The Pennsylvania State University

The Graduate School

PERFORMANCE MODELING AND RESOURCE ALLOCATION FOR ADAPTIVE AGENT-BASED SYSTEMS

A Thesis in
Industrial Engineering
by
Shanmuga-Nathan Gnanasambandam

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

December 2007

The thesis of Shanmuga-Nathan Gnanasambandam was reviewed and approved* by the following:

Soundar R.T. Kumara
Allen E., and Allen, M., Pearce Professor of Industrial Engineering
Thesis Adviser
Chair of Committee

M. Jeya Chandra
Professor of Industrial Engineering

Tao Yao
Assistant Professor of Industrial Engineering

Jun Shu
Assistant Professor of Supply Chain
and Information Systems

Natarajan Gautam
Associate Professor of Industrial Engineering
Texas A&M, College Station
Special Member

M. Jeya Chandra
Professor of Industrial Engineering
Acting Head of the Harold and Inge Marcus
Department of Industrial and Manufacturing
Engineering

*Signatures are on file in the Graduate School.

Abstract

Applications are increasingly becoming networks of individual application components spread over an infrastructure of physical resources (servers and computing entities). Such distributed agent-based applications are not only prevalent in military domains but also in commercial domains such as data centers. To assure the survivability of a distributed application, repeatedly estimating what sort of multi-dimensional guarantees (such as response time or availability) can be made as a function of the offered load and environmental conditions is paramount. This thesis studies how such guarantees can be computed and negotiated in a distributed MAS that is situated on an unreliable infrastructure.

Firstly, this work concentrates on identifying relevant service-level attributes that can be estimated for the failure-prone infrastructure. Queueing models with single and multi-class traffic are studied in scenarios with breakdown, repair and catastrophic failure, and are utilized for the estimation of performance and reliability attributes in steady state. These analytical results serve as internal models for the agents which aid them in evaluating the quality of service that can be attained given the environmental conditions (stresses and information load). Secondly, two resource allocation mechanisms are studied that are used to assign the agents to the nodes forming the infrastructure which are constrained by capacity, and are non-homogeneous in terms of their capability and the stresses experienced. Since the assignment problem is in general NP-hard, a few allocation heuristics with varying roles on the part of the agents and the infrastructure are proposed. Finally, in order to use resources efficiently, a usage price is computed based on the applications demand for the service-level attributes (which are the information goods sold by the infrastructure).

The main contributions of this work are the analytical solutions for the queueing models which include multi-class traffic, service disruption (both temporary and catastrophic failure) and non-preemptive priority scheduling. The analytical models pave the way for rapid negotiation between the MAS and the infrastructure as opposed to relatively slower simulation models. The analytical solutions are applicable to other domains including web-servers and hosting. The other contribution is the study of decentralized mechanisms that aid in the negotiation of quantitative service-level agreements in multi-agent systems and, in general, service-oriented architectures. While negotiation protocols and methods are widely studied, automatic negotiation using internal models are novel to this work, especially to distributed MAS.

Contents

List of Figures

List of Tables

Acknowledgments

I am deeply indebted to my adviser, Professor Soundar Kumara for his guidance and support during my stay at Penn State. Working with him on this dissertation has been a rewarding experience, and I am very grateful for his flexibility and co-operation during the course of this research. I thank Professor Kumara for including me in the DARPA UltraLog team and introducing me to several research topics in advanced information systems and logistics.

I am very thankful to Dr.Gautam for introducing me to the field of queueing and mathematical modeling of agent-based systems as well as for being a constant source of inspiration throughout my graduate school years.

I would like to thank Professor Jeya Chandra and Dr. Tao Yao for serving on my committee and helping me overcome several stumbling blocks during my thesis. I have greatly benefited from the discussions I have had with Professor Chandra and Professor Yao.

I would like to thank Dr. Jun Shu for serving on my committee, his insightful comments and helping me with aspects of my thesis relating to game theory.

I would like to thank Dr. Naveen Sharma for guiding me during my internship in the summer of 2005 and for continously providing encouragement afterwords.

I would like to thank Dr. Hari Srihari for advising me to pursue my Ph.D. and in particular at Penn State.

I would like to thank DARPA for their generous support and the UltraLog grant (MDA972-1-1-0038) through which I was supported. I owe thanks to Dr. Mark Greaves and Mr. Marshall Brinn who have helped me in the initial stages of problem formulation. I would like to thank Dr. Vikram Manikonda and Dr. Wilbur Peng for helping me understand COUGAAR and other software aspects of agent-based computing.

My excellent labmates are wonderful friends and have always helped me and kept me exposed to various exciting research problems.

I am deeply indeted to my wife for her love, support, enthusiasm and advice. She has helped me in innumerable ways during my thesis including coding and typesetting. She has been a constant source of motivation for me and we have had several interesting technical discussions. I thank her for her utmost patience and trust.

None of this would have been possible without the constant love and support of my parents and parents-in-law. I will always be indebted to them. My brother has always been supportive and helpful in all my endeavors and I thank him very much for his love and affection.

To my wife

Chapter 1

Introduction

Consider this example from everyday life. Imagine a personal assistant software application such as the ones on hand-held devices and PDAs. These applications are increasingly becoming more sophisticated with software features such as web-services, location- and context- aware functionalities and highly intuitive interfaces. In fact, with continuing technological advances, applications started on a desktop can be continued on a hand-held and subsequently finished back on the desktop, with barely noticeable transitions. These highly sophisticated software applications are possible because of a gamut of proliferating information and content sources (search engines, semantic knowledge bases, mobile databases etc), and highly self- and peer-aware software.

Let us assume that this trend is going to continue and the demand for such software is going to increase. Then it is natural to assume that there are going to be more such software components on our PDAs, laptops and desktops. To scale better, the algorithms for accessing and retrieving data and information from all the sources, as well as planning (scheduling calendar appointments, looking for travel deals, booking tickets) from them have to improve constantly. The other important consideration is that these applications have to be cognizant of their resources' capabilities and the quality of service (QoS) that can be delivered when transitioning from device to device and context to context. Why is that an important consideration? It is because devices are extremely non-homogeneous and contexts are dissimilar. In other words, QoS sensitive resource allocation is extremely important for context-sensitive mobile applications - otherwise these applications will be unusable inspite of their usefulness.

Matters are further complicated if the application considered is *mission-critical* such as those in military logistics. Being situated (in a battle-field environment/context), embedded with increased autonomy (to take decisions locally) and mobility, the application components tend resemble agents more closely than traditional software components (see [1, 2] for agent definitions). The number of agents are higher, the environment is harsher and ridden with catastrophes, and the performance requirements are more stringent. The actual applications may be distributed logistics planning and mission control instead of, perhaps more forgiving, personal assistance software. Failure, such as the distributed agent network not being able to plan or perform effectively, is usually met with very high costs.

Performance analysis is the domain for this thesis and the agent-based military logistics application is the one under consideration. ***The problem we address is that of modeling the performance and reliability interactions of these software agents in harsh environments, and subsequently allocating them to computing resources (such as those in military transportation hardware) in an autonomous fashion.*** The solution must consider the QoS sensitivity of the agents in different contexts (with varying battle-field stresses) and allocate them in such a way that the overall application experiences the desired QoS. The goal, therefore, is to estimate the performance of the agents using queueing models and subsequently use the predictions for QoS-sensitive resource allocation.

1.1 Distributed Application And its Multi-Scale Nature

We refer to the agents in the logistics application as the *Multi Agent System* (MAS) or the *application*. The hardware and middle-ware is referred to as *infrastructure* (providing computing power). Figure 1.1 is a depiction of our domain i.e. military logistics. Groups of agents are invoked on military vehicles which are engaged in battle-field operations in a remote location. Being in a harsh environment, the computing infrastructure in these vehicles experiences heavy informational load and catastrophic failure. The agents present on this infrastructure form the backbone of a multi-agent application involved in logistics planning. UltraLog is one such application [3]. The challenge here is that, inspite of battle-field conditions, the MAS must function effectively.

1.1.1 Abstraction

Figure 1.2 presents a multi-scale abstracted view of our domain. The application (top layer) is a network of individual agents spread over an infrastructure of physical resources (CPU, network bandwidth). The agents send tasks of various kinds (or classes) to the infrastructure through their respective *nodes*. The infrastructure consisting of a logical layer (COUGAAR nodes) and physical layer process the tasks. Catastrophic failure occurs in the physical layer. When this occurs, nodes can be re-

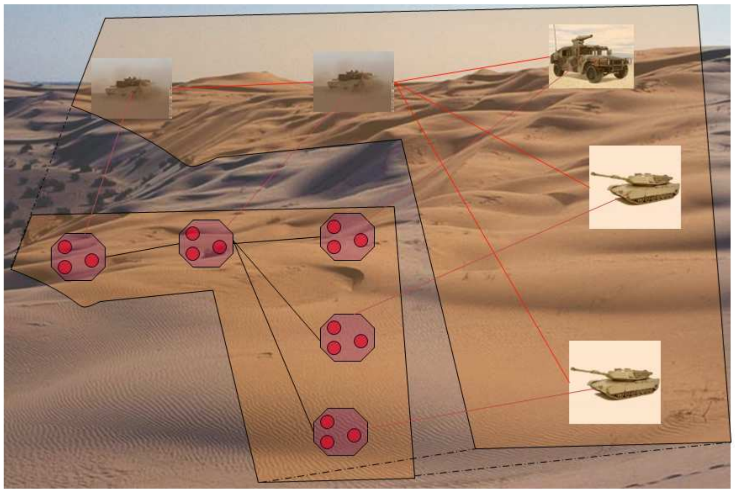
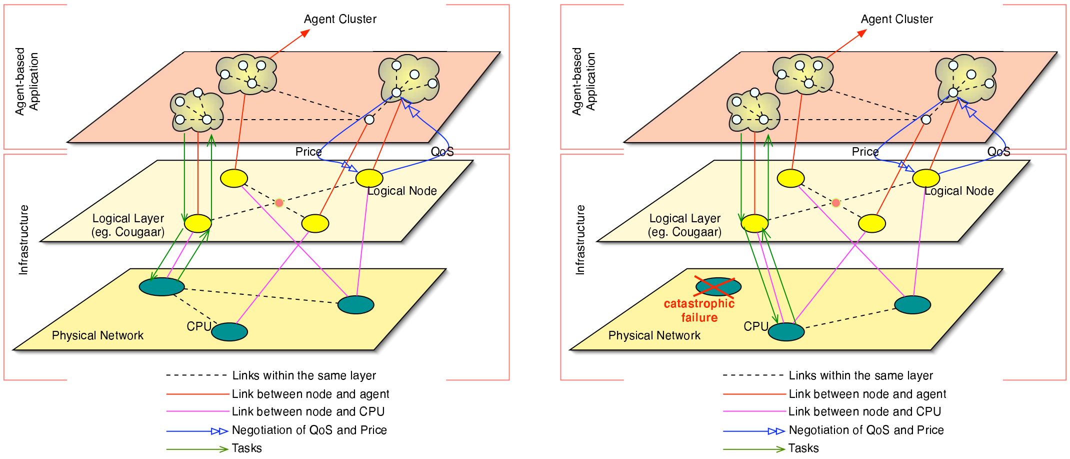


Figure 1.1: An agent-based application and its infrastructure

hydrated (reinvoked from existing data snap-shots) on other resources. Other failures are usually repairable through reconfiguration in software. We consider such failures in the logical layer. The agents experience a certain degree of QoS, performance and reliability, while the tasks are processed. The variation is because of resource availability at each node and different stress levels. Since nodes keep breaking down, the agents prefer to request a certain repair rate (through software - alternate algorithms, priority scheduling etc) for functioning effectively. If the demand cannot be met, the agents will migrate to a different node. For the QoS experienced, agents are willing pay a monetary price. To assure the survivability of such a distributed application, repeatedly estimating what sort of multi-dimensional guarantees (such as performance or reliability) can be made as a function of the offered load and environmental conditions is paramount. This thesis studies how such guarantees can be negotiated and priced in a distributed MAS that is situated on an unreliable and


Agent Cluster
Agent-based Application
Price
QoS
Logical Node
Infrastructure
Logical Layer (eg. Cougaar)
Physical Network
CPU
Links within the same layer
Link between node and agent
Link between node and CPU
Negotiation of QoS and Price
Tasks
Agent Cluster
Agent-based Application
Price
QoS
Logical Node
Infrastructure
Logical Layer (eg. Cougaar)
catastrophic failure
Physical Network
CPU
Links within the same layer
Link between node and agent
Link between node and CPU
Negotiation of QoS and Price
Tasks

(a) Before catastrophic failure

(b) After catastrophic failure

Figure 1.2: Operational Layers forming the Application and the Infrastructure: Agents located on a logical node (i.e. middle-ware) send task packets to CPU which get processed and returned to the agents. In case the CPU breaks down, all tasks currently in the queue are lost but the agents can be moved and *rehydrated* on another logical node on a different CPU.

failure-prone infrastructure.

1.1.2 Research overview

Firstly, this work concentrates identifying relevant service level attributes that can be provided and estimated in this domain. Queueing models with single and multi-class traffic are studied in scenarios with breakdown, repair and catastrophic failure, and are utilized for the estimation of performance and reliability attributes. Secondly, a pricing scheme is studied that is used to sell the service-level attributes as information goods to an application that is willing to pay varying amounts to ensure it will survive. Pricing provides an incentive for the application to use the resources efficiently, a mechanism to control the congestion of the infrastructure and also a contract for the infrastructure to get compensated for the service it is providing. Thirdly, reaching a contract between the parties involved is studied as a multi-agent negotiation problem. The specific focus is on agents autonomously negotiating contracts using the QoS estimation methods developed in the first step.

1.1.3 Multi-Agent Application

A multi-agent application means that distributed agents are used as the backbone of an application. MASs are becoming preferred choices of large-scale application deployment in an increasing number of scenarios including supply networks, data-centers and sensor networks [4, 5, 6]. Due to the inherent distributed nature of these scenarios, agents are spread across non-homogeneous infrastructures each possessing varying degrees of performance and robustness (resistance to failure) capability. However, the MAS's performance and robustness needs are still high, thereby requiring the application to be survivable in dynamic and loss-prone situations. Often times, the entire multi-agent application (such as UltraLog [3] with thousands of agents) is so huge that they cannot be controlled centrally - hence the larger society of agents is subdivided into smaller *communities*. We present a prototypical multi-agent application in Chapter 3.

1.2 Research Objectives

The main research objective of the proposed work is to negotiate QoS contracts between a group of agents (viz. the application) and an infrastructure that charges the former a price to provide service. We divide this into two sub-problems. The first problem will investigate models that aid the prediction of expected quality of service. The second sub-problem will examine collaborative mechanisms that the application (the agents) and the infrastructure (the principal) engage in for achieving the allocation of the agents to the principal's nodes. We formally define the sub-problems and present a birds-eye view of the solution methodology.

1.2.1 Problem 1: Performance models for the MAS

Agents need a tool to predict the performance impact of various parameters on themselves and their peers. Once agents are allocated onto the infrastructure, agents are subject to the environmental conditions faced by that node. Since they are co-located with other agents, they are impacted by the load caused by their peers. Moreover, every agent is an adaptive program that requests support from infrastructure for repairing itself. In total, there are several parameters and interactions that need to taken into account to compute the expected steady state performance at a node.

In the first problem, we develop two models based on queueing theory. These are listed as sub-problems below.

1. **Sub-problem 1.1 - Single-class Queueing Model** M_j^1(Chapter 4)
    - To develop a single class model $M_j^1(\beta_j,\ \lambda_j,\ \phi_j)$ for node j containing i agents, where β_j is the repair rate provided by the infrastructure node j to the i agents, λ_j is the total rate at which requests from all the i agents arrive and ϕ_j is a collection of parameters that characterize the node and the environment, assumed not to be under the agents' control.
    - Using the model, certain quality of service metrics need to be analytically derived. The metrics of interest are response time (W_j), loss probability (LP_j) and availability (R_j).

2. **Sub-problem 1.2 - Multi-class Queueing Model** M_j^2(Chapter 5)

   - To develop a two-class model $M_j^2(\beta_j,\ \lambda_j^1,\ \lambda_j^2,\ \phi_j)$ where λ_j^k is the total arrival rate of requests from all the i agents with k indicating the type (or class) of the request, and all other parameters are the same as the single-class model.
   - The quality of metrics that will be derived analytically are W_j^k, LP_j^k and R_j where $k = \{1,\ 2\}$. These quantities have the same meanings as above with k referring to the class of the request (or task).

We will henceforth refer to QoS enjoyed by the i^{th} agent as x_i which, in general, is a vector comprising of the various QoS components such as response time or availability. All agents on node j will receive the same QoS when we consider the single-class case. However, in the multi-class case, agents on the same node j may experience differentiated service.

1.2.2 Problem 2: Allocation of agents on the computing infrastructure

Let there be n agents that need to be allocated. Let there be m nodes that comprise the infrastructure (the principal). Assuming that agents can predict the expected QoS using M_j^1 or M_j^2, the question we seek to answer now is as follows. What infrastructural nodes (among the M nodes) do the agents reside on and what QoS do they receive, if the principal has to maximize its revenue? Therefore, we design a *mechanism* using game theory for the agents and the principal to negotiate a QoS contract. By *mechanism, we* refer to a set of rules according to which the agents and the infrastructure (the principal) play the game of determining the allocation [7, 8]. We list the sub-problems below along with the research objectives.

1. **Sub-problem 2.1 - Mechanism A** (Chapter 6)
    - To develop a mechanism by which agents can participate in a game and self-select the QoS they will attain on the infrastructure. To utilize the predictive models M_j^p, $p = \{1, 2\}$ in the allocation of the agents and estimation of QoS levels which are not precomputed (to propose algorithms for computing allocation-dependent QoS levels).
    - To compute the usage prices p_j, where $j = \{1,\ 2,\ \ \ldots,\ m\}$ that will be paid to the principal for the QoS delivered.
2. **Sub-problem 2.2 - Mechanism B** (Chapter 6)
    - To develop a decentralized mechanism by which agents bargain at each of the j nodes to select the QoS they will attain on the infrastructure.
    - To propose an algorithm that will simultaneously determine the allocation and QoS levels by utilizing the models M_j^p, $p = \{1, 2\}$.
    - To contrast Mechanism B with Mechanism A.

1.2.3 Solution Methodology and Scope

For the queueing models in Problem 1, we will utilize the properties of generating functions to derive the steady state performance measures of the Markov process that describes the models. We will validate the models using simulation. The models are subsequently used in the allocation problem (Problem 2). Assigning an agent to a node affects the QoS of the other agents already present on the node. So, the model is used to determine the impact on QoS at the time of allocation as well as to check constraints based on quantities that the model estimates. To the extent the model helps the allocation process, we may refer to the related algorithms as model-based allocation algorithms and problem as the *model-based allocation problem*. This is in some sense is similar to model-predictive control (see [9, 10]) where a model is utilized to evaluate the space for candidate solutions. In this work we use queueing models for resource allocation and not (real-time) control. The models capture various interactions (explained in Chapter 3) in the MAS and assist in quantifying the impact, so that a good allocation is possible. So the control of the MAS and, for that matter, agent-based planning are outside the scope of this research.

In the second problem, we try to address mechanism design for the allocation problem from a game theoretic perspective. Model-based allocation algorithms form part of the mechanism, because the QoS levels available cannot be efficiently precomputed. While more explanation is offered in Chapter 6, it should be said that agents' private information (such as bids), the participants and the environmental conditions are revealed only at the time of allocation. This renders statistical models less useful in this domain, especially when a predictive analytical model is available to probe the solution space (of allocation) rapidly.

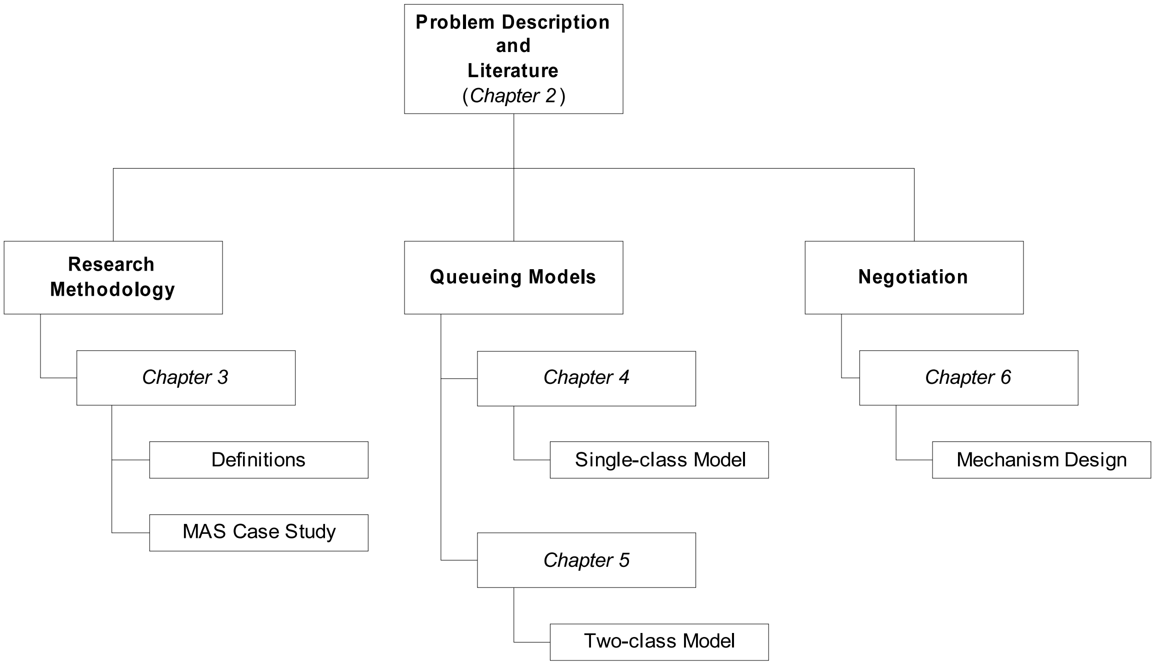


Figure 1.3: Organization of this thesis

1.3 Organization of Research and Problems Addressed

This organization of this thesis is shown in Figure 1.3. In Chapter 2, we relate our problem to various approaches in literature. In Chapter 3, we address two main goals. One of them is to provide a crisp definition for a MAS, describe associated constraints and optimization problems from the perspectives of both the infrastructure and the MAS. The second goal is to provide a more practical understanding by means of an implementation example of an agent-based system.The two main problems that are addressed in this thesis are in Chapters 4-6. In Chapter 4 and Chapter 5 , we address the problem of performance modeling of an agent-based system (Problem 1). In Chapter 6, we solve a version of the resource allocation problem (Problem 2) as applied to MASs.

