is classified as dynamic priority approach in [25].

   The priority adjustment algorithm is improved in this thesis. The new priority adjustment algorithm indicates a higher success ratio according to the experimental results.

   New metrics for optimization are proposed by heuristic reasons after examining and discussing the original metrics.

3. Implement and provide the optimization algorithms that begin from LQN models

H. El-Sayed's original optimization approach started with scenarios that could be used to generate an LQN model. It required a specific tool to make the optimizer work. The optimization approach in this thesis starts with ordinary LQN models with scenario deadlines. It is more applicable than the previous one. The original and new algorithms for optimization are implemented using Java language. All the optimization algorithms are provided in this thesis.

1.4 Thesis Organization

This thesis is organized as follows. Chapter 2 gives the background on task allocation algorithms, priority assignment algorithms and issues related to LQN models. Chapter 3 proposes algorithms to handle communication overhead and network delay in the general LQN models. Chapter 4 describes and examines the optimization part in [11] in detail. The improvements and modifications are also provided in this chapter. Chapter 5 evaluates the new optimization approach. Experiment results achieved from the new optimization approach are compared with those from the original one on some hard real-time examples. The new optimization approach is extended to soft real-time systems with stochastic execution demands. Experimental results for an application are provided in chapter 5. A case study on an e-commerce prototype is provided in chapter 6. It shows the effectiveness of the optimization approach on practical systems. Chapter 7 gives the conclusions finally.

CHAPTER 2: BACKGROUND

Real-time system with deadlines has been studied using the following concepts and notations. An independent task T_i may be characterized by its release time instant r_i, execution time e_i and deadline D_i. The time between its release time instant r_i, and time instant completing the amount of e_i execution time is the response time R_i of that task. If R_i D_i, the task T_i meets its deadline requirement. Otherwise, it misses its deadline requirement. If 100% of the response times in the real-time system must meet the deadline requirements, the system is considered as a "hard" real-time system. If some deadlines can be missed without causing system failure, the system is called a "soft" real-time system. Sometimes the task T_i may consist of a chain of dependent subtasks. The response time R_i of T_i is then calculated from the release of the first subtask to the completion of the last subtask in the chain. The deadline D_i for T_i is called the end-to-end deadline respectively. As mentioned in chapter 1, real-time systems that are deployed on multiple processors are called distributed real-time systems in this thesis.

Priority assignment and task allocation are the two most important issues configuring distributed real-time systems. The related works on these two issues and the introduction of the Layered Queueing Network (LQN) are presented in this chapter.

2.1 Priority Assignment Algorithms

J. W. S. Liu overviewed the real-time systems and classified three commonly used approaches to scheduling real-time systems in [25]: clock driven, weighted round robin and priority-driven approach. The priority-driven approach is the one that is used the most. The priority-driven algorithms are classified into two types for scheduling periodic tasks: fixed-priority algorithms and dynamic-priority algorithms. Due to the poor performance of dynamic-priority scheduling under overload conditions and the complexity of its implementation, this thesis focuses on the fixed-priority algorithms.

2.1.1 Uniprocessor Scheduling Algorithms

C.L. Liu and J.W. Layland proposed a uniprocessor real-time scheduling algorithm which is well known as rate monotonic algorithm (RMA)[24]. According to the rate monotonic algorithm, “tasks with higher request rate will have higher priorities”. The tasks with the same request rate will break the tie arbitrary. They proved that the rate monotonic priority assignment is optimal for the preemptive, independent and periodic tasks with fixed priorities if the deadlines of the tasks are the same as their periods. It is assumed that the tasks can’t be blocked and self suspended. The term “optimal” means that the rate monotonic algorithm always produces a feasible scheduling if a feasible scheduling exists. Liu and Layland provided a sufficient condition to guarantee all periodic tasks to meet their deadlines if the total utilization of all the tasks is no more than the utilization bound 0.693. The exact characterization for rate monotonic algorithm to meet the deadlines is given by J. Lehoczky; L. Sha and Y. Ding in [22]. For randomly generated and uniformly distributed tasks, 0.88 is suggested as reasonable utilization bound to meet all the deadlines.

There are many constraints in the rate monotonic algorithm. These constraints have been weakened by later works.

The constraint that the tasks can’t be blocked and self suspended is weakened by L. Sha, R. Rajkumar and J. Lehoczky. They pointed out that the priority inversion problem might occur if synchronization is permitted for the tasks [29]. An algorithm called priority ceiling protocol is provided to deal with task synchronization. When a task enters a critical section, its priority will be raised to the ceiling priority of the critical section, and restores its original priority when it leaves the critical section. The priority ceiling protocol can avoid mutual deadlock and bound the block time of a higher priority task at most once by lower priority tasks. The utilization bound is adjusted respectively.

The deadline constraint is weakened by J.Y.T. Leung and J. Whitehead. They proposed deadline monotonic algorithm for tasks with deadlines no more than their periods [23]. The higher priorities are assigned to tasks with shorter deadlines instead of shorter request rates. They proved that deadline monotonic algorithm is optimal for the fixed-priority algorithms. I.e. if any other fixed-priority algorithm is feasible for scheduling a set of tasks, the deadline monotonic algorithm is also feasible for those tasks. If the deadline of every task is proportional to its period, deadline monotonic

algorithm is equivalent to rate monotonic algorithm. The deadline monotonic algorithm generalizes the rate monotonic algorithm and weakens the deadline constraint. J. P. Lehoczky weakened the deadline constraint further [21]. By introducing the "level *i* busy period", the maximum time when only tasks with priorities greater than and equal to *i* are executed, he gave a formula to calculate the worst-case response time of task with priority *i*. Whether a task is schedulable could be determined by comparing the deadline and the worst-case response time. He also pointed out that the rate monotonic algorithm and deadline monotonic algorithm are no longer optimal if deadlines of the tasks are greater than their periods.

The proofs of the algorithms mentioned above assume a worst-case scenario in which all tasks are released at the same time. If the tasks are permitted to have arbitrary start times, the common release time among all tasks may not occur. N. C. Audsley pointed out that both rate-monotonic and deadline-monotonic priority assignments cease to be optimal if the tasks have arbitrary release times, and provided the optimal priority ordering algorithm to find a feasible priority assignment [2]. The optimal priority ordering algorithm divides all the tasks into a sorted lower priority tasks part and an unsorted higher priority tasks part. It chooses an arbitrary task from the unsorted higher priority tasks part and assigns the lowest priority level among these unsorted tasks. If the chosen task is schedulable, the task is put into the sorted part and its priority will be kept. Otherwise it will be put back to the unsorted part and another unsorted task will be chosen. This continues until unsorted tasks part is empty (a feasible solution is found) or all the unsorted tasks have been checked and found unschedulable (the system is not feasible). By extending J. P. Lehoczky's formula [21] and N. C. Audsley's algorithm [2], K. W. Tindell provided a more general solution which could cope with release jitter and sporadic periodic tasks [35].

2.1.2 Multiprocessor Scheduling Algorithms

When considering multiprocessors and "end-to-end" tasks, which have a chain of subtasks in distributed system, the priority assignment becomes much more complicated. R. Bettati proved that the problem of assigning priority to end-to-end tasks (transactions)

is NP-hard [3]. Efficient optimal solutions are not likely available, and heuristic algorithms must be created to find out feasible solutions.

B. Kao and H. Garcia-Molina proposed several heuristic algorithms to assign relative deadlines to subtasks over sequential [19] and parallel [20] transactions for soft real-time systems using Earliest Deadline First (EDF) scheduling. J. Sun adopted these algorithms and proposed two new algorithms called Proportional-Deadline-Monotonic (PDM) assignment and Normalized- Proportional-Deadline-Monotonic (NPDM) assignment [33]. In PDM, The relative deadline of a task is obtained by dividing the end-to-end deadline among the tasks proportionally according to their execution times. In NPDM, which is similar to PDM, takes into account the processor utilization when determining the relative deadlines of the tasks. These two new algorithms are reported to have better performance than those of Kao's [33].

There are more complicated algorithms using different kinds of optimization methods which take more time but usually produce feasible schedules in more difficult ones. K.W. Tindel, A. Burns, and A.J. Wellings used a technique called simulated annealing to find a solution for priority assignment and task allocation at the same time [36]. It is suitable for general hard real-time problems.

J.J.G. Garcia and M. Gonzalez Harbour proposed the Heuristic Optimized Priority Assignment (HOPA) algorithm which schedules hard real-time transactions consisting of a chain of actions (subtasks) in a distributed system to meet end-to-end deadlines [16]. The HOPA is based on the distribution of the end-to-end deadlines among the actions. Once each action is assigned a local deadline, deadline monotonic algorithm is used to assign a priority to each action. By calculating the worst-case response time using formula provided in [35], the new intermediate local deadlines will be reached by taking into account the "excess" of each action which measures the distance that separates each action from schedulability, and the priority of each action is reassigned respectively. This continues until a feasible solution is found or some stopping conditions are satisfied.

H.M. El-Sayed et al. proposed an optimization approach for priority assignment and task allocation for distributed hard real-time systems in [10][11]. The tasks are assigned priorities according to certain algorithm. Based on the simulations result of the LQN model, a metric which measures the contributions to deadline miss rate for every task is

calculated. The priority of the task with the worst metric will be raised to a higher level. The priorities are adjusted until a feasible solution is found or the termination conditions are satisfied. Its procedure of priority adjustment is similar to the HOPA to some degree. This optimization approach is adopted in this thesis, and more details of this approach are provided in chapter 4.

2.1.3 Soft Real-Time Approaches

Up till now, only a little research on soft real-time scheduling is reported.

T.-S. Tia, T. S., Z. Deng, M. Shankar, M. Storch, J. Sun, L. C. Wu, and J. W. S. Liu described the Probabilistic Time Demand Analysis (PTDA) to provide probabilistic schedulability guarantees to semi-periodic tasks which are released periodically and have widely varied computation times [34]. The algorithm extends the Time Demand Analysis (TDA) methods to semi-periodic tasks. It assumes that the deadlines of all the tasks are no more than the periods.

M. K. Gardner used Stochastic Time Demand Analysis (STDA) to calculate the lower bound frequencies of missed deadlines for the uniprocessor systems with independent tasks and shared resources. The STDA extends the Time Demand Analysis (TDA) in another way. Along with the Release Guard Protocol (RGP), the STDA is applied to distributed systems of independent tasks with end-to-end deadlines [17].

Alia K. Atlas and Azer Bestavros presented Stochastic Rate Monotonic Scheduling (SRMS) to deal with periodic tasks with highly variable execution times and statistical QoS requirements [1]. A job admission controller, which is responsible for maintaining the QoS requirements of the tasks, guarantees the admitted job to meet their deadlines, and discards the jobs that are not admitted or let them run in lower priority than the admitted jobs. The SRMS is restricted to rate monotonic scheduling policy. And discarding a job may not be a good choice in real life applications. E.g. in an e-commerce system, a late response should be better than discarding a request.

All these approaches extend hard real-time analysis to soft real-time systems to some degree. However they are not suitable for general soft real-time systems due to their constraints.

N. J. Dingle, P. G. Harrison and W. J. Knottenbelt presented a technique for the numerical determination of response time densities in Generalized Stochastic Petri Nets (GSPNs) models which are popularly used for investigating complex concurrent systems [9]. GSPN is able to model general systems with deterministic or exponential transitions. Its numerical technique should be able to deal with general real-time systems including the soft real-time systems. However the state explosion of the Petri Net may limit its use on large systems and certain tool must be implemented to describe the systems by GSPN efficiently.

2.2 Allocation Algorithms

It is understandable that the task allocation affects the performance of distributed systems. An improper task allocation may cause the systems to miss deadlines where it is possible to meet them in another allocation scheme.

Task allocation with minimum processors is an NP-hard problem even without precedence constraints [23]. Heuristic algorithms are usually used for allocating task in multiprocessor systems. Variants of bin packing algorithms [6] are used for task allocation where the processors are considered as bins and tasks are considered as items with different size. S.K. Dhall and C.L. Liu proposed two heuristic algorithms, Rate-Monotonic First-Fit (RMFF) and Rate-Monotonic Next-Fit (RMNF) algorithms, to allocate independent periodic tasks to a minimal number of processors [8]. The tasks are sorted in non-increasing order of their periods, and then assigned to processors using bin packing algorithms without violating rate monotonic schedulability conditions. The worst-case performance was improved by another algorithm called First-Fit Decreasing-Utilization Factor (FFDUF) algorithm where the tasks are sorted by the non-increasing order of the utilization factor [7]. A. Buchard, J. Liebeherr, Y. Oh, and S. H. Son proposed heuristic algorithms based on the tighter schedulability conditions [5]. The performance improvement is significant for tasks with small load factors. M. F. Storch and J.W.S. Liu proposed heuristic algorithms which cluster tasks to minimize the total communication cost when the communication among the tasks is considered [32]. The

tasks are assigned to a processor whose total utilization factor is less than the maximum utilization factor.

All the algorithms mentioned above assume independent periodic tasks with hard deadlines. These algorithms are too pessimistic for soft real-time systems, and the assumption is too restrictive for general systems.

There are many other allocation algorithms concentrating other issues instead of schedulability. H. S. Stone suggested an optimal algorithm for allocating tasks to two processors to minimize the total communication cost by transforming the allocation problem into a solvable network flow problem [31]. Optimal solutions with certain constraints are extended to three or more processors where the general N-processor problem is NP-complete [4].

A number of researchers use branch-and-bound (B & B) technique to allocate tasks. D.T. Peng and K.G. Shin et al proposed two branch-and-bound algorithms to allocate periodic tasks with precedence constraints to minimize the maximum response time [26]. C.J. Hou and K.G. Shin proposed an algorithm to find an assignment which maximizes the probability of meeting deadlines by making use of a branch-and-bound technique called Module Allocation Algorithm (MAA) [18]. Usually the branch-and-bound technique also schedules the tasks while allocating the task. It takes more time than general allocation algorithms.

C.M. Woodside and G.M. Monforton proposed Multifit-Com algorithm which aims at the maximum throughput considering the communication cost [38]. The Multifit-Com algorithm generalizes the Multifit algorithm on a bus-based multiprocessor system taking account of the interprocessor communication requirements. The Multifit-Com algorithm develops four heuristic policy choices to decide an allocation and each policy choice has several options (Figure 2.1). In total, 36 combinations with different options in each policy are studied on 680 small randomly generated examples, and a reduced set of 4 polices with less than 2% degradation is recommended. The Multifit-Com is chosen for initial task allocation in this thesis. A more detailed description is provided in chapter 4.

Policies	Options
Initial Task Sizing (ITS): which estimates the task size	1 Upper Bound (S_u) execution time plus all potential communications 2 Lower Bound (S_l) execution time only 3 Communication Based (S_c) only the potential interprocessor communication costs.
Task Ordering Technique (TOT): which sorts the tasks in decreasing ordering	1 Absolute (A) decreasing order according to task size 2 Communication-Directed (C) special heuristic algorithm in [38]
Intermediate Bin Weights (IBW): which represents the workload allocated to the bin	1 Upper Bound (W_u) special heuristic algorithm in [38] 2 Lower Bound (W_l) special heuristic algorithm in [38] 3 Processing Based (W_p) special heuristic algorithm in [38]
Placement Criteria (PC): which determines the bin to which the tasks are placed	1 Greedy (G), or Largest Fit calculate the intermediate bin weights for all possible placements of the task in question. Choose the placement which, to date, satisfies the goal of minimizing the bottleneck workload. 2 First Fit (F) starting at the first bin, calculate the intermediate bin weights. If they are all below the capacity, choose this processor (bin); otherwise move on to the next processor and repeat.

Figure 2.1 Policies and Options of Multifit-Com in [38]

2.3 Layered Queueing Network (LQN) Model

The layered queueing network (LQN) model, presented by C. M. Woodside et al, is a performance model for systems with distributed software servers [13][14][28][37][39]. It extends the queueing network model and can be used to model distributed systems with a variety of different system behavior (e.g. hardware devices, software processes, nested

services, precedence constraints and multithreads). It represents the software resources in a natural way, and the LQN model is easy to understand and create.

In LQN model, a task (or process) represents a software or hardware object which may execute concurrently. The tasks in real-time system could be easily described by the tasks in LQN model. In the later part of this thesis, the word “task” represents the task in the LQN model.

In LQN model, there are three types of task: pure client task, pure server task and active server task.

- Pure client tasks: only send requests. Typical examples: actual users and input drivers. In Figure 2.2, *Env1* and *Env2* are pure client tasks.
- Pure server tasks: only receive requests. Typical examples: the final stage servers or hardware devices. *TaskE* is a pure server task in Figure 2.2.
- Active server tasks: can both send and receive requests. Typical examples: the middle stage servers, such as *TaskD* in Figure 2.2.

Three types of communications are supported in LQN model: synchronous call, asynchronous call, and forwarding call:

- Synchronous call: sender blocks when waiting for a reply from receiver. This is the pattern of standard Remote Procedure Call (RPC). There are two kinds of messages in synchronous call: request messages and reply messages. A synchronous call is indicated by a solid line with a filled arrowhead, such as the call from *a2* to *en_e1* in Figure 2.2.
- Asynchronous call: sender doesn’t block after sending a request. No replies are required. Only request messages exist in asynchronous call. An asynchronous call is indicated by a solid line with an open arrowhead, such as the call from *a1* to *en_b1* in Figure 2.2.
- Forwarding call: The receiver might forward a synchronous call to the third task instead of replying it directly. The third task might reply it or forward it further. The receiver doesn’t block after forwarding. The final receiver will send a reply to the blocked sender directly. Forwarding call contains request messages. Whether it contains reply messages depends on the details of the receiver and the third task. A

forwarding call is indicated by a dash line with a filled arrowhead, such as the call from *d2* to *en_e2* in Figure 2.2

A task may have some “entries” which provide different services. An entry consists of some “activities” or “phases” which are the smallest execution units. The activities may have complex precedence relationships (e.g. AND fork, join, OR fork, join) each other [15]. Most of the precedence constraints can be expressed by using activities. Figure 2.2 is an example of the LQN model.

Both analytic and simulation tools are supported for LQN models. The input format of LQN model can be found in [15]. The important performance results such as response times, throughputs, waiting times (the time when task is ready but not able to run) and missed probabilities are listed in the output report.

In this thesis, all the results are produced by an LQN simulation tool called “parasrvn”. The “max confidence interval” of every model is no more than 10%, meaning that all the results should be accurate within ±10% with 95% confidence.

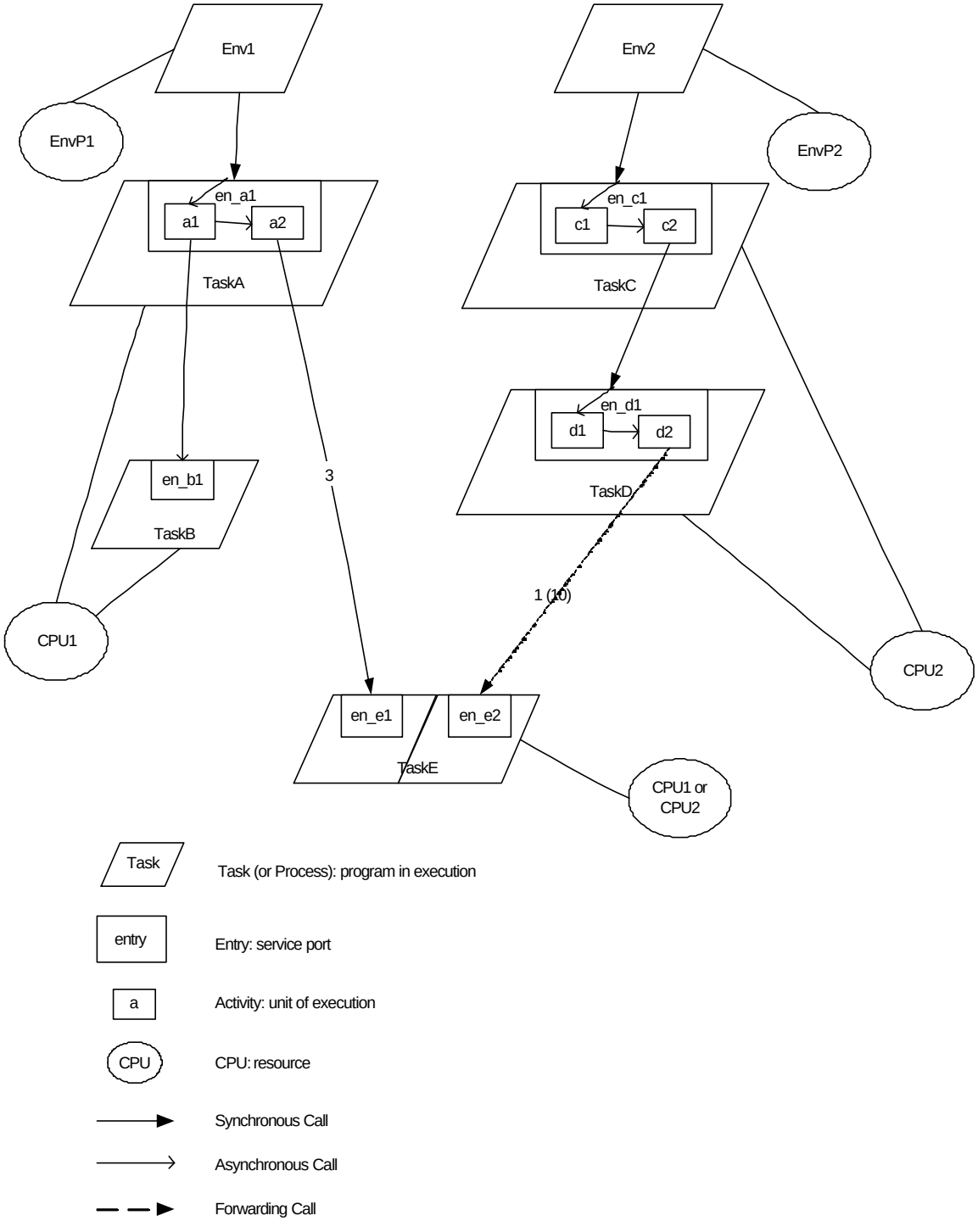


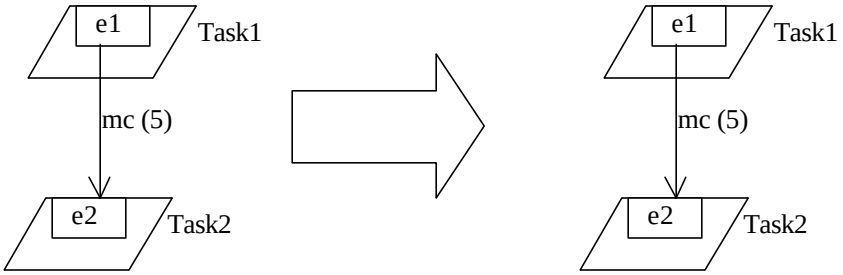
Figure 2.2 An Example of LQN Model

CHAPTER 3: A COMMUNICATION COST APPROACH IN LQN MODEL

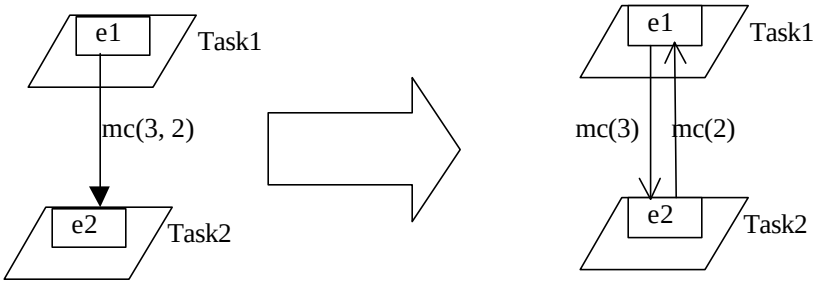
In order to make full use of the available resources, real-time systems are probably deployed on multiprocessors. The tasks on different processors are synchronized by the messages between them. The communication cost should be considered to describe the systems more precisely. This chapter provides an approach to handle the communication cost in LQN model. Two types of the communication cost are considered in this chapter: the communication overhead sending and receiving messages, and the network delay.