Chapter 2

Problem Description and Literature

Although MASs, performance modeling or negotiation have been individually focused on by several researchers, there is not much work that connects these areas. We examine a representative body of work and describe how our problem is related to these in an effort to connect the aforementioned areas.

2.1 Analytical Queueing Models for Determining QoS

Our work is related to the problem of single server queueing models under breakdown and catastrophes. The QoS metrics are the information goods used for pricing are computed analytically from the queueing models. We examine work by Li et al. [11] and Chao [12] where the nodes can undergo breakdown and subsequent repair without loss of customers, or face catastrophes thereby losing all the customers in the server. In the former case an $M/G/1$ queue is examined while the latter is analyzed as a queueing network with product form solutions. In [13], a single server queue with catastrophes (customers are lost in this case) is studied as a continuous time Markov chain for deriving steady state performance metrics. The proposed research will extend [13] by having (1) breakdowns without customer loss in addition to catastrophes and (2) more than one class of customers. For transient analysis in related problems see for example [14, 15].

2.1.1 Queueing systems with breakdown: single-class case

Chao examines a server with catastrophic failures where all customers are lost [12]. But his assumptions are different in the following ways: (1) the catastrophe occurs with the arrivals, however in this thesis catastrophe randomly toggles the server into a state where processing does not occur; (2) repair times from catastrophic breakdowns are zero, however in this work rehydration (restarting logical node) can occur from catastrophic breakdowns; (3) Chao considers the breakdown and catastrophe synonymous, although this thesis differentiates them; and (4) the methodologies considered in this thesis are different. Gautam [13] considers a server with breakdown and repair where all customers are lost when the server breaks down. In this research, we differentiate breakdown and catastrophe and allow the system to accept customers while in breakdown - while the CPU is down requests can still fill the data buffers. In this way, we hope to extend the work done in [13]. This type of system with catastrophic breakdowns have received little attention - as has also been noted in [12, 13]. Other recent work (such as [15, 14, 11] etc.) consider transient analysis of such systems.

2.1.2 Queueing systems with catastrophic failure: multi-class case

When analyzing multiple classes of traffic for the aforementioned system, we generally encounter multi-dimensional Markov chains. Determining closed form solutions for multi-dimensional Markov chains is hard. In examples where a special structure is seen, for example, a QBD process, then we could resort to approximation techniques such as MGM [16]. A common method to deal with the state space is to often truncate it along some dimensions. For example, in the case of the two-dimensional state space we encounter for the two-class model it is feasible to truncate it along the y-axis. For example, Green [17] and Stanford and Grassman [18] analyze cycle stealing without switching cost in a multi-processor scenario by truncating the state space. Instead of simple truncation, one could resort to approximating the state space beyond a particular phase using moment matching algorithms. Osogami [19] has shown this type of dimensionality reduction for multi-dimensional Markov chains with QBD structure. Another approach that is seen is the use of generating functions to solve for the stationary probabilities using uniformization [20] or by reducing the functional equation to a boundary value problem [21]. Generating functions also seem to address a specific class of problems, wherein there seem to be no examples for state-spaces with more than two dimensions. Our approach reduces the dimensionality by truncation because approximating the state space would require a special structure which our problem seems to lack. Subsequently, we employ generating functions to solve for the steady state “boundary” probabilities. Note that the size of the queue can be truncated at a suitable value of n (where n is the maximum number of class 2 packets upon truncation). As such the use of generating functions in this way allows for a convenient representation for the derivation of many performance measures.

2.2 Brief Tutorial on Generating Functions

2.2.1 Generating functions

Generating functions have been widely used in the literature [22, 23]. In particular, they are useful in queueing problems for compactly representing summations of combinatorial state probabilities. We will review some relevant concepts and list some examples of generating functions that may be useful for this work.

Definition 2.2.1. Let a_0, a_1, a_2 be an arbitrary (infinite) sequence of numbers. The generating function (generating series) for this sequence is the expression of the form

$$a_0 + a_1 z + a_2 z^2 + \dots,$$

or, briefly,

$$\textstyle\sum_{n=0}^{\infty} a_n z^n \quad .$$

If beyond a certain n, the coefficients a_{n+1}, a_{n+2}, $\dots$ are zero then the generating function is called a generating polynomial. The elements of the sequence a_n may be of arbitrary nature. Of interest to this work is when these elements are real because they denote probabilities.

Example 2.2.2. $\phi(z) = \sum_n p_n z^n$ is the generating function of the probabilities p_n. For example, upon simplification of the balance equations for the Markov chain in

[13], we get a generating function of the form:

$$\phi(z) = \frac{c_0 - c_1 z}{z^2 - c_1^{'} z + c_0^{'}}$$

where $\{c_0,\ c_1,\ c_0^{'},\ c_1^{'}\}$ are all constants that can be represented in terms of the parameters in the Markov chain and the p_ns.

Example 2.2.3. A case where the a_ns are integer valued is the Fibonacci sequence $1,\ 1,\ 2,\ 3,\ 5,\ 8,\ \ldots$, which can be represented as $Fib(z) = 1 + z + 2z^2 + 3z^3 + \ldots$. Upon simplification this sequence can be written as $Fib(z) = \frac{1}{1-z-z^2}$.

Let X be a non-negative integer valued random variable. From the probability perspective the generating function can be written as follows.

$$\begin{aligned} \phi_X(z) &= E(z^X) \\ &= \sum_{k=0}^{\infty} z^k P\{X = k\} \end{aligned}$$

Properties of Generating Functions

1. The probability mass function is uniquely defined by its generating function. If two random variables have the same generating function, then must have the same probability mass function.

2. The probability mass function can be derived from the generating function as follows:

$$P\{X = k\} = \frac{1}{k!} \frac{d^k}{dz} \phi_X(z) \mid_{z=0}.$$

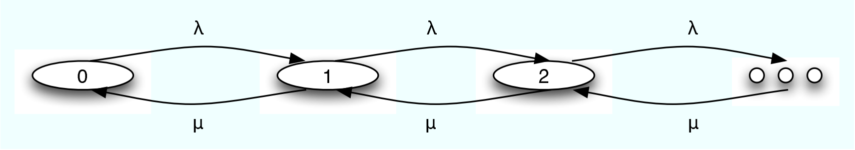


Figure 2.1: A birth-and-death process

3. Moments of X can be derived from its generating function as follows. Let $(X)_r = X\ (X-1)\dots(X-r+1)$ be the r^{th} factorial power of X. Then

$$E((X)_{r)} \quad = \quad \frac{d^r}{dx^r}\ \phi_X(z)\ |_{z=1}.$$

   In particular,

$$\begin{aligned} E(X) \quad &= \quad \frac{d}{dz}\ \phi_X(z)\ |_{z=1}, \\ E(X^2) \quad &= \quad E(X(X-1)) + E(X) = \frac{d^2}{dz^2}\ \phi_X(z)\ |_{z=1} + \frac{d}{dz}\ \phi_X(z)\ |_{z=1}. \end{aligned}$$

4. Let X_1 and X_2 be independent random variables. Then

$$\phi_{X_1+X_2}(z) \quad = \quad \phi_{X_1}(z)\ \phi_{X_2}(z)$$

2.2.2 Quasi-Birth and Death Processes

In this section, we will provide a brief tutorial on quasi-birth-and-death (QBD) processes. We will also provide an overview on using generating functions with QBD processes.

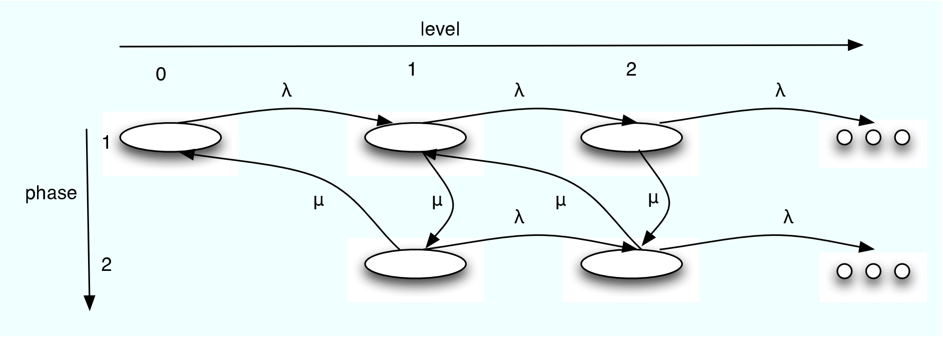


Figure 2.2: A quasi-birth-and-death process

2.2.2.1 Examples of QBD processes

We provide a few illustrative examples of QBD processes. A birth-and-death (BD) process is a QBD process. Figure 2.1 shows an example of a BD process. This BD process models the number of customers in an $M/M/1$ queue where the customers arrive according to a Poisson process with rate λ and are serviced according to an exponential distribution with parameter μ. In general, a BD process is a Markov chain on the states $\{0,\ 1,\ 2,\ 3,\ \ldots\}$, where transitions can occur between neighboring states. The generator matrix of a BD process is of the form:

$$\begin{pmatrix} -f_0 & f_0 & 0 & \\ b_1 & -(b_1+f_1) & f_1 & \\ & b_2 & -(b_2+f_2) & \ddots \\ & & \ddots & \ddots \end{pmatrix},$$

where f_l is the transition from state l to $l+1$, and b_l denotes the transition from l to $l-1$. In the figure $f_l = \lambda$ and $b_l = \mu$ for all l.

Figure 2.2 is a QBD process but not a BD process. In a QBD process, transitions are allowed between neighboring levels and between neighbors within each level. This process models the number of customers in a $M/Er/1$ queue where customers arrive according to a Poisson process with rate λ and are processed according to an Erlang-2 distribution which has parameter μ. In other words, the Erlang-2 distribution is the distribution of the time until absorption into state 0 in a Markov chain on the states $\{0,\ 1,\ 2\}$ with initial probability vector $(0,\ 1,\ 0)$ and infinitesimal generator:

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & -\mu & \mu \\ \mu & 0 & -\mu \end{pmatrix} .$$

Generally speaking, a QBD process is a Markov chain on the state space $\{(i,\ l) \mid 1 \leq i \leq n_l,\ \ l \geq 0\}$, where the Markov chain is divided into levels and each level l has n_l states. For example, in Figure 2.2, $n_0 = 1$ and $n_l = 2\ \forall\ l \geq 1$. Thus, a QBD process will have a generator matrix has the form:

$$Q \;=\; \begin{pmatrix} L^{(0)} & F^{(0)} & 0 & \\ B^{(1)} & L^{(1)} & F^{(1)} & \\ & B^{(2)} & L^{(2)} & F^{(2)} \\ & \ddots & \ddots & \ddots \end{pmatrix},$$

where sub-matrix $F^{(l)}$ denotes the forward transitions from level l to level $l+1$ for $l \geq 0$, sub-matrix $B^{(l)}$ denotes the backward transitions from level l to level $l-1$ for $l \geq 1$ and sub-matrix $L^{(l)}$denotes the transitions within level l for $l \geq 0$. For example, in Figure 2.2,

$$F^{(0)} = \begin{pmatrix} \lambda & 0 \end{pmatrix} \text{ and } F^{(l)} = \begin{pmatrix} \lambda & 0 \\ 0 & \lambda \end{pmatrix} \text{ for } l \geq 1$$

$$B^{(1)} = \begin{pmatrix} 0 \\ \mu \end{pmatrix} \text{ and } B^{(l)} = \begin{pmatrix} 0 & 0 \\ \mu & 0 \end{pmatrix} \text{ for } l \geq 2$$

$$L^{(0)} = -\lambda \text{ and } L^{(l)} = \begin{pmatrix} -(\lambda+\mu) & u \\ 0 & -(\lambda+\mu) \end{pmatrix} \text{for } l \geq 1.$$

Since transitions can occur only between neighboring levels, the process is said to be “skip-free between levels”.

2.2.3 Matrix Geometric Method (MGM)

The structure of QBD processes enables these systems to be solved using matrix geometric methods. It is to be noted that the application of MGM requires that we deal with QBD processes. In MGM, an auxiliary matrix called R is utilized for computing steady state performance measures such as waiting time or mean queue length. The matrix R is often determined numerically using the following relation (Ramaswami [24], Riska [25]):

$$F^{(l)} + RL^{(l+1)} + R^2B^{(l+2)} \quad = \quad 0. \tag{2.1}$$

Equation (2.1) is used recursively to solve for R. Neuts [16] defines infinite -state Markov chains with a repetitive structure with states partitioned into the boundary states $S^{(0)} = \{s_0^1,\ \ldots,\ s_0^n\}$ and a set of states $S^{(i)} = \{s_i^1,\ \ldots,\ s_i^n\}\ \forall i \geq 1$, that correspond to the repetitive portion of the chain. Let $\pi^{(i)}$be the steady state probability vector of states $S^{(i)}$. Then

$$\pi^{(i)} \quad = \quad \pi^{(1)}R^{i-1}\ \forall i \geq 1.$$

For $\pi = [\pi^{(0)} \ \pi^{(1)} \ \ldots]$, solving $\pi Q = 0$ will give both $\pi^{(0)}$ and $\pi^{(1)}$. Let e be a column vector of ones. The following set of equations are obtained:

$$\begin{aligned}
\pi^{(0)} L^{(0)} + \pi^{(1)} B^{(1)} &= 0, \\
\pi^{(0)} F^{(0)} + \pi^{(1)} (L^{(1)} + R B^{(2)}) &= 0, \\
\pi^{(0)} e + \pi^{(1)} (I - R)^{-1} e &= 1.
\end{aligned}$$

2.2.4 Systems with breakdown

A system with breakdown can go into breakdown from any state. For example, in [13] a Markov chain is described on states $\{D, \ 0, \ 1, \ 2, \ \ldots\}$ where there can be transitions between levels and states that are not necessarily neighbors. In particular, there is a transition for every state i to state D (which denotes the breakdown state). The generator matrix of this system will have the form:

$$\begin{pmatrix}
-\delta & \delta & & & \\
\gamma & -(\gamma+\lambda) & \lambda & & \\
\gamma & \mu & -(\gamma+\lambda+\mu) & \lambda & \\
\gamma & & \mu & -(\gamma+\lambda+\mu) & \ddots \\
\vdots & & & \ddots & \ddots
\end{pmatrix}.$$

From the generator matrix, it is obvious that **??** is not a QBD process. We stated that a QBD process has to be skip-free in the levels which does not seem to be the case here. This lack of structure is the reason why we cannot apply MGMs to the above problem.

2.2.5 Using Generating functions

We give an example of how a generating function is used for a QBD. Consider the example in Figure 2.1 which represents an $M/M/1$ queue. Let $X(t)$ represent the number in system at time t. $\{X(t),\ t \geq 0\}$ is a CTMC on state-space $\{0,\ 1,\ 2,\ 3, \ldots\}$. Let $p_j = \lim_{t\to\infty} P\{X(t) = j\}$. The balance equations are as follows.

$$\begin{aligned} p_0\lambda &= p_1\mu \\ p_1(\lambda+\mu) &= p_2\mu + p_0\lambda \\ p_2(\lambda+\mu) &= p_3\mu + p_1\lambda \\ \vdots &= \vdots \end{aligned}$$

We multiply the first equation by z^0, second by z^1, third by z^2, and so on add up both sides of the equations. If we denote $\psi(z) = \sum_0^\infty p_i z^i$, we can more compactly represent the summation on both sides on the equation as follows:

$$(\lambda+\mu)\psi(z) - \mu p_0 = \frac{\mu}{z}\psi(z) - \frac{\mu}{z}p_0 + \lambda z\psi(z).$$

Rearranging,

$$\psi(z) = \frac{\mu p_0}{\mu - \lambda z}.$$

So we get a very compact notation for probabilities multiplied by z^i. It we let $z = 1$, then $\psi(z) = 1$ since we are summing up all probabilities in the Markov chain. This fact will allow us to find the value of p_0 as follows:

$$\frac{\mu p_0}{\mu - \lambda} = 1,$$

therefore $p_0 = 1 - \frac{\lambda}{\mu}$. Using this expression for $\psi(z)$ we can determine the performance measures of the $M/M/1$ queue. However, we will see later in models discussed in Chapter 4 and Chapter 5, the expression $\psi(1) = 1$ does not yield the desired benefits. Moreover, in we will use more than one generating function for each of the models which are not QBDs such as the one we discussed.

2.3 Allocation Problems

The allocation problem is so rich that it has attracted the interest of researchers from Artificial Intelligence, Software Engineering, Economics and Operations Research to name just a few. With the proliferation of large-scale computing applications in seemingly diverse areas such as net-centric logistics [26], commercial data centers, and elastic computing clouds (Amazon EC2[27]) , the problem of efficiently and dynamically allocating software components to hardware resources is gaining prominence. The Generalized Assignment Problem (GAP) can be described using terminology from the familiar Knapsack Problem. The 0-1 Multiple Knapsack Problem arises when n items (or agents) have to assigned to m containers (or servers), each of given given capacity c_j $j = \{1,\ 2, \ldots,\ m\}$. Let x_{ij} be the binary decision variable that is 1 when item i is assigned to container j , 0 otherwise. It is formulated as Problem 1 where p_i and w_i are the profit and weight associated with item i.

If the profit and weight associated with item i depends on the container j, the

$$
\begin{aligned}
&\quad max\ \sum_{j=1}^{m}\sum_{i=1}^{n} p_i x_{ij}\\
&subject\ to\\
&\qquad \sum_{i=1}^{n} w_i x_{ij} \leq c_j \quad \forall\ j \in \{1, \dots,\ m\}\\
&\qquad \sum_{j=1}^{m} x_{ij} \leq 1 \quad \forall\ i \in \{1, \dots,\ n\}\\
&\qquad x_{ij} = 0\ or\ 1\ \ \forall\ i \in \{1, \dots,\ n\},\ \forall\ j \in \{1, \dots,\ m\}
\end{aligned}
$$

Problem 1

$$
\begin{aligned}
&\quad max\ \sum_{j=1}^{m}\sum_{i=1}^{n} p_{ij} x_{ij}\\
&subject\ to\\
&\qquad \sum_{i=1}^{n} w_{ij} x_{ij} \leq c_j \quad \forall\ j \in \{1, \dots,\ m\}\\
&\qquad \sum_{j=1}^{m} x_{ij} \leq 1 \quad \forall\ i \in \{1, \dots,\ n\}\\
&\qquad x_{ij} = 0\ or\ 1\ \ \forall\ i \in \{1, \dots,\ n\},\ \forall\ j \in \{1, \dots,\ m\}.
\end{aligned}
$$

Problem 2

GAP arises, which is formulated similarly as Problem 2. p_{ij}, w_{ij} and c_j are normally known in advance. Both problems above are combinatorial in nature as it involves making choices from a total of m^n choices for $(i,\ j)$ pairs. The problem we discuss in Chapter 6 is one in which the profit p_{ij} (also w_{ij}) are dependent not only on i but also the allocation of other items. Because of the dependencies amongst the allocations (i.e. x_{ij}s), we have to allocate as a set $\widetilde{x} = (\widetilde{x}_1,\ \widetilde{x}_2,\ \dots,\ \widetilde{x}_n)$ comprising of all the individual assignments for every agent (here, if the value of $\widetilde{x}_i = j$ it would correspond to $x_{ij} = 1$ in the above GAP). Since w_{ij} (perhaps a QoS measure) varies for every $\widetilde{x}$, we will use a model to dynamically determine them during the allocation process.

There is a separate literature on stochastic knapsack problems (SKP) [28, 29, 30, 31]. Some of them are the so called online stochastic knapsack problems (SKPs) (see for example [32, 30, 28]) where items arrive over time. In several of these online SKPs the weights (w_{ij}) and profits (p_{ij}) are independent random variables with known distribution.

2.4 Pricing Information Goods

Pricing provides an incentive for the application to use the resources efficiently, a mechanism to control the congestion of the infrastructure and also a contract for the infrastructure to get compensated for the service it is providing. Congestion sensitive pricing can cause the user to consume resources more conservatively, thereby reducing the load on the infrastructure. Pricing is also a means by which the infrastructure can collect money or charge the application for the information goods it is consuming. Another motivation for pricing is that an application's cost savings can be an indicator of how well it can survive stressful environments, particularly because applications are becoming increasingly resourceful in selecting operational settings. However, issues of who should be in control of which component is always under question. Traditionally, the service provider has been the one that provides whatever quality he can provide and prices it in a rather monopolistic way. For example, consider data centers or supply networks - where they charge for response time (in ms) or delay (in days) through posted-price negotiation. The *consumer* has little role to play than to accept or reject.

Within the problem of negotiation of contracts in MAS, we explore pricing because of its ability to influence or control a multitude of attributes especially in large decentralized systems. Pricing and its applications has been widely studied in a variety of areas including Internet, computer networks and services for the web (see [33, 34, 35, 36, 37, 38]). More recently pricing has been applied in the contexts of pricing of web servers, services oriented computing and congestion control of networks. These schemes involve pricing based on the QoS experienced, the congestion faced by the network, a flat fee or combinations thereof. Davies et al. study a framework for capacity planning and pricing IP networks support various kinds of services [39]. Caesar et al. compare usage based pricing methods to charge the user on the basis of QoS received, congestion or both as applied to IP telephony [40]. Gautam considers a three-tier architecture consisting of users, a client application and the infrastructure that hosts it, and provides a mechanism for the pricing the client on the basis of QoS received by the client's users [13]. Yaucoubi et al. discuss pricing in network access providers where a queueing model is used to reserve a minimum bandwidth for each customer such that the provider's revenue is maximized while request blocking is guaranteed to be under a negotiated percentage [41]. Chen et al. study pricing policies and admission control methodologies that impact performance of web-servers [42]. Henderson et al. discuss the pros and cons of existing congestion based pricing mechanisms which primarily aims at internalizing economic externalities (eg. using a toll in the highway system). They mention that congestion based pricing may actually replace flat and usage based pricing, but is also fraught with issues such as complexity in the case of bidding based methods, inaccurately charging based on expected congestion in edge mechanisms and routers adding their own marks on IP packets in the case of ECN [43]. Paschalidis and Tsitsiklis use a dynamic programming formulation for determining a congestion-dependent fee in the case of a communication network by using concepts of welfare maximization. They also note that an appropriately chosen *time-of-day* pricing may be enough in many cases with the advantage of being very simple to implement [44]. Lin and Shroff show that when a large enough network exists an appropriate static pricing scheme will be close to a dynamic pricing scheme [45].

Numerous *market-based control mechanisms* are available in literature such as [46, 47, 48, 49]. In market-based control systems, agents emulate buyers and sellers in a market acting only with locally available information yet helping us realize global behavior for the community of agents. While these methods are very effective and offer desirable properties such as decentralization, autonomy and control hierarchy, they have been used for resource allocation [46, 47] and resource control [49]. The Challenger [47] system seeks to minimize *mean flow time (job completion time - job origination time)*, the task is allocated to an agent providing least processing time. Load balancing is another application as applied by Ferguson et al. [48]. Within large information systems, Wellman has studied market based methods as well as illustrated the concepts of competitive equilibria and auction-based methodologies [50, 51].

2.5 Negotiation of Price and QoS

Once QoS and price are determined, the application and the infrastructure have to negotiate the prices such that the infrastructure is compensated for the service provided. This problem leads us in to the realm of Microeconomic analysis where the negotiation of prices and commodities are generally studied [52, 38]. In studies such as [41, 42] the server is responsible for setting the prices and its resources, thereby restricting the negotiation capability of the customers. In [13], the server gets paid according to QoS experienced by the users of a web-server application (viz. the client). Here the activity of negotiation happens between the client and server for deciding the price and acceptable service level. Other market-based methods involving scheduling to achieve better QoS has been examined in [53] in the context of component-based systems. One aspect of negotiation deals with identifying the quantitative parameters involved (such as delay, response time or reliability). Another aspect of negotiation dealing with protocols, representation and languages has received considerable attention in literature (see for example, [54, 55], more recently [56]). Many studies incorporate Rao and Georgeff's BDI model [57, 58] in some form and are primarily logic based.

2.6 Non-Pricing Based Control Mechanisms for MAS

Because of the diversity of literature on control frameworks and performance evaluation, we examined a representative subset primarily on the basis of control objective, (component) interdependence and autonomy, generality, composability, real-time capability (off-line/on-line control) and layering in control architecture.

In some *AI based approaches* such as [59, 60], behavioral or rule based controllers are employed to make the system exhibit particular behavior based upon logical reasoning or learning. While performance is not the objective, layered learning is an interesting capability that may be helpful in a large scale MAS. Learning may be from a statistical sense as well where the parameters of a transfer function are learnt from empirical data to subsequently enforce feedback control [61]. Another architectural framework called MONAD [62], utilizes a hierarchical and distributed behavior-based control module, with immense flexibility through scripting for role and resource allocation, and co-ordination. While many of these approaches favor the "sense-plan-act" or "sense and respond" paradigm and some partially support flexibility through scripting, some important unanswered questions are what happens when system size changes, can all axioms and behaviors be learnt a priori and what is the performance impact of size (i.e. scalability)?

Control theoretic approaches in software performance optimization are becoming important [63, 64], with software becoming increasingly more complex, multi-layered and having real-time requirements. However, because of the dynamic system boundaries, size, varying measures of performance and non-linearity in DMAS it is very complex to adopt a strict control theoretic process [65]. Some approaches such as [65, 5] take the heuristic path, with occasional analogs to control theory, with an emphasis on application or domain-specific utility. Kokar et al. [63] refer to this utility as *benefit function* and elaborate on various analogs between software systems and traditional control systems. From the perspective of autonomic control of computer systems, Bennani and Menasce [66] study the robustness of self-management techniques for servers under highly variable workloads. Although queueing theory has been used in this work, any notion of components being distributed or agent-based along with the occurrence of catastrophes seems to be absent. Furthermore, exponential smoothing or regression based load-forecasting may not be sufficient to address situations caused by wartime dynamics, catastrophic failure and distributed computing. Nevertheless, in our approach we have a notion of controlling a distributed application's utility using queueing theory.

Using *finite state machines*, hybrid automata and their variants have been the foci of many research paths in agent control as in [67, 68]. The idea here is to utilize the states of the multi-agent system to represent, validate, evaluate, and choose plans that lead the system towards the goal. Often, the drawback here is that as the number of agents increase, the state-space approaches tend to become intractable.

Heuristics have widely been used in controlling multi-agent systems primarily in the following sense: searching and evaluating options based on domain knowledge and picking a course of action (maybe a compound action composed of a schedule of individual actions) eventually. The main idea in recent heuristics based control as exemplified by [69, 70, 71] is that schedules of actions are chosen based upon requirements such as costs, feasibilities for real-time contexts, complexity, quality etc. Opportunistic planning is an interesting idea as mentioned in Soto et al. [71] refers to the best-effort planning (maximum quality) considering available resources. These meta-heuristics offer very effective, special-purpose solutions to control agent behavior, however to be more flexible, we separate the performance evaluation and the domain-specific application utility computation.

Given that we have a model for performance estimation (whose parameters and state-space are known), dynamic programming (DP) and its adaptive version - reinforcement learning (RL), and model predictive control (MPC) have been used to find the control policy [72, 73, 74, 9, 10]. The complexity of finding the optimal policy grows exponentially with the state space [72] and convergence has to be ensured in RL [73, 74].

In large scale MAS applications, performance estimation and modeling itself can be a formidable task as illustrated by [75] in the UltraLog [3] context. UltraLog [3], built on Cougaar [76], uses for heuristic control a host of architectural features such as operating modes, conditions, and plays and play-books as described in [65]. Helsinger et al. [77] incorporate the aforementioned features into their closed-loop heuristic framework that balances the different dimensions of system survivability through targeted defense mechanisms, trade-offs and layered control actions. The importance of high-level, system specifications (interchangeably called *TechSpecs*, specification database, component database) has been emphasized in many places such as [2, 65, 78]. These specifications contain component-wise, static input/output behavior, operating requirements and control actions of agents along with domain measures of performance and computation methodologies [78]. Also, queueing network based methodologies for offline and design-time performance evaluation have been applied and validated in [78, 79].

2.7 Distributed computing versus agent-based computing

Distributed computing traditionally involves allocating jobs from a central location to various computing entities and then accumalating the results. The inherent assumption for executing this paradigm of distributed computing is that these entities forming the computing infrastructure have to connected using highly specialized software as determined by the application in question (see for example, [50, 51]). Another paradigm, agent-based software engineering [1, 2], uses software agents as the building blocks of distributed computing. In other words, instead of the distributing light-weight jobs, interconnected software agents form a network of functionally-distinct components. The agent network, also referred to a distributed multi-agent system, acts like a traditional software application, except that its components are distributed on various computing entities (for example, servers). The agent network can handle relatively general purpose problems such as maneuver planning for logistics, distributed numerical computations in mathematics and task-assignments without major reconfiguration by suitably adapting to contexts and users [76].

While new applications of agent-based computing are growing by the day, large-scale commercial deployments of agent-based computing are scarce. This is because this field is relatively new and is fraught with problems. Below, we examine the questions surrounding one such problem for agents operating in harsh environments, i.e. agent allocation and its control.

The natural question that arises is who allocates these agents on the computing entities? Typically, the allocation of these agents is done manually on the chosen collection of nodes. These agents once coupled to the infrastructure, act somewhat like brokers (say for multiple users, a collection of sensors etc.) and send tasks to the CPU for processing and receive the results. Understandably, the manual technique of agent allocation does not scale well - i.e. when there are a large number of agents, manual allocation is not practical. Secondly, what if the computing entities are not operating under normal environmental settings? Often times, agent-based computing is applied to, military logistics scenarios where there are many abnormal types of loads and breakdowns. Under such situations, allocation is an even more difficult problem because the state of the computing infrastructure has to be factored in. Thirdly, what if the agents themselves are overloading the computing infrastructure? Since this is plausible, the allocation problem has to deal with mechanisms to control over-usage because the resulting can serverely affect the performance of the computing infrastructure.

Chapter 3

Research Methodology

3.1 Introduction

In this chapter, we provide a high-level description of the research problem, discuss the sub-problems relate to each other. Simultaneously, we will describe the motivating scenarios that this research draws from. We introduce terminology that will be used subsequent chapters in two ways:

1. by providing mathematical definitions and other descriptions, and
2. by providing an example of a multi-agent application.

3.1.1 The Problem at a Glance

The research problem studied in this thesis can be divided broadly into two areas. The first area studied relates to distributed software components and the impact of harsh environmental conditions on these components. These software entities, known as agents, operate as a group (community) and achieve their goals collectively, giving rise to the term multi-agent system (MAS). Due to particular nature of the goal and collective functioning of the agents, the MAS is not unlike a software application except for differences in the scale of distribution and the characteristics of the agents. The goal of a MAS could be a mission-critical application such as military logistic planning (our motivation). The scale of the application is of the order of thousands of agents (UltraLog, the military logistics multi-agent based planner used over 10,000 agents for battle-field scenarios [3]). Agents have many characteristics such as situatedness, autonomy, interactivity and adaptivity as described in [1, 2]. In this thesis we focus on the characteristic of adaptivity when there are changes in the performance experienced by agents while operating in a stressful environment.

The second area relates to analyzing the process of reaching a (service) *contract* between the agents and the computing infrastructure (say, a collection of servers). Agents being software programs, require hardware to situate them and provide processing power. This processing is provided by the computing infrastructure. However, infrastructure capacity is constrained and various (battle-field) stresses bring down either the agents, the computing infrastructure or both. In order to prevent congestion and hence degraded service, agents look to the infrastructure to propose a *mechanism* through which a contract can be negotiated and subsequently adhered to. On one side of the contract is the quality of service (QoS) the agents request while on the other side there is the payment they make for experiencing that QoS. With these characteristics in mind, this thesis seeks to formalize the process of negotiating quantitative performance contracts automatically within the realm of autonomous software agents.

3.1.2 Practical Questions

If we combine the aforementioned areas, certain practical issues come to mind. Foremost among them - is distributed computing effective in harsh operating environments? To what extent can agent-based software systems be of help in this sort distributed computing - given their characteristics of autonomy, self- and peer-awareness and cooperation? While we cannot say anything conclusive, we can definitely claim that it may be effective in certain domains. In particular, the domain of military logistics and, in general, commercial supply chains seems to promising. The design of UltraLog using an agent-based backbone is based on the aforementioned claim. In this work, we augment the claim by developing models and mechanisms pertinent to this domain, which the agents can adopt or internalize into their knowledge bank and/or playbooks (strategy pool) to maximize the performance they experience.

3.1.3 Structure of this chapter

This chapter provides a definition for an agent-based application and lists several constraints that such an application may face. This is followed by a concise description of the application's and the infrastructure's problems. The application and the infrastructure are primarily responsible for the software and hardware respectively. The two parties control different variables that affect the performance achieved by the agent-based application.

First, it is important for the agent-based application to be capable of predicting its own performance. The performance prediction is used by the application to select and tune the parameters under its control. So it is assumed that certain analytical models are contained within the application that are run rapidly to evaluate the alternatives (different parameter settings in the space of parameters) and select the parameters. This concept is similar to model-predictive control (MPC) [72, 73] where, for example, a policy may be used to guide the course of future action in a fine-grained time-scale (i.e. real-time control). We use steady state estimates from queueing models because the application is selecting parameters for a courser time-scale (window to window operation). After explaining the attributes of the application in this chapter, we go into the details for the queueing models (single- and two-class) in Chapter 4 and Chapter 5 respectively. In Chapter 6, we discuss how the models are used in the context of the MAS.

The second goal in this chapter is to give a birds-eye view of a multi-agent application. The application we provide as an example is actually a MAS designed within the UltraLog framework. We start from defining what is referred to as the Continuous, Planning and Execution (CPE) agent society. We describe the agents and list their Technical Specifications (TechSpecs). The case-study highlights agent-based software design steps, TechSpecs formulation and a simple performance estimation example. While the case-study illustrates a systematic method to approach the MAS design and performance estimation problem, *it is an example distinct from the single- and two-class queueing models that explicitly factor breakdowns in operation.* As mentioned earlier, the queueing models (Chapter 4 and Chapter 5) and their utilization (Chapter 6) are the main topics in subsequent chapters. This structure is also explained in Figure 3.1.

3.2 Definitions and Domain Information

In this section, we define some terms and describe the model parameters.

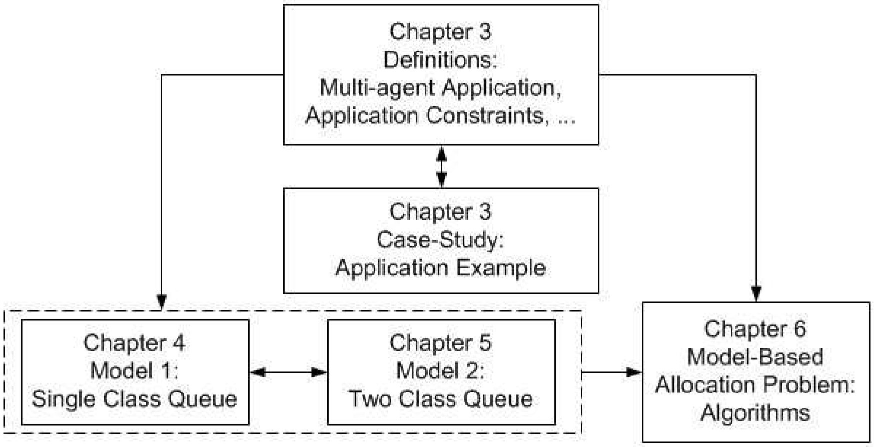


Figure 3.1: Structure of the Chapters 3-6

3.2.1 Definitions

Definition 3.2.1. Application. An agent-based application is a set of interacting software agents $A = \{a_1,\ a_2,\ \ldots,\ a_n\}$ having the specialized requirements tuple $\{I,\ C,\ S,\ U,\ M,\ T\}$ where

- a_i refers to agent i, $i \in \{1,\ 2,\ \ldots,\ n\}$;
- If $\{n_1,\ n_2,\ \ldots,\ n_n\}$ is an assignment of each of the n agents belonging to the application to one of the m nodes that constitute the infrastructure where $n_i \in \{1,\ 2,\ \ldots,\ m\}\ \ \forall i \in \{1,\ 2,\ \ldots,\ n\}$, I is the set of all such assignments;
- $C = \{c_1,\ c_2,\ \ldots, c_k\}$ constitute a set of constraints demanded by the agents;
- S is a set of strategies (also called play-books), say for bidding, adaptation or self-healing;
- $U = \{u_1,\ u_2,\ \ldots,\ u_N\}$ is a set of private utility functions for the agents (defined in Chapter 6);

- $M_j(p_j, \phi_j)$ is a shared performance model used by the agents belonging to n_j, $j \in \{1,\ 2,\ \ldots,\ m\}$($p_j$ is decided by the agents on node j, ϕ_j describes node j's characteristics which are not under application control);
- $T = \{t_1,\ t_2,\ \ldots,\ t_n\}$ is a set of technical specifications (TechSpecs) for the agents that, among many domain specific needs, details the required processing speeds, types of requests, desired repair rates and so on.

By interacting agents, we do not *necessarily* refer to a group of agents through which a specific task is routed. Agent interactions might also refer to communication between agents to share model M_j's parameters. At other times, the agents may communicate to share and compose individual TechSpecs, into a group specification. For example, if the set of agents $\{a_1,\ a_2,\ a_3\}$ are assigned to n_1, the group specification for repair rate is $\{\beta_1,\ \beta_2,\ \beta_3\}$ - which are the only acceptable repair rates on node n_1. By self-healing, we mean a set of actions that an agent can implement to improve the performance or availability it offers to the whole system i.e. the application. For example, the agent could request software reconfiguration from the operating system, thereby *repairing* states such as deadlocks or thread contentions and therefore improving the performance it experiences. In Section 3.4, we go through a case study of an agent-based application where in concepts such as TechSpecs are explained in detail for a particular case.

Definition 3.2.2. Agent Constraints. The set of all constraints (quantitative or qualitative) required by individual agents or groups of interacting agents for the application to function. If all constraints are satisfied, the application is said to function normally.

Without domain-specific information, it is hard to describe agent constraints are important. Hence, we give few examples from our domain of military logistics.

Definition 3.2.3. Flow Constraint. A flow constraint is an upper bound (Δ) on the absolute value of worst-case average delay for a set of agents $\Lambda = \{\Lambda_1,\ \Lambda_2,\ \ldots,\ \Lambda_k\}$, i.e.

$$\sum_{i=1}^{k} d(a_i) \leq \Delta$$

where $d(a_i)$ denotes the average delay of agent i.

When two agents are assigned to nodes p and q having an average response time w_p and w_q respectively, the worst case average delay for a flow consisting of these two agents is $w_p + w_q$. This is because it can be assumed that in the worst case the tasks are processed with zero parallelism - say all tasks of node p are processed after node q. This constraint is a strong one because in most cases there is some degree of parallelism even though the asynchronous computation is common in the agent literature. In the limiting case, when there is only agent per flow, the flow constraint becomes the delay constraint for an individual agent (because the flow is a sum of delays).

Certain types of agents may not be co-located i.e. residing on the same node. This could be because of security reasons. Nevertheless, it affects the allocation of agents. We call this type of constraint a grouping constraint.

Definition 3.2.4. Grouping Constraint. If the set $G = \{\Lambda_1,\ \Lambda_2,\ \ldots,\ \Lambda_k\}$ is a set of agents and $IG = \{ig_1,\ ig_2,\ \ldots,\ ig_3\}$ are the nodes to which the agents are assigned, then

$$ig_r \neq ig_s \;\; \forall \; \{g_r, \; g_s\} \in G, \; IG \subseteq I, \; G \subseteq A.$$

The constraint could be defined in terms of the allocation of agents as well (see Chapter 6).

Definition 3.2.5. Fairness Constraint. If b_i denotes the bid of agent i,.

$$b_i \geq b_{i+1} \implies x_i \geq x_{i+1} \quad \forall \; i \; \in \; (1, \ldots, \; n-1),$$

which denotes that a higher bidding agent is entitled to atleast as much QoS as another agent with a lesser or equal bid.

In addition, the principal may have constraints on its own (see Chapter 6).

Definition 3.2.6. Allocation. An allocation of the application A is one choice $\hat{I} \in I$, i.e.

$$\hat{I} = \{n_1, \; n_2, \; \ldots, \; n_n\} \; \forall n_i \in \{1, \; 2, \; \ldots, \; m\}, \; i \in \{1, \; 2, \; \ldots, \; n\}$$

which is one among m^n such possibilities.

Some allocations may be infeasible given the constraints of the agent and the principal. Let A_{ij} denote the decision variable that agent i is allocated to node j such that,

$$A_{ij} \;\; = \;\; \left\{ \begin{array}{ll} 1 & if \; agent \; i \; is \; allocated \; to \; node \; j \\ 0 & \qquad\qquad otherwise \end{array} \right\}$$

for all $1 \leq i \leq n$ and $1 \leq j \leq m$ (here i is the index for agents and j for nodes). For the allocation $\hat{I}$, only $A_{in_i} = 1$.

3.2.2 Model Parameters

Here we present, what we call, a multi-scale view of a system of computing agents. Typically, each scale in a large system would have its own self-healing capabilities. For modeling the interactions in our MAS, we use queueing models because of our particular interest in performance. After sensing the environment, planning for the future actions would based on predictions from appropriate queueing models. These queueing models at a single node are sometimes referred to as "micro-models" (refer Cohen [20]). The queueing models that are developed here describe a scenario of partial operation in the agent as is typical for a software entity. The agent can be partially incapacitated by software issues such as thread contentions, deadlocks and exceptions all of which would degrade performance without disrupting the agent completely.

We now describe the model M_j and the interactions it seeks to capture. The interactions are diagrammatically represented in Figure 3.2.

3.2.3 Objectives of the Application and the Infrastructure

Since we are developing a pricing scheme that models the economic interaction between an application that buys services and an infrastructure that sells them in the form of information goods, we examine their individual objectives.