3.1 Identify Physical Messages and Message Counts in LQN Model

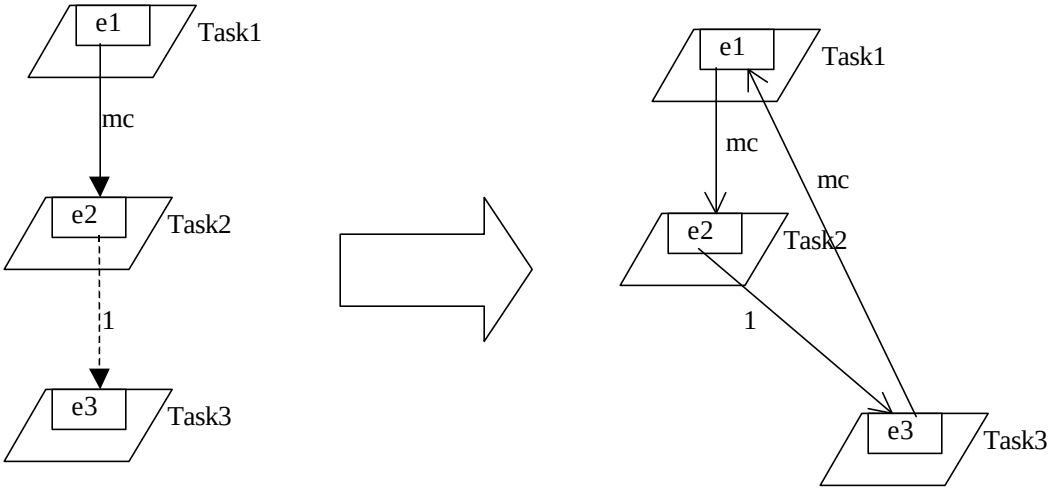
The LQN model supports three types of communication calls: synchronous call, asynchronous call, and forwarding call. It is not straightforward to calculate the communication cost from those types of calls directly. According to the properties of those calls, the physical messages and message counts must be identified before calculating the communication cost. An extended LQN notation and physical message notation is used in Figure 3.1. On the left-hand side in Figure 3.1, the extended LQN notation adds the information of the message length of the LQN call. A number in parentheses beside the request frequency is the message length of the request in some suitable units (such as bytes), e.g. the "(5)" in Figure 3.1 (a). If there are two numbers in parentheses beside the request frequency, the first number indicates the message length of the request and the second number indicates the message length of the reply, such as the "(3, 2)" in Figure 3.1 (b). If there is no number in parentheses beside the request frequency, the default message length of the request and reply is 1. On the right-hand side in Figure 3.1, the LQN calls are interpreted as one-way request messages and reply messages. In Figure 3.1, the request messages go down from entry *e1* or *e2* (in Figure 3.1 (c)), the reply messages go up to entry *e1*. The right-hand side of Figure 3.1 uses a physical message notation for a single complete interaction, which is used for calculation of communication overhead. In this notation, messages are indicated by an open headed



(a) Asynchronous Call



(b) Pure Synchronous Call



(c) Combined Synchronous and Forwarding Call

Note:

1. Left-hand side: Extended LQN Notation. Right-hand side: Physical Message Notation
2. The request frequency in the LQN notation (left-hand side) is the number of messages sent when the **sender** is invoked once. The message number of the physical message notation (right-hand side) is the number of messages sent when the **client** or **forwarder** is invoked once.

Figure 3.1: Basic Rules Identifying Physical Messages and Message Counts with Request and Reply Messages

arrow.The number beside the arrow is the mean number of messages *messNum(m)* for a physical message *m* which is transmitted between the sender and receiver when the client (*e1* in Figure 3.1) or forwarder (*e2* in Figure 3.1 (c)) in the interaction is invoked once. For the request message, the client or forwarder sends the message. For the reply message, the client receives the message. This notation is different from the notation of LQN asynchronous call whose frequency value is always related to the sender. The reason to use this notation is that the notation of asynchronous call doesn't carry enough information which is necessary to calculate the communication cost. This notation is only used for calculating communication cost in this thesis. Another number in parentheses beside the arc (e.g. "(5)" in Figure 3.1 (a) and "(3)", "(2)" in Figure 3.1 (b)) indicates the length of the messages *messLen(m)* in physical message *m*. The default length of a message is assumed 1 if it is not specified. In this thesis it is also assumed that the overhead of transmitting one message with unit message length is a known constant: *messUnitOV* when the sender and receiver are on different processors. Thus the total overhead for sending and receiving a physical message *m* is *CommOV(m)* defined by:

$$\text{CommOV}(m)=\begin{cases} 0 & \text{if sender and receiver are on the same processor} \\ \text{messNum}(m)\times\text{messLen}(m)\times\text{messUnitOV}, & \text{if on different processors} \end{cases} \quad (3.1)$$

The message length *messLen(m)* is not defined in the current version of LQN modeling language. In this research, it is described in an additional file along with the other information such as the overhead of transmitting a unit length message *messUnitOV*. The additional file is explained in section 4.7 and examples of additional file could be found in Appendix A.2 and Appendix B.2.

It is straightforward to identify the request messages and message counts. For each call in the LQN model, a respective request message is produced. The sender and receiver of the request message are the same as those of the LQN call. For each synchronous call or asynchronous call, the message number of the respective request message is the request frequency of that LQN call. For each forwarding call, the message number of the respective request message is the forwarding probability of that forwarding call. The

message length of the request message is defined in an additional file. The set of produced request messages is identified as *reqCall(M)* in this thesis.

However, identifying the reply message and message counts from synchronous calls and forwarding calls is not so simple. Different types of calls should be treated differently:

1. It is straightforward to identify the reply message for a pure synchronous call. For each pure synchronous call (e.g. the call in Figure 3.1 (b)), the message number of the respective reply message is the request frequency of that pure synchronous call while the sender and receiver of the reply message are opposite to those of the pure synchronous call. The number of reply messages is the number when the client (e.g. *e1* in Figure 3.1 (b)) is visited once, instead of the server (e.g. *e2* in Figure 3.1(b)).
2. When forwarding calls are combined with synchronous calls (e.g. in Figure 3.1), the identification is more complicated. A set of synchronous calls that send messages from the client to the final servers directly will be constructed according to the forwarding calls and relative synchronous calls. After transformation, these constructed synchronous calls could be considered as pure synchronous calls, and the reply messages can be identified using the method mentioned in 1.

In order to identify the reply messages and their message counts, an algorithm is provided to construct a series of transformed messages, ending up with reply messages. The algorithm is provided as following:

1. Clone an LQN model and remove all the asynchronous calls in the cloned model.
2. If there is no forwarding calls in the model, then go to step 5. Otherwise, locate an entry *e* which is a secondary server forwarding synchronous calls to later stage servers.
3. (a) Consider all forwarding calls from entry *e,* naming them as *forCall(e, *)* and synchronous calls sent to *e,* naming them as *synCall(*, e).* Note that:
   - M is the number of synchronous calls in *synCall(*, e)*
   - mc_i (i = 1 to M) is the request frequency of the i^{th} synchronous call in *synCall(*,e)*
   - a_i (i = 1 to M) is the sender of the i^{th} synchronous call in *synCall(*, e)*
   - N is the number of forwarding calls in *forCall(e, *)*

- $prob_j$ (j = 1 to N) is the forwarding probabilities of the j^{th} forwarding call in *forCall(e, *)*
- e_j (j = 1 to N) is the receiver of the j^{th} forwarding call in *forCall(e, *)*

(b) For each synchronous call from a_i in *synCall(*, e)*

For each forwarding call to e_j in *forCall(e, *)*

Delete forwarding call to e_j in *forCall(e, *)*

Add a synchronous call from a_i to e_j with request frequency $mc_{i*} prob_j$

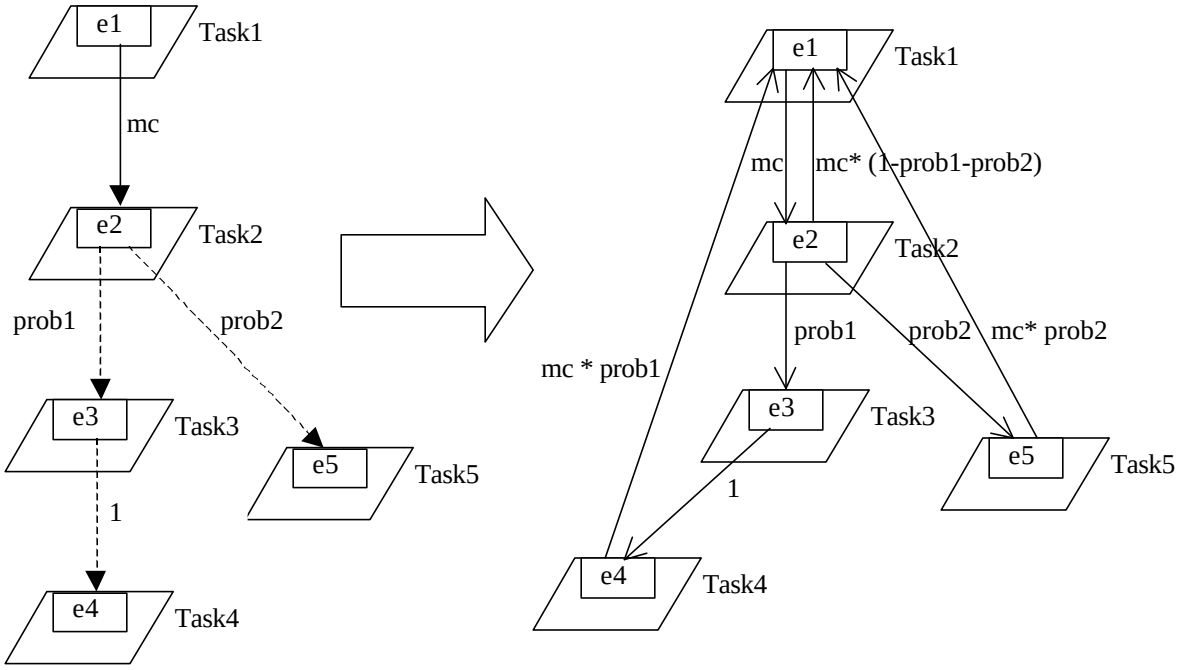
Next

Change the request frequency of synchronous call from a_i in *synCall(*,e)* to $mc_{i*}(1 - prob_j)$. If the result is 0, delete that synchronous call

Next

4. If the set of forwarding calls is not empty, go to step 3.
5. For each remaining synchronous call, a reply message is produced. The set of the produced reply messages is identified as *replyCall(M)* in this thesis. The number of messages of the respective reply message is the request frequency of that transformed synchronous call while the sender and receiver of the reply message are opposite to those of the synchronous call. The message numbers of the reply message are the number when the client (receiver of the reply message) is visited once, instead of the server (the sender of the reply message).

The request messages and reply messages can be used to calculate communication overhead and network delay. Figure 3.2 gives an example showing the identified physical messages and message counts from an LQN model. Figure 3.3 shows how the request messages and message counts are identified for the example model in detail, and Figure 3.4 shows how the reply messages and message counts are identified for the example model in detail.

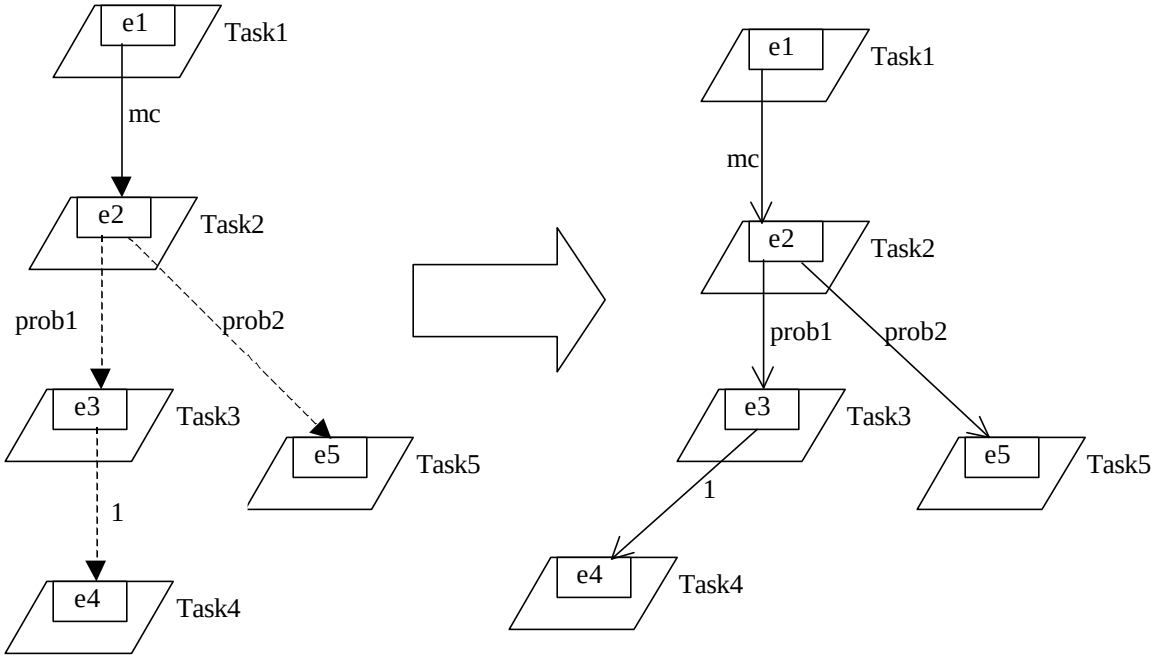


Note:

1. Left-hand side: Extended LQN Notation. Right-hand side: Physical Message Notation
2. The request frequency in the LQN notation (left-hand side) is the number of messages sent when the **sender** is invoked once. The message number of the physical message notation (right-hand side) is the number of messages sent when the **client** or **forwarder** is invoked once.
3. prob1, prob2: forwarding probabilities (prob1 + prob2 < 1)

.

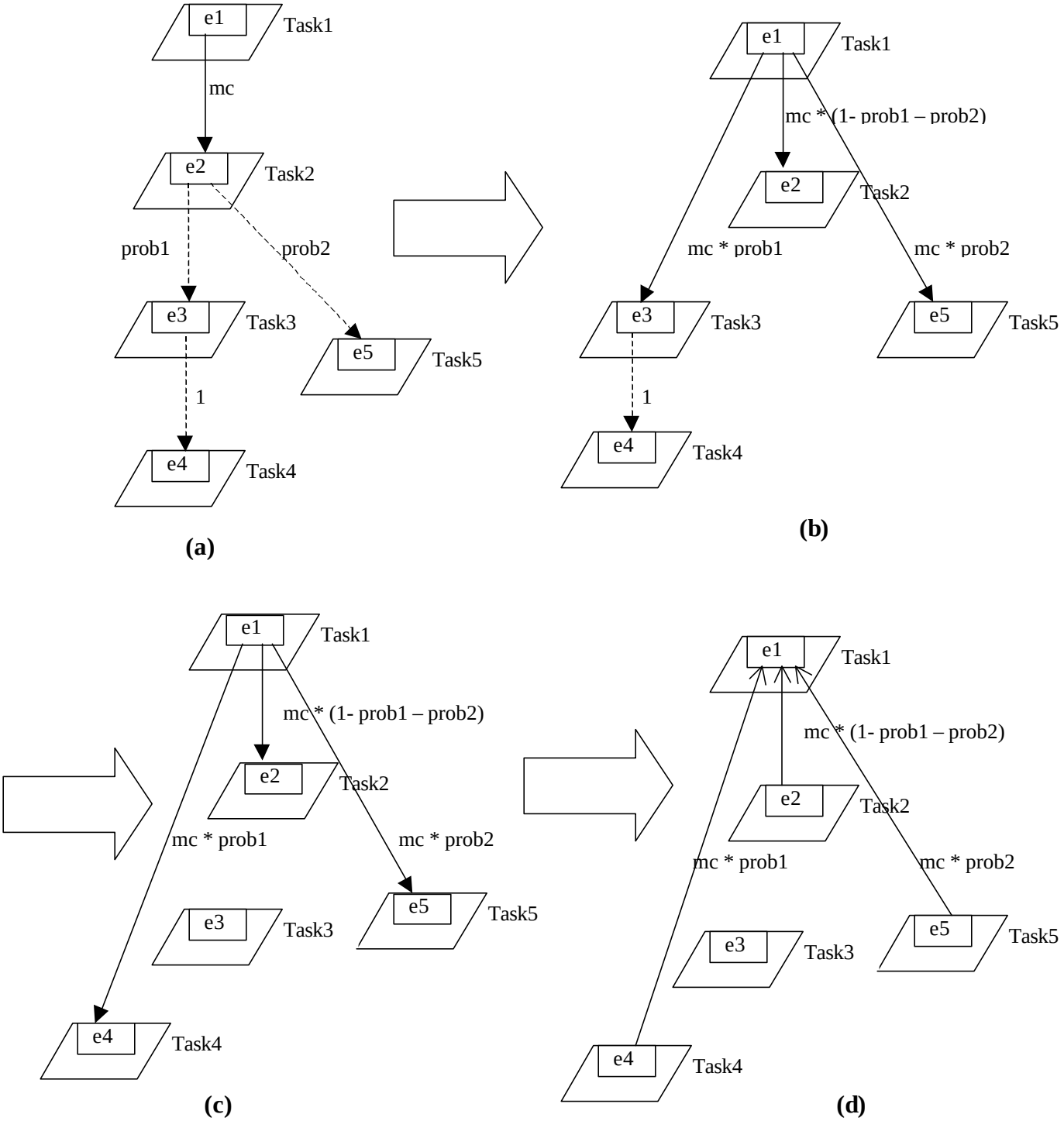
Figure 3.2 Identification of Physical Messages and Message Counts (An Example)



Note:

1. Left-hand side: Extended LQN Notation. Right-hand side: Physical Message Notation
2. prob1, prob2: forwarding probabilities (prob1 + prob2 < 1)

Figure 3.3 Identification of Request Messages and Message Counts (An Example)



Note:

1. Left-hand side: Extended LQN Notation. Right-hand side: Physical Message Notation
2. prob1, prob2: forwarding probabilities (prob1 + prob2 < 1)

Figure 3.4 Identification of Reply Messages and Message Counts (An Example)

3.2 Calculate Communication Overhead in LQN Model

In this thesis, the communication overhead is the overhead when client sends request messages to server and server returns reply messages back to client, if applicable. The client is defined as the one that makes requests and receives replies. The server is the one that receives requests and sends replies, if applicable. If a client makes a request to the 1st server (forwarder) and the 1st server forwards the request to the 2nd server, then for the forwarding calls, the 1st server is considered as a client of the 2nd server.

The calculation of communication overhead is under the following assumptions:

1. The communication overhead occurs when client and server are allocated on different processors. If they are allocated on the same processor, the communication overhead is assumed to be zero.
2. For each physical message *m*, if there is a communication overhead, it is divided evenly to both the client side and the server side. I.e. the communication overhead on the client side *CommOV(m, client)* and the communication overhead on the server side *CommOV(m, server)* are:

   CommOV (m, client) = CommOV(m, server) = CommOV (m) / 2 (3.2)

3. As mentioned in equation (3.1), the communication overhead is assumed not only proportional to the number of messages, but also proportional to the message length and a constant cost for a unit length message.

In the "High Level Flow Chart of Optimization Approach" (Figure 1.1), the communication cost is modified when allocations are changed during optimization. The communication overhead transmitting the messages must be added or removed respectively. The following section discusses how to add or remove the communication overhead in LQN model.

In LQN model, the sender of an LQN call might be an entry or an activity, while the receiver of an LQN call is always an entry. However this is not always true for the physical messages identified in the previous sections. For a request message, the sender could be an entry or activity, and the receiver must be an entry. For a reply message, the

sender must be an entry and the receiver could be an entry or activity. There are some rules on where to add (remove) the communication overhead in LQN model:

1. For each request message identified from a synchronous call or asynchronous call, the communication overhead on the client side (sender) is added (or removed) to an activity or the phase of an entry that makes the request. The communication overhead on the server side (receiver) is added (or removed) to the first phase of an entry that receives the call.
2. For each request message identified from a forwarding call, the communication overhead on the client side (sender) is added (or removed) to the first phase of an entry that forwards the request or the last activity in an entry if the entry consists of a chain of activities. The communication overhead on the server side (receiver) is added (or removed) to the first phase of an entry that receives the call.
3. For each identified reply message, the communication overhead on the client side (receiver) is added (or removed) to the phase of an entry that causes the request or the reply activity in an entry if the entry consists of a chain of activities. The communication overhead on the server side (sender) is added (or removed) to the first phase of an entry that replies the call.

For every entry or activity in LQN model that related to the LQN calls, it might be on the client side or on the server side. The identified physical messages could be request messages or reply messages. Thus the communication overhead incurred by the phase of an entry or an activity *a* consist of four parts:

$$\begin{aligned} \text{CommOV}(a) = {} & \text{CommOV}_{\text{req,client}}(a) + \text{CommOV}_{\text{req,server}}(a) + \\ & \text{CommOV}_{\text{reply,client}}(a) + \text{CommOV}_{\text{reply,server}}(a) \end{aligned} \qquad (3.3)$$

Where

CommOV(a) is the total communication overhead incurred by *a*

$CommOV_{req,client}(a)$ is the communication overhead incurred by *a* from the request messages on the client side

$CommOV_{req,server}(a)$ is the communication overhead incurred by *a* from the request messages on the server side

$CommOV_{reply,client}(a)$ is the communication overhead incurred by *a* from the reply messages on the client side
$CommOV_{reply,server}(a)$ is the communication overhead incurred by *a* from the reply messages on the server side

Those four parts of communication overhead can be calculated from the following equation (3.4-3.7) weighted by the invocation rate of the physical messages and the invocation rate of *a*:

$$CommOV_{req,client}(a) = \frac{\sum_{m \in reqCall(a,*)} (invokeRate(m) * CommOV(m, client))}{invokeRate(a)} \quad (3.4)$$

$$CommOV_{req,server}(a) = \frac{\sum_{m \in reqCall(*,a)} (invokeRate(m) * CommOV(m, server))}{invokeRate(a)} \quad (3.5)$$

$$CommOV_{reply,client}(a) = \frac{\sum_{m \in replyCall(a,*)} (criticalInvokeRate(m) * CommOV(m, client))}{criticalInvokeRate(a)} \quad (3.6)$$

$$CommOV_{reply,server}(a) = \frac{\sum_{m \in replyCall(*,a)} (criticalInvokeRate(m) * CommOV(m, server))}{criticalInvokeRate(a)} \quad (3.7)$$

where *reqCall(a, *)* is the set of request messages sent by *a*
reqCall(, a)* is the set of request messages sent to *a*
*replyCall(a, *)* is the set of reply messages sent by *a*
replyCall(, a)* is the set of reply messages sent to *a*
invokeRate(m) is the normalized sending rate of physical message *m*

invokeRate(a) is the normalized invocation rate of *a*

criticalInvokeRate(m) is the normalized critical invocation rate of physical message *m*

criticalInvokeRate(a) is the normalized critical invocation rate of *a*

CommOV(m, server) and *CommOV(m, client)* are provided in equation (3.2)

$$\text{invokeRate}(m) = \text{invokeRate}(a') \quad \text{if } a' \text{ sends } m \qquad (3.8)$$

$$\text{criticalInvokeRate}(m) = \text{criticalInvokeRate}(a') \quad \text{if } a' \text{ sends } m \qquad (3.9)$$

i.e.

$\text{invokeRate}(m) = \text{invokeRate}(a)$ in equation (3.4)

& $\text{criticalInvokeRate}(m) = \text{criticalInvokeRate}(a)$ in equation (3.6)

The equation (3.4) and (3.6) could then be simplified as (3.4.1) and (3.6.1)

$$\text{CommOV}_{req,client}(a) = \sum_{m \in \text{reqCall}(a, *)} \text{CommOV}(m, client) \qquad (3.4.1)$$

$$\text{CommOV}_{reply,client}(a) = \sum_{m \in \text{replyCall}(a, *)} \text{CommOV}(m, client) \qquad (3.6.1)$$

invokeRate(a) is the total normalized invocation rate (per second) among all scenarios.

$$\text{invokeRate}(a) = \sum_{\forall S \in \text{Scenarios}} \text{invokeRate}(a, S) \qquad (3.10)$$

invokeRate(a, S) could be calculated recursively:

$$\text{invokeRate}(a,S) = \begin{cases} \text{arrivalRate}(S) & \text{if } S \text{ has an open arrival} \\ 1 / \text{period}(S) & \text{if } S \text{ is periodic} \\ \sum_{\forall a_0 \in \text{predecessor}(a_s)} \text{messNum}(a_0, a_s) \times \text{repeats}(a_s, a) \times \text{invokeRate}(a_0, S) & \end{cases} \qquad (3.11)$$

where

a_s is the start activity of the entry where a stays if a is an activity.

repeats(a_s, a): The mean repetitions of activity a related to its start activity a_s

If a is the phase of an entry, a_s and a are considered the same and the number *repeats(a_s, a)* is always 1

messNum(a_0, a_s) is the number of messages sent from a predecessor a_0,

a_0 might be an activity or a phase of an entry

predecessor(a_s) is the set of senders that makes calls to the entry begin with start activity a_s

The similar equations could be reached for the criticalInvokeRate(a):

$$\text{criticalInvokeRate(a)} = \sum_{\forall S \in \text{Scenarios}} \text{criticalInvokeRate(a, S)} \qquad (3.12)$$

$$\text{criticalInvokeRate(a, S)} = \begin{cases} \text{arrivalRate(S)} & \text{if S has an open arrival} \\ 1 \,/\, \text{period(S)} & \text{if S is periodic} \\ \sum \text{messNum}(a_0, a_s) \times \text{repeats}(a_s, a) \times \text{criticalInvokeRate}(a_0, S) & \end{cases} \qquad (3.13)$$

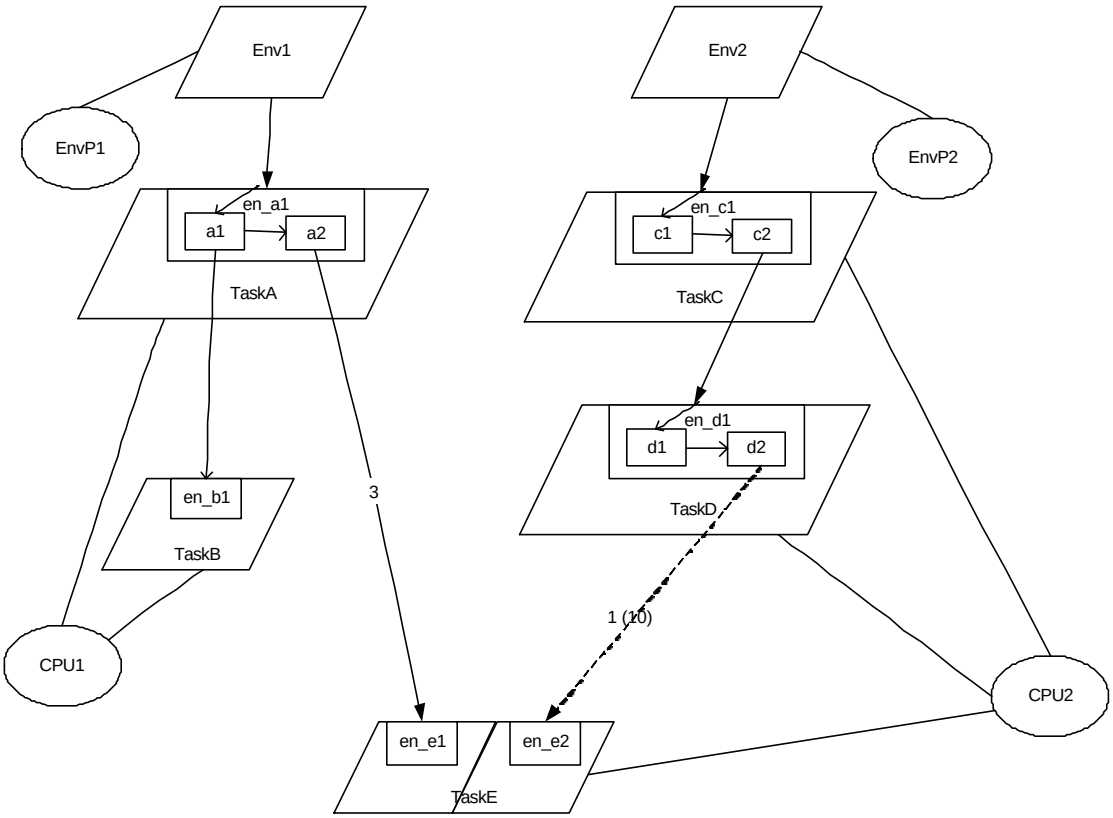
$\forall\, a_0 \in \text{predecessor}(a_s)$

The only difference between the *invokeRate(a)* and *criticalInvokeRate(a)* is that the calculation of *criticalInvokeRate(a)* comes from the identified reply messages where asynchronous calls are not counted.

Figure 3.5 indicates how to calculate the communication overhead in an example.

The final communication overhead incurred by the activities or the phase of entries is:

Activity *a2*:	2
Entry *en_e1* (phase 1):	2
Others:	0



Communication Overhead For One Message With Message Length Unit: 1

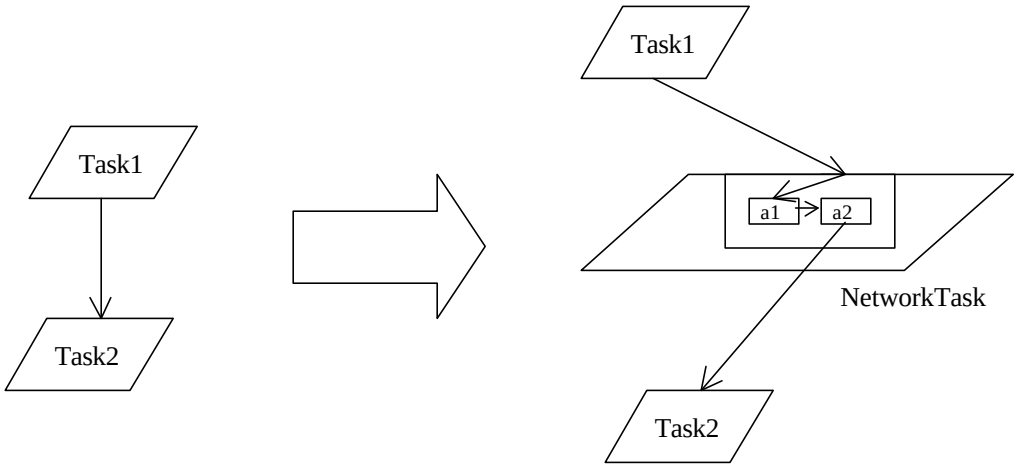
Network Delay: 0

The Call in LQN Model			Identified Request message		Identified Reply message	
Type	Client	Server	Client Side	Server Side	Client Side	Server Side
Asynchronous	a1	en_b1	0	0	NA	NA
Synchronous	a2	en_e1	3*1*1/2 = 1.5	3*1*1/2/3=0.5	3*1*1/2 = 1.5	3*1*1/2/3=0.5
Synchronous	c2	en_d1	0	0	0	NA
Forwarding	en_d1	en_e2	0	0	NA	0

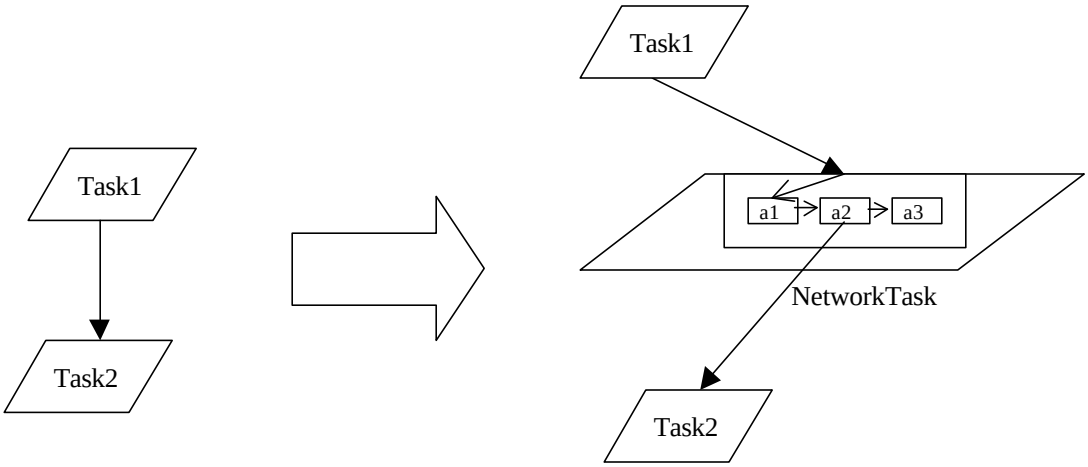
Figure 3.5 Communication Overhead Calculation (Example)

3.3 Insert Network Delay in LQN Model

When messages are transmitted between different processors connected by certain network, a delay may occur during the transmission in network. The network delay must be considered to get a more precise description. In this thesis, a pseudo NetworkTask with several activities in each entry is inserted for each request or reply message between different processors. Figure 3.6, Figure 3.7 indicates the insertion of NetworkTask in LQN model. In Figure 3.6, the execution demand of activity *a2* is 0, and the execution demand of activity *a1* and *a3* are the network delay. In Figure 3.7, the execution demands of activity *a2* and *a5* are always 0. The execution demand of *a4* is the network delay. There are some discussions to decide the execution demand of activity *a1* and *a3*. Task2 and Task3 are assumed on different processors. If Task1, Task2 and Task3 are on different processors, then the execution demands of *a1* and *a3* are the network delay. If Task1 and Task2 are on the same processor, then the execution demand of *a1* is 0, and the execution demand of *a3* is the network delay. If Task1 and Task3 are on the same processor, then the execution demand of *a1* is the network delay, and the execution demand of *a3* is 0.

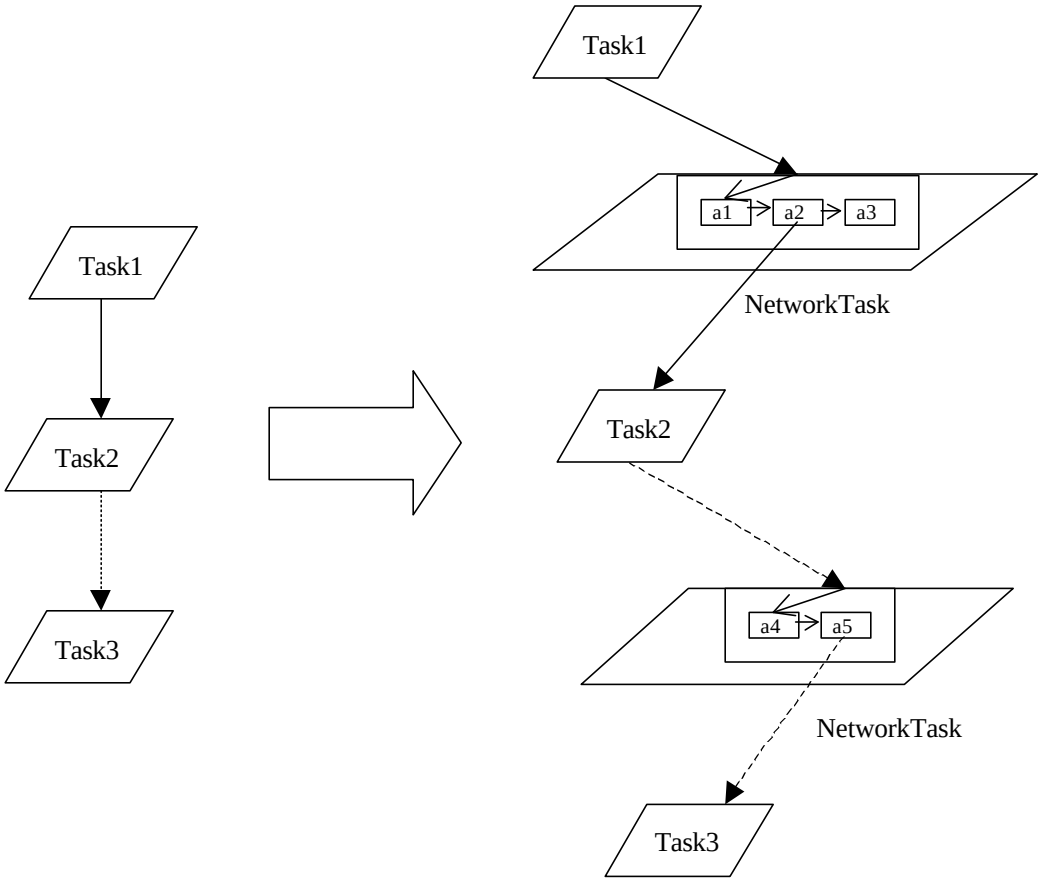


(a) Asynchronous Call



(b) Pure Synchronous Call

Figure 3.6 Network Delay Insertion in LQN Model (1)



(c) Combined Synchronous and Forwarding Call

Figure 3.7 Network Delay Insertion in LQN Model (2)

This thesis used deterministic network delay in the examples. The value of the network delay is defined in an additional file whose format can be found in Appendix A.2 and Appendix B.2. However this doesn't limit the use of stochastic network delays in the optimization approach. The simulation used in this thesis supports stochastic execution demands. The network delay could even be extended to any other distributions by creating specific generators.

CHAPTER 4: OPTIMIZATION STRATEGY: ORIGIN AND IMPROVEMENT

This chapter proposes the strategies to optimize distributed real-time systems. H. El-Sayed's original optimization strategies described in [11] are introduced, and the improvements and modifications are suggested after examining and discussing these strategies in detail.

4.1 Procedure Of Optimization

This section introduces the optimization procedure proposed in [11] and the new optimization procedure which could deal with the communication cost in the LQN model.

H. El-Sayed described a methodology of optimizing LQN model to meet scenario deadlines in [11]. The methodology consists of two phases: an initialization phase and an optimization phase. The initialization phase suggests a good initial design as a starting point, and makes the optimization more efficient. The optimization phase, the core of the methodology, modifies the initial design gradually in order to meet all the scenario deadlines.

The initialization phase provides the initial task allocation and priority assignment. The Multifit-Com algorithm [38] is used for the initial task allocation, and the Proportional- Deadline-Monotonic algorithm is used for the initial priority assignment.

The optimization phase first adjusts the priorities, by raising the priority of the task which is recommended by calculating the task metric. If the priority adjustment is not able to find a feasible solution, the system is "reshaped" by changing the allocation of tasks to other processors and modifying the task structure, and then repeats the priority adjustment. The procedure stops if a feasible solution is found or the termination conditions are satisfied.

The original procedure of optimization is kept and yet is modified in the present thesis. In order to describe the model more precisely, the communication costs (communication overhead and network delay) are added before solving the model. When the model is modified during the optimization, the communication costs will be removed and added automatically to identify the effect of communication cost correctly. The high level flow chart of the new optimization procedure is shown in Figure 1.1.

This work adds the communication cost handling during the procedure of optimization.

4.2 Initial Allocation Using Multifit-Com Algorithm

This section introduces the initial allocation used in [11]. It is kept the same in this thesis.

The Multifit-Com algorithm was originally designed for allocating independent communicating tasks to processors sharing certain communication links. It uses the total execution time and the communication cost during one scenario or response to estimate the tasks. In [11], the starting point of the analysis was a scenario model which was used to determine the execution and communication cost communication cost parameters. However in this thesis, the starting point is already in LQN form, and the sequence of operations implied by LQN model must be used. In LQN model, the calls are sent to entries, which are the service ports of the tasks. The tasks may have more than one entry to receive calls. One entry might be invoked by more than one scenario with different arrival rates, and the visit ratios of the entries might be different from each other. In order to allocate tasks in the LQN model using Multifit-Com algorithm, the task sizes and communication overheads must be calculated specifically considering the issues mentioned. The task sizes are calculated by the following equations in [11]:

$$\text{TaskSize}(T) = \sum_{\forall e \in \text{ entries}(T)} \text{executionDemand}(e) \times \text{invokeRate}(e) \qquad (4.1.1)$$

$$\text{executionDemand}(e) = \sum_{\forall \text{activity} \in \text{ activities}(e)} \text{executionTime}(\text{activity}) \qquad (4.1.2)$$

where *entries(T)* is the set of entries in task *T*
activities(e) is the set of activities (or phases) in entry *e*

The calculation of *invokeRate(e)* is already provided in equation (3.10) and (3.11), using entry *e* in place of *a*.

The *TaskSize(T)* is the task size estimated for task T. The execution demand of an entry *executionDemand(e)* is the total execution times among all the activities. Its calculation is straightforward.

The interprocessor communication costs are calculated as the following equations if they exist in [11]:

$$\text{CommOV}(T_1, T_2) = \sum_{\forall e \in \text{entries}(T_1)} \text{CommOV}(e, T_2) + \sum_{\forall e \in \text{entries}(T_2)} \text{CommOV}(e, T_1) \qquad (4.2.1)$$

$$\text{CommOV}(e, T) = \left(\sum_{\forall m \in \text{physical messages from e to T}} \text{CommOV}(m)\right) \times \text{invokeRate}(e) \qquad (4.2.2)$$

CommOV(T_1, T_2) is the communication overhead between task T_1 and T_2, and *CommOV(e, T)* is the communication overhead between entry *e* and task *T*. In this thesis it assumes that the communication overhead sending and receiving messages depends on the number of physical messages, the length of the message and whether the sender and receiver are on the same processor. The communication overhead is charged on both the sender side and the receiver side evenly (See chapter 3 for details).

The calculation of *CommOV(m)* is provided in equation (3.1)

TaskSize(T) and *CommOV(T_1 , T_2)* are normalized by invocation rate, and they are actually the CPU utilization consumed by the task and communication. Thus they are suitable for the Multifit-Com algorithm.

Multifit-Com uses a large set of different heuristics called “policies” in [38], to create candidate allocation, and then chooses the best one according to its criteria (which is to maximize a throughput bound). The policies are identified as a 4-Turple (PC, TOT, ITS, IBW) which is defined in [38] and mentioned in Figure 2.1. This work follows in El-Sayed’s steps using the following eight policies keeping all eight solutions as separate starting points for the optimization phase.

Policy No.	Policy Options = (PC, TOT, ITS, IBW)
2	(G, A, S_u, W_l)
13	(F, A, S_u, W_u)
6	(G, A, S_c, W_l)
12	(G, C, S_c, W_l)
8	(G, C, S_u, W_l)
10	(G, C, S_l, W_l)
19	(F, C, S_u, W_u)
4	(G, A, S_l, W_l)

Figure 4.1 The Policy Set Used in [11]

4.3 Initial Priority Assignment Using Modified Proportional-Deadline-Monotonic Algorithm

This section describes the modified Proportional-Deadline-Monotonic algorithm. The calculation of the proportional deadline which is not shown in [11] is also provided.