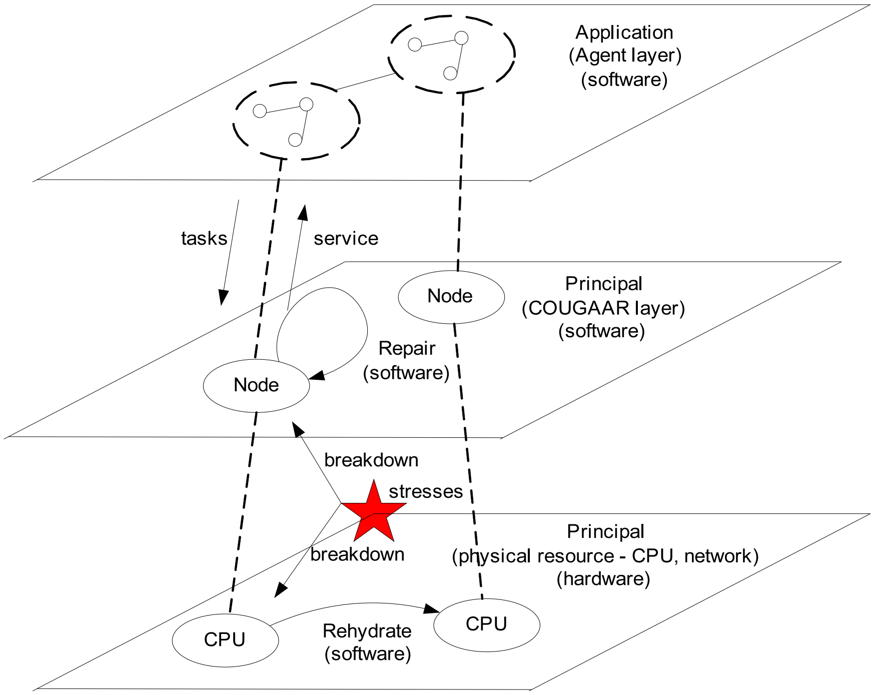


Figure 3.2: Interactions in the MAS

3.2.3.1 Infrastructure's Problem

The revenue Υ earned by the infrastructure by selling information goods is based on the usage fee charged on a per request basis for the agents served. The information good is a vector of expected QoS components (usually a mixture of performance and reliability guarantees) and is denoted x_i for agent i. Some examples of QoS components are *response time, queue lengths, loss probability, mean time to failure (MTTF) and availability.* Let $\hat{a}$ be an allocation of n agents and β^p the repair rate provided by the principal. The total expected revenue per unit time is

$$\Upsilon(\hat{a},\ \beta^p) \;=\; \sum_{i=1}^{n} \lambda_i \{p_i x_i(\hat{a},\ \beta^p) - \hat{p_i}\hat{x_i}(\hat{a},\ \beta^p)\}. \tag{3.1}$$

Equation (3.1) forms the objective function of the optimization problem of the infrastructure. Usually, the infrastructure control the tuple $(\hat{a},\ \beta^p)$. The infrastructure is expected to satisfy the agents' constraints as well as those of its own. We discuss all the constraints in the context of the mechanism in Chapter 6. $\hat{p_i}\hat{x_i}$ can be thought of as a production or penalty cost. This cost could be such that it drastically brings down the Principal's profit, if it delivers poor service. For example, one component of $\hat{x_i}$ could be response time which will be high if the infrastructure delivers poor performance. As seen in Chapter 6, we will predict $\hat{x_i}$ using the same queueing model. However, it remains to be seen as to how $\hat{p_i}$ can be computed. We will assume that $\hat{p_i}$ is known for each agent i and only compute p_i. (However, we offer some intuitive explanation[1] as to how $\hat{p_i}$ can be computed).

---

[1] Since we are talking about software agents, $\hat{p_i}$ could be a function of qualitative aspects such

Given this pricing structure, the variables under the control of the infrastructure are $\hat{a}$, β^p and p_i. By controlling these variables the infrastructure ensures that the application traffic is well-behaved while accumulating revenue for the performance and reliability offered. We will provide details on how to compute each of the above in Chapter 6.

3.2.3.2 Application's Problem

The application's utility[2] U depends on its valuation of the negotiated contract $x = \{x_1,\ x_2,\ \ldots,\ x_n\}$ and the micro-payments $p = \{p_1,\ p_2,\ \ldots,\ p_n\}$ it has to make to the infrastructure. In other words, x is the QoS bundle consumed by the entire application and p is overall payment per request. purchased by it. The total utility to the application per unit time is

$$U(\lambda,\ b,\ \theta,\ x) = \sum_{i=1}^{n} \lambda_i \{\theta_i x_i - p_i(b)\}$$

where θ_i is the true type of agent i, $b = \{b_1,\ b_2,\ \ldots,\ b_n\}$ are the bids. Notice that price paid depends the market valuation of the good - i.e. p depends on all bids in b. This utility is not optimized centrally. In fact, the bids b_i are the agents' private information. x_i is predicted using the model. The $\lambda_i$values are computed using historical information (say, from the arrival rates from previous time periods) by the principal. Table 6.1 summarizes the parameters controlled by the application

---

as the type of the agent (i.e. defaulting on the QoS of mission-critical agents results in loss of revenue). Another factor that could affect $\hat{p_i}$ is its memory footprint which we do not explicitly model. Generally speaking, the computation of this quantity can take into account marketing and historical data, especially when we are discussing about commercial infrastructure. Understandably, users in the commercial realm migrate away from web-servers if the response time is too high. This results in loss of revenue to the infrastructure which can be reflected in the penalty $\hat{p_i}$.

[2] also called consumer surplus (CS)

Table 3.1: Parameters controlled by the Application and the Infrastructure for Mechanism A

Controlling Entity	Controlled Quantities
Application / MAS	$\{b, \beta^a\}$ i.e. the bids and the associated repair rates
Infrastructure	$\{\ p,\ \ \beta^p,\ \hat{a}\}$ i.e. the infrastructure controls the prices, the assigned repair rates and the allocation

and the infrastructure.

3.2.3.3 Who controls what?

While we are not talking about a control problem, we have to specify the roles or who is in charge of which action. For the most part, the *mechanism design* problem answers this question. However, we make the following comment. Once the bids b_i are announced, every aspect of the allocation is in the Principal's control. Notice that this case is not unlike the *take-it-or-leave-it* option usually provided to the customer - there is no room for negotiating the contractual parameters. There are checks in place to ensure that the principal is truthful. For example, the penalties $\hat{p}$ are precisely for this reason. Moreover, the agents' can choose not to participate. But the main point is that it is too complex computationally for the principal to allocate the agents because it faces a combinatorial optimization problem which is known to be NP-Hard because it can be reduced to the GAP [80]. We provide heuristics to mitigate this problem. But in a different vein, we also propose a paradigm where the application is in almost full control of the allocation (rather than the infrastructure). We call this the *negotiation problem* where the role of the principal is to accept or reject an offer made by the application. We will revisit this topic in detail in Chapter 6.

3.3 Overview of UltraLog

A distributed (software) application can be considered as consisting of several components allocated on a distributed (service-providing) infrastructure, each performing its pre-established function. Each component (or *agent)* or group of components is located on a computing resource (eg. a server) whose computing power is limited. The computing powers of different resources are usually non-homogeneous[3]. Each agent can be thought of as a user supplying requests to the infrastructure node on which it is located. Each infrastructure node processes tasks from its users thereby providing service. The varying processing requirements of the tasks imposes load on the infrastructure. Although the memory footprint of agents may also cause loading of the infrastructure, we assume that processing causes greater loading and hence is the subject of our analysis. Furthermore, we consider that the computing resources on which the agents are located are under attack. These attacks may cause temporary (and so recoverable) loss of processing capability or permanent loss of information. In case of permanent damage, the only recourse is to replace the agent (by re-spawning it), sometimes on a different computing resource. Every node is equipped with capability to recover from some damage. Systems that are capable of recovering from damage incurred while operation are sometimes referred to as *self-healing* systems. We are interested in analyzing the performance of every such computing resource.

In the next section, we discuss the design process of a real multi-agent system. The goal is to explain by way of example terms such as classes of tasks, class-switching, TechSpecs and so on.

---

[3] i.e. servers can vary in the processing capability of their CPUs and operating conditions.

3.4 DMAS based Application Development: A Case Study

If the modeling problems we state in Chapter 1 are of immediate interest, readers may choose to revisit this section. In this section, we illustrate the development of a multi-agent application in military logistics and examine the relevance of specifying an agent system using a case-study called CPE. This section would serve a quick primer for the TechSpecs based MAS design.

Within the research domain of military logistics, we are conducted our studies using a continuous planning and execution (CPE) agent society. The CPE society is constructed using the COUGAAR MAS development platform developed under DARPA's leadership [76]. From the modeling perspective, the CPE society (or otherwise) is nothing but a collection of distributed agents that lend themselves to be represented by a network of queues. With this motivation, we analytically modeled the CPE society using queueing theory. In doing so, we realized that if the TechSpecs were suitably specified, the generation of the queueing model could be accomplished with lesser human intervention. The primary function of the model is to help evaluate the performance of the MAS and provide alternatives to steer the agent society towards optimal regions of operation boosting performance in a distributed environment. Therefore the main focus of this research lies in specifying the MAS in a systematic fashion so that queueing models can be derived from the specification.

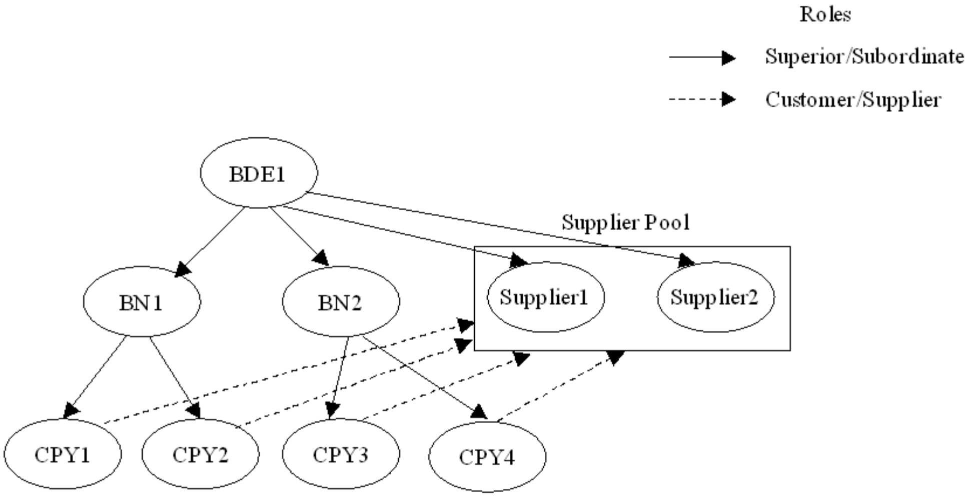


Figure 3.3: Agent Hierarchy in CPE Society

3.4.1 Continuous Planning and Execution Society Overview

The CPE society comprises of agents and a world model. Agents in the CPE society assume a combination of command and control, and customer-supplier roles as required in a military logistics scenario. The world model is an artificial source that provides the agents with external stimuli. Figure 3.3 represents the superior-subordinate and the customer-supplier relations between the brigade (BDE), battalion (BN), company (CPY) and supplier (SUPP) agents as modeled in this research. Each agent in the society constantly performs one or more of the following tasks: 1) Evaluates its own perception of the world state through local sensors and remote inputs; 2) Performs planning, re-planning, plan reconciliation and plan refinement; 3) Executing plans, either through local actuators or through sending messages to other agents; 4) Adapting to the environment, e.g. centralizing or decentralizing planning as computational resources permit.

3.4.2 Terminology

The following definitions are in order when relating to the system under consideration.

Stresses occur due to the operation of the MAS in battlefield environments where events such as permanent infrastructure damage and information attacks adversely affect overall system performance.

Based on the planning activity in CPE, we simply base our *measures of performance (MOPs)* on timeliness or freshness of a plan at the point of usage and on the quality of the plan. Based on the requirements of UltraLog [3], a broad series of performance measures categorized according to timeliness, completeness, correctness, accountability and confidentiality is available but is outside the requirements of CPE. Some insights about these MOPs can be gained from [81]. The MOPs are the components of the *quality of service (QoS)* expected from the system.

Survivability of a distributed agent based system (or otherwise) is the extent to which the quality of service (QoS) of the system is maintained under stress [81].

Although we consider a survivable MAS, we only concern ourselves with performance analysis in this work. We assume that a global controller exists that coordinates between threads relating to performance, robustness and security. The contents of this paper are organized in the following way. In Section 3.4.3, we introduce the concept of TechSpecs based design and some of the benefits associated with this approach. We then discuss the components of the CPE society in detail and organize the TechSpecs for CPE into various categories in Section 3.4.4. The discussion on TechSpecs leads us further in the direction of how to utilize them to form models. We discuss an analytical method using queueing networks to model a small example in CPE in Section 3.4.6 and verify it using an Arena simulation.

3.4.3 The Concept of TechSpecs Based Design

Technical Specifications (or TechSpecs) refer to component-wise, static information relating to agent input/output behavior, operating requirements, control actions and their consequences for adaptivity [82]. In addition to outlining a comprehensive set of functionalities, the TechSpecs are responsible for the definition of domain MOPs, their respective computational methodologies and QoS measurement points. The construction of TechSpecs helps us proceed in the following direction:

1. Use the *specs* to ensure a close mapping between MAS functionality and an abstracted model. An apparent choice here is a queueing model because of similarities between multi-class traffic in queueing networks and the different types of *flows* in CPE.

2. Establish the parameters of the queueing model - from TechSpecs directly (eg. update rate at a node) as well as by collecting empirical data from sample runs (eg. processing times).

3. As the queueing model provides an indication of system performance for a given configuration, use it to quickly explore options for control (choices resulting from adjusting (queueing) parameters or configurations). Once a suitable candidate is obtained, this choice is translated back into the application level knob settings (for control) to result in better QoS for the MAS.

The direction that TechSpecs motivates us to take is illustrated in Figure 3.4. Figure 3.4 indicates that we could use the *specs* in an online or offline fashion. Because the functionality is clearly defined using TechSpecs, offline analysis can be independently carried out to remove instabilities from the MAS design. Assuming automatic

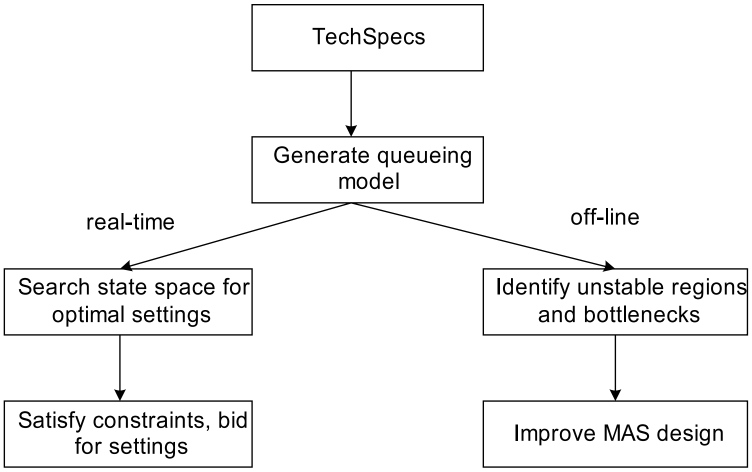

TechSpecs
Generate queueing model
real-time
off-line
Search state space for optimal settings
Identify unstable regions and bottlenecks
Satisfy constraints, bid for settings
Improve MAS design

Figure 3.4: TechSpecs based MAS Design

conversion from TechSpec to a model is feasible, TechSpecs have a real-time use as well - i.e. use the *specs* as a template to derive the model. As noted above, the candidate parameters from the queueing model (parameters that may lead to performance improvement) cannot be used directly. Reconverting these choices to actual control knob settings may be handled by a separate global controller. We allude to this in Section 3.4.5.

It can be noted that the idea of TechSpecs bears analogy to the conventional control problems in electronic or hardware realms where the technical specification or rating could be leveraged to effect better design and control. This was one of the motivating factors for TechSpecs based design for MAS.

3.4.4 CPE Society TechSpecs

In this section we discuss the formulation of TechSpecs. In order to build TechSpecs, the functionalities of the components of the CPE society are defined. We then categorize the capabilities of CPE components in a manner that would lend itself to easy

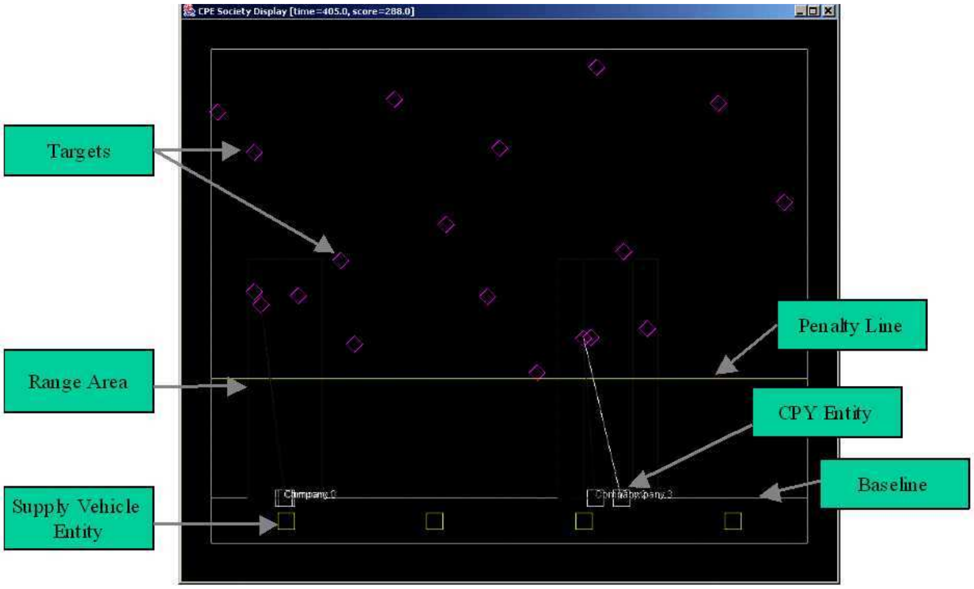


Figure 3.5: The World Model

translation into the queueing models. We then show through examples how the mapping process between TechSpecs and a queueing model could be interpreted. This would enable us to analyze the MAS using the models we develop in Section 3.4.6.

3.4.4.1 Description of CPE Society Components

- The World Model: The world model refers to the conceptual set-up that provides the agents with external stimuli. It captures a military engagement scenario using a 2-dimensional model of the world. As shown in Figure 3.5, CPY agents moving along the x-axis engage an unlimited supply of targets that move along the y-axis. The targets move at a fixed rate but engagement slows them down. While a probabilistic model is chosen to create targets and engaging them, a deterministic model is chosen for fuel consumption (which is dependent on the distance moved). A logistics model for resupplying the units with fuel or ammunition is based on the demand generation from maneuver plans.

Currently, the world model is also implemented as an agent.

- CPY Agent: Each CPY unit is designated a target area for engaging in combat actions. These action require a superior agent (BN) to supply a maneuver plan to each of the CPY agents. This plan enables the CPY agent to move along the x-axis and engage the enemy by firing. Each of these agents simulate sensors and actuators. The CPY agents consume resources and subsequently forward the demand to SUPP agents. The current status is reported to superior agents to enable replanning.

- BN Agent: The BN agent maintains situational awareness of all the agents under its direct command and performs (re)planning for them using a consistent set of observations that is collected continuously. The BN agent has to execute a branch and bound algorithm of a specified planning depth and breadth to generate a maneuver plan for its subordinates. The BN agent serves as a medium for transferring orders from superiors to subordinates.

- BDE Agent: The BDE agent is responsible for generating maneuver plans for the BN and CPY agents although this implementation does not empower the BDE with that functionality.

- SUPP Agent: SUPP agents represent an abstracted set of supply and inventory and sustainment services. These agents take maneuver plans from the CPY agents and supply them with fuel or ammunition. It is currently assumed that the SUPP units have infinite inventory. Projected and actual consumption depend on the sustainment plan generated from orders and the presence of enemy targets.

3.4.4.2 TechSpecs Organization

Right at the outset, our goal is to embed enough transparency in the TechSpecs to allow the generation of models (queueing models). Hence, we extract the input/output behavior, state, actions and QoS for each entity within CPE and form the following categories within the TechSpecs :

- Internal State of an Agent: Corresponds to continuously updated variables or data structures corresponding to the actual working of the agent.
- Inputs: Relates to distinct classes of information received or sent to or from an agent respectively.
- Outputs: Information provided to other agents.
- Actions: Determines the actions that need to be taken as a result of state changes or the dependencies introduced by input/output operations.
- Operating Modes: The fidelity or the rate at which outputs are sent may relate to the operating mode of an agent. Switching operating modes may be necessary to alter QoS requirements or as counter-measure for stress.
- QoS Measurement (QoS Measurement Points): Indicates the measure of performance that needs to be monitored or measured in order to compute the QoS at the designated measurement point. For example, when we consider queueing models, we would be interested in measuring the average waiting times at different agents to compute a quantity such as the freshness of the maneuver plan.

- Tradeoffs: While these may not pertain to every agent, some agents have the capability to trade-off a certain measure of performance to gain another. These are specified explicitly in TechSpecs.

This categorization facilitates the delineation of specific *flows* of jobs between agents. For example, consider the following *flow*: External stimuli at CPY gets converted to update tasks at CPY, delivered to BN as updates, converted to a maneuver plan at BN, delivered to CPY and then forwarded to SUPP for sustainment. From a queueing theory perspective, the update tasks that originates at CPY and end up at BN for the purpose of planning could constitute a class of traffic with CPY and BN acting as servers to process these tasks. Similarly, consider the flow where external stimuli received at CPY end up as updates at BDE through BN. This could be regarded as another class of traffic. At this point it is important to notice that classes of traffic could be derived form the input/output details embedded within TechSpecs. We describe how we handle these flows in the queueing network formulation in Section 3.4.6.

Another example of how we could describe something in the application domain (say a QoS metric) with the queueing model is as follows. If one is interested in how fresh a maneuver plan is at its usage point (i.e. CPY), the model could describe it in terms of the queueing delays for a particular class of traffic. In our application, this very quantity happens to be a QoS metric called maneuver plan freshness. In the actual MAS, this metric is calculated directly from the timestamps that are tagged to the tasks.

3.4.4.3 TechSpecs Representation

Although an elaborate discussion of the format of TechSpecs representation is outside the scope of this discussion, we present some aspects of the specification directly relating to the application and some infrastructural requirements that need to be part of the specification.