After task allocation, the Proportional-Deadline-Monotonic algorithm is used for initial priority assignment. The longer the proportional deadlines, the lower the priority assigned. The equation of calculating the proportional deadlines are provides as follows:

$$\text{ProportionalDeadline}(T) = \min_{\forall S \in \text{Scenarios}} (\text{ProportionalDeadline}|_T^S) \qquad (4.3.1)$$

$$\text{ProportionalDeadline}|_T^S = \frac{\text{TaskSize}|_T^S \times \text{Deadline}_S}{(\sum_{\forall T_i \text{ relates to } S} \text{TaskSize}|_{Ti}^S) \times \text{visitRate}|_T^S} \qquad (4.3.2)$$

$$\text{visitRate}|_T^S = \begin{cases} 1 & \text{if } T \text{ is called directly by S} \\ \sum_{\forall T_0 \in \text{predecessor}(T)} \text{meanCalls}(T_0, T) \times \text{visitRate}|_{T_0}^S & \end{cases} \qquad (4.3.3)$$

If one task is included in more than one scenario, the proportional deadlines of Task T *ProportionalDeadline(T)* in different scenarios are calculated separately and the minimal one will be chosen as its proportional deadline. $TaskSize|_T^S$ is the part of the *TaskSize(T)* where only the arrival rate of scenario *S* is counted. The visit rate of task *T* in scenario *S* $visitRate|_T^S$ must be considered if the task *T* is called more than one time in scenarios.

Proportional-Deadline-Monotonic algorithm is originally created for hard real-time systems where tasks are modeled as infinite server tasks in LQN model. When a general distributed real-time system is modeled by LQN, the task may have a finite number of threads. The Proportional-Deadline-Monotonic algorithm should also consider the number of threads according to the experience of the author. Heuristically, a single threaded task is easier to be a bottleneck that delays the response time than a multithreaded task. In this thesis, the modified proportional deadline is described as follows:

1. Sort all the tasks according to their number of threads. The lower the number of threads, the higher the priority.
2. If more than one task have the same number of threads, those tasks will be assigned priorities according to their proportional deadlines. The shorter the proportional deadline, the higher the priority.

This work extends the Proportional-Deadline-Monotonic algorithm to general systems, by taking into account the number of threads in the tasks.

4.4 Estimation for Solution Quality

This section describes the original and the new formulas to estimate the solution quality. The difference between these two estimations is discussed and compared.

In order to measure the quality of a solution, a criticality metric is established according to the probability of missed deadlines in [11]:

$$\text{SolutionPenalty} = \sum_{\forall(S \in \text{Scenarios})} \text{Criticality}_S \qquad (4.4.1)$$

$$\text{Criticality}_S =$$

$$\begin{cases} 0 & \text{probExceedDeadline} \le \alpha \\ e^{\beta \times \text{ProbExceedDeadline}} & \text{probExceedDeadline} > \alpha \end{cases} \quad (4.4.2)$$

where α is the probability of missed deadlines.

e.g. $\alpha = 0$ for hard real-time systems and $\alpha = 0.05$ for soft real-time systems

β is a design decision parameter. e.g $\beta = 15$

The exponential power indicates the huge penalty on the scenarios with big missed probabilities. The zero solution penalty means a feasible solution.

The calculation of solution penalty is straightforward. However in this work the calculation of criticality is modified as equation (4.4.3) in this thesis:

$$\text{Criticality}_S = \begin{cases} 0 & \text{probExceedDeadline} \le \alpha \\ e^{\beta \times (\text{ProbExceedDeadline} - \alpha)} & \text{probExceedDeadline} > \alpha \end{cases} \quad (4.4.3)$$

The two different functions to calculate the criticality metric are visualized in Figure 4.2.

The reason behind the modification is explained by the following example. A system with two scenarios has the following deadline requirements:

Deadline requirements for the system:

scenario 1: 100% requests must meet deadlines

scenario 2: 95% requests must meet deadlines

There are two candidate solutions:

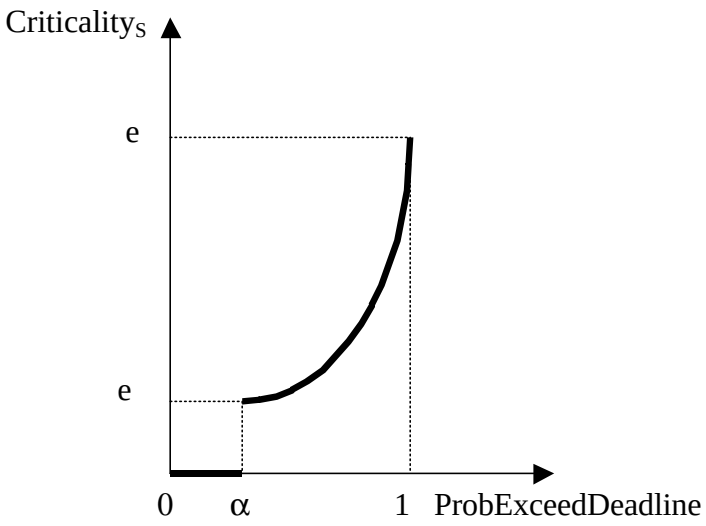
Solution 1 results:

scenario 1: 96% requests meet deadline

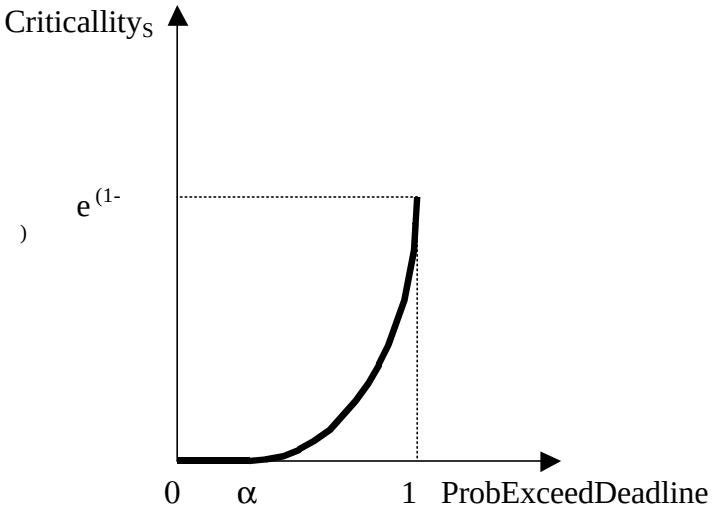
scenario 2: 100% requests meet deadline

Criticality$_S$ using equation(4.4.2): 1.822

Criticality$_S$ using equation (4.4.3): 1.822



(a) Original Function for Criticality Metric, Equation (4.4.2)



(b) New Function for Criticality Metric, Equation (4.4.3)

Figure 4.2 Original and New Criticality Metric

Solution 2 results:

scenario 1: 100% requests meet deadline

scenario 2: 94% requests meet deadline

$Criticality_S$ using equation(4.4.2): 2.46

$Criticality_S$ using equation(4.4.3): 1.162

If we use the original equation to calculate the solution penalty, the solution 1 is better than solution 2. However solution 2 is better than solution 1 if the new equation is applied. Although the absolute missed probability in solution 2 (6%) is larger that of solution 1 (4%), the missed probability in solution 2 is closer to the requirements (1% difference) than that of solution 1 (4% difference). This is what is important to the optimization. Moreover, as we can see in Figure 4.2, the new criticality metric uses a continuous function instead of a discontinuous function. The continuous function is supposed to optimize the model more smoothly.

Whenever a model is solved, the solution quality will be calculated, and the best configuration, which has the least solution penalty value, will be kept until the optimization is finished. This will improve the optimization behavior and avoid thrashing.

This work uses a more "reasonable" continuous function to estimate the criticality of an LQN model. The justification is heuristic.

4.5 Task Metric and Priority Adjustment Strategy

This section introduces an improved priority adjustment strategy and modified task metric after discussing the original ones.

When a solution is not feasible, optimization is required to improve the solution. The optimization is the core part of the methodology.

The priority adjustment is the first consideration for optimization due to its low overhead. A task metric, which measures the task's contributions to problems failures, suggests which task should have a higher priority. Task metric and priority adjustment

strategies are important parts of the optimization. The original metrics and strategies will be examined carefully to bring about a better solution.

4.5.1 Task Metric

The calculation of the task metric is given in [11] by the following equations:

$$\text{TaskMetric}_T = 1/U_T \times \sum_{\forall S \in \text{Scenarios}} \text{TaskScenarioMetric}_T|_T^S \times \text{Criticality}_S \quad (4.5.1)$$

$$\text{TaskScenarioMetric}_T|_T^S = \text{TaskWaitMetric}|_T^S + \text{askCommOVExcessMetric}|_T^S \quad (4.5.2)$$

$$\text{TaskWaitMetric}|_T^S = \sum_{\forall (a \in \text{Activities}|_T^S)} \text{WaitingTime}_a \quad (4.5.3)$$

$$\text{TaskCommOVExcessMetric}|_T^S = \frac{\sum_{\forall (m \in \text{nonLocalMsgs}|_T^S)} \text{CommOV}_m}{\text{noTargetCPUs}|_T^S} - \sum_{\forall (m \in \text{LocalMsgs}|_T^S)} \text{CommOV}_m \quad (4.5.4)$$

Where

* $Activities|_T^S$ ≡ set of activities of task *T* that are on the critical path of scenario *S*
* nonLocalMsgs ≡ set of non-local messages of task T that are on the critical path of scenario *S*
* $LocalMsgs|_T^S$ ≡ set of local messages of task *T* that are on the critical path of scenario *S*

$noTargetCPUs|_T^S$ ≡ no of CPUs that task *T* communicates with during the execution of the critical path of *S*

* U_T is the CPU utilization of task *T* which is provided in the output file of the simulation

The original task metric combines the task waiting time during execution and the potential average communication overhead excess if reallocation occurs. The optimization does need to consider these two issues to adjust the priority and reshape the design, but just for priority adjustment, it is not necessary to consider the communication overhead excess. In this thesis, only the task waiting times will be considered for calculating the task metric. It's heuristic that the waiting times (when a task is ready to

run but not able to run) for the tasks along the scenario critical paths cause the scenario to have a longer service time and hence to miss the deadline. Obviously the task which contributes the largest waiting time should have its priority raised. The following new equations (4.5.5 – 4.5.6) are used to calculate the task metric:

$$\text{TaskMetric}_T = 1/\,U_T \times \sum_{\forall S \in \text{ Scenarios}} \text{TaskWaitMetric}_T|_T^S \times \text{Criticality}_S \qquad (4.5.5)$$

$$\text{TaskWaitMetric}|_T^S = \sum_{\forall e \in \text{ Entries}|_T^S} \text{WaitingTime}_e \times \text{criticalInvokeRate}(e, S) \qquad (4.5.6)$$

The new equations remove the communication overhead excess from the task metric and consider the visit rates along the scenario critical paths *criticalInvokeRate(e, S).* Note that *criticalInvokeRate(e, S)* is the part of the *invokeRate(e)* where only the synchronous calls and forwarding calls from scenario *S* are counted. The calculation equation of *criticalInvokeRate(e, S)* is provided in (3.13).

4.5.2 Priority Adjustment Strategy

When a task is chosen as the candidate task according to the task metric, its priority should be raised in order to get a better solution. A static priority discipline which allows equal priority level for different tasks is used in [11]. However, the “static” priority algorithm is a little bit different from the fixed priority algorithm which requires different priority levels for different tasks. FIFO (First In First Out) scheduling discipline for the tasks with same priority levels is considered as a dynamic priority algorithm in [25]. The hard real-time systems always use fixed priority algorithms which could predict the task behaviors at any time. This optimization approach is targeted at the general real-time systems (including the hard real-time systems), the fixed priority algorithm is adopted in this thesis.

The new priority adjustment strategy assumes different priority levels for different tasks. If the equal priority levels are assigned to different tasks, the priority levels will be sorted by initialization. The procedure of priority adjustment is described as follows:

1. Raise the priority level of the candidate task one level higher.

If the priority combination is visited at the first time, a new priority state is found.

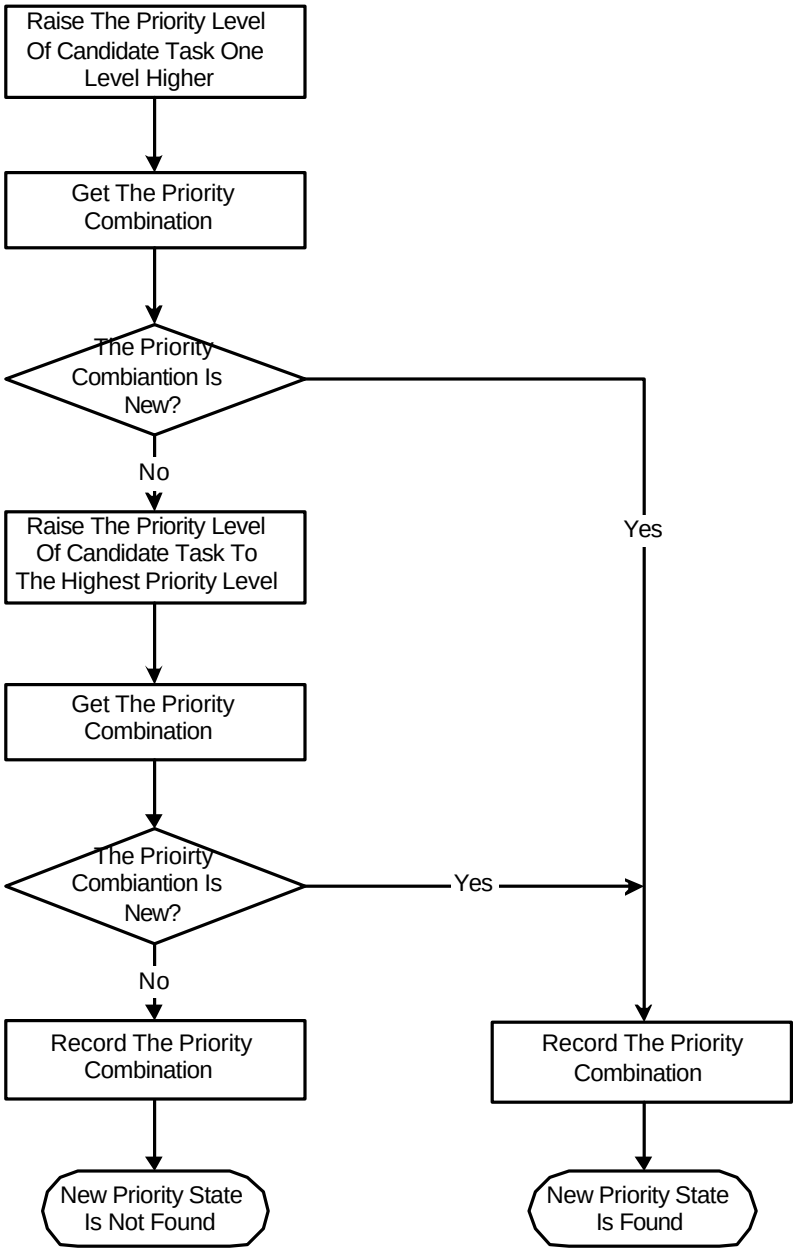


Figure 4.3 Detailed Flow Chart of Priority Adjustment

2. Otherwise, raise the priority level of the candidate task to the highest priority level.
3. If the priority combination is visited at the first time, a new priority state is found. Otherwise, new priority state is not found.

The "Adjust Priority For Candidate Task" block in Figure 1.1 is described in the flow chart (Figure 4.3) in detail.

This work uses the distinct fixed priority approach in place of the priority approach which allows equal priority levels. The task metric for priority adjustment is modified heuristically.

4.6 Task Reallocation and "Task Reshaping"

This section introduces a modified task reallocation strategies and task allocation metric after discussing the original ones.

The design will be reshaped if the priority adjustment could not provide a feasible solution. Two strategies are used for reshaping in [11]: task reallocation and task restructuring. Task reallocation is to reallocate the candidate task to another processor. Task restructuring is to split an entry from the candidate task.

Although task splitting gives benefits to reduce priority inversion, it is not used in the optimization phase in this thesis. The entries are the service ports provided by the task. In some situations, they can't be split and must be kept together. If task splitting is necessary, a simple way is to split tasks in the initialization phase instead of the optimization phase. In the initialization phase, more than one task could be split at the same time and more than one entry could be split from a task. (In this research, task splitting is not automated). The case study in chapter 6 tried this attempt. The task reshaping is hence simplified as task reallocation during optimization phase.

Before task reallocation, the best configuration so far will be restored. By heuristics, the model with better solution quality may have a better chance to be feasible using optimization. The tasks will then be sorted by the scenario metric $Criticality_S$ defined in equation (4.4.3) and the task metric. I.e. the tasks that are along the path of the scenario with the worst scenario metric will be considered first. The task with be worst task metric among the selected tasks is the candidate task for reallocation. The criteria of choosing the destination CPU are different from the criteria in [11]. The metric is described by the equation (4.6.1) – (4.6.3):

$$CPUMetric(T, C) = CommOVGainMetric(T, C) + UtilGainMetric(T, C) \quad (4.6.1)$$

$$\text{CommOVGainMetric}(T, C) = \left(\sum_{\forall m \in \text{nonLocalMsgs}(C, C_0)} \text{CommOV}(m) - \sum_{\forall m \in \text{LocalMsgs}} \text{CommOV}(m) \right) * \text{invokeRate}(m) \quad (4.6.2)$$

$$\text{UtilGainMetric}(T, C) = \text{Util}_{C0} - \text{TaskSize}(T) - \text{Util}_{C} \quad (4.6.3)$$

* *nonLocalMsgs(C, C_0)* ≡ set of non-local physical messages of task *T* between CPU *C* and CPU C_0 (the local CPU)
* *LocalMsgs* ≡ set of local physical messages of task *T* that are on the critical path of scenario *S*

CPUMetric(T, C) is the metric estimating the benefits of reallocating task *T* (originally located on processor C_0) to a new processor *C*. It consists of two parts: the utilization decreased due to communication overhead *CommOVGainMetric(T,C)* and the decreased utilization of the processor on which task *T* locates *UtilGainMetric(T, C)* after reallocation. The *CommOVGainMeric(T, C)* is the processor utilization saved in the system if task *T* is reallocated to processor *C*. *CommOV(m)* could be calculated using equation (3.1) and *invokeRate(m)* could be calculated using equation (3.8) and (3.10). By heuristics, if the total processor utilization in the system is decreased, the performance of the system will be better. *UtilGainMetric(T, C)* estimates whether the utilization of the processor on which Task *T* locates decreases. By heuristics, if a task is reallocated to a processor whose processor utilization is less than that of the original one, the system has more chances to be feasible. $Util_{C0}$ and $Util_C$ are the processor utilization of C_0 and *C* before reallocation. They are included in the simulation output file.

The candidate task will be reallocated to the target processor and the candidate task will be recorded so that it will not be reallocated next time.

The “Reallocate Candidate Task” block in Figure 1.1 is described in the flow chart (Figure 4.4) in detail.

This work simplifies the task reshaping and modifies the metric for task reallocation by heuristics.