Table 3.2 represents some TechSpecs categories specific to this application. Simply speaking, this is a tabular representation of the information contained in Section 3.4.4.1 organized using the aforementioned categories. From Table 3.2 one can understand that an output called *update* originates from CPY agent and travels up at BN because BN is CPY's superior. Similarly, an output called *maneuver plan* would reach CPY from BN. One assumption that is being made here is that *updates* travel up the hierarchy and *plans* downward. These outputs form part of the different classes of traffic if observed from a queueing perspective. Another example would be that the *plan action* in the BN agent relates to a functionality in the MAS domain and would simply be abstracted by a *processing time* in the queueing domain.

In addition to the above specification, static requirements of the agents in terms of infrastructure are also embedded into TechSpecs. Some of these requirements for BDE, BN, CPY and BDE agents shown in Table 3.3.

3.4.5 Translating TechSpecs to the Queueing Domain

In order to translate the specs into queueing models we first use the following rules:

1. Inputs and outputs are regarded as tasks;
2. The rate at which external stimuli are received is captured by the arrival rate (λ);

Table 3.2: TechSpecs Categories: Application Perspective

Property / Attribute	BDE	BN	CPY	SUPP
Superior	-	BDE	BN	CPY
Internal State	Overall Status	World State	Maneuver Plan, Fuel, Ammunition	Sustainment Plan, World State
Inputs	Update	Update	Update, Maneuver Plan	Maneuver Plan
Outputs	Update	Update, Maneuver Plan	Update, Maneuver Plan	Maneuver Plan
Actions	Update	Plan, Update	Update	Plan, Update
QoS Measurement Points	-	-	Maneuver plan freshness (MPF)	Sustainment plan freshness (SPF)
Operating Modes	-	High, Medium, Low	High, Medium, Low	-
Operating Mode Trade-offs	-	Planning depth versus breadth	-	-

3. Actions take time to perform so they get abstracted by processing times (μ_i);

4. QoS Metrics such as freshness are in terms of average waiting times at several nodes ($\sum W_{ij}$, i is the node, j is the class of traffic);

5. If tasks follow a particular route (or flow as described in Section 3.4.4.2), then that route gets associated to a class of traffic;

6. If a particular task goes into the node and gets converted to another task, we say class-switching has occurred. For example, in our application *update* tasks go to BN and get converted to *plan* tasks;

7. If a connection exists between two nodes, this is converted to a transition prob-

Table 3.3: TechSpecs: Infrastructure Perspective

Property/Attribute	Node	Agent	Plug-in
hasProcessor	Yes	depends on node	depends on agent
hasBandwidth	Yes	depends on node	depends on agent
Processor Speed	update	depends on node	depends on agent
Bandwidth	1 Mb/s	depends on node	depends on agent
Location	IP address	NodeID	AgentID
Operating System	Win2000/Linux	-	-
Memory	1GB	depends on node	depends on agent

ability p_{ij}, where i is the source and j is the target node.

Using the above rules as well as the aforementioned representations of TechSpecs we develop a mapping between the TechSpecs and a queueing model. Although the current procedure is manual, in theory this procedure could be automated. Such an automatic capability of translating TechSpecs would prove very beneficial for predicting performance of the MAS in real-time. Table 3.4 captures the queueing model abstraction from TechSpecs for the CPY agents. Similarly, we can establish the mapping for other agents as well. Some useful guidelines that were followed in order to translate the TechSpecs into models are as follows:

- Identify *flows* of traffic: Trace the route followed by each type of packet completely within the system boundary i.e. from the entry into the system until it exits the system. These would subsequently form classes of traffic in the queueing model. Care has to be taken to note any class switching.

- Identify the network type: The network could be closed (fixed number of tasks) or open. The CPE is an open system because tasks constantly enter and exit the system.

- Does any parameter of the model require empirical data from the actual society?

Although some aspects in this research are currently being resolved, the following observations can be made.

- Who does the TechSpecs translation? Where does the model run? In our case the translation is done manually at present. The model would run at a place visible to the controller (possibly as a separate agent at the highest level). The controller we refer to here is the actual effector of control actions throughout the CPE society and is separate from all we have discussed so far. The role of the controller is also to balance between other threads such as robustness and security.
- The identification of control alternatives is currently centralized. However, we visualize a decentralized, hierarchical controller for effecting the changes.

3.4.6 Queueing Network Models (QNMs)

A complex logistics system such as the CPE society has numerous interactions. Yet, if the functionalities are abstracted to capture some application level specifics in terms of queueing model elements (example as shown in Table 3.4), analytical predictions on the behavior of the MAS can be made. Analytical models are good candidates for enforcing adaptive control quickly and in real-time. Each agent behaves like a server that process jobs waiting in line. Hence, the mapping between an agent and a server with a queue is easily established. Because of the task flow structure and the superior-subordinate relationships in the TechSpecs, queues can be connected in tandem with jobs entering and exiting the system. This results in the formation of an open queuing network.

We conducted initial experiments using an actual COUGAAR based MAS, an analytical formulation and an Arena simulation. We used this experiment to bootstrap our modeling process in terms of parameter estimation and calibration. However, working with the MAS was time-consuming as our goal was to identify modeling alternatives and control ramifications. Hence we continued our experimentation with a scaled up queueing model and simulation with the insight gained from working with the actual society.

Thus the open queueing network's parameters were carefully chosen and tasks sub-divided into mutiple-classes to denote a particular task within the MAS. The TechSpecs clearly delineate the input and output tasks facilitating the mapping to arrivals and services in a queueing network. Application level QoS measures of the MAS are calcuated in terms of the waiting times (or other equivalent perfromance measures) at the individual nodes of the QNM.

Figure 3.6 is a representation of the CPE society from a queueing perspective. We show two types of tasks flowing in the network namely the *plan* (denoting maneuver and sustainment) and the *update* tasks. These tasks can be divided further into three classes of traffic. The first class refers to *update* packets entering at the CPY nodes and proceeding further as *updates* to BDE through BN. Class 2 relates to those *update* packets that are converted to *plan* tasks. There is class-switching at nodes 2 and 3 and we introduce approximations to deal with this later in the paper. The third class relates to the maneuver *plan* tasks that reach SUPP nodes through CPY. Although we know multiple task types exist in the MAS, by making the simplifying assumption and treating all job classes alike we analyze the MAS using Jackson networks [83] in Section 3.4.7.

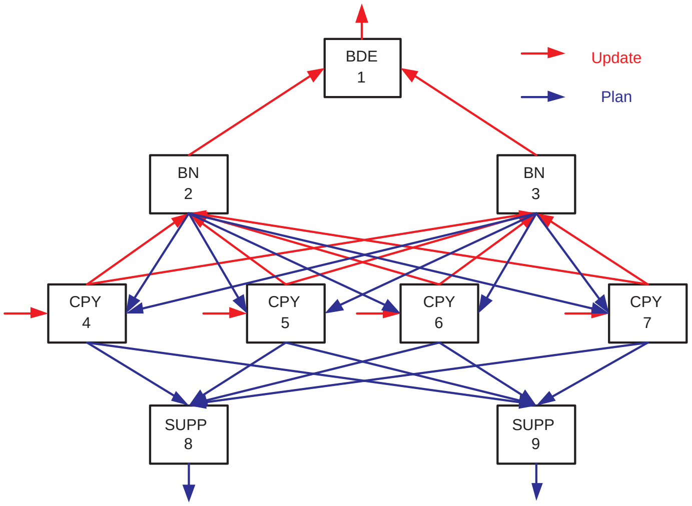


Figure 3.6: Task Flow in the MAS

3.4.7 Jackson Network Model

We apply a single class Jackson network [83] formulation for open queuing networks to our example by choosing a weighted average service time for nodes with multiple classes. The nine agents of the MAS considered here can then be assumed to be M/M/1 systems. The arrival rates of the open network can be computed by solving the traffic equations. Assuming the load is balanced to start-with, the routing probabilities are also known. If each node of the system is ergodic, we can calculate the steady state probabilities and performance measures of the entire network by computing these measures for every agent exactly as in an M/M/1 system.

We consider a simple example. For this queueing model, we assume all tasks are of a single type and do not distinguish between classes as shown in Figure 3.6. Let λ_{0i} and λ_{i0} be the rate of arrival and exit into and from the i^{th} node respectively. Since the routing probabilities are known we can calculate the arrival rates λ_i of each of the nodes of the open network by solving the following traffic equations:

$$\lambda_i = \lambda_{0i} + \sum_{j=1}^{9} \lambda_j p_{ji} \,, \; i = 1, ..., 9 \,.$$

The routing probabilities (p_{ji}: probability from i (column index) to j (row index)) for the balanced case are as follows:

$$\begin{matrix}
0 & 1/5 & 1/5 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1/4 & 1/4 & 1/4 & 1/4 & 0 & 0 \\
0 & 0 & 0 & 1/4 & 1/4 & 1/4 & 1/4 & 0 & 0 \\
0 & 1/5 & 1/5 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1/5 & 1/5 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1/5 & 1/5 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1/5 & 1/5 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1/4 & 1/4 & 1/4 & 1/4 & 0 & 0 \\
0 & 0 & 0 & 1/4 & 1/4 & 1/4 & 1/4 & 0 & 0
\end{matrix}$$

Note that the customer exits from a node i with probability $1 - \sum_j p_{ji}$. Once the arrival rates are known, we can calculate the average waiting times at the nodes by using the following formula:

$$W_i = \frac{1/\mu_i}{1 - (\lambda_i/\mu_i)} \,, \; i = \{1, ..., 9\}.$$

The QoS metrics namely maneuver plan freshness (MPF) and sustainment plan freshness (SPF) are calculated in terms of the average waiting times of the nodes at each level (W_{CPY}, W_{BN}, W_{SUPP}) as follows:

$$MPF = 2W_{CPY} + W_{BN} \,,$$

$$SPF = 2W_{CPY} + W_{BN} + W_{SUPP} \,.$$

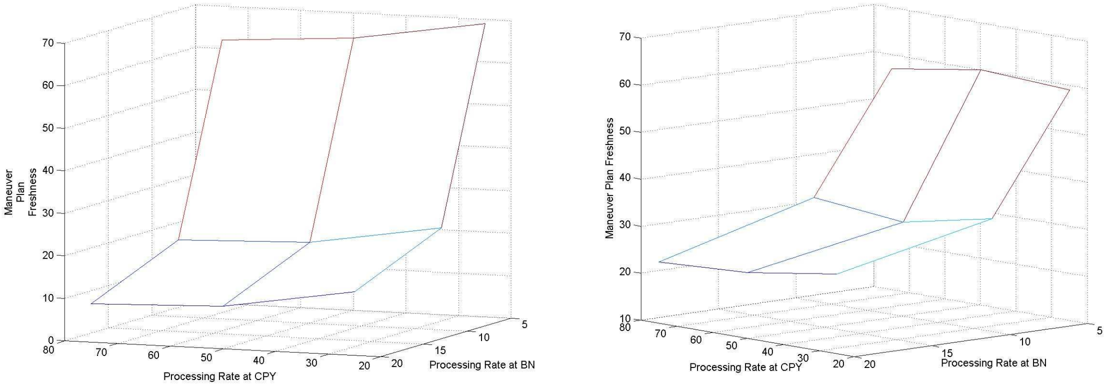


Figure 3.7: Maneuver Plan Freshness using Jackson Network

If the load is not balanced and the waiting times are different for the different branches, the QoS measures are accordingly calculated. It can be observed that two methods of control are straightaway obvious: 1) Adjust the μ_i so that we could process faster if possible, 2) Alter the transition probabilities p_{ji} to divert traffic to nodes that are less loaded.

We studied the impact of changing the processing rates at the nodes to illustrate the benefit of deriving a online queueing model that could form an integral part of a controller. The Jackson network model is used to compute the maneuver plan and sustainment plan freshness from the average waiting times of the individual nodes. We assume the processing rate for *class 1* tasks, $\mu_{update\_tasks} = 10Mb/s$ at all the nodes. We assume that the overall arrival rate from the environment is according to a Poisson Process with $\lambda = 2\,Mb/s$.

Table 3.4: Queuing Model Abstraction from TechSpecs for CPY Agent

TechSpecs Entry	Description	Queueing Model Abstraction
Update (input)	input called update arrives from BN	update packet enters CPY node from environment
Update (output)	update packet leaves CPY and enters BN	route exists from CPY to BN for update packet, reflects as a probability of transition
New Maneuver Plan (input)	Input received at CPY from BN, yet to be processed	plan packet enters CPY and waits in queue
Maneuver Plan (output)	output from CPY to SUPP. Plan is processed.	Plan packet is ready to exit queue after service
Update (action)	perform this action at CPY	maneuver plan is processed at a particular rate. determined by experiments.
Maneuver Plan Freshness (QoS point)	compute this QoS at this node by respective agent sing timestamps for the flows.	sum of waiting times in the system that the packet has visited
Status update rate (operating mode)	rate at which stimuli from the environment is received	queueing model's arrival and processing rate is adjusted

Chapter 4

Analyzing a Single-Class Agent-Based System

4.1 Introduction

In this chapter, we develop a performance model for an agent-based system that can cope with different types of breakdown (in service) and process tasks all of the same kind. The main motivation for developing this model is to analyze agents functioning on a single computing resource (or CPU node) and to quantify the impact of their self-healing behavior on performance. We assume that the CPU infrastructure on which agents function is inherently unstable, which is typical in most computing domains now-a-days. As a result, the waiting line models developed here are applicable to other computing domains prone to interruptions in service.

The basic premise is that if an agent can influence (by requesting the service provider, negotiating etc.) how frequently or quickly a breakdown is repaired, it can improve the performance of the system. Software components, and in particular, agent-based systems, exhibit many adaptive capabilities among which self-healing is crucial. Due to harsh operating conditions, the agent system should be able to heal from different kinds of failure having varying impact on the service received. The goal in this chapter is to explicitly factor two kinds of failure (see Section 4.1.2) into the model and gauge the failures' impact on performance. To validate our claim that self-healing affects the measures of effectiveness of the agent, we propose waiting line models to predict the extent to which performance and availability are affected. Subsequently, in Chapter 6 we illustrate how the agents can use these predictive models internally for decision-making in the context of an multi-agent allocation problem.

4.1.1 Agent-based computing in harsh environments

A distributed (software) application can be considered as consisting of several components allocated on a distributed (service-providing) infrastructure, each performing its pre-established function. Each component (or *agent*[1]*)* or group of components is located on a computing resource (eg. a server) whose computing power is limited. The computing powers of different resources are usually non-homogeneous[2]. Each agent can be thought of as a user supplying requests to the infrastructure node on which it is located. Each infrastructure node processes tasks from its users thereby providing service. The varying processing requirements of the tasks imposes load on the infrastructure. Although the memory footprint of agents may also cause loading of the infrastructure, we assume that information load and processing demanding tasks stresses the resources more. Furthermore, we consider that the computing re-

---

[1] An agent based intelligent supply chain is the application considered.

[2] i.e. servers can vary in the processing capability of their CPUs.

sources on which the agents are located fail frequently due to the heavy loads and other disruptions. There is either temporary (and so recoverable) loss of processing capability or permanent loss of information. In case of permanent damage, the recourse is to replace the software agent (by re-spawning it), sometimes on a different computing resource. Every node is equipped with capability to recover from some damage by performing software reconfiguration. This recovery process is abstracted for the purpose of modeling using the term *repair*. Such systems that are capable of recovering from damage incurred during operation are referred to as *self-healing* systems. In this chapter, *we are interested in analyzing the performance and availability of a self-healing agent from the queueing persepctive.*

4.1.2 Definitions

We will now describe a few terms that we will use throughout. Depending on the types of breakdown and the processing requirements for tasks, the choice of model will vary. However, we will assume that all tasks have similar processing requirements (i.e. single-class) in this chapter. We consider multi-class traffic in Chapter 5. Further, we assume that there are two kinds of breakdown.

Definition 4.1.1. Catastrophic Breakdown: A catastrophic breakdown in a queue will lead to loss of tasks in the queue. Service will be disrupted until the agent is restarted.

Definition 4.1.2. Temporary Breakdown: A temporary breakdown in a queue will lead to the queue's capability to service tasks. However, the queue will continue to accept tasks and maintain state information.

This means that the queue is in a state of partial failure and hence is not fully operational.

Definition 4.1.3. Class of traffic: A class of traffic refers to the particular type of tasks whose service time requirement, x will conform to a given probability distribution $F(x)$.

A model that contains both these two types of breakdown is described below.

4.1.3 Performance model with temporary and catastrophic breakdowns

In this model, we have both catastrophic and temporary breakdowns. We have a queue whose state-space is a combination of a birth-death process, a (pure) birth process and a breakdown state. There states constituting the birth only proces are the states in which the system is in temporary breakdown. However, while in these states the number in queue is increased by newly arriving tasks. Tasks arrive according to a Poisson process $PP(\lambda)$ and get serviced for a mean time of $1/\mu$ distributed exponenentially. The queue goes into breakdown after a mean time of $1/\gamma$ and gets repaired from the down state after a mean time of $1/\delta$, both times distributed exponentially. In addition, there are transitions to states of temporary breakdown after an exponentially distributed mean time of $1/\alpha$. From these states, the system heals after a mean time of $1/\beta$, distributed exponentially. The rate diagram for this model is shown in Figure 4.1. In Figure 4.1, T and D refer to temporary and catastrophic breakdown respectively.

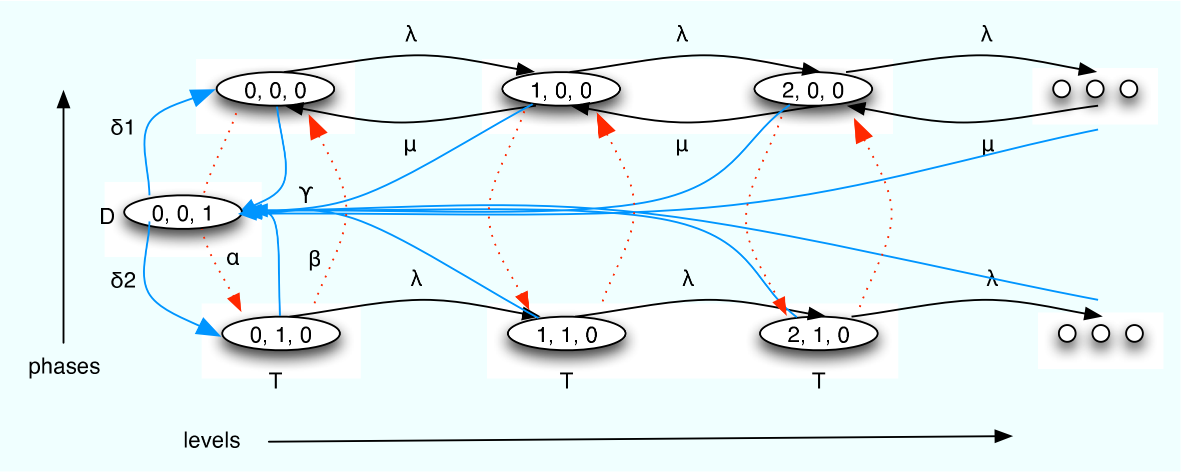


Figure 4.1: Rate diagram of a system with catastrophic and temporary breakdown

4.1.4 Organization of this chapter

The details of the modeling are presented in the next section. We provide a solution technique for computing the steady state probabilities in Section 4.2.2. We will obtain measures of effectiveness based on the model in Section 4.3. The provide numerical examples and validate the theoretical model against an Arena simulation model in Section 5.4. Conclusions are provided in Section 4.5.

4.2 Modeling details

We first consider a node in which all requests are identically distributed and hence belong to a single class. The arrival of tasks[3] from a particular agent i is according to a Poisson Process with parameter λ_i (represented by $PP(\lambda_i)$). Summing the arrivals from all the agents on the node, the total arrival at the node can be computed as $\lambda = \sum_{i=1}^{N} \lambda_i$ if there are N agents allocated to this node. The service time requirement

---

[3] Used interchangeably with packets to denote a quantum of work. From the queueing sense, we will refer to each task as a customer waiting in the queue.

is assumed to be exponentially distributed with mean $1/\mu$.

Each node on which tasks are processed may be under attack. Whatever the cause of attack may be, the effect of the attack is either temporary disruption or permanent destruction of the node. Every node is attacked repeatedly separated by random amounts of time, so attacks can be considered to be according to a $PP(\alpha)$ process. Since the nodes are *self-healing,* they repair some damage and are up again after an exponential time with mean $1/\beta$. Furthermore, we consider catastrophic attacks on regular or damaged nodes (according to $PP(\gamma)$) leading to loss of information with no chance of repair. In this situation, replacement of nodes is an option. In agent-based systems, replacement of agents is synonymous to restarting them on other fully (or partially) functional nodes. This process is referred to as *re-hydration* and occurs after a switch-over time distributed exponentially with mean $1/\delta$.

4.2.1 Summary of solution technique

Firstly we represent the queueing system as a Markov chain[4]. Subsequently, we write the balance equations of the system. Several generating functions are defined for different *phases* in the Markov chain. We obtain the generating functions, which sum up the state space probabilities, in terms of known quantities. In this step, a set of simultaneous equations of the form $A(z)\psi(z) = B(z)$ have to be solved, where $\psi(z)$ represents the matrix of generating functions from all the phases, and $A(z)$ and $B(z)$ are both matrices representing model parameters. After obtaining $\psi(z)$, the boundary probabilties cannot be obtained using usual techniques (see 4.2.1). First we find $z^* = \{z_1^*,\ z_2^*, ...,\ z_n^*\}$ where $Det(A(z*)) = 0,\ z^* \in [0,\ 1)$. Then, an arbitrary row j of $A(z)$ and $B(z)$ is selected into which $z^*$is plugged to give the

---

[4] Throughout, a Markov chain is assumed to be a *continuous time* Markov chain.

$A_j(z^*)\psi(z) = B_j(z^*)$. Using this equation (set of equations in general), we can obtain the boundary probabilities such as p_{000}. At this point, the expressions for the generating functions are obtained. Subsequently, the measures of effectiveness, such as W (response time in the system) and L (number of tasks in the system), are derived.

4.2.2 Analysis

This model is described by the stochastic process $\{P(t),\ Q(t),\ R(t)\}$ where $P(t)$ denotes the number in the system, and $Q(t)$ and $R(t)$ are binary variables indicating whether or not a temporary and catastrophic breakdown have occured respectively (1 implies breakdown). We assume that there is infinite room to wait and that the system can accept (but not process) packets when it is in temporary breakdown.

We can analyze this stochastic process as a continuous time Markov chain (CTMC). Let $Z(t) = \{P(t),\ Q(t),\ R(t)\} = (p,\ q,\ r)$ (for $p = 0,\ 1,\ 2,\ ...,\ q \in \{0,\ 1\}, r \in \{0,\ 1\}$). Clearly, $\{Z(t),\ t \geq 0\}$ is a CTMC with rate diagram shown in Figure **??**. The CTMC is ergodic, and for $p = 0,\ 1,\ 2,\ ...;\ q = 0,\ 1;\ r = 0,\ 1$, let

$$p_{pqr} = \lim_{t\to\infty} P\{Z(t) = (p,\ q,\ r)\}.$$

Consider the balance equations:

$$\begin{aligned}
p_{001}(\delta_1+\delta_2) &= (1-p_{001})\gamma \\
p_{000}(\gamma+\alpha+\lambda) &= p_{100}\mu + p_{010}\beta + p_{001}\delta_1 \\
p_{100}(\gamma+\alpha+\lambda+\mu) &= p_{200}\mu + p_{110}\beta + p_{000}\lambda \\
p_{200}(\gamma+\alpha+\lambda+\mu) &= p_{300}\mu + p_{210}\beta + p_{100}\lambda \\
\vdots &= \vdots
\end{aligned}$$

and

$$\begin{aligned}
p_{010}(\gamma+\beta+\lambda) &= p_{000}\alpha + p_{001}\delta_2 \\
p_{110}(\gamma+\beta+\lambda) &= p_{100}\alpha + p_{010}\lambda \\
p_{210}(\gamma+\beta+\lambda) &= p_{200}\alpha + p_{110}\lambda \\
\vdots &= \vdots
\end{aligned}$$

Let $\psi(z) = \sum_i p_{i00}z^i$ and $\phi(z) = \sum_i p_{i10}z^i$. Multiplying the above system by z^i and summing we get the following.

$$\psi(z)[(\gamma+\alpha+\lambda+\mu) - \frac{\mu}{z} - \lambda z] = (-\frac{\mu}{z}+\mu)p_{000} + \beta\phi(z) + p_{001}\delta_1 \tag{4.1}$$

$$\phi(z)(\gamma+\beta+\lambda-\lambda z) = p_{001}\delta_2 + \alpha\psi(z). \tag{4.2}$$

We know from the first equation that

$$p_{001} = \frac{\gamma}{\delta_1+\delta_2+\gamma}. \tag{4.3}$$

Let $(\gamma+\alpha+\lambda-\lambda z+\mu-\frac{\mu}{z}) \equiv (A_0-\lambda z-\frac{\mu}{z})$ and $(\gamma+\beta+\lambda-\lambda z) \equiv B_0-\lambda z$. Equation (4.1) and Equation (4.2) can be represented as

$$\begin{bmatrix} A_0-\lambda z-\frac{\mu}{z} & -\beta \\ -\alpha & B_0-\lambda z \end{bmatrix} \begin{bmatrix} \psi(z) \\ \phi(z) \end{bmatrix} = \begin{bmatrix} \mu p_{000}(1-\frac{1}{z})+p_{001}\delta_1 \\ p_{001}\delta_2 \end{bmatrix}$$

The generating functions $\psi(z)$ and $\phi(z)$ are given by

$$\psi(z) = \frac{p_{000}(\mu B_0(1-1/z)+\lambda\mu(1-z))+p_{001}(\delta_1(B_0-\lambda z)+\delta_2\beta)}{(A_0-\lambda z-\frac{\mu}{z})(B_0-\lambda z)-\alpha\beta} \tag{4.4}$$

and

$$\phi(z) = \frac{p_{000}\mu\alpha(1-1/z)+p_{001}(\delta_1\alpha+\delta_2(A_0-\lambda z-\frac{\mu}{z}))}{(A_0-\lambda z-\frac{\mu}{z})(B_0-\lambda z)-\alpha\beta}. \tag{4.5}$$

These quantities obtained by solving Equation (4.1) with Equation (4.2). p_{000} is yet be determined.

Remark 4.2.1. Obtaining boundary probabilities: Now $\psi(z)=p_{000}+p_{100}z+p_{200}z^2$..... That gives $\psi(0)=p_{000}$. But a standard technique such as this does not yield a solution for p_{000}. Any equations obtained from $\psi(z)$ or $\phi(z)$ for finding p_{000} are redundant. So p_{000} is not obtained directly. Let $D(z) \equiv (A_0-\lambda z-\frac{\mu}{z})(B_0-\lambda z)-\alpha\beta$. For some z^*, $D(z^*)=0$ as it is a polynomial. To obtain the boundary probabilty p_{000}, we try to take advantage of the zeros z^* and the following result due to [13]. For completeness, we review this result from [13].

Lemma 4.2.2. *Consider a series of real positive-valued numbers a_0, a_1, a_2, etc. For any function of the form $\phi(z)=\sum_{i=0}^{\infty} a_i z^i$ such that $\phi(z)$ can be written as a fraction*

$\phi(z) = \frac{A(z)}{B(z)}$, where $A(z)$ and $B(z)$ are polynomials, if $B(z^*) = 0$ for any $z^* \in [0, \infty)$ then $A(z^*) = 0$.

Proof. By definition, $\phi(z)$ is a continuous, differentiable and increasing function over $z \in [0,\ \infty)$. For some $z^* \in [0,\ \infty)$, let $B(z^*) = 0$. If $A(z^*) > 0$, then $\phi(z^*-) = \phi(z^*+)$ and $|\phi(z^*-)| \to \infty$. This contradicts the fact that $\phi(z)$ is continuous, differentiable and increasing function over $z \in [0,\ \infty)$. Hence $A(z^*) = 0$. (Similarly, we can prove for the case $A(z^*) < 0$.) □

Theorem 4.2.3. *The value of p_{000} is given by*

$$p_{000} = \frac{p_{001}(\delta_1 \lambda z^* - (\delta_1 B_0 + \delta_2 \beta))}{(\mu B_0(1 - 1/z^*) + \lambda\mu(1 - z^*))} \tag{4.6}$$

where z^* is the positive real root among

$$\begin{aligned} z_1 &= \frac{A_0 + B_0}{3\lambda} + S + T \\ z_2 &= \frac{A_0 + B_0}{3\lambda} - \frac{S+T}{2} + i\ \sqrt{3}\frac{S-T}{2} \\ z_3 &= \frac{A_0 + B_0}{3\lambda} - \frac{S+T}{2} - i\ \sqrt{3}\frac{S-T}{2} \end{aligned}$$

and the intermediate values S, T, R and Q are given by

$$S = \sqrt[3]{R + \sqrt{Q^3 + R^2}} \quad ,$$

$$T = \sqrt[3]{R - \sqrt{Q^3 + R^2}} \quad ,$$

$$R = \frac{\mu B_0}{2\lambda^2} + \frac{(A_0+B_0)^3}{27\lambda^3} - \frac{(\mu\lambda - \alpha\beta + A_0B_0)(A_0+B_0)}{6\lambda^3}$$

and

$$Q = \frac{3(\mu\lambda - \alpha\beta + A_0B_0) - (A_0+B_0)^2}{9\lambda^2}.$$

Proof. Using the fact the root of the denominator of $\psi(z)$ must also be the root of the numerator from Lemma 4.2.2, we seek to find the root of $A(z)B(z) - \alpha\beta = 0$. In other words, we seek to find the positive root of the cubic equation $(A_0z - \lambda z^2 - \mu)(B_0 - \lambda z) - \alpha\beta z = 0$. This equation is of the form $a_1z^3 - a_2z + a_3z - a_4 = 0$ where $a_1 = \lambda^2$, $a_2 = \lambda(A_0 + B_0)$, $a_3 = \mu\lambda - \alpha\beta + A_0B_0$ and $a_4 = \mu B_0$. The roots of this cubic polynomial can be written as above using Cardano's formula [84]. The polynomial will have at least one real root. This is the case when the discriminant $D = Q^3 + R^2 > 0$. In other cases (i.e. when $D \leq 0$), it will have more than one real root. In all cases, we have to determine $z^* \in [0, \infty)$. Rearranging the numerator of $\psi(z)$, collecting the p_{000} terms and setting it equal to zero, we get the value of p_{000} as above. □

Corollary 4.2.4. *If $z^* \in [0,1)$, $p_{000} > 0$.*

Proof. Substituting for $B_0 = \gamma + \beta + \lambda$ and rearranging Equation (4.6), we get

$$p_{000} = \frac{p_{001}z^*(\delta_1\lambda(1-z^*) + \delta_1(\gamma+\beta) + \delta_2\beta)}{(1-z^*)(\mu(\gamma+\delta) + \lambda\mu(1-z^*))},$$

from which it can be seen that $p_{000} > 0$ if $z^* \in [0,1)$. □

4.3 Measures of Effectiveness

As before, the performance measures considered are (1) the response time of the customers that receive service (Δ) and (2) the probability that a customer is lost (P_l). In both cases, a lower value means better QoS.

Theorem 4.3.1. *The average number of requests in the computing node is given by*

$$\begin{aligned} L &= \frac{p_{001}\delta_1(\beta+B_0)+p_{000}B_0\mu-2\lambda p_{001}\delta_1-\lambda p_{000}\mu}{A_1}+A_2\frac{p_{001}\delta_1(\beta+B_0)-\lambda p_{001}\delta_1}{(A_1)^2}+ \\ &\quad \frac{p_{001}\delta_2(\alpha+A_0)-2\lambda p_{001}\delta_2-\alpha p_{000}\mu}{A_1}+A_2\frac{p_{001}\delta_2(\alpha+A_0)-p_{001}\delta_2(\lambda+\mu)}{(A_1)^2} \end{aligned}$$

where the A_1 and A_2 are given by $A_1 = A_0B_0 - \lambda(A_0+B_0)+\lambda^2-\mu B_0+\mu\lambda-\alpha\beta$ and $A_2 = A_0B_0-2\lambda(A_0+B_0)+3\lambda^2+\mu\lambda-\alpha\beta$.

Proof. The number of requests in the system at steady state is given by

$$L = 0p_{000}+0p_{001}+0p_{010}+1p_{100}+1p_{010}+2p_{200}+2p_{020}+.....,$$

We can write $L=\psi^{'}(1)+\phi^{'}(1)$. Now we take the derivative of $\psi(z)$ and $\phi(z)$ and allow $z=1$. This results in the following expressions.

$$\psi^{'}(1) = \frac{p_{001}\delta_1(\beta+B_0)+p_{000}B_0\mu-2\lambda p_{001}\delta_1-\lambda p_{000}\mu}{A_1}+A_2\frac{p_{001}\delta_1(\beta+B_0)-\lambda p_{001}\delta_1}{(A_1)^2}$$

$$\phi^{'}(1) = \frac{p_{001}\delta_2(\alpha+A_0)-2\lambda p_{001}\delta_2-\alpha p_{000}\mu}{A_1}+A_2\frac{p_{001}\delta_2(\alpha+A_0)-p_{001}\delta_2(\lambda+\mu)}{(A_1)^2}$$

where A_1 and A_2 are as above. ☐

Theorem 4.3.2. *The QoS measures* P_l *and* Δ, *in terms of* L *are given by*

$$P_l = \frac{\lambda p_{001}(1 - p_{001}) + \gamma L}{\lambda(1 - p_{001})}$$

and

$$\Delta = \frac{L(1 - p_{001})}{\lambda(1 - p_{001}) - \gamma L}.$$

Proof. The number of requests that were blocked per unit time, when the server was unavailable is λp_{001}. Furthermore, the fraction that was lost because of catastrophic failure is $\frac{\gamma L}{(1-p_{001})}$ (conditioning on the fact that the server was up when the catastrophe occured). Adding thse quantities and dividing the actual arrival rate λ gives P_l as above. The net departure rate from the system is

$$\lambda^{net} = \lambda - \lambda p_{001} - \frac{\gamma L}{(1 - p_{001})}$$

So, the response time for the served customers alone is given by

$$\Delta = \frac{L}{\lambda^{net}}$$

according to Little's law, which when simplified gives the quantity above. ☐

Theorem 4.3.3. *The probability of being in termporary breakdown is*

$$p_T = \frac{\delta_1 \alpha + \delta_2(\gamma + \alpha)}{(\delta_1 + \delta_2 + \gamma)(\gamma + \alpha + \beta)}.$$

Proof. The probability of being "down" due to temporary breakdown is

$$p_{010} + p_{110} + p_{210} + \dots$$

This is nothing but $\phi(1)$ where $\phi(z)$ is the generating function given in Equation (4.5).Letting $z = 1$ in Equation (4.5) we get p_T as above. □

Theorem 4.3.4. *The availability of the agent is*

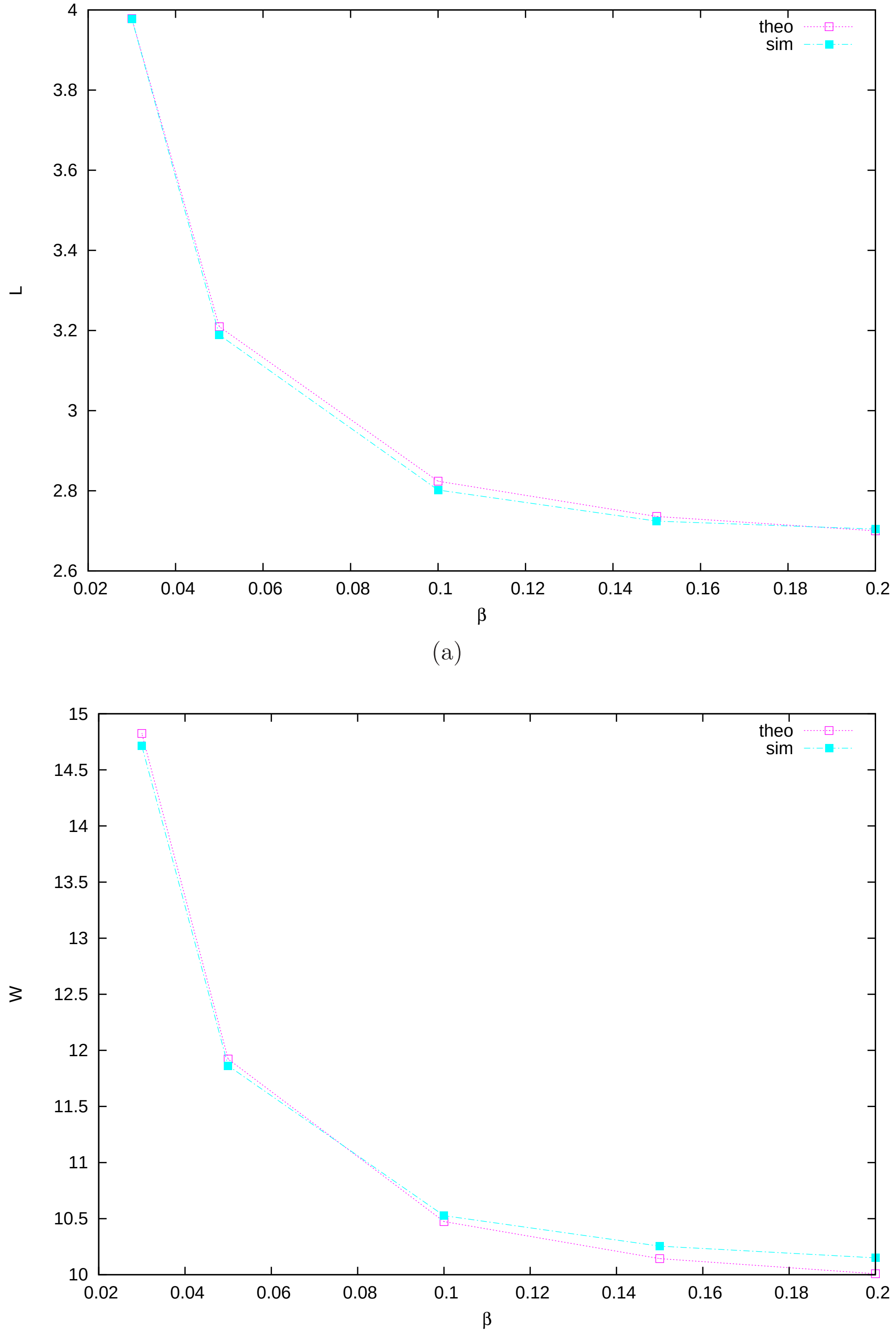
$$R = \frac{\delta_1(\gamma + \beta) + \delta_2\beta}{(\delta_1 + \delta_2 + \gamma)(\gamma + \alpha + \beta)}.$$

Proof. The availability of the system i.e. the system is available for processing incoming tasks, is nothing but $1 - p_{001} - p_T$. It is also equal to $\psi(1)$ where $\psi(z)$ is the generating function given in Equation (4.4). Allow $z = 1$ in $\phi(z)$ to get R as above. □

4.4 Numerical Examples and Validation

We now provide numerical examples for the queueing model in this chapter. In parallel, we compare these examples with a simulation model and the model in [13]. In this section, we examine the variation of some performance measures with respect to changes in β, the repair rate.

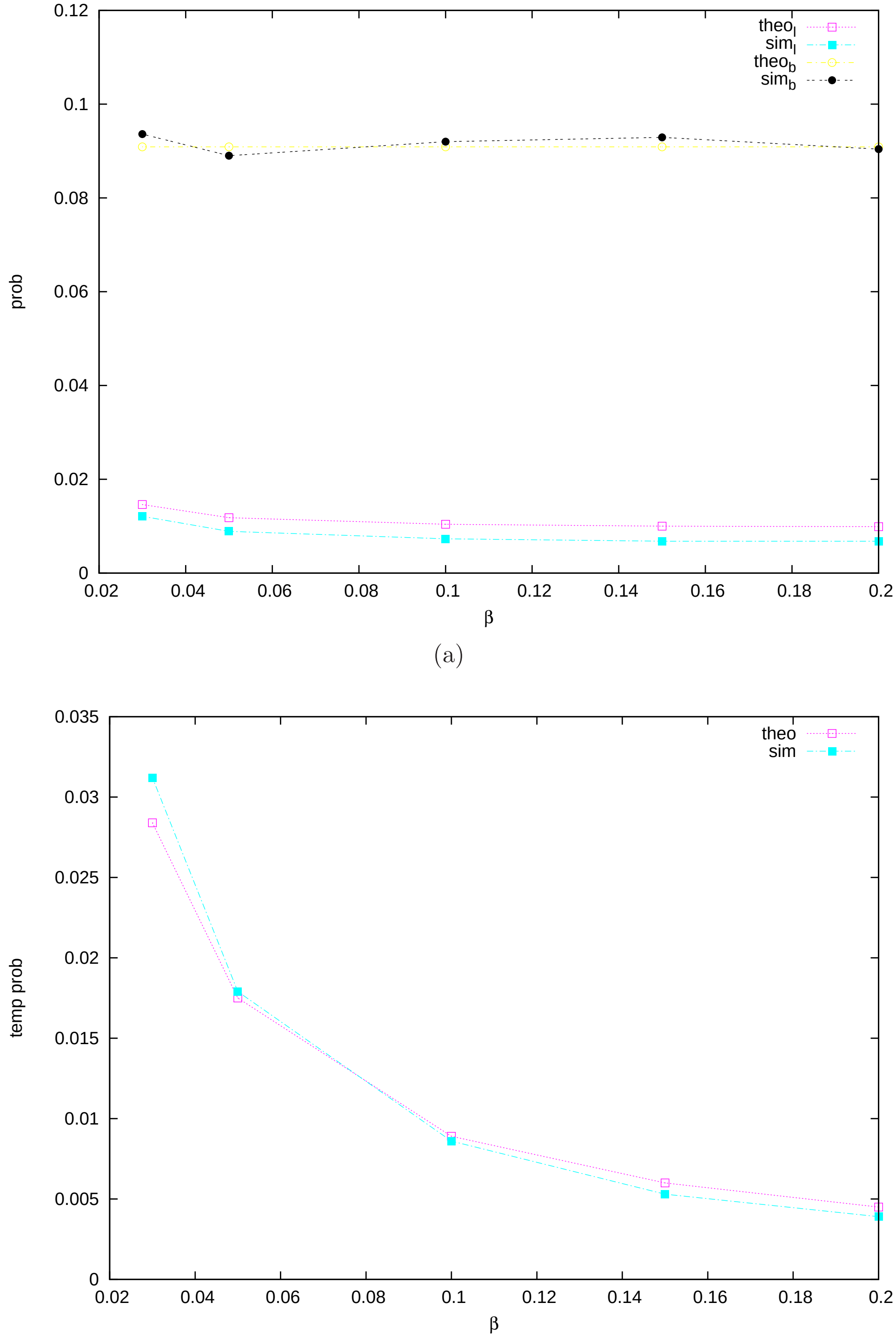
Example 4.4.1. The numerical example for the aforementioned model assumes values for $\{\lambda,\ \mu,\ \alpha,\ \beta,\ \gamma,\ \delta\}$ as $\{0.3,\ 0.4,\ 0.001,\ \beta,\ 0.001,\ 0.01\}$. β, the repair rate of the agent, is varied in the range $0.03 - 0.20$ keeing everything else fixed. It can be seen from Figure 4.2 that an increase in β would drive down the steady state average queue length and average waiting time. In Figure 4.3a, $theo_l$ and $theo_b$ refer to the loss and blocking probabilities. $theo_b$ is more-or-less flat because the number of tasks that are blocked is going to depend only on p_{001} which is same when γ and δ $(= \delta_1 + \delta_2)$ are kept constant. However, $theo_l$ depends on the number of tasks already in the queue which itself changes with β. Therefore, there is a drop in $theo_b$ as β in increased. The sum of $theo_b$ and $theo_l$ is P_l - the total loss probability of the agent. The probability that the system is in temporary breakdown (temp prob or p_T) steadily drops with increasing β as shown in Figure 4.3b. In both Figure 4.2 and Figure 4.3, the values obtained from the Arena simulation is plotted side-by-side. In Table 4.1 we show the numerical figures obtained from both the simulation and the theoretical model. We also compare the theoretical model with the model in [13]. In this comparison, we illustrate two cases. The first is when we ignore the temporary breakdown (by setting $\alpha = 10^{-5}$). In this case, the two models are almost the same but for the small probability with which temporary breakdown will occur in the model given in this chapter. The second case is when both temporary and catastrophic breakdown are ignored. This is done by setting $\gamma = 10^{-5}$ and $\alpha = 10^{-5}$. In doing so, both models are almost the same as the standard $M/M/1$ queueing model. In Figure 4.4, we examine the variation of response time W by varing λ(and hence ρ) and β, and keeping all else as stated above. As per expection, W is worst when β is minimum and ρ is maximum.