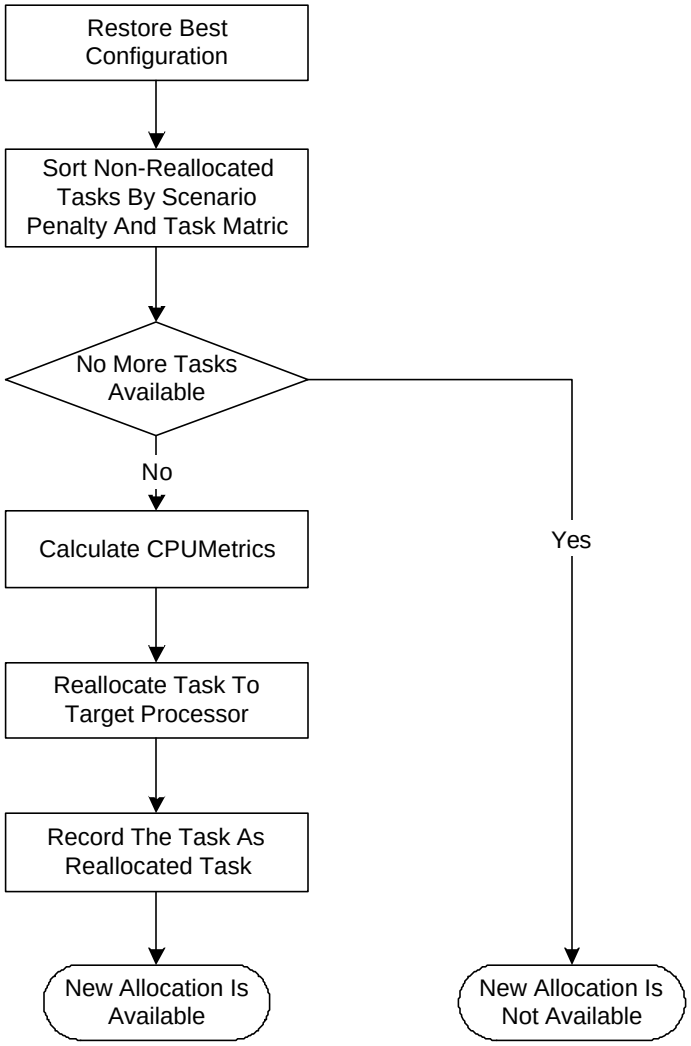


Figure 4.4 Detailed Flow Chart of Task Reallocation

4.7 Optimizer Implementation

The optimization approach was implemented using Java programming language. Figure 4.5 shows the information flow of the implementation. The core part of the optimization is the “Optimizer” module. It collects information from LQN model file and an additional file and controls the running of other programs according to the optimization parameters. The “Optimization Output” will be printed on screen or

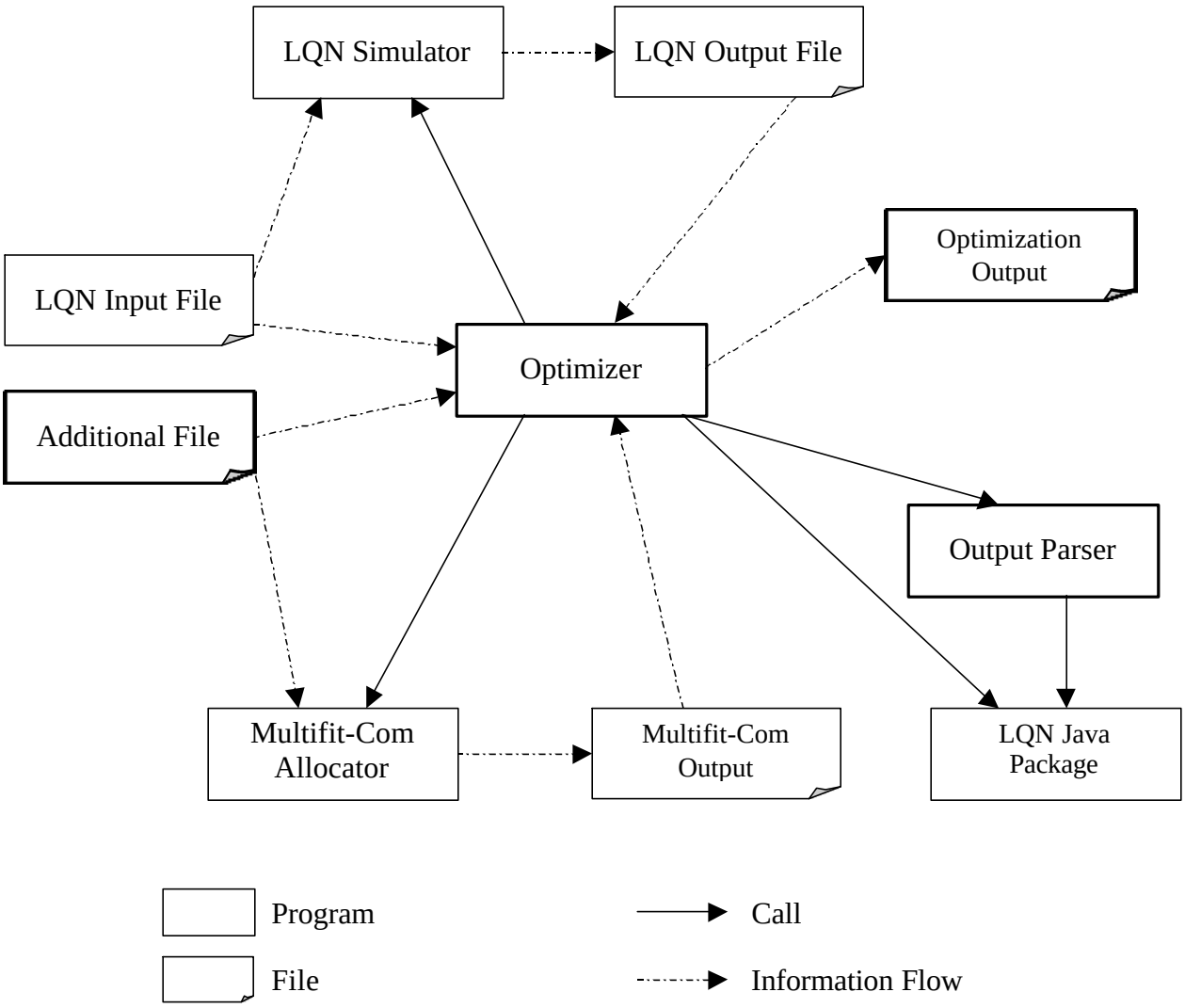


Figure 4.5 Information Flow Diagram for Optimizer Implementation

be redirected to an output file. The blocks with bold border are the work done in this research. When using the optimizer, there are some command line options to decide:

- Whether to initialize priorities
- Whether to initialize task allocation. If initial task allocation is necessary, which policy should be used or all the eight policies should be used.
- Whether the communication cost should be handled during optimization.

- Whether to reallocate the tasks
- The miss rate requirement

The “LQN Input File” is a standard LQN model file. It requires scenario deadlines in pseudo scenario entries to set the deadline requirements for LQN model.

The “Additional File” defines information that is necessary for the optimization, but can’t be expressed in the standard LQN model file. This file contains information for communication cost calculation, task allocation and some concerns for optimization. It can be extended for other purposes (e.g. automatic task splitting) for future uses.

The optimizer uses classes in the “LQN Java Package” developed by former students in the lab. However that LQN Java package only provides basic functions. The “Output Parser” class is created upon the LQN Java package to provide high level functions for the optimizer, hiding the details of LQN model. The metrics and algorithms are also implemented in this class. There are about 6000 lines of Java source code in the Optimizer and “Output Parser” class.

The “LQN Simulator” is a program called “parasrvn”, and “Multifit-Com Allocator” is the program provided by [38] to perform Multifit-Com algorithm.

4.8 Summary

This chapter introduces every part of the optimization approach proposed in [11] in detail. After examining and discussing the original strategies, the optimization approach framework, and algorithms related to the initial task allocation are kept. Priorities are forced to be different for all tasks in present thesis. An improved priority adjustment algorithm with new task metric is proposed. And the original priority initialization algorithm is modified for tasks with finite threads. Due to heuristic reasons, the estimation function for the scenario criticality metric, the metrics for task reallocation and task reallocation strategies are changed as well.

CHAPTER 5: EVALUATION OF THE OPTIMIZATION APPROACH

This chapter evaluates the optimization approach provided in this thesis. The purposes of evaluation are:

- To verify that the re-implementation is reasonably close or better in capability to the original one.
- To verify that the new changes are satisfactory on the test cases
- To evaluate the approach on the stochastic systems with soft deadlines which was not included in [11]

5.1 Evaluation of New Algorithms on Hard Real-Time Systems

The effectiveness of the new optimization technique was compared to the "original" optimization described in [11]. Since [11] only considered deterministic hard real-time systems, this section was based on this kind of system, with two sets of examples:

- Independent periodic tasks on a single processor.
- An example described by Etemadi in [12]

5.1.1 Independent Periodic Tasks Models

In order to check the strength of the priority adjustment strategies, a number of test sets were generated randomly. All the test cases have 2-8 independent periodic tasks with certain utilizations, on one processor. The tasks are assumed to have a fixed priority using the new algorithm and a static priority using the original algorithm. These test cases satisfy the requirements of rate monotonic algorithm, and hence the optimization results could be compared with the results using the rate monotonic algorithm to check the effectiveness. The following table lists the test sets generated and the respective utilization bounds:

Number Of Tasks	Group1 Utilization	Group2 Utilization	Rate Monotonic Utilization Bound
2	0.82	0.90	0.828
3	0.77	0.90	0.779
4	0.75	0.90	0.756
5	0.74	0.90	0.743
6	0.73	0.90	0.734
7	0.72	0.90	0.728
8	0.72	0.90	0.724

Figure 5.1 Test Sets Parameters and Respective Utilization Bounds

For each number of tasks, two groups of test cases are generated. The group1 test cases whose utilization is less than the rate monotonic utilization bound could be definitely scheduled to meet all the deadlines according to the rate monotonic algorithm [24]. The group2 test cases whose utilization is more than rate monotonic utilization bound but less than 1.0 can't be determined simply if they are schedulable. However the rate monotonic algorithm is proved to be optimal among all the fixed-priority algorithms. It could be used as a standard to check the effectiveness of other algorithms. The successful number of cases using the optimization strategy over the successful number of cases using the rate monotonic algorithm is called the "success ratio" of the optimization strategy. In order to increase some difficulties on the test cases, the initial priorities of the tasks are set in a reverse order according to the rate monotonic algorithm. I.e. the task with the shorter period has the lower priority. These task sets are supposed to be difficult ones due to the bad initial priority assignment. 50 models are generated for each task number and utilization combination; hence 700 models are generated totally.

Figure 5.2 lists the experiments results using the original priority adjustment strategy in [11], and Figure 5.3 lists the experiments results using the new priority adjustment strategy in this thesis.

Task Number	Utilization	Total Number of Cases	Number of Successful Cases (RM)	Number of Successful Cases (Optimization)	Success Ratio
2	0.82	50	50	50	100.00%
2	0.9	50	41	41	100.00%
3	0.77	50	50	50	100.00%
3	0.9	50	37	37	100.00%
4	0.75	50	50	49	98.00%
4	0.9	50	24	22	91.67%
5	0.74	50	50	47	94.00%
5	0.9	50	20	16	80.00%
6	0.73	50	50	49	98.00%
6	0.9	50	5	2	40.00%
7	0.72	50	50	48	96.00%
7	0.9	50	7	3	42.86%
8	0.72	50	50	47	94.00%
8	0.9	50	3	1	33.33%

Figure 5.2 Experimental Results Using Original Priority Adjustment Strategy

Task Number	Utilization	Total Number of Cases	Number of Successful Cases (RM)	Number of Successful Cases (Optimization)	Success Ratio
2	0.82	50	50	50	100.00%
2	0.9	50	41	41	100.00%
3	0.77	50	50	50	100.00%
3	0.9	50	37	37	100.00%
4	0.75	50	50	50	100.00%
4	0.9	50	24	24	100.00%
5	0.74	50	50	50	100.00%
5	0.9	50	20	20	100.00%
6	0.73	50	50	50	100.00%
6	0.9	50	5	5	100.00%
7	0.72	50	50	50	100.00%
7	0.9	50	7	7	100.00%
8	0.72	50	50	50	100.00%
8	0.9	50	3	3	100.00%

Figure 5.3 Experimental Results Using New Priority Adjustment Strategy

The results are quite encouraging. The new strategy finds out a feasible solution every time when the rate monotonic algorithm succeeds. The success ratio of the original algorithm is less than 100% especially when the task number and the utilization increase. The new algorithm indicates a better performance than the original one in addition to its predictability.

5.1.2 Etemadi's Transaction Model

A sonar signal processing example originally described by Etemadi [12] is discussed in [11]. It consists of 13 transactions running on 8 processors. Figure 5.4 shows the model of Etemadi's example.

A conventional task graph notation is used in Figure 5.4. The arrows show the precedence dependency, and blocks are tasks in the task graph. The LQN model file was provided by H. El-Sayed, although it is not listed in [11]. For each task in the task graph, it is represented by a task in LQN model. Each transaction is represented as a scenario. Each driver is described by a reference task to generate arrival rate and a pseudo task to collect response time and set deadline. For the scenarios with a single task (e.g. scenario 2 – scenario 11), the arrows are synchronous calls in the LQN model file. For the scenarios with multi branches (e.g scenario 0 and scenario 1), one branch is expressed by a forwarding call, and the other branches are expressed by asynchronous calls. All these branches join together and send a reply to the pseudo task to get a total execution time of a transaction.

Three experiments with different resource requirements are studied in [11]. These three experiments were repeated using the new algorithm introduced in this thesis.

- Experiment #1: all the transactions are run on 8 processors. Every task is required to be allocated to a specific processor. This is same as the one described in [12].
- Experiment #2: the number of processors is reduced to 7. The tasks could be allocated to any processors.
- Experiment #3: only 6 processors are available. The tasks could be allocated to any processors.

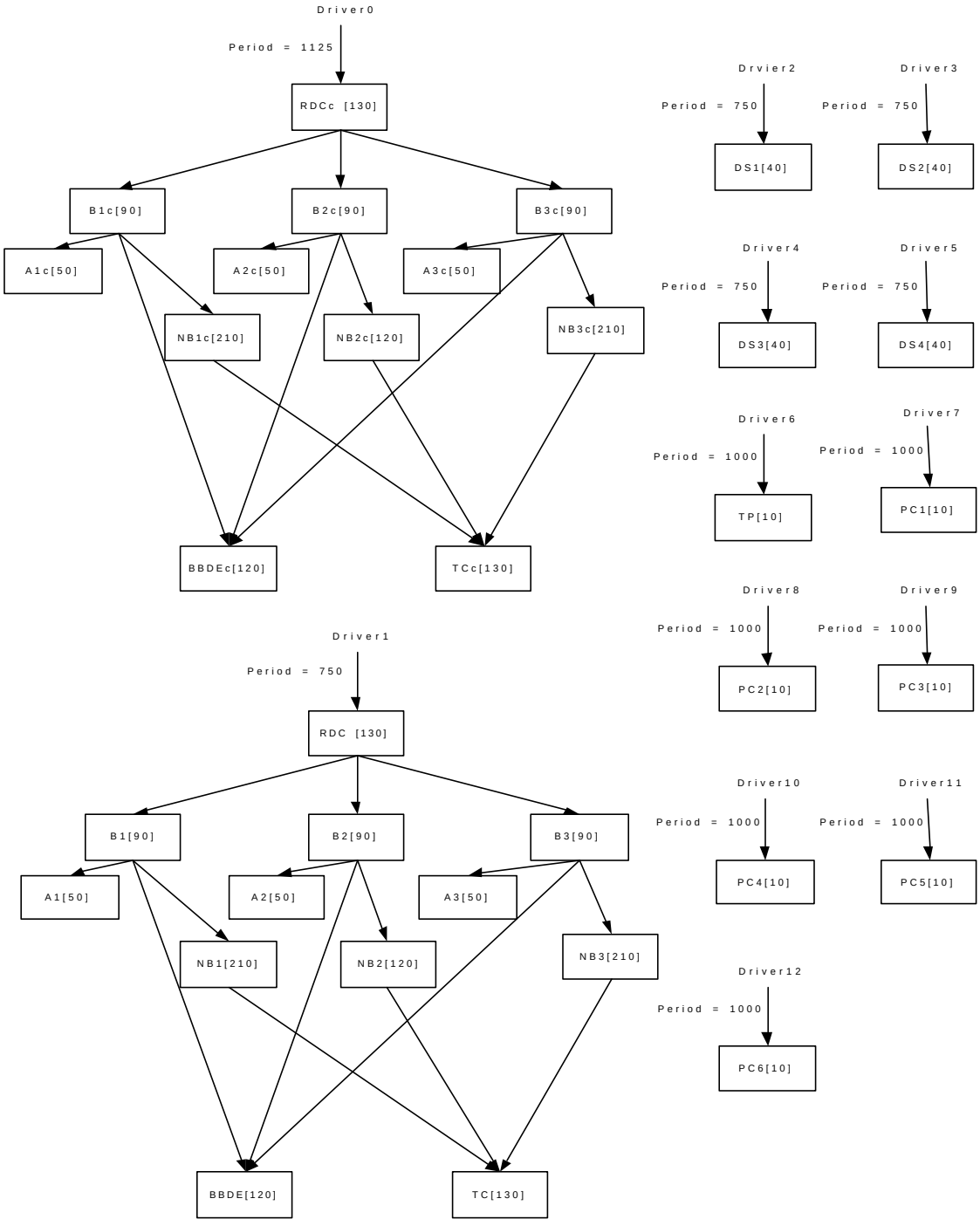


Figure 5.4 The Model for Etemadi's Example

These three experiments can check not only the priority adjustment strategy, but also the task reallocation strategy. Figure 5.5 summarizes the experimental results using the original algorithms and the new algorithms.

		Original Algorithms		New Algorithms	
Experiment #	# of CPUs	Success Ratio	Avg. Steps	Success Ratio	Avg. Steps
1	8	8/8	0	8/8	0
2	7	8/8	2	8/8	9
3	6	7/8	60	8/8	38

Figure 5.5 Experiment Results for Etemadi's Transaction Model

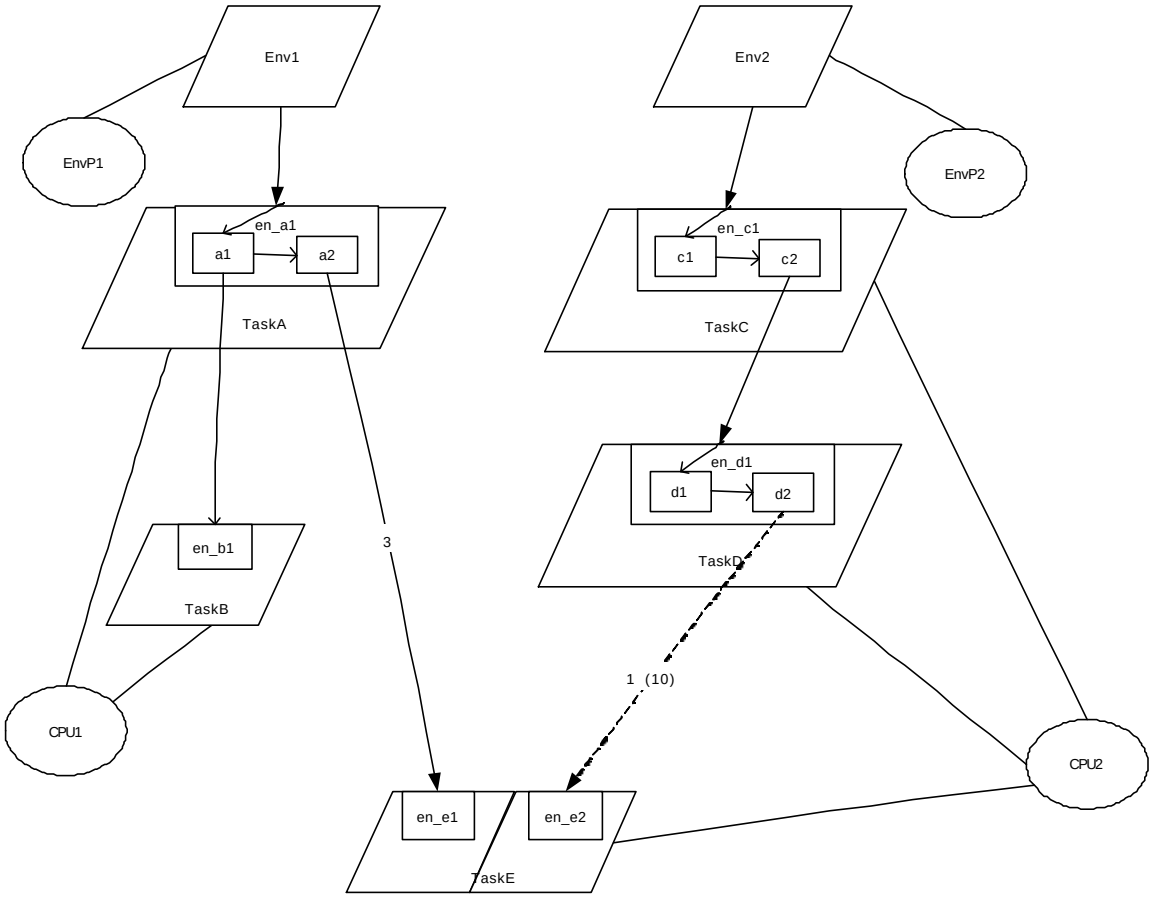
Both the original and the new optimization algorithms find the feasible solution in experiment #1 just after initialization. It shows the algorithms used for initialization are quite good. The new algorithms found a feasible solution every time from 8 different starting points in experiment #3 while the original algorithms succeeded from 7 of the 8 starting points. Along with the experimental results of the independent periodic tasks in the previous section, it indicates that the new algorithms may provide a higher success ratio than the original ones.

5.2 Effectiveness of Communication Cost Approach

In order to check the effectiveness of the communication cost approach, a tutorial model with communication cost is created (Figure 5.6). The overhead of sending and receiving one message with unit message length and the network delay are modified in two scenarios to indicate different communication costs in the model.

In the tutorial model, the call from *a2* to *en_e1* has the value of 3 for request frequency (i.e. 3 requests per invocation), and every message has one message length unit (by default). The call from *d2* to *en_e2* has 1 request per invocation (by default), while this message has a length of 10 message length units. Other calls have the default values, i.e. 1 request per invocation and the message length is 1 message length unit.

In scenario 1, the network delay is 0. The communication cost is only the communication overhead sending and receiving messages between different processors.



Scenario1:
Communication Overhead for one message per message length unit: 0.7
Network Delay: 0
The deadlines will be met when the priorities of TaskE is raised.

Scenario2:
Communication Overhead for one message per message length unit: 0.01
Network Delay: 1.7
The deadlines can't be met just under priority adjustment.
The deadlines will be met when TaskE is reallocated to CPU1.

Execution Demands:	
a1:	35
a2:	0
en_b1:	3
c1:	20
c2:	0
d1:	20
d2:	0
en_e1:	1
en_e2:	1

Periods and Deadlines:
Env1: 50
Env2: 50

Initial Priorities:
TaskA: 2
TaskB: 1
TaskC: 3
TaskD: 2
TaskE: 1

Figure 5.6 Tutorial Model for Communication Cost Approach

The total communication cost between CPU1 and CPU2 is the communication overhead for the call from *a2* to *en_e1*:

Total communication cost (*TaskE* on CPU2) = 3 * 0.7 + 3 * 0.7 = 4.2

If *TaskE* is assigned to CPU1, then the total communication cost will be the communication overhead for the call from *d2* to *en_e2* and the reply from *en_e2* to *c2*:

Total communication cost (*TaskE* on CPU1) = 10 * 0.7 + 1 * 0.7 = 7.7

The scenario 1 (*TaskE* is assigned to CPU2) has lower communication cost, and has a better chance to meet the deadlines. It has been checked that if *TaskE* is assigned to CPU1, the deadline requirements can't be met 100%.

Figure 5.7 indicates the steps to optimize the model in scenario 1:

Step	Candidate Task	Actions And States	Resp. Times of Scenarios	Prob. of Missed Deadlines	Solution Penalty
0		CPU1: TaskA > TaskB CPU2: TaskC > TaskD > TaskE	50 44.8	0.0005 0	1.0075
1	TaskE	Raise priority of TaskE CPU1: TaskA > TaskB CPU2: TaskC > TaskE > TaskD	45 49.6	0 0	0

Figure 5.7 Optimization Steps in Scenario 1

The LQN model file, the additional file and the optimization output of the tutorial example in scenario 1 are listed in Appendix A.

In scenario 2, the network delay is 1.7 and the overhead sending and receiving one message with one message length unit is 0.01. The network delay dominates the communication cost. The total communication cost between CPU1 and CPU2 is the communication overhead for the call from *a2* to *en_e1* and the network delay during message transmissions:

Total communication cost (*TaskE* on CPU2) = 3*0.01 +3*0.01 +3*1.7 +3*1.7 =10.26

If *TaskE* is assigned to CPU1, then the total communication cost will be the communication overhead for the call from *d2* to *en_e2* and the reply from *en_e2* to *c2* and the respective network delay:

Total communication cost (TaskE on CPU1) = 10*0.01 +1*0.01 + 1*1.7 +1*1.7 =3.42

The *TaskE* should be reallocated to CPU1 to have a better chance to meet the deadlines. The optimization result confirms this calculation.

Figure 5.8 indicates the steps to optimize the model in scenario 2.

Step	Candidate Task	Actions And States	Resp. Times of Scenarios	Prob. of Missed Deadlines	Solution Penalty
0		CPU1: TaskA > TaskB CPU2: TaskC > TaskD > TaskE	75 46.8	1 0	3269017
1	TaskE	Raise priority of TaskE CPU1: TaskA > TaskB CPU2: TaskC > TaskE > TaskD	53.847 48.451	1 0	3269017
2	TaskE	Raise priority of TaskE CPU1: TaskA > TaskB CPU2: TaskE> TaskC > TaskD	51.724 48.69	1 0	3269017
3	TaskE	TaskE has the highest priority, its priority can't be raised any more. The best configuration is restored (step0). The TaskE has the largest task metric, and is reallocated to CPU1 with the highest priority. CPU1: TaskE > TaskA > TaskB CPU2: TaskC > TaskD	41.42 49.63	0 0	0

Figure 5.8 Optimization Steps in Scenario 2

The optimization really gives the result that is expected. It shows that the optimization algorithms have adaptabilities to configure the systems to meet the deadline requirements according to different environment parameters.

The LQN model file, the additional file and the optimization output of the tutorial example in scenario 2 are listed in Appendix B.

5.3 Evaluation on Soft Real-Time Systems with Stochastic Execution Demands

Figure 5.9 is a random software architecture introduced in [11] to show the robustness of optimization on hard real-time applications with parameters that were generated randomly. The evaluation is extended here to soft deadlines and stochastic execution demands to study the performance of the architecture. Statistical models with soft

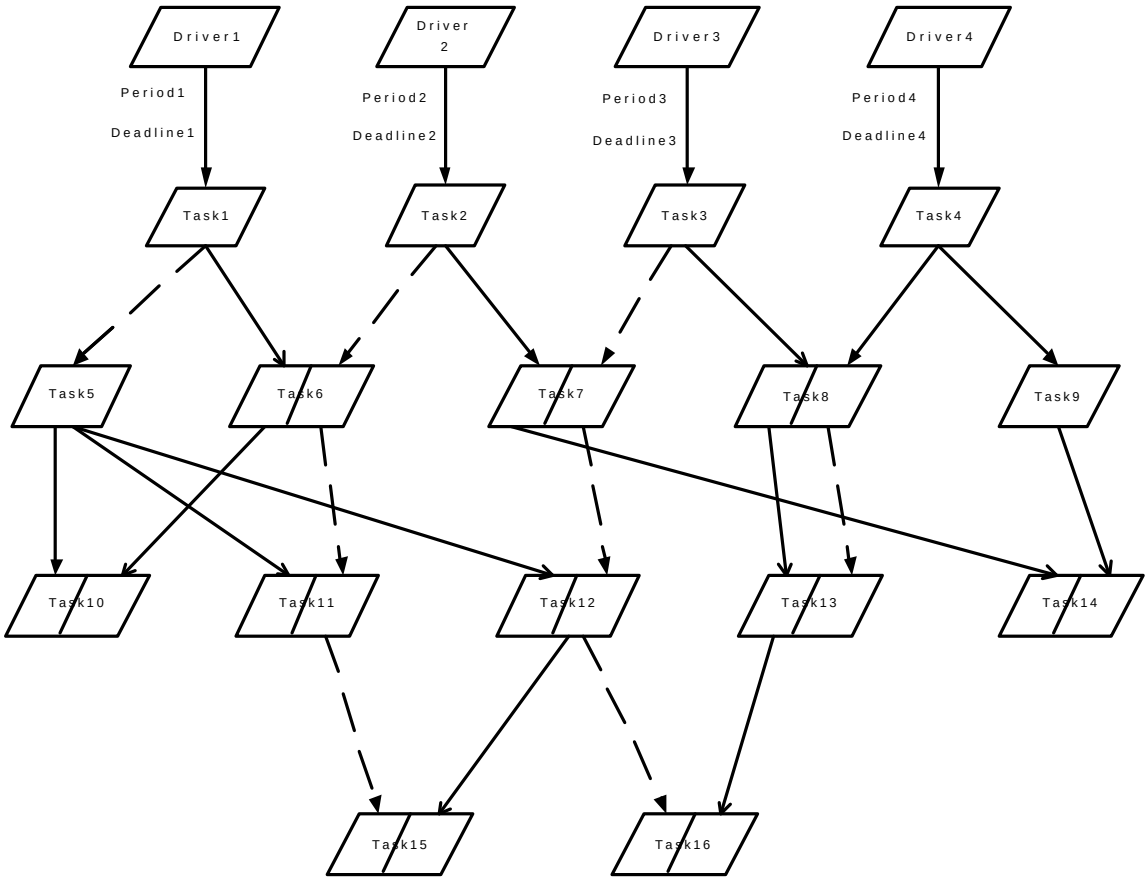


Figure 5.9 Random Statistical Models

deadlines and stochastic execution demands are generated randomly based on this architecture. The characteristics of the statistical models are described as follows:

- Each model has 4 scenarios with 16 tasks on 4 processors
- The average execution demand for every entry is uniformly distributed between [80, 120] time units
- The coefficient of variation of the execution demand is chosen from 0.0 (deterministic), 0.1, 0.5 or 1.0 (exponential)
- The request frequencies or forwarding probabilities of the calls are 1 (by default)
- Communication cost is omitted
- Every scenario has a fixed period and deadline, the scenario periods and deadlines are calculated as the equations (5.1 – 5.3). The deadline requirement is that the deadline miss rate is less than 5%

$$Deadline_s = CriticalDemand_s * L \quad \forall \text{ Every Scenario } s \tag{5.1}$$

$$Deadline_s / Period_s = Constant \quad \forall \text{ Every Scenario } s \tag{5.2}$$

$$\sum (Demand_s / Period_s) = 4 * U \quad \forall \text{ Every Scenario } s \tag{5.3}$$

where $Deadline_s$ is deadline for scenario *s*

$Demand_s$ is the sum of the execution demands in scenario *s*

$CriticalDemand_s$ is the summation of the execution demands along the critical path in scenario *s* (the asynchronous calls are not counted)

L is the laxity factor, taking values between 1.9 and 6

U is the average utilization of the processors, taking value between 0.4 and 0.8

There are altogether 240 different combinations for the coefficient of variation, laxity factor and utilization. 50 models were generated for each combination. All the 12000 models were optimized by the algorithms introduced in this thesis. The success ratio of each combination is listed in Appendix C. For each coefficient of variation and utilization

combination, we can see the minimum laxity factor value for which all cases were feasible. This gives us a heuristic guideline for feasibility of deadlines in general, based on a system's utilization and coefficient variation values. Figure 5.10 and Figure 5.11 are the table and diagram for the result.

	Coefficient Of Variation			
Utilization	0.0	0.1	0.5	1.0
0.4	2.1	2.1	2.5	3.0
0.5	2.1	2.3	2.7	3.0
0.6	2.1	2.3	3.0	3.5
0.7	2.3	2.7	3.5	4.5
0.8	2.5	2.7	4.5	6.0

Figure 5.10 Minimum Laxity Factor Value Providing Feasibility for Different Coefficient of Variation and Utilization Combination (Table)

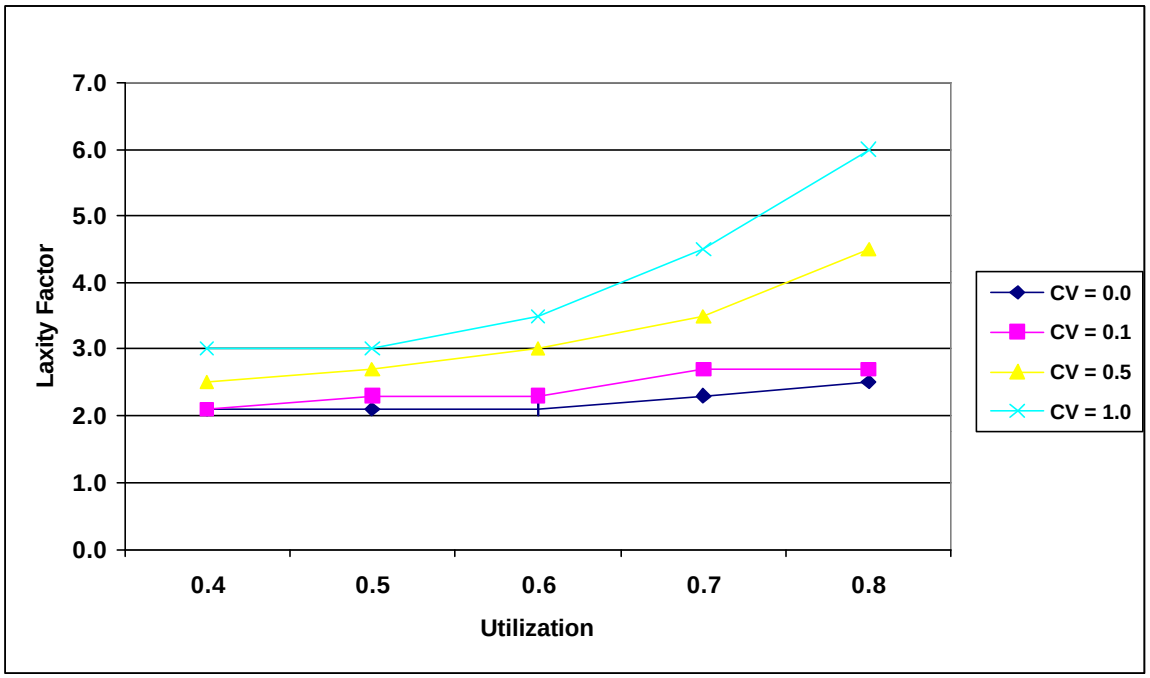


Figure 5.11 Minimum Laxity Factor Value Providing Feasibility for Different Coefficient of Variation and Utilization Combination (Diagram)

From Figure 5.10 and Figure 5.11, we can see:

- For a given utilization, the required minimum laxity factor increases as the coefficient of variation increases.
- For a give coefficient of variation, the required minimum laxity factor increases as the utilization increases.
- The minimum laxity factor with large coefficient of variation (e.g. 1.0) increases much faster than that with small coefficient of variation (e.g. 0.0) as the utilization increases. This indicates the extreme difficulties to meet deadline requirements with large coefficient of variation and high utilization.

CHAPTER 6: CASE STUDY

This chapter studies an e-commerce based real-time system with soft deadlines and stochastic execution demands. The comparison between the baseline model (which assumes all task have the same priority level) and the optimized model shows the effectiveness of the optimization approach on practical applications.

6.1 RADS Bookstore Model

A simplified e-commerce site example called RADS bookstore is described by D. Petriu and M. Woodside in [27]. The RADS bookstore model is established to analyze the software performance in that paper. The model is designed as a 3-tier client-server system (client, application and database tiers). The customer has 7 scenarios: browsing item list, viewing detailed item description, adding or removing items to or from shopping cart, checking out the items in shopping cart, registering and logging into the RADS bookstore. The administrator can update the inventory and fill the outstanding back orders. This model is adopted and modified to some degree to study the optimization issues on soft real-time systems with stochastic execution demands in this thesis.

Figure 6.1 is the simplified LQN model of RADS bookstore. The diagram suppresses the detail of entries, activities and workload parameters. The multiple interaction arrows show the different numbers of different kinds of access made from one task to another. In Figure 6.1, in processor *BookstoreProc,* task *RADSbookst*ore is the interface for customers and administrator to access the application server. The task Server is the application task with 5 threads. *CustomerAccount, ShoppingCart* and *InventoryMgr* are applications to manage customer accounts, shopping cart objects and inventory. *BackorderMgr* is a subsystem to track and fill back-orders. *Database* and *CustomerDB* in processor *DatabaseProc* are the databases for the inventory and customer accounts. The diagram originates from a diagram in [27], and the model has been modified as follows:

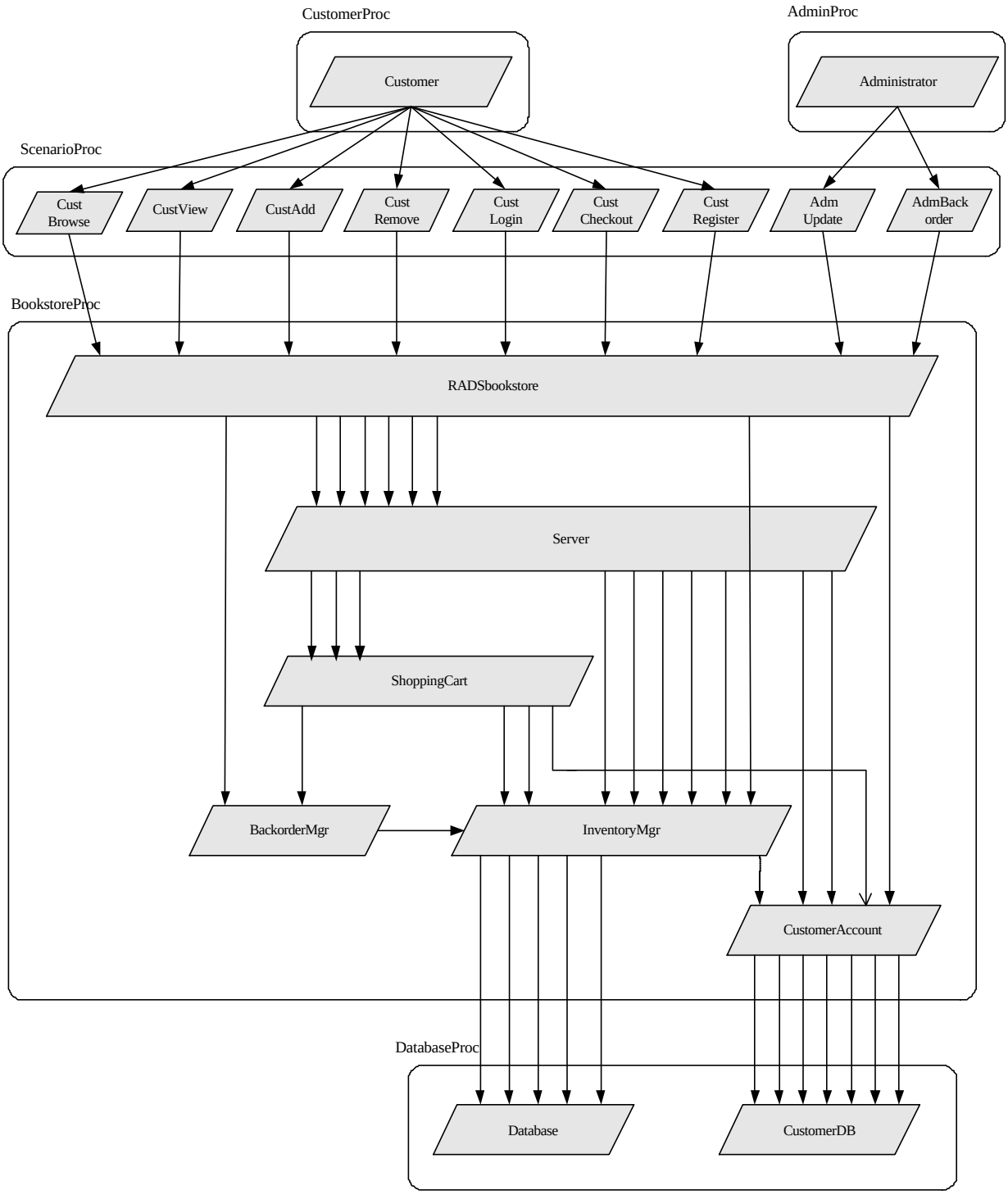


Figure 6.1 Simplified LQN Model of RADS Bookstore

- Each scenario for the Customer and Administrator is separated by a pseudo task running on a pseudo processor *ScenarioProc*, The pseudo tasks are used to collect response time and set deadlines. The scenario deadlines for Customer are set to 500ms, and the scenario deadlines for Administrator are set to 6000 ms. The deadline miss rates for scenarios are required to be no more than 5%.
- The speed rate of the processor *DatabaseProc* is increased to remove the bottleneck in the processor *DatabaseProc*.
- The coefficients of variation of execution demands on processor *BookstoreProc* and processor *DatabaseProc* are decreased to 0.2 from 1. However, the think times of the Customers and the Administrator are still exponential distribution.

In this case study, the model is simulated with number of customers set to 50, 100, 150, 200, 250, 300 and 350. When the number of customers increases to 350, the utilization of processor *BookstoreProc* is around 87%. It is not necessary to increase the number of customer any more. Because there is only one processor in the application tier, task reallocation is impossible. Priority adjustment is the only optimization option in this model. The baseline model is assumed that all the tasks on processor *BookstoreProc* and processor *DatabaseProc* are scheduled by the FIFO discipline (i.e. all the tasks are assigned the same priority). The optimized model will be compared to baseline model.

The average response times and miss rates of the baseline model and optimized model with different number of customers are listed in Figure 6.2.

Scenario	Baseline Model		Optimized Model	
	Average Response Time (ms)	Average Miss Rate (%)	Average Response Time (ms)	Average Miss Rate (%)
CustBrowse	10.571	0	10.643	0
CustView	3.5695	0	3.6019	0
CustAdd	3.5921	0	3.7944	0
CustRemove	3.5968	0	3.791	0
CustLogin	18.766	0	31.576	0.61409
CustCheckout	7.7461	0	9.3125	0
CustRegister	35.849	0	50.834	0.67979
AdmUpdate	1796.9	3.5864	1676.9	2.9079
AdmBackorder	2053.8	5.2072	1900.7	4.4059

(a) RADS Bookstore Model Results (Number of Customers = 50)

Scenario	Baseline Model		Optimized Model	
	Average Response Time (ms)	Average Miss Rate (%)	Average Response Time (ms)	Average Miss Rate (%)
CustBrowse	11.378	0	11.519	0
CustView	4.3706	0	4.4533	0
CustAdd	4.5133	0	4.8611	0
CustRemove	4.5181	0	4.8666	0
CustLogin	21.17	0	33.021	0.60952
CustCheckout	8.7876	0	11.005	0
CustRegister	38.215	0	53.594	0.6623
AdmUpdate	1961.2	4.5876	1673.2	2.8546
AdmBackorder	2207.5	6.5274	1987.1	5.0409

(b) RADS Bookstore Model Results (Number of Customers = 100)

Scenario	Baseline Model		Optimized Model	
	Average Response Time (ms)	Average Miss Rate (%)	Average Response Time (ms)	Average Miss Rate (%)
CustBrowse	12.47	0	12.729	0
CustView	5.4583	0	5.6421	0
CustAdd	5.938	0	6.3035	0
CustRemove	5.9272	0	6.3034	0
CustLogin	24.309	0	35.058	0.63827
CustCheckout	10.338	0	13.194	0
CustRegister	41.354	0	55.139	0.64972
AdmUpdate	2166.5	6.1436	1668.6	2.7692
AdmBackorder	2417.3	8.3916	1994.2	5.1146

(c) RADS Bookstore Model Results (Number of Customers = 150)

Scenario	Baseline Model		Optimized Model	
	Average Response Time (ms)	Average Miss Rate (%)	Average Response Time (ms)	Average Miss Rate (%)
CustBrowse	13.955	0	14.432	0
CustView	6.9421	0	7.3235	0
CustAdd	8.1431	0	8.29	0
CustRemove	8.1292	0	8.2893	0
CustLogin	28.675	0	35.936	0.61535
CustCheckout	12.644	0	16.106	0
CustRegister	45.851	0	55.921	0.63776
AdmUpdate	2351.2	7.93293	1622.7	2.3129
AdmBackorder	2794.1	11.845	1979.6	5.0221

(d) RADS Bookstore Model Results (Number of Customers = 200)

Scenario	Baseline Model		Optimized Model	
	Average Response Time (ms)	Average Miss Rate (%)	Average Response Time (ms)	Average Miss Rate (%)
CustBrowse	16.038	0	16.911	0
CustView	9.0201	0	9.7851	0
CustAdd	11.506	0	11.129	0
CustRemove	11.498	0	11.132	0
CustLogin	35.128	0	39.174	0.66128
CustCheckout	16.141	0	20.175	0
CustRegister	52.768	0	60.609	0.77711
AdmUpdate	2615.2	10.008	1690	2.7164
AdmBackorder	3241.3	15.593	2086.9	5.1115

(e) RADS Bookstore Model Results (Number of Customers = 250)

Scenario	Baseline Model		Optimized Model	
	Average Response Time (ms)	Average Miss Rate (%)	Average Response Time (ms)	Average Miss Rate (%)
CustBrowse	19.067	0	20.749	0
CustView	12.049	0	13.609	0
CustAdd	16.634	0	15.414	0
CustRemove	16.633	0	15.418	0
CustLogin	45.376	0.00017026	43.127	0.66045
CustCheckout	21.488	0	26.017	0
CustRegister	63.08	0.0016807	65.917	0.73239
AdmUpdate	3033	13.778	1684.4	2.8488
AdmBackorder	3968.3	22.023	2097.1	5.6549

(f) RADS Bookstore Model Results (Number of Customers = 300)

Scenario	Baseline Model		Optimized Model	
	Average Response Time (ms)	Average Miss Rate (%)	Average Response Time (ms)	Average Miss Rate (%)
CustBrowse	21.606	0	27.312	0.000010253
CustView	14.588	0	20.156	0.000017948
CustAdd	14.715	0	22.534	0.000019237
CustRemove	14.708	0	22.534	0
CustLogin	45.817	0.006392	48.176	0.66982
CustCheckout	18.834	0	34.847	0.000076878
CustRegister	51.392	0.063707	71.993	0.75517
AdmUpdate	3069.1	14.045	1668.7	2.809
AdmBackorder	4165.9	23.798	2143.1	5.97

(g) RADS Bookstore Model Results (Number of Customers = 350)

Figure 6.2 Response Times and Miss Rates of All Scenarios in Baseline Model and Optimized Model

Discuss of The Results (Figure 6.2)

The deadline requirements for the customer's scenarios are easy to meet. In every case, All the deadlines of customer's scenario are really met in both the baseline models and optimized models. The laxity factors of customer's scenarios range from 10-100. The easiness of meeting deadline requirements can be explained by the laxity factor, as defined in section 5.3.