(a)

(b)

Figure 4.2: L and W versus β - theoretical and simulation



(a)

(b)

Figure 4.3: p_{loss}, p_{block}, and p_{temp} versus β - theoretical and simulation

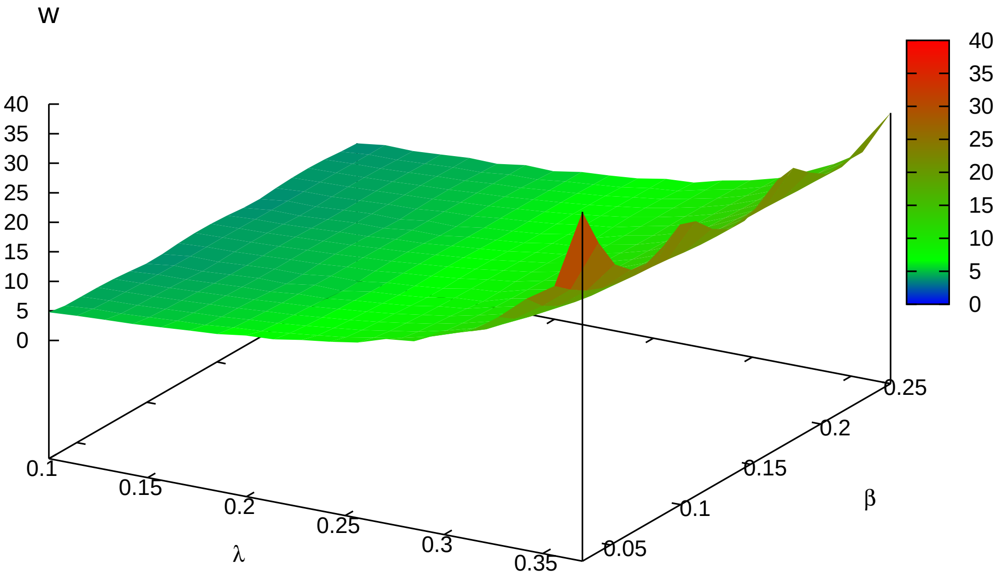
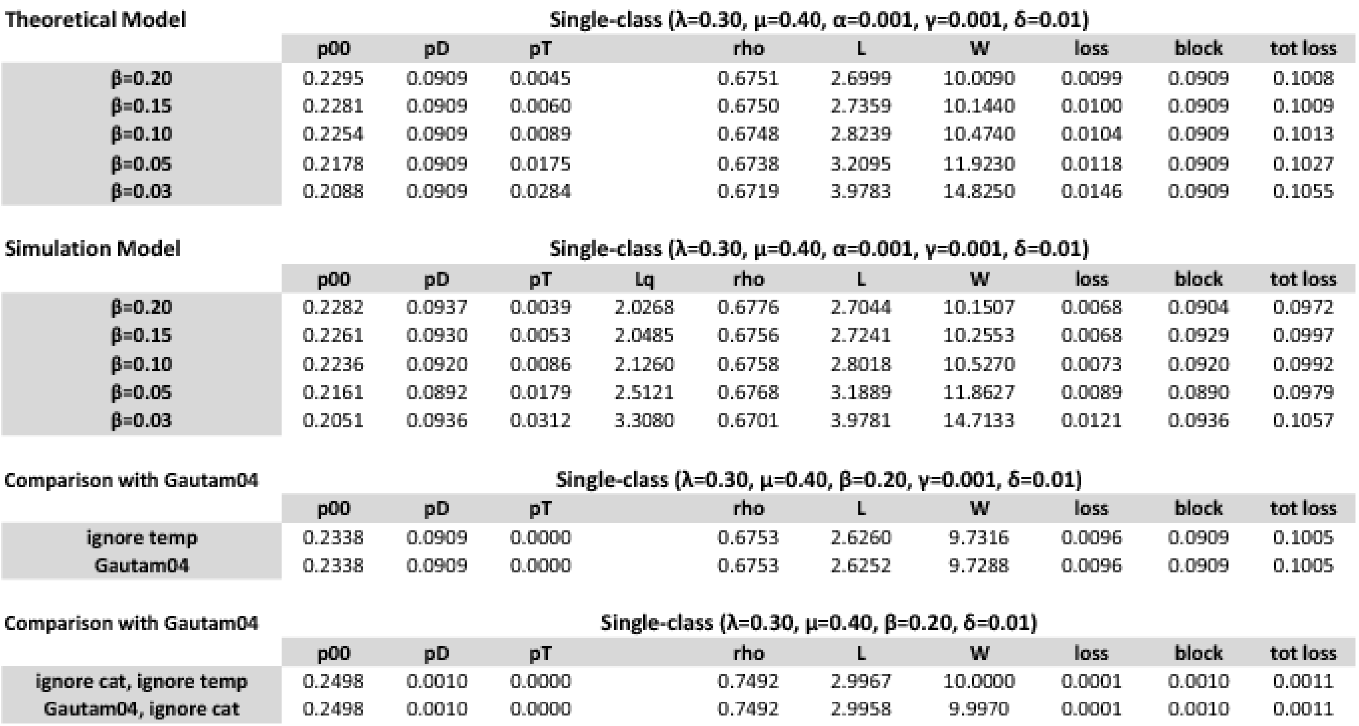


Figure 4.4: variation of response time (W) with λ and β

Table 4.1: Summary of results



Theoretical Model	Single-class (λ=0.30, μ=0.40, α=0.001, γ=0.001, δ=0.01)									
	p00	pD	pT		rho	L	W	loss	block	tot loss
β=0.20	0.2295	0.0909	0.0045		0.6751	2.6999	10.0090	0.0099	0.0909	0.1008
β=0.15	0.2281	0.0909	0.0060		0.6750	2.7359	10.1440	0.0100	0.0909	0.1009
β=0.10	0.2254	0.0909	0.0089		0.6748	2.8239	10.4740	0.0104	0.0909	0.1013
β=0.05	0.2178	0.0909	0.0175		0.6738	3.2095	11.9230	0.0118	0.0909	0.1027
β=0.03	0.2088	0.0909	0.0284		0.6719	3.9783	14.8250	0.0146	0.0909	0.1055

Simulation Model	Single-class (λ=0.30, μ=0.40, α=0.001, γ=0.001, δ=0.01)									
	p00	pD	pT	Lq	rho	L	W	loss	block	tot loss
β=0.20	0.2282	0.0937	0.0039	2.0268	0.6776	2.7044	10.1507	0.0068	0.0904	0.0972
β=0.15	0.2261	0.0930	0.0053	2.0485	0.6756	2.7241	10.2553	0.0068	0.0929	0.0997
β=0.10	0.2236	0.0920	0.0086	2.1260	0.6758	2.8018	10.5270	0.0073	0.0920	0.0992
β=0.05	0.2161	0.0892	0.0179	2.5121	0.6768	3.1889	11.8627	0.0089	0.0890	0.0979
β=0.03	0.2051	0.0936	0.0312	3.3080	0.6701	3.9781	14.7133	0.0121	0.0936	0.1057

Comparison with Gautam04	Single-class (λ=0.30, μ=0.40, β=0.20, γ=0.001, δ=0.01)									
	p00	pD	pT		rho	L	W	loss	block	tot loss
ignore temp	0.2338	0.0909	0.0000		0.6753	2.6260	9.7316	0.0096	0.0909	0.1005
Gautam04	0.2338	0.0909	0.0000		0.6753	2.6252	9.7288	0.0096	0.0909	0.1005

Comparison with Gautam04	Single-class (λ=0.30, μ=0.40, β=0.20, δ=0.01)									
	p00	pD	pT		rho	L	W	loss	block	tot loss
ignore cat, ignore temp	0.2498	0.0010	0.0000		0.7492	2.9967	10.0000	0.0001	0.0010	0.0011
Gautam04, ignore cat	0.2498	0.0010	0.0000		0.7492	2.9958	9.9970	0.0001	0.0010	0.0011

Validation of single-class model with temporary and catastrophic breakdown

An Arena simulation was used to validate the theoretical model. The code for the simulation model is available at this site.[5] Each simulation was run for $5 * 10^6$ seconds (1-2 million arrivals). Each simulation was replicated three times with different seeds. For a few cases, the results of the simulation runs and their theoretical counterparts are tabulated in Table 4.1. The average error between the compared models across all the quantities reported is less than 5%.

4.5 Conclusions

In this chapter, we addressed a "micro-model" for the performance prediction of an agent with different kinds of failure. We examined a class of queueing models with catastrophic breakdown that has not received much attention in the literature. In particular, we develop a methodology to analyze queues with two kinds of failure - temporary and catastrophic breakdown. We consider a single-class of traffic and obtain performance measures using generating functions. In this particular case, we are able to obtain closed form results. In the next chapter, we consider a case with two classes of traffic. Furthermore, we introduce priorities by making one class more important than the other. These analytical models can be utilized to rapidly obtain performance and availability estimates as is required in agent-based systems.

---

[5] http://www2.ie.psu.edu/Kumara/Research/lisq/index_files/Algorithms.htm

Chapter 5

A Multi-Class Performance Model for an Agent-Based System with Breakdown

5.1 Introduction

In this chapter, we develop a performance model for a system that can handle two types of traffic while healing from different types of breakdown. We will first describe the scenario and elaborate on some application areas for this model. In specific, we will describe the application of this performance model to multi-agent systems. The basic purpose of modeling the system is to evalaute to what degree the performance and availability will deteriorate when there are breakdowns. Since we are able to obtain a closed-form solution for our performance model, it could be used in a computationally efficient manner as an internal-model for system optimization. This multi-class waiting-line model is applicable to several applications in manufacturing and telecommunications where there are interruptions in service.

5.1.1 Overview of the modeled scenario: Agent-Based System

A software application such as a agent system can process different types of tasks. The difference in the types is characterized by varing arrival rates in to the system and the time required to process them. The application which is composed of various components is susceptible to failure. The first type of failure considered causes a loss of processing capability while the system state (number in system, the types of tasks entering etc.) is maintained. The second type of failure is catastrophic in nature and causes all state information and queued up tasks to be lost. The assumptions made with respect to repair (and rehydration), multiple levels of tasks that can be class-switched or inter-converted, and the types of failures make the model applicable to a software system, in specific, an agent-based system.

The failure in the application is caused by environmental conditions that are inherently unstable, such as conditions in a battle-zone. When computing resources and software are operating in such environments, it gives rise to severe loads that adversely affect the software components. The model in this chapter pertains to a single software component (henceforth called agent) that operates under aforementioned battlefield conditions. The software agent we consider generally processes two types of tasks - *update* and *planning* tasks. Update tasks result due to the agent sampling the environment for changes in physical attributes and sensor readings. Planning tasks are more computation intensive and generally relate to utilizing sensed input and stored operational tactics into a plan of action for a subsequent time duration.

In any case, we will concentrate of developing a micro-model (see Cohen [20]) for analyzing the performance of an agent system operating in a stressful environment. In the next few sections we will detail the modeling objectives and describe the system from the perspective of queueing theory.

5.1.2 Modeling Objectives

The objective of this chapter is to develop a multi-class queueing model for each *agent* of a system comprising of multitude of agents potentially under varying operating conditions. Each agent accepts two types of tasks, namely low (class 1) and high (class 2) priority tasks.[1] Tasks wait in a single queue and are processed in a first-come first-served basis within their types. If a temporary breakdown occurs, all tasks already in the queue wait until the system is repaired. Once the agent heals (i.e. repaired), the next task to be processed is picked as per its priority. Under temporary breakdown, the system continues to accept new tasks i.e. arriving tasks are not blocked. Whenever a catastrophic failure occurs, the queue is emptied. The model has to account for two kinds of adaptive behaviors of the agent:

1. Drop tasks that require high processing beyond a particular threshold of n packets; or
2. Convert tasks that require high processing to the other type which requires lesser processing at a particular threshold of n tasks (referred to as *class-switching*).

---

[1] In agent-based systems such as Ultra-Log, the tasks are referred to as *level-i* where i indicates the priority and/or procesing time requirement. The system therefore can handle tasks of different levels. Furthermore, tasks can be converted to different levels. This is the motivation for considering *class-switching*.

In this way, the agent can modulate its *stress* level and survive severe information load. Accepting a limited number of tasks requiring high level of processing has performance benefits. This is the motivation behind having a finite buffer space n for class 2. In an agent-based system like Ultra*Log, class 2 tasks are treated are treated with high priority to the maximum extent possible. However, if the demand for high priority processing goes beyond n, they should be switched to the lower priority and be processed at the corresponding QoS. Class-switching should automatically cease once there are not more than n class 2 packets in the queue. We now describe the above modeling requirments more formally. We also factor the environmental conditions and healing capability of the agent (referred to as repair as in Chapter 4) into the model.

5.1.2.1 Performance model with catastrophic and temporary breakdowns, and multiple classes: Model parameters

In the simplest case, we model each agent as a queue that can process two classes of traffic. However, several agents could exist on a computational resource. In that case, we bunch all the similar agents residing on the computational resource and represent it as a single queue. Agents located on a node (i.e. a computational resource) could supply tasks that belong to two classes of traffic. This means that tasks could be classified as those that involve light (class 1) or heavy (class 2) computation. The service time requirements for the two kinds of tasks are exponentially distributed with mean $1/\mu^k$, $k = 1,\ 2$. The arrival of tasks from agent i for each class of traffic k is according to a Poisson process with parameter λ_i^k $\big(PP(\lambda_i^k)$. The total arrival for class k can be calculated as $\lambda^k = \sum_{i=1}^{N} \lambda_i^k$, where N is the total number of agents allocated to the node. We asume that there is infinite room for the tasks that require light computation (class 1) to wait in the queue while only a maximum of n class 2 tasks can reside in the system.

The node on which agents are located may be experiencing information overload (high λ_i) and/or other kinds of information attacks - causing either temporary or permanent destruction to the processing at the node. These information attacks are modeled as being separated by random amounts of time, or according to a $PP(\alpha)$ process. Since the agents and nodes are *self-healing*[2], they repair some damage and are up again after time distrbuted exponentially with mean $1/\beta$. Furthermore, we assume that catastrophic attacks occur according to $PP(\gamma)$ leading to loss of information. In this situation, the replacement of agents is an option. In agent-based systems, replacement of agents is synonymous to restarting them on other fully functional computing entities. This process is referred to as *re-hydration* and occurs after a switch-over time distributed exponentially with mean $1/\delta$.

5.1.2.2 Measures of effectiveness

The performance metrics we need to compute from the model are (1) the response time of the customers that receive service (Δ) for either class and (2) the probability that a task is lost (P_l) for either class. In both cases, a lower value means better quality of service (QoS). From a reliability standpoint, the availability (R) of an agent is the probability that it is in the *up* state. A higher value of R indicates a better QoS.

In the rest of this chapter, we develop the queueing model and a solution technique to arrive at the aforementioned metrics.

---

[2] We are referring to software reconfiguration such as alternate algorithms, resolving deadlocks, thread contentions and scheduling issues in the infrastrucuture's operating system.

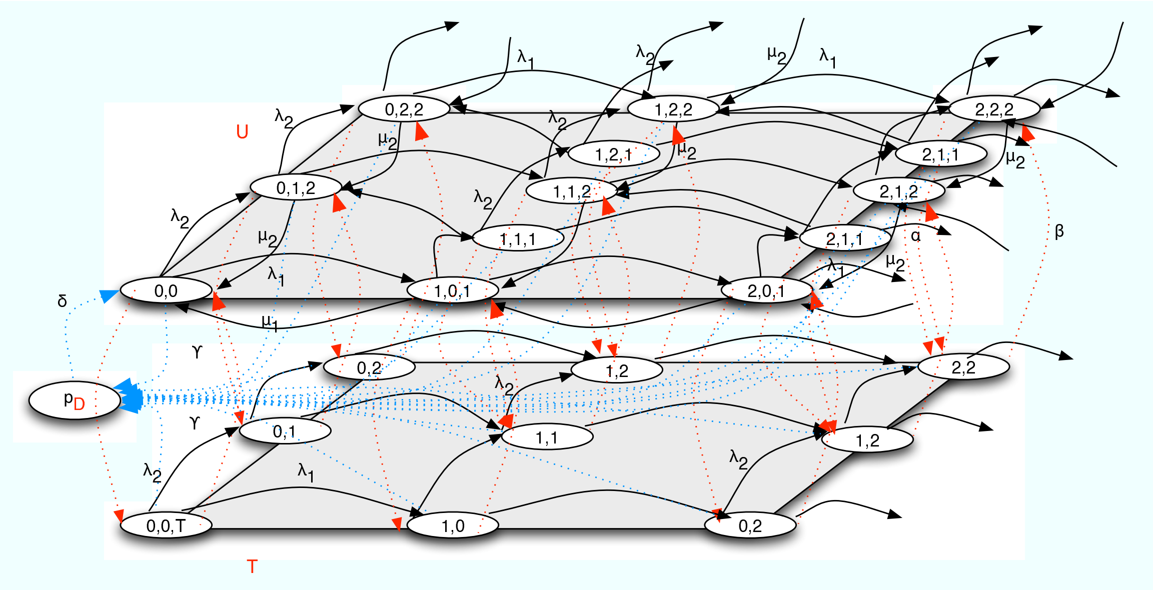


Figure 5.1: Rate diagram of an agent with temporary and catastrophic breakdown, and two classes of traffic on a computing resource

5.2 Queueing Model

5.2.1 Two-Class Model

The aforementioned scenario is described by a stochastic process $\{P(t),\ K(t),\ Q(t),\ R(t)\}$ where $P(t)$ denotes the number of tasks in both classes as $(p_1(t),\ p_2(t))$. $K(t)$ denotes the class in service in the priority queue. $Q(t)$ and $R(t)$ denote temporary and catastrophic breakdown respectively. We can analyze this stochastic process as a continuous time Markov chain. Let $Z(t) = \{P(t),\ K(t),\ Q(t),\ R(t)\} = ((p_1,\ p_2),\ k,\ q,\ r)$ (for $p_1 = 0,\ 1,\ 2,\ ...,\ p_2 = 0,\ 1,\ 2,\ ...,\ k \in \{0,\ 1,\ 2\}$ where $k = 0$ is the don't care condition *w.r.t.* which class is in service, $q \in \{0,\ 1\}$, $r \in \{0,\ 1\}$). Clearly, $\{Z(t),\ t \geq 0\}$ is a CTMC with rate diagram shown in Figure 5.1. The CTMC is ergodic. However, this CTMC is two-dimensional, which makes it hard to obtain the solution of system of differential-difference equations corresponding the CTMC in closed-form .

5.2.1.1 Modeling considerations

In order to obtain performance measures and to accommodate the adaptive behavior of the agent, we make the assumption of finite buffer space for class 2. Thereby, we are looking at a $1-$*dimensional* Markov chain unlike a $2-$*dimensional* Markov chain which would have resulted by assuming infinite waiting room for class 2. From a functional perspective, this assumption can be viewed as the agent degrading the performance of one of the classes as per a threshold policy - a threshold that the agent can change. Alternately, we can consider dropping packets after the threshold point of n is reached for the class 2 packets. This is similar to Erlang loss models with finite buffer spaces. Neither of these assumptions affects the analysis methodology. We solve both the cases making the aforementioned assumption regarding the threshold and obtain measures of effectiveness in each case, namely the *switch* and *no-switch* cases.

5.2.1.2 Analysis

We now utilize the generating function methodology to sum our probabilities. We first introduce some notation which are also tabulated in Table 5.1.

Let

$$p_{p_1p_2kqr} \quad = \quad \lim_{t\to\infty} P\{Z(t) = ((p_1,\ p_2),\ k,\ q,\ r)\}$$

which are the steady state probabilities we will be solving for. Let

$$\psi_j^1(z) = \sum_i p_{ij100} z^i, \quad j \in \{0,\ 1,\ \ldots,\ n\}\ , \tag{5.1}$$

Table 5.1: Notation

Symbol	Description
i	Index for class 1
j	Index for class 2
$((p_1,\ p_2),\ k,\ q,\ r)$	number of class 1 tasks, number of class 2 tasks, type of task in service, if temporary breakdown has occured (1) or not (0), if catastrophic breakdown has occured (1) or not (0)
n	buffer capacity for class 2
$\psi_j^1(z)$	generating function for the j^{th} phase on the top plane, class 1 is in service
$\hat{\psi}_j^2(z)$	generating function for the j^{th} phase on the top plane, class 2 is in service
$\hat{\psi}_j(z)$	generating function for the j^{th} phase on the bottom plane where $q = 1$, $k = 0$ denotes do not care which class was in service
$Z(t)$	queue state at time t
$p_{p_1 p_2 kqr}(t)$	$P\{Z(t) = ((p_1,\ p_2),\ k,\ q,\ r)\}$

$$\psi_j^2(z) = \sum_i p_{ij200} z^i, \quad j \in [0,\ 1, \ldots,\ n] \tag{5.2}$$

and

$$\hat{\psi}_j(z) = \sum_i p_{ij010} z^i, \quad j \in [0,\ 1, \ldots,\ n]. \tag{5.3}$$

By level i we mean the set of states $\{(i,\ j,\ k\ ,\ l,\ m) :\ j = 0, 1, 2, \ldots, n;\ k = 1\ or\ 2;\ l = 0\ or\ 1;\ m = 0\}$. By phase j we mean the set of states $\{(i,\ j,\ k\ ,\ l,\ m) : i = 0, 1, 2, \ldots;\ k = 1\ or\ 2;\ l = 0\ or\ 1;\ m = 0\}$. By top plane we refer to those states that have $l = 0$ and by bottom plane we refer to those states that have $l = 1$. Equation (5.2) represents the generating function for the j^{th} phase in the top plane of Figure **??**. Equation (5.3) represents the generating function for the j^{th} phase in the bottom plane of Figure **??**. We first perform the analysis for the no-switch case. This is easily extended to the case where there is class-switching. We drop the z in all $\psi(z)$s with the understanding that they are all functions of z.