However the laxity factors of two administrator's scenarios are around 3. According to the guideline reached in section 5.3, the deadline requirement for the administrator's scenarios might be a problem, by considering that the distribution in the reference is exponential instead of deterministic, as in the section 5.3. The response time and miss rate for the administrator's scenarios should be focused in this case study. Figure 6.3, Figure 6.4 summarizes the average miss rates of two administrator's scenarios in the baseline model and optimized model.

	Miss Rate For Scenario AdmBackorder (%)		Miss Rate For Scenario AdmUpdate (%)	
User Number	Baseline Model	Optimized Model	Baseline Model	Optimized Model
50	5.2072	4.4059	3.5864	2.9079
100	6.5274	5.0409	4.5876	2.8546
150	8.3916	5.1146	6.1436	2.7692
200	11.845	5.0221	7.93293	2.3129
250	15.593	5.1115	10.008	2.7164
300	22.023	5.6549	13.778	2.8488
350	25.796	5.97	21.6	2.809

Figure 6.3 Miss Rates of Baseline and Optimized Model for Scenarios AdmBackorder and AdmUpdate (Table)

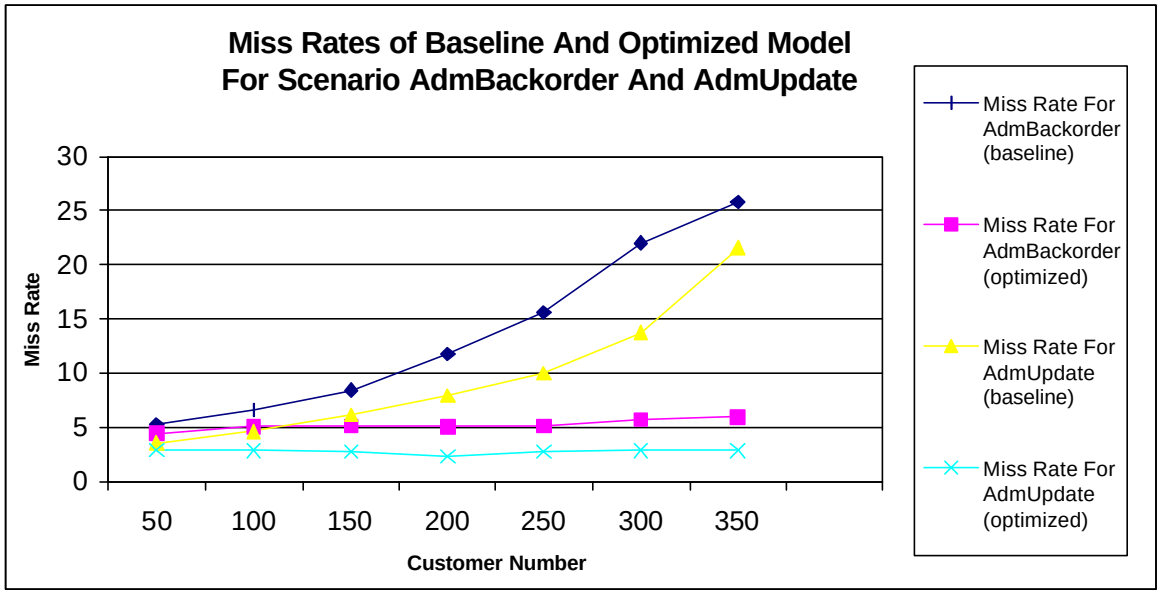


Figure 6.4 Miss Rates of Baseline and Optimized Model for Scenarios AdmBackorder and AdmUpdate (Diagram)

As we can see, the miss rates for both the administrator's scenario are improved in the optimized model especially when the number of customers increases. In the baseline model, the miss rates for the two administrator's scenarios increase rapidly when the number of customers increases. In the optimized model, the miss rates of the two

scenarios are kept almost the same. However the optimized model couldn't meet the deadline requirements when the number of customers is more than 50.

In order to meet the deadline requirements, the system must be modified to improve the response time of the administrator's scenarios.

6.2 Modified RADS Bookstore System: Task Splitting

In the previous section, the administrator scenarios and customer scenarios share the tasks in both the processor *BookstoreProc* and *DatabaseProc*. Because of the limited number of threads in the tasks, the administrator's execution may be blocked by the customer's execution. The blocking increases the response times of the administrator's scenarios and cause the system to miss the deadline requirements. In order to decrease the interference, for the *InventoryMgr*, *CustomerAccount, BackorderMgr* task in processor *BookstoreProc* and *Database*, *CutomerDB* task in *DatabaseProc*, a single thread copy for each of those tasks is created for the administrator. The simplified LQN model of the modified system is indicated in Figure 6.5. The simulation results (response times and miss rates for all the scenarios) are listed in Figure 6.6:

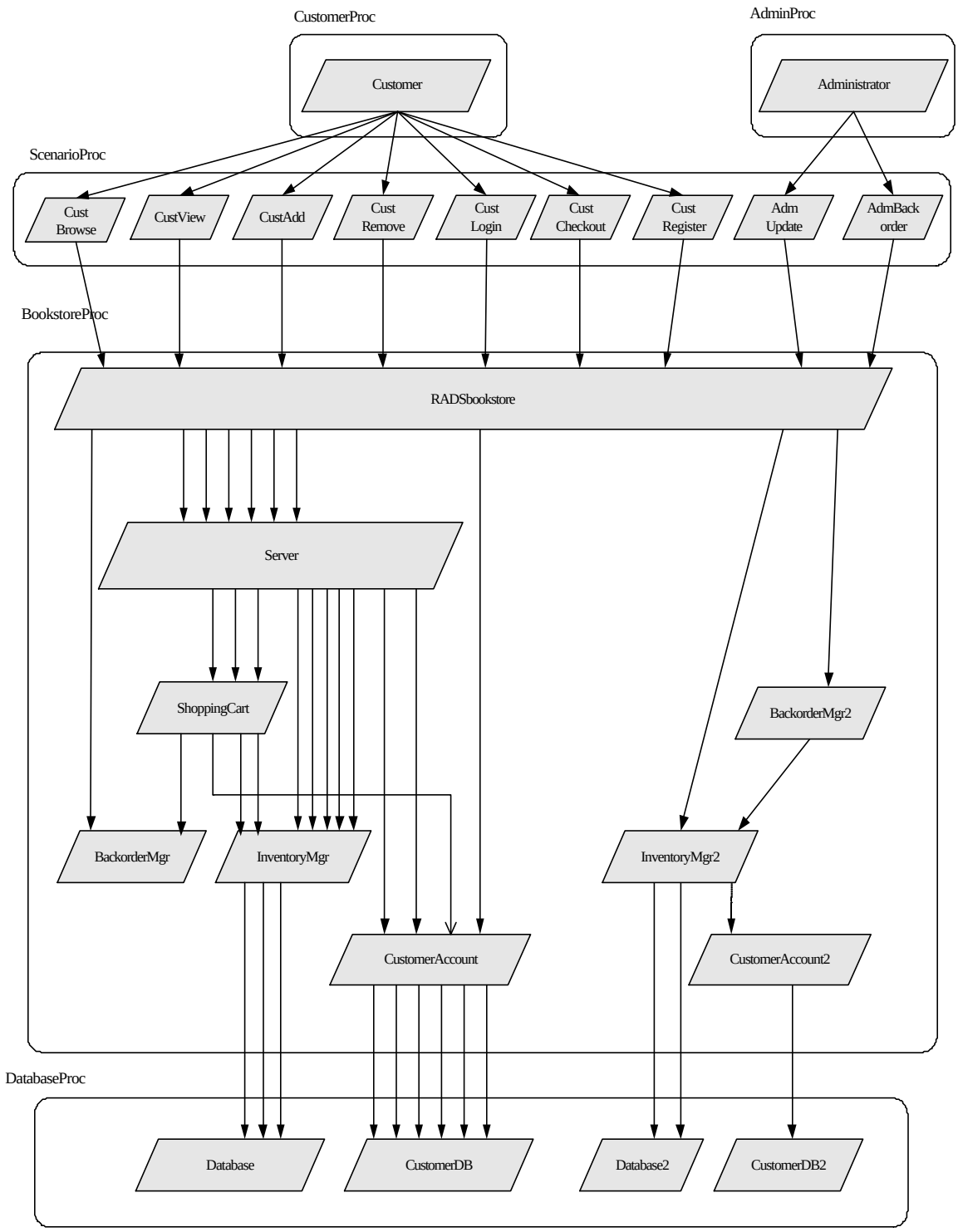


Figure 6.5 Simplified LQN Model of RADS Bookstore (After Splitting)

Scenario	Original Optimized Model (Without Task Splitting)		New Optimized Model (With Task Splitting)	
	Average Response Time (ms)	Average Miss Rate (%)	Average Response Time (ms)	Average Miss Rate (%)
CustBrowse	10.575	0	10.642	0
CustView	3.5693	0	3.6014	0
CustAdd	3.5959	0	3.7933	0
CustRemove	3.5998	0	3.792	0
CustLogin	18.809	0	32.051	0.61437
CustCheckout	7.7496	0	9.3199	0
CustRegister	35.327	0	49.987	0.63392
AdmUpdate	1827.7	3.7402	1666	2.7156
AdmBackorder	2021.1	5.6261	1889.3	4.311

(a) RADS Bookstore Model Results (Number of Customers = 50)

Scenario	Original Optimized Model (Without Task Splitting)		New Optimized Model (With Task Splitting)	
	Average Response Time (ms)	Average Miss Rate (%)	Average Response Time (ms)	Average Miss Rate (%)
CustBrowse	11.376	0	11.517	0
CustView	4.3702	0	4.4528	0
CustAdd	4.5136	0	4.8616	0
CustRemove	4.5022	0	4.8674	0
CustLogin	21.198	0	33.733	0.61521
CustCheckout	8.7843	0	11.014	0
CustRegister	37.994	0	52.696	0.65362
AdmUpdate	1999.8	4.8459	1662.1	2.8096
AdmBackorder	2139.2	6.1576	1902.2	4.4374

(b) RADS Bookstore Model Results (Number of Customers = 100)

Scenario	Original Optimized Model (Without Task Splitting)		New Optimized Model (With Task Splitting)	
	Average Response Time (ms)	Average Miss Rate (%)	Average Response Time (ms)	Average Miss Rate (%)
CustBrowse	12.472	0	12.731	0
CustView	5.4609	0	5.6421	0
CustAdd	5.9348	0	6.3048	0
CustRemove	5.9389	0	6.3093	0
CustLogin	24.346	0	36.2	0.65784
CustCheckout	10.35	0	13.191	0
CustRegister	41.068	0	54.611	0.60249
AdmUpdate	2211.3	6.87	1704.9	2.9029
AdmBackorder	2445.6	8.4844	1906.1	4.3949

(c) RADS Bookstore Model Results (Number of Customers = 150)

Scenario	Original Optimized Model (Without Task Splitting)		New Optimized Model (With Task Splitting)	
	Average Response Time (ms)	Average Miss Rate (%)	Average Response Time (ms)	Average Miss Rate (%)
CustBrowse	13.962	0	14.433	0
CustView	6.9459	0	7.3251	0
CustAdd	8.1403	0	8.291	0
CustRemove	8.155	0	8.2966	0
CustLogin	28.679	0	37.243	0.63424
CustCheckout	12.68	0	16.124	0
CustRegister	45.552	0	57.918	0.73044
AdmUpdate	2456.8	8.7315	1667.7	2.6265
AdmBackorder	2734.1	11.337	1943.6	4.3972

(d) RADS Bookstore Model Results (Number of Customers = 200)

Scenario	Original Optimized Model (Without Task Splitting)		New Optimized Model (With Task Splitting)	
	Average Response Time (ms)	Average Miss Rate (%)	Average Response Time (ms)	Average Miss Rate (%)
CustBrowse	16.037	0	16.918	0
CustView	9.0215	0	9.7905	0
CustAdd	11.505	0	11.137	0
CustRemove	11.524	0	11.129	0
CustLogin	35.149	0	40.169	0.65878
CustCheckout	16.171	0	20.168	0
CustRegister	52.501	0	62.239	0.77721
AdmUpdate	2797.4	11.584	1690.5	2.7736
AdmBackorder	3027.7	14.235	1923.1	4.7231

(e) RADS Bookstore Model Results (Number of Customers = 250)

Scenario	Original Optimized Model (Without Task Splitting)		New Optimized Model (With Task Splitting)	
	Average Response Time (ms)	Average Miss Rate (%)	Average Response Time (ms)	Average Miss Rate (%)
CustBrowse	18.798	0	20.763	0
CustView	11.784	0	13.623	0
CustAdd	15.932	0	15.424	0
CustRemove	15.924	0	15.423	0
CustLogin	44.492	0	43.424	0.66401
CustCheckout	20.834	0	26.008	0
CustRegister	61.286	0	67.221	0.85625
AdmUpdate	3146.3	14.69	1661.5	2.6023
AdmBackorder	3580.6	18.872	1929.6	4.4412

(f) RADS Bookstore Model Results (Number of Customers = 300)

Scenario	Original Optimized Model (Without Task Splitting)		New Optimized Model (With Task Splitting)	
	Average Response Time (ms)	Average Miss Rate (%)	Average Response Time (ms)	Average Miss Rate (%)
CustBrowse	23.897	0	27.362	0.000087758
CustView	16.888	0	20.198	0.00009912
CustAdd	24.594	0	22.587	0.00010252
CustRemove	24.575	0	22.592	0.000045555
CustLogin	62.471	0.006392	48.838	0.6783
CustCheckout	29.804	0	34.883	0.00013678
CustRegister	80.798	0.063707	73.932	0.96696
AdmUpdate	3906.8	21.6	1656.8	2.7533
AdmBackorder	4397.8	25.796	1975.7	4.4955

(g) RADS Bookstore Model Results (Number of Customers = 350)

Figure 6.6 Response Times and Miss Rates of All Scenarios in Both Optimized Model (with and without Task Splitting)

The miss rates for administrator's scenarios in the optimized model after task splitting is shown in Figure 6.7 and Figure 6.8, comparing to those in the optimized model before task splitting. Before task splitting, the couldn't be feasible using optimization. After task splitting, the model is initially feasible.

	Miss Rate For Scenario AdmBackorder (%)		Miss Rate For Scenario AdmUpdate (%)	
User Number	Before Splitting	After Splitting	Before Splitting	After Splitting
50	4.4059	4.311	2.9079	2.7156
100	5.0409	4.4374	2.8546	2.8096
150	5.1146	4.3949	2.7692	2.9029
200	5.0221	4.3972	2.3129	2.6265
250	5.1115	4.7231	2.7164	2.7736
300	5.6549	4.4412	2.8488	2.6023
350	5.97	4.4955	2.809	2.7533

Figure 6.7 Miss Rates of Optimized Model for Administrator's Scenarios Before and After Task Splitting (Table)

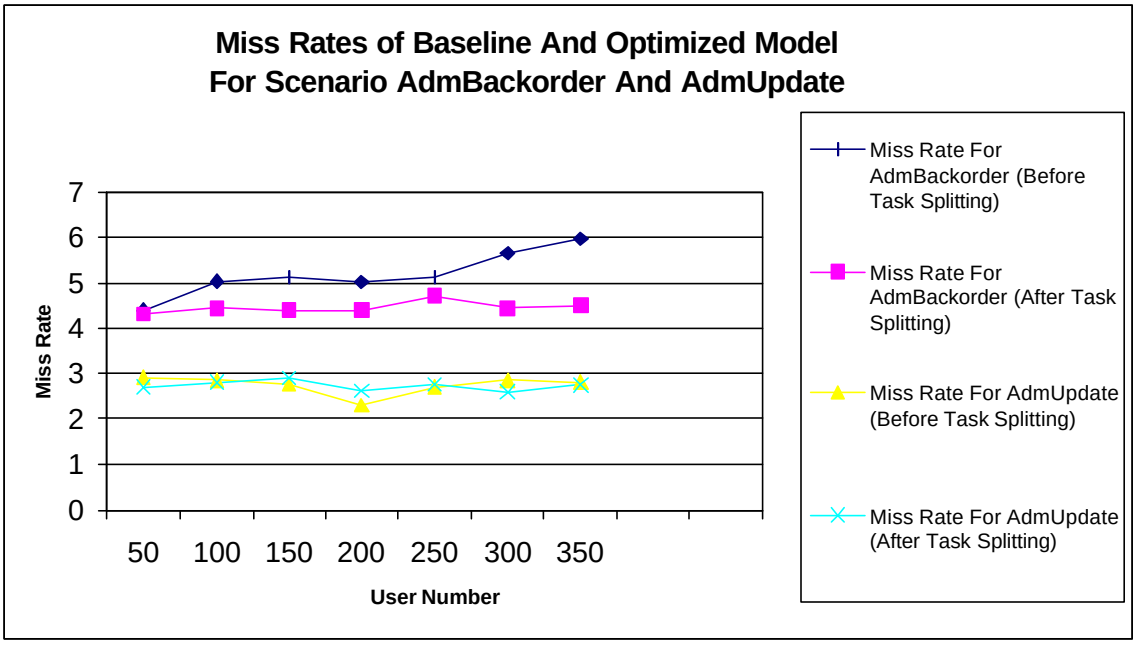


Figure 6.8 Miss Rates of Optimized Model for Administrator's Scenarios Before and After Task Splitting (Diagram)

Discuss of The Results with Task Splitting:

From Figure 6.7 and Figure 6.8, the miss rates for scenario *AdmUpdate* in both optimized models are almost the same. For scenario *AdmBackorder*, the miss rate in the original optimized model increases when the number of customers is increased. The deadline requirements can't be met all the time. However, the miss rate in the new optimized model is kept almost the same. This makes the RADS bookstore model meet all the deadline requirements even when the number of customers is increased to 350. The optimized model after splitting shows better scalability than the optimized model without task splitting.

In the modified RADS bookstore system, the administrator always has a thread reserved for him. Whenever the executions in the administrator's scenarios are ready to run, it might have a higher priority than that of the customers to preempt the customer's executions. The response times of the administrator's scenarios will not be delayed by the customers even when the number of customers is increased. Task splitting helps to improve response time of the task with higher priority.

Although task splitting is used in the previous work in [11], it is used in the optimization phase to split one entry from the candidate task at a time. For a complicated model like this case study, more than one task should be considered to be split and more than one entry should be split from a task. The entries related to the customer's scenarios should not be considered to split. The task splitting in the optimization phase is difficult to apply, whereas it is easy to do it in the initialization phase.

This case study shows that the optimization approach is also suitable for practical soft real-time systems with stochastic execution demands. Task splitting in the initialization phase could improve the performance of higher priority tasks easily.

CHAPTER 7: CONCLUSIONS

7.1 Summary

This thesis is concerned with re-implementing and improving an optimizer for distributed systems with deadlines, and studying systems with soft deadlines.

The re-implemented optimizer is capable of analyzing and improving an existing system model described by LQN modeling language while its predecessor required input in a now unavailable scenario language.

The five modifications to the optimization, that are described in chapter 4, are in the details of determining the starting point, and the improvement steps. They are:

- Add the communication cost handling during the procedure of optimization.
- Extend the Proportional-Deadline-Monotonic algorithm to more general systems, taking account of the number of threads of the task.
- Uses a modified function to estimate the scenario criticality of the LQN model.
- Use the “distinct fixed priority” approach to setting task priorities, and modify the task metric for priority adjustment
- Simplify the task reshaping and modify the metric for task reallocation

The evaluation in section 5.1 and 5.2 shows the effectiveness and improvement of the modifications.

New evaluations were performed on systems with soft deadlines and stochastic execution demands in section 5.3. A guideline for laxity values, coefficient of variation and utilization values was determined based on 12000 experiments with randomly generated parameters, and is given in Figure 5.10 and Figure 5.11. The guideline can be used to assess the feasibility of a system with soft deadlines.

The case study in chapter 6 shows the effectiveness of the optimization approach on a practical application.

7.2 Contributions

The optimization approach provided in this thesis is first proposed by other researchers. It has been improved and modified in several aspects. The contributions of this thesis include:

- This thesis applies the optimization approach to more general distributed real-time systems which could include soft deadlines, stochastic execution demands and different numbers of threads in the tasks. An algorithm is proposed to handle communication costs in LQN model automatically during optimization (this was not clear in the previous algorithm).
- Some algorithms and metrics used in optimization are improved and modified. The fixed priority approach which is required in hard real-time systems replaces the priority approach which allows equal priorities in the optimization approach. The priority adjustment algorithm is improved to have a higher success ratio (section 5.1.1 and 5.1.2) and the Proportional-Deadline-Monotonic algorithm is extended to be suitable for systems with different number of threads. The metrics for priority adjustment, task reallocation and model estimation are modified for certain reasons. The modified algorithms were more successful than the pre-existing ones, on the test cases in section 5.1.
- The studies on systems with soft deadlines and stochastic demands, provide an guideline for feasibility according the laxity factor, utilization and coefficient of variation. It is hypothesized that these guidelines can be used before attempting an optimization, to judge the difficulty of the problem and likelihood of the success.
- A case study on an e-commerce application and the exploration of task splitting in the case study show that the optimization approach is suitable for the practical soft real-time systems.
- The research re-implements the optimization approach to accept an LQN model as a starting point. It makes the optimization approach applicable to ordinary LQN models.

7.3 Future Work

Although the optimization approach is shown effective in this thesis, it will be more practical and valuable if it is improved in the following direction:

- The simplification of LQN model. If the LQN model is very large (e.g. automatically generated LQN models), the simulation time will be very long to get accurate results. It will be valuable to provide algorithms to simplify the LQN model without losing the fundamental characteristic.
- A better metric for task reallocation. Two factors are considered for task reallocation: the communication overheard and CPU utilization. In this thesis, both of them have the same weight. A better metric considering different weights or adding other factors may improve the efficiency of the optimization approach.
- A better initial task allocation. It's helpful for efficiency.
- Automatic task splitting in initialization phase. The case study indicates that the task splitting in the initialization phase is effective. An automation of task splitting according to the scenarios or other criteria is beneficial to the optimizer.

- **References:**

[1] Alia K. Atlas, Azer Bestavros, “Statistical Rate Monotonic Scheduling”, IEEE 19th Real-Time System Symposium, RTSS’98. Madrid, Spain. December 1998

[2] N.C. Audsley, “Optimal priority assignment and feasibility of static priority tasks with arbitrary start times”, Department of Computer Science, Report no. YCS 164, University of York, England, Dec. 1991.

[3] R. Bettati, “End-to-end scheduling to meet deadlines in distributed systems”, Ph.D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA, March 1994.

[4] S.H. Bokhari. “Assignment problems in Parallel and Distributed Computing”. Kluwer Academic Publishers, 1987.

[5] A. Buchard, J. Liebeherr, Y. Oh, and S.H. Son, “Assigning real-time tasks to homogeneous multiprocessor systems”, Tec. Rep. CS-94-01, University of Virginia, Jan. 1994.

[6] E.G. Coffman, M.R. Garey and D.S. Johnson, “An application of bin-packing to multiprocessor scheduling”, SIAM J. Comput., vol. 7, pp. 1-17, Feb. 1978.

[7] S. Davari and S. Dhall, “An on line algorithm for real-time task allocation”, in Proc. of the IEEE Real-Time System Symposium, pp. 194-200, 1986.

[8] S.K. Dhall and C.L. Liu, “On a real-time scheduling problem”, in Operations Research, vol. 26(1), pp. 127-140, Feb. 1978

[9] N. J. Dingle, P. G. Harrison, W. J. Knottenbelt, “Response time densities in Gneralised Stochastic Petri net models”, In Proceeding of the Third Workshop on Software and Performance, Rome, July, 2002

[10] H.M. El-Sayed, D. Cameron, and C.M. Woodside, "Automated performance modeling from scenarios and SDL designs of distributed systems", In Proc. of the Int. Symposium on Software Engineering for Parallel and Distributed Systems (PDSE'98), Kyoto, April 1998

[11] Hesham M. El-Sayed “A Framework For Automated Performance Engineering of Distributed Real-Time Systems” Thesis of Doctor of Philosophy. Ottawa-Carleton

Institute for Electrical Engineering, Faculty of Engineering, Department of Systems and Computer Engineering Carleton University 1999

[12] Reza Etemadi, “End-to-end scheduling in hard real-time multiprocessor systems”, Ph.D. Thesis, Dept. of Systems and Computer Engineering, Carleton University, 1996.

[13] G. Franks, A. Hubbard, S. Majumdar, D. Petriu, J. Rolia, and C.M. Woodside, “A toolset for performance engineering and software design of client-server systems”, Performance Evaluation, 24 (1-2):117-135, November 1995.

[14] G. Franks, S. Majumdar, J. Neilson, D. Petriu, J. Rolia, and M. Woodside, “Performance analysis of distributed server systems”, Proceedings of The Sixth International Conference on Software Quality

[15] G. Franks, “Performance analysis of distributed server systems”, Ph.D. thesis proposal, Dept. of Systems and Comp. Eng., Carleton University, Feb. 1997.

[16] J.J.G. Garcia and M. Gonzalez Harbour, “Optimized priority assignment for task and messages in distributed hard real-time systems”, Proc. of the IEEE Workshop on Parallel and Distributed Real-Time Systems, California, pp. 124-132, April 1995.

[17] Mark K. Gardner, “Probabilistic Analysis and Scheduling of Critical Soft Real-Time Systems”, Thesis of Doctor of Philosophy, in Computer Science in the Graduate College of the University of Illinois at Urbana-Champaign, 1999

[18] C.J. Hou and K.G. Shin, “Allocation of periodic task modules with precedence and dead-line constraints in distributed real-time systems”, in Proc. of the Real Time system symposium, pp. 146-155, 1992

[19] B. Kao and H. Garcia-Molina, “Deadline assignment in a distributed soft real-time system”, Proc. of the IEEE International Conference on Distributed Computing Systems, Pittsburgh, Pennsylvania, pp. 428-437, May 1993.

[20] B. Kao and H. Garcia-Molina, “Subtask deadline assignment for complex distributed soft real-time systems”, Proc. of the IEEE International Conference on Distributed Computing Systems, Poznan, Poland, pp. 172-181, June 1994.

[21] J.P. Lehoczky, “Fixed priority scheduling of periodic task sets with arbitrary deadlines”, Proceedings IEEE Real-Time Systems Symposium, pp. 201-209, 1990.

[22] Lehoczky, J.; Sha, L.; Ding, Y, "The rate monotonic scheduling algorithm: exact characterization and average case behavior", Real Time Systems Symposium, 1989., Proceedings.

[23] J.Y.T. Leung, J. Whitehead, "On the complexity of fixed-priority scheduling of periodic real-time tasks", Performance Evaluation, 2, (4), pp. 237-250, Dec. 1982.

[24] C.L. Liu and J.W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment", in Journal of the Assoc. Computing Mach., vol. 20(1), pp. 46-61, 1973.

[25] Jane W. S. Liu, "Real-Time Systems", Prentice Hall, 2000

[26] D.T. Peng and K.G. Shin, "Static allocation of periodic tasks with precedence constraints in distributed real-time systems", In Proc. of the 9th Intl. Conf. On Distributed computing systems, pp. 190-198, 1989.

[27] Dorin Petriu, Murray Woodside, "Analysing Software Requirements Specifications for Performance", Report SCE-02-02, Dept of Systems and Computer Engineering, submitted for publication, Feb. 2002

[28] J. R. Rolia and Kenneth Sevcik, "The method of layers", IEEE Transactions on Software Engineering, Vol. 21, No. 8, pp. 689-700, 1995.

[29] Lui Sha, Raghunathan Rajkumar and John Lehoczky , "Priority Inheritance Protocols: An Approach to Real-Time Synchronization", IEEE Trans. on Computers, September 1990, vol. 39 no. 9, pp. 1175-1184

[30] C.U. Smith, "Performance Engineering of Software Systems", Addison-Wesley Publishing Co., New York, NY, 1990.

[31] Harold S. Stone, "Multiprocessor scheduling with the aid of network flow algorithm", IEEE Transactions on Software Engineering, 3(1):85-93, Jan. 1979

[32] 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, pp. 245-251, Apr. 1993.

[33] Jun Sun, "Fixed Periodic Scheduling of Periodic Tasks with End-To-End Deadlines", Ph.D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA, March 1996.

[34] T.–S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L. C. Wu, and J. W. S. Liu, ``Probabilistic Performance Guarantee for Real-Time Tasks with Varying Computation Times," Proceedings of IEEE Real-Time Technologies and Applications Symposium, pp. 164-173, Chicago, Illinois, May 1995.

[35] K. W. Tindell, “Extendible Approach For Analyzing Fixed Priority Hard Real-Time Tasks”,

[36] K.W. Tindel, A. Burns, and A.J. Wellings, “Allocating hard real-time tasks: an NP hard problem made easy”, Real-Time Systems, 4(2):145-165, June 1992.

[37] C.M. Woodside, “Throughput calculation for basic stochastic rendezvous networks”, Performance Evaluation, Vol. 9, No. 2, pp. 143-160, 1989.

[38] C.M. Woodside and G.M. Monforton, “Fast allocation of processes in distributed and parallel systems”, IEEE Transactions on Parallel and Distributed Systems, vol. 4, no. 2, Feb. 1993.

[39] C. M. Woodside, J. E. Neilson, D. C. Petriu, and S. Majumdar, “The stochastic rendezvous network model for performance of synchronous client server-like distributed software”, IEEE Transactions on Computers, 44(1):20-34, January 1995.

Appendix A: LQN Model File, Additional File and Optimization Output of Tutorial Example in Section 5.2, Scenario 1

A.1 LQN Model File:

```
 G "Tutorial Example In Thesis, Section 5.2, Scenario 1"
0.000001
100
1
0.9
-1

P 0
p EnvP1 f i
p EnvP2 f i
p CPU1 p
p CPU2 p
-1

T 0
t Env1 r env1 -1 EnvP1
t Env2 r env2 -1 EnvP2
t Resp1 n resp1 -1 EnvP1
t Resp2 n resp2 -1 EnvP2
t TaskA n en_a1 -1 CPU1 2 i
t TaskB n en_b1 -1 CPU1 1 i
t TaskC n en_c1 -1 CPU2 3 i
t TaskD n en_d1 -1 CPU2 2 i
t TaskE n en_e1 en_e2 -1 CPU2 1 i
-1

E 0
s env1 0 50 -1
c env1 0 0 -1
f env1 1 0 -1
M resp1 50 0 -1
z env1 resp1 1 0 -1
s resp1 0 0 -1
c resp1 0 0 -1
f resp1 1 0 -1
y resp1 en_a1 1 0 -1
```

```
A en_a1 a1
s en_b1 3 0 -1
c en_b1 0 0 -1
f en_b1 1 0 -1

s env2 0 50 -1
c env2 0 0 -1
f env2 1 0 -1
z env2 resp2 1 0 -1
s resp2 0 0 0 -1
c resp2 0 0 -1
f resp2 1 0 -1
M resp2 50 0 -1
y resp2 en_c1 1 0 0 -1
A en_c1 c1
A en_d1 d1
F en_d1 en_e2 1 -1
s en_e1 1 0 0 -1
c en_e1 0 0 0 -1
f en_e1 1 0 0 -1
s en_e2 1 0 0 -1
c en_e2 0 0 0 -1
f en_e2 1 0 0 -1

-1
A TaskA
s a1 35
c a1 0
f a1 1
s a2 0
c a2 0
f a2 1
z a1 en_b1 1
y a2 en_e1 3
:
a1 -> a2;
a2[en_a1]
-1

A TaskC
s c1 20
c c1 0
f c1 1
c c2 0
f c2 1
s c2 0
```

```
y c2 en_d1 1
:
c1 -> c2;
c2[en_c1]
-1

A TaskD
s d1 20
c d1 0
f d1 1
c d2 0
f d2 1
s d2 0
:
d1 -> d2;
d2[en_d1]
-1
```

A.2 Additional File:

```
#Tutorial Example In Thesis, Section 5.2, Scenario 1
Special Processors: (Processor1, Processor2, ...)
EnvP1, EnvP2
-1
Special Tasks: (Task1, Task2, ...)
Env1, Env2, Resp1, Resp2, TaskB
-1
Default Communication Cost:
0.7
-1
Default Network Delay:
0.0
-1
Default Message Numbers For Calls:
1
-1
Special Message Numbers For Calls: (Source Destination: Message Numbers)
en_d1 en_e2: 10
-1
Restricted allocation: (Task: Processor1 Processor2 ...)
-1
Forced coallocation: (Task Task)
-1
Forced uncoallocation: (Task Task)
-1
```

A.3 Optimization Output:

```
Communication Cost has been added to Tutorial/step0.lqn
Solver is running. Please wait....
Model Criticality: 1.0075312
Processor CPU2:
  Task TaskC's priority: 3  Metric: 0.0
  Task TaskD's priority: 2  Metric: 0.0
  Task TaskE's priority: 1  Metric: 0.6143236
Processor CPU1:
  Task TaskA's priority: 2  Metric: 0.0
  Task TaskB's priority: 1  Metric: 6.0828013
Candidate task: TaskE
------------------
Solver is running. Please wait....
Model Criticality: 0.0
Processor CPU2:
  Task TaskC's priority: 3  Metric: 0.0
  Task TaskD's priority: 1  Metric: 0.0
  Task TaskE's priority: 2  Metric: 0.0
Processor CPU1:
  Task TaskA's priority: 2  Metric: 0.0
  Task TaskB's priority: 1  Metric: 0.0
------------------
Optimization succeeded!
------------------
The optimized lqn file is: Tutorial_opt.lqn
------------------
Communication Cost has been removed from Tutorial/step1.lqn
The optimized priorities:
Processor CPU2:
```

Task TaskC: 3

Task TaskD: 1

Task TaskE: 2

Processor CPU1:

Task TaskA: 2

Task TaskB: 1

The best model metric is: 0.0 (in step 1)

Total steps: 1

The optimization is done....

Appendix B: LQN Model File, Additional File and Optimization Output of Tutorial Example in Section 5.2, Scenario 2

B.1 LQN Model File:

```
G "Tutorial Example In Thesis, Section 5.2, Scenario 2"
0.000001
100
1
0.9
-1

P 0
p EnvP1 f i
p EnvP2 f i
p CPU1 p
p CPU2 p
-1

T 0
t Env1 r env1 -1 EnvP1
t Env2 r env2 -1 EnvP2
t Resp1 n resp1 -1 EnvP1
t Resp2 n resp2 -1 EnvP2
t TaskA n en_a1 -1 CPU1 2 i
t TaskB n en_b1 -1 CPU1 1 i
t TaskC n en_c1 -1 CPU2 3 i
t TaskD n en_d1 -1 CPU2 2 i
t TaskE n en_e1 en_e2 -1 CPU2 1 i
-1

E 0
s env1 0 50 -1
c env1 0 0 -1
f env1 1 0 -1
M resp1 50 0 -1
z env1 resp1 1 0 -1
s resp1 0 0 -1
c resp1 0 0 -1
f resp1 1 0 -1
y resp1 en_a1 1 0 -1
```

```
A en_a1 a1
s en_b1 3 0 -1
c en_b1 0 0 -1
f en_b1 1 0 -1

s env2 0 50 -1
c env2 0 0 -1
f env2 1 0 -1
z env2 resp2 1 0 -1
s resp2 0 0 0 -1
c resp2 0 0 -1
f resp2 1 0 -1
M resp2 50 0 -1
y resp2 en_c1 1 0 0 -1
A en_c1 c1
A en_d1 d1
F en_d1 en_e2 1 -1
s en_e1 1 0 0 -1
c en_e1 0 0 0 -1
f en_e1 1 0 0 -1
s en_e2 1 0 0 -1
c en_e2 0 0 0 -1
f en_e2 1 0 0 -1
-1

A TaskA
s a1 35
c a1 0
f a1 1
s a2 0
c a2 0
f a2 1
z a1 en_b1 1
y a2 en_e1 3
:
a1 -> a2;
a2[en_a1]
-1

A TaskC
s c1 20
c c1 0
f c1 1
c c2 0
f c2 1
s c2 0
```

```
y c2 en_d1 1
:
c1 -> c2;
c2[en_c1]
-1

A TaskD
s d1 20
c d1 0
f d1 1
c d2 0
f d2 1
s d2 0
:
d1 -> d2;
d2[en_d1]
-1
```

B.2 Additional File:

```
#Tutorial Example In Thesis, Section 5.2, Scenario 2
Special Processors: (Processor1, Processor2, ...)
EnvP1, EnvP2
-1
Special Tasks: (Task1, Task2, ...)
Env1, Env2, Resp1, Resp2, TaskB
-1
Default Communication Cost:
0.01
-1
Default Network Delay:
1.7
-1
Default Message Numbers For Calls:
1
-1
Special Message Numbers For Calls: (Source Destination: Message Numbers)
en_d1 en_e2: 10
-1
Restricted allocation: (Task: Processor1 Processor2 ...)
-1
Forced coallocation: (Task Task)
-1
Forced uncoallocation: (Task Task)
-1
```

B.3 Optimization Output:

```
Communication Cost has been added to Tutorial-1/step0.lqn
Solver is running. Please wait....
Model Criticality: 3269017.2
Processor CPU1:
  Task TaskA's priority: 2  Metric: 0.0
  Task TaskB's priority: 1  Metric: 2.8616056E7
Processor CPU2:
  Task TaskC's priority: 3  Metric: 0.0
  Task TaskD's priority: 2  Metric: 0.0
  Task TaskE's priority: 1  Metric: 2.5055502E7
Candidate task: TaskE
-------------------
Solver is running. Please wait....
Model Criticality: 3269017.2
Processor CPU1:
  Task TaskA's priority: 2  Metric: 0.0
  Task TaskB's priority: 1  Metric: 2.0545994E7
Processor CPU2:
  Task TaskC's priority: 3  Metric: 0.0
  Task TaskD's priority: 1  Metric: 0.0
  Task TaskE's priority: 2  Metric: 1826936.6
Candidate task: TaskE
-------------------
Solver is running. Please wait....
Model Criticality: 3269017.2
Processor CPU1:
  Task TaskA's priority: 2  Metric: 0.0
  Task TaskB's priority: 1  Metric: 1.9736186E7
Processor CPU2:
```

Task TaskC's priority: 2 Metric: 0.0

Task TaskD's priority: 1 Metric: 0.0

Task TaskE's priority: 3 Metric: 19959.766

Candidate task: TaskE

Task TaskE has already the highest priority.

The model needs reallocating.

The up-to-now best model (Tutorial-1/step0) is restored.

The best model metric: 3269017.2

Candidate task: TaskE

Communication Cost has been removed from Tutorial-1/step3.lqn

Task TaskE is reallocated to CPU1 Priority: 3

Communication Cost has been added to Tutorial-1/step3.lqn

Solver is running. Please wait....

Model Criticality: 0.0

Processor CPU1:

Task TaskA's priority: 2 Metric: 0.0

Task TaskB's priority: 1 Metric: 0.0

Task TaskE's priority: 3 Metric: 0.0

Processor CPU2:

Task TaskC's priority: 2 Metric: 0.0

Task TaskD's priority: 1 Metric: 0.0

Optimization succeeded!

The optimized lqn file is: Tutorial-1_opt.lqn

Communication Cost has been removed from Tutorial-1/step3.lqn

The optimized priorities:

Processor CPU1:

Task TaskA: 2

Task TaskB: 1

Task TaskE: 3

Processor CPU2:

Task TaskC: 2

Task TaskD: 1

The best model metric is: 0.0 (in step 3)

Total steps: 3

The optimization is done....

Appendix C: Experiment Results for Statistical Models in Section 5.3

Laxity: 1.9			
Utilization	Number of Total Cases	Number of Successful Cases	Success ratio
0.4	50	47	0.94
0.5	50	31	0.62
0.6	50	15	0.3
0.7	50	10	0.2
0.8	50	1	0.02
Laxity: 2.1			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	40	0.8
0.8	50	21	0.42
Laxity: 2.3			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	50	1
0.8	50	45	0.9
Laxity: 2.5			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	50	1
0.8	50	50	1
Laxity: 2.7			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	50	1
0.8	50	50	1
Laxity: 3.0			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	50	1
0.8	50	50	1
Laxity: 3.5			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	50	1

0.8	50	50	1
Laxity: 4.0			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	50	1
0.8	50	50	1
Laxity: 4.5			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	50	1
0.8	50	50	1
Laxity: 5.0			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	50	1
0.8	50	50	1
Laxity: 5.5			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	50	1
0.8	50	50	1
Laxity: 6.0			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	50	1
0.8	50	50	1

Figure C.1 Statistical Model Results (CV = 0.0)

Laxity: 1.9			
Utilization	Number of Total Cases	Number of Successful Cases	Success ratio
0.4	50	33	0.66
0.5	50	13	0.26
0.6	50	1	0.02
0.7	50	0	0
0.8	50	0	0
Laxity: 2.1			
0.4	50	50	1
0.5	50	49	0.98
0.6	50	40	0.8
0.7	50	15	0.3
0.8	50	0	0
Laxity: 2.3			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	50	1
0.8	50	23	0.46
Laxity: 2.5			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	49	0.98
0.8	50	44	0.88
Laxity: 2.7			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	50	1
0.8	50	50	1
Laxity: 3.0			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	50	1
0.8	50	50	1
Laxity: 3.5			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	50	1
0.8	50	50	1
Laxity: 4.0			

0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	50	1
0.8	50	50	1
Laxity: 4.5			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	50	1
0.8	50	50	1
Laxity: 5.0			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	50	1
0.8	50	50	1
Laxity: 5.5			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	50	1
0.8	50	50	1
Laxity: 6.0			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	50	1
0.8	50	50	1

Figure C.2 Statistical Model Results (CV = 0.1)

Laxity: 1.9			
Utilization	Number of Total Cases	Number of Successful Cases	Success ratio
0.4	50	0	0
0.5	50	0	0
0.6	50	0	0
0.7	50	0	0
0.8	50	0	0
Laxity: 2.1			
0.4	50	3	0.06
0.5	50	0	0
0.6	50	0	0
0.7	50	0	0
0.8	50	0	0
Laxity: 2.3			
0.4	50	45	0.9
0.5	50	17	0.34
0.6	50	1	0.02
0.7	50	0	0
0.8	50	0	0
Laxity: 2.5			
0.4	50	50	1
0.5	50	49	0.98
0.6	50	33	0.66
0.7	50	2	0.04
0.8	50	0	0
Laxity: 2.7			
0.4	50	50	1
0.5	50	50	1
0.6	50	48	0.96
0.7	50	20	0.4
0.8	50	0	0
Laxity: 3.0			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	47	0.94
0.8	50	1	0.02
Laxity: 3.5			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	50	1
0.8	50	40	0.8
Laxity: 4.0			
0.4	50	50	1
0.5	50	50	1

0.6	50	50	1
0.7	50	50	1
0.8	50	48	0.96
Laxity: 4.5			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	50	1
0.8	50	50	1
Laxity: 5.0			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	50	1
0.8	50	50	1
Laxity: 5.5			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	50	1
0.8	50	50	1
Laxity: 6.0			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	50	1
0.8	50	50	1

Figure C.3 Statistical Model Results (CV = 0.5)

Laxity: 1.9			
Utilization	Number of Total Cases	Number of Successful Cases	Success ratio
0.4	50	0	0
0.5	50	0	0
0.6	50	0	0
0.7	50	0	0
0.8	50	0	0
Laxity: 2.1			
0.4	50	0	0
0.5	50	0	0
0.6	50	0	0
0.7	50	0	0
0.8	50	0	0
Laxity: 2.3			
0.4	50	0	0
0.5	50	0	0
0.6	50	0	0
0.7	50	0	0
0.8	50	0	0
Laxity: 2.5			
0.4	50	7	0.14
0.5	50	0	0
0.6	50	0	0
0.7	50	0	0
0.8	50	0	0
Laxity: 2.7			
0.4	50	47	0.94
0.5	50	16	0.32
0.6	50	0	0
0.7	50	0	0
0.8	50	0	0
Laxity: 3.0			
0.4	50	50	1
0.5	50	50	1
0.6	50	15	0.3
0.7	50	0	0
0.8	50	0	0
Laxity: 3.5			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	19	0.38
0.8	50	0	0
Laxity: 4.0			
0.4	50	50	1
0.5	50	50	1

0.6	50	50	1
0.7	50	49	0.98
0.8	50	0	0
Laxity: 4.5			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	50	1
0.8	50	8	0.16
Laxity: 5.0			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	50	1
0.8	50	39	0.78
Laxity: 5.5			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	50	1
0.8	50	48	0.96
Laxity: 6.0			
0.4	50	50	1
0.5	50	50	1
0.6	50	50	1
0.7	50	50	1
0.8	50	50	1

Figure C.4 Statistical Model Results (CV = 1.0)