Upon summing up the balance equations using the generating functions, we get the following.

$$\begin{aligned}
[a - \frac{\mu_1}{z} - \lambda_1 z]\psi_0^1 &= \mu_1 p_{00000}(1 - \frac{1}{z}) + \mu_2 \psi_1^2 + \beta\hat{\psi}_0 + p_{00001}\delta_1 && (5.4)\\
[c - \lambda_1 z]\hat{\psi}_0 &= \alpha\psi_0^1 && (5.5)\\
[a - \lambda_1 z]\psi_1^1 &= \lambda_2\psi_0^1 - \lambda_2 p_{00000} && (5.6)\\
[b - \lambda_1 z]\psi_1^2 &= \frac{\mu_1}{z}\psi_1^1 + \lambda_2 p_{00000} + \mu_2\psi_2^2 + \beta\hat{\psi}_1 && (5.7)\\
[c - \lambda_1 z]\hat{\psi}_1 &= \alpha(\psi_1^1 + \psi_1^2) + \lambda_2\hat{\psi}_0 && (5.8)\\
\vdots \quad &\vdots \quad \vdots
\end{aligned}$$

$$\begin{array}{rcll} \vdots & \vdots & \vdots & \\ [a' - \lambda_1 z]\psi_n^1 & = & \lambda_2 \psi_{n-1}^1 & (5.9) \\ [b' - \lambda_1 z]\psi_n^2 & = & \frac{\mu_1}{z}\psi_n^1 + \lambda_2 \psi_{n-1}^2 + \beta \hat{\psi}_n & (5.10) \\ [c' - \lambda_1 z]\hat{\psi}_n & = & \alpha(\psi_n^1 + \psi_n^2) + \lambda_2 \hat{\psi}_{n-1} & (5.11) \end{array}$$

The constants $a \equiv \gamma+\lambda_1+\lambda_2+\mu_1+\alpha$, $b \equiv \gamma+\lambda_1+\lambda_2+\mu_2+\alpha$ and $c \equiv \gamma+\lambda_1+\lambda_2+\beta$. The other constants $a' \equiv a - \lambda_2$, $b' \equiv b - \lambda_2$ and $c' \equiv c - \lambda_2$. If the buffer capacity for class 2 tasks is n, there will we $3n + 2$ rows. Furthermore,

$$p_{00001}\delta \quad = \quad (1 - p_{00001})\gamma.$$

This gives p_{00001} (also called p_D, the down state) as $\gamma/(\gamma + \delta)$. Multiplying the equations (**??**)-(5.11) by z, we can write them in matrix form $A(z)\psi(z) = B(z)$ as follows(for a buffer-capacity of n for class 2 packets):

$$A(z) \equiv \begin{bmatrix} F^0 & F_r^0 & 0 & 0 & 0 & 0 & 0 & 0 \\ M_1^1 & M^1 & M_r^1 & 0 & 0 & 0 & 0 & 0 \\ 0 & M_l^2 & M^2 & M_r^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 & 0 & 0 \\ 0 & 0 & 0 & \ddots & \ddots & \ddots & 0 & 0 \\ 0 & 0 & 0 & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & M_l^{n-1} & M^{n-1} & M_r^{n-1} \\ 0 & 0 & 0 & 0 & 0 & 0 & M_l^n & L^n \end{bmatrix}$$

where the sub-matrices are defined as

$$F^j \equiv \begin{bmatrix} az-\mu_1-\lambda_1 z^2 & -\beta z \\ -\alpha z & cz-\lambda_1 z^2 \end{bmatrix},\ F_r^j \equiv \begin{bmatrix} 0 & -\mu_2 z & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

$$M_1^j \equiv \begin{bmatrix} -\lambda_2 z & 0 \\ 0 & 0 \\ 0 & -\lambda_2 z \end{bmatrix},\ M^j \equiv \begin{bmatrix} az-\lambda_1 z^2 & 0 & 0 \\ -\mu_1 & bz-\lambda_1 z^2 & -\beta z \\ -\alpha z & -\alpha z & cz-\lambda_1 z^2 \end{bmatrix},$$

$$M_r^j \equiv \begin{bmatrix} 0 & 0 & 0 \\ 0 & -\mu_2 z & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

$$M_l^j \equiv \begin{bmatrix} -\lambda_2 z & 0 & 0 \\ 0 & -\lambda_2 z & 0 \\ 0 & 0 & -\lambda_2 z \end{bmatrix},\ L^j \equiv \begin{bmatrix} a^{'}z-\lambda_1 z^2 & 0 & 0 \\ -\mu_1 & b^{'}z-\lambda_1 z^2 & -\beta z \\ -\alpha z & -\alpha z & c^{'}z-\lambda_1 z^2 \end{bmatrix}$$

and j denotes the phase.

$$B \equiv \begin{bmatrix} B^0 \\ B^1 \\ B^2 \\ \vdots \\ \vdots \\ B^{n-1} \\ B^n \end{bmatrix}$$

where $B^0 = [\{\mu_1 p_{00000}(z-1) + p_{00001}\delta_1 z\} \;\; 0]^T$, $B^1 = [-\lambda_2 p_{00000} \;\; \lambda_2 p_{00000} \;\; 0]^T$ and $B^j = [0 \; 0 \; 0]^T$ for $j \geq 2$.

$$\psi(z) \equiv \begin{bmatrix} \psi^0 \\ \psi^1 \\ \psi^2 \\ \vdots \\ \vdots \\ \psi^{n-1} \\ \psi^n \end{bmatrix}$$

where $\psi^0 \equiv [\psi_0^1 \; \hat{\psi}_0]^T$ and $\psi^j \equiv [\psi_j^1 \; \psi_j^2 \; \hat{\psi}_j]^T$ for $j \geq 1$.

5.2.2 Finding the boundary probabilities

It is straightforward to obtain

$$\psi_j^k(z) = \frac{C_j^k(z)}{D(z)}$$

and

$$\psi_j(z) = \frac{\hat{C_j}(z)}{D(z)}$$

. The ψ_j^k and ψ_j represents the generating function for the j^{th} phase on the top and bottom layers as shown in Figure 5.1 in terms of the known input parameters (λ_1, μ_1 etc.) as well as p_{00000} which we must compute. Note that p_{00001} is already known to be $\gamma/(\gamma + \delta)$. The denominator

$$D(z) = det(A).$$

$D(z)$ in our model usually a large degree polynomial in z.For example, for $n = 2$, it would be of the order 8. We are interested in the zeros of $D(z)$ to obtain the boundary probability p_{00000}. In this particular case, 2-class priority queueing model, we have only one unknown. The zeros of $D(z)$ are evaluated numerically and denoted z^*. We are interested only in $z^* < 1$.

Next we find $C_j^k(z^*)$ and/or $C_j(z^*)$. These quantities are equated to zero to solve for the boundary probabilities. This procedure is pretty general and can be used to obtain the boundary probabilites for other types of similar models such as preemtive priority multi-class case or those with multiple levels of temporary breakdown. The commonality between these models is that they all have multiple phases with a generating function ψ_j representing phase j. In those cases, the different zeros $z^*(\leq 1)$ are utilized to obtain several equations which are solved simultaneously to obtain the boundary probabilities.

5.2.3 Class-switching case

The class-switching case is almost the same as above. The only difference is with the matrix A where the constants have to be redefined slightly. In order to obtain A for this case, first define $a^{'} \equiv a$, $b^{'} \equiv b$ and $c^{'} \equiv c$. Subsequently, replace λ_1 by $(\lambda_1 + \lambda_2)$ when $j = n$ (i.e. the last three rows).

5.2.4 Case where n=2

We illustrate the above steps for $n = 2$ for the class-switching case. This means that when $n = 2$, the net arrival rate of class 1 packets into the system will be $\lambda_1 + \lambda_2$ and that of class 2 will be zero. The matrices are

$$A(z) = \begin{bmatrix} az-\mu_1-\lambda_1 z^2 & -\beta z & 0 & -\mu_2 z & 0 & 0 & 0 & 0 \\ -\alpha z & cz-\lambda_1 z^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\lambda_2 z & 0 & az-\lambda_1 z^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\mu_1 & bz-\lambda_1 z^2 & -\beta z & 0 & -\mu_2 z & 0 \\ 0 & -\lambda_2 z & -\alpha z & -\alpha z & cz-\lambda_1 z^2 & 0 & 0 & 0 \\ 0 & 0 & -\lambda_2 z & 0 & 0 & az-(\lambda_1+\lambda_2)z^2 & 0 & 0 \\ 0 & 0 & 0 & -\lambda_2 z & 0 & -\mu_1 & bz-(\lambda_1+\lambda_2)z & -\beta z \\ 0 & 0 & 0 & 0 & -\lambda_2 z & -\alpha z & -\alpha z & cz-(\lambda_1+\lambda_2)z^2 \end{bmatrix},$$

$$B \quad = \quad \begin{bmatrix} \mu_1 p_{00000}(z-1) + p_{0001}\delta_1 z \\ 0 \\ -\lambda_2 p_{00000} \\ \lambda_2 p_{00000} \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

and $\psi(z) = [\psi_0^1 \ \hat{\psi}_0 \ \psi_1^1 \ \psi_1^2 \ \ \hat{\psi}_1 \ \psi_2^1 \ \psi_2^2 \ \hat{\psi}_2]^T$.

We will use this matrix to illustrate our theoretical procedure.

At this point, we know the generating functions and the boundary probabilites. These are used to derive performance measures.

5.3 Measures of Effectiveness

We are interested in L_k, Δ_k , P_l^k and R for $k = 1,\ 2$ which are the steady state number in system, response time, loss probability and system availability respectively where k denotes the class of traffic. We will derive the the two cases (1) when the buffer capacity for class 2 is n, and (2) when class 2 tasks are switched to class 1 when the buffer-capacity of class 2 tasks reaches n.

Theorem 5.3.1. *The average number of tasks of class k in steady state*

$$L_1 = \frac{d}{dz}\sum_{j=0}^{n}(\sum_{l=1}^{2}\psi_j^l(z) + \hat{\psi}_j(z))|_{z=1},$$

$$L_2 \ = \ \sum_{j=1}^{n} j(\sum_{l=1}^{2}\psi_j^l(1) + \hat{\psi}_j(1)).$$

Proof. Follows directly from definition. □

Case 1: Class 2 tasks are dropped at threshold n

This case is akin to having a finite buffer space for class 2, i.e. when there n packets of class 2, any additional packets are dropped. This case is represented in Figure 5.2

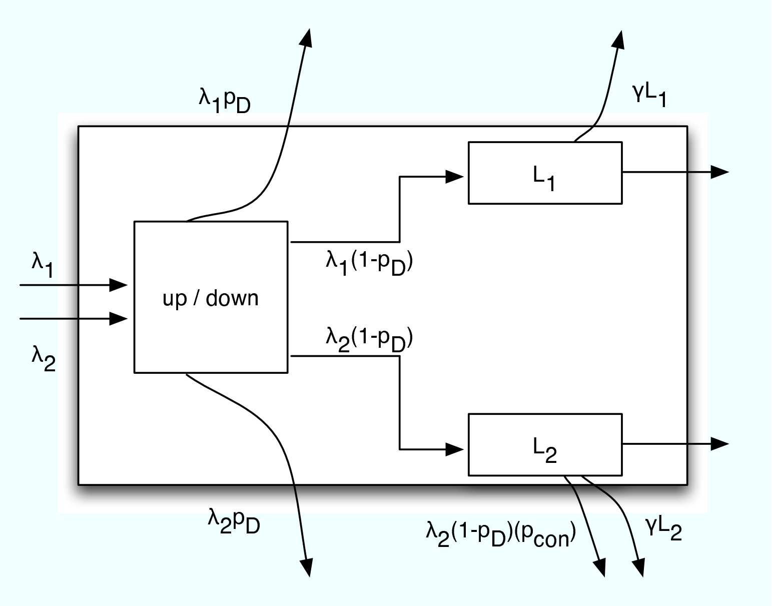


Figure 5.2: Drop class 2 packets at threshold n

where p_{con} is the probability of getting dropped. .

Theorem 5.3.2. *The average waiting times are*

$$\Delta_1 \quad = \quad \frac{L_1(1-p_D)}{\lambda_1(1-p_D)^2 - \gamma L_1}$$

and

$$\Delta_2 \quad = \quad \frac{L_2(1-p_D)}{\lambda_2(1-p_D-p_{drop})(1-p_D) - \gamma L_2}$$

where $p_{drop} = \sum_{l=1}^{2} \psi_n^l(1) + \hat{\psi}_n(1)$.

Proof. Of the λ_1 class 1 tasks that arrive, a portion $\lambda_1 p_D$ are blocked because the system was down due to catastrophic failure. L_1 is the long run average of the number of class 1 tasks in the system. In addition, γL_1 tasks that are already in the queue are lost due to catastrophic failure. Conditioning on the fact the the system was up when catastrophic failure occured, a portion $\frac{\gamma L_1}{(1-p_D)}$ is lost. The expected rate of departing customers (those that completed service) would therefore be

$$\lambda_1^{net} \quad = \quad \lambda_1 - \lambda_1 p_D - \frac{\gamma L_1}{(1-p_D)}$$

. Since the average time a class 1 customer spends in the system is related to L_1 by Little's Law, the response time for those tasks that finish processing can be computed as

$$\Delta_1 = \frac{L_1}{\lambda_1^{net}}$$

which is stated above. For the class 2 tasks, an additional stream of loss is $\lambda_2 p_{drop}$. Since tasks for type class 2 are dropped as soon as the buffer space reaches n, the probabaility of dropping tasks is the probabaility of being in the states p_{ink00} where $i = 0,\ 1,\ 2, \ldots$ and $k = \{1,\ 2\}$.This is easily obtained from the generating functions $\psi_n^k(z)$ and $\psi_n(z)$ by putting $z = 1$. This sum is defined as p_{drop} and written as above. Following similar steps,

$$\lambda_2^{net} \quad = \quad \lambda_2 - \lambda_2 p_D - \lambda_2 p_{drop} - \frac{\gamma L_2}{(1-p_D)}.$$

Applying Little's Law using the net departure rate λ_2^{net}, we obtain Δ_2 as stated above. □

Theorem 5.3.3. *Let P_l^k be the loss probability for class k.*

$$P_l^1 \quad = \quad \frac{\lambda_1 p_D (1-p_D) + \gamma L_1}{\lambda_1 (1-p_D)},$$

$$P_l^2 = \frac{\lambda_2(p_D + p_{drop})(1 - p_D) + \gamma L_2}{\lambda_2(1 - p_D)},$$

Proof. The total rate at which class 1 tasks are lost is

$$\lambda_1 p_D + \tfrac{\gamma L_1}{(1-p_D)}.$$

Dividing it by incoming arrival rate λ_1 gives the loss probability as experienced by the class 1 traffic. Class 2 traffic experiences additional losses due to finite buffer space. So the total rate of losses are

$$\lambda_2 p_D + \tfrac{\gamma L_2}{(1-p_D)} + \lambda_2 p_{drop}$$

which when divided by incoming arrival rate λ_2 gives P_l^2 as stated above. □

Case 2: Class switching at threshold n

This case is akin to a single-class queue after threshold n. The value of n can be adjusted by the agent as a function of desired performance. $L_k,\ k = 1, 2$ are the same in this case as well. This case is represented in Figure 5.2 where where p_{con} is probability of switching.

Theorem 5.3.4. *The average waiting times are*

$$\Delta_1 = \frac{L_1(1 - p_D)}{(\lambda_1(1 - p_D) + \lambda_2 p_{switch})(1 - p_D) - \gamma L_1}$$

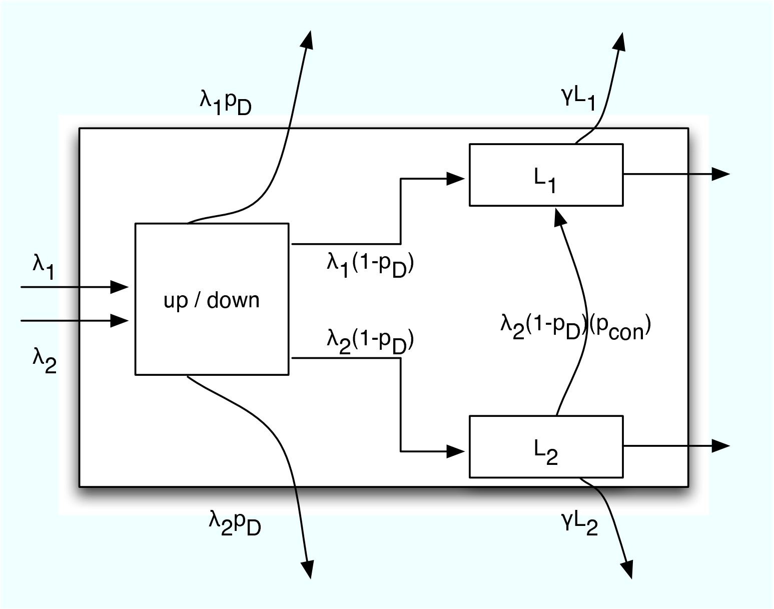


Figure 5.3: Switch class 2 to class 1 at threshold n

and

$$\Delta_2 \quad = \quad \frac{L_2(1-p_D)}{\lambda_2(1-p_D-p_{switch})(1-p_D)-\gamma L_2}$$

where $p_{switch} = \sum_{l=1}^{2} \psi_n^l(1) + \hat{\psi}_n(1)$.

Proof. Because of class switching, the effective arrival rates of both classes of traffic are affected. The net arrival rate of class 1 packets is

$$\lambda_1^{net} = \lambda_1 + \lambda_2 P\{X_2 = n\} - \lambda_1 p_D - \frac{\gamma L_1}{(1-p_D)}$$

where X_2 is the random variable indicating the number of class 2 packets in the system and $P\{X_2 = n\}$ is the probability that there are exactly n class 2 tasks in the system. Since type 2 tasks switch class at the threshold n, the probability of switching $p_{switch} = P\{X_2 = n\}$ which is stated in terms of generating functions as above (this probability is same as the drop probability of case 1). The effective arrival rate of class 2 tasks is

$$\lambda_2^{net} \quad = \quad \lambda_2 - \lambda_2 P\{X_2 = n\} - \lambda_2 p_D - \frac{\gamma L_2}{(1-p_D)}.$$

Now

$$\Delta_k \quad = \quad \frac{L_k}{\lambda_k^{net}}$$

by Little's Law which gives the expressions above. ☐

Theorem 5.3.5. *The loss probabilities for the two classes of traffic are*

$$P_l^1 \quad = \quad \frac{\lambda_1 p_D(1-p_D) + \gamma L_1}{\lambda_1(1-p_D)},$$

and

$$P = \frac{\lambda_2(p_D + p_{switch})(1-p_D) + \gamma L_2}{\lambda_2(1-p_D)}$$

where $p_{switch} = \sum_{l=1}^{2} \psi_n^l(1) + \hat{\psi}_n(1)$.

Proof. Similar to case 1. ☐

Theorem 5.3.6. *The steady state availability R of the 2 class queue*

$$R \quad = \quad \sum_{j=0}^{n} \sum_{l=1}^{2} \psi_j^l(1).$$

Proof. The availability of the system is the steady state probability the system is available for processing tasks. In the 2 class queue, the system is not serving any customer when the system is in temporary or catastrophic breakdown. The sum of steady state probabilities of the remaining states i.e. the total probability that the system is in the states p_{ijk00} where $i = \{0,\ 1,\ 2, \ldots\}$, $j = \{0,\ 1, 2\ , \ldots,\ n\}$ and $k = \{1,\ 2\}$. Alternately, this result can be stated in terms of a Markov Reward Model (MRM) [83]. Let the random variable

$$Z^{'}(t) \quad = \quad r_{Z(t)}$$

refer to the instantaneous reward rate of the MRM corresponding to $Z(t)$. The instantaneous availability is

$$\begin{aligned} R(t) &= E[Z^{'}(t)] \\ &= \sum_{i,\ j,\ k} (r_U p_{ijk00}(t) + r_T p_{ijk10}(t)) + r_D p_D(t) \end{aligned}$$

where r_U, r_T and r_D are the reward rates assigned to the "up" states, temporary breakdown states and catastrophic breakdown state respectively. By assigning a reward rate of zero to states where $Q(t) = 1$ or $R(t) = 1$, and a reward of one to states where $Q(t) = 0$ and $R(t) = 0$ the steady state availability is computed [83]. Setting set $r_U = 1$ and $r_T = r_D = 0$ and allowing $t \rightarrow \infty$

$$\begin{aligned} R &= \lim_{t \rightarrow \infty} R(t) \\ &= \sum_{i,\ j,\ k} p_{ijk00} \end{aligned}$$

which is the quantity given above. □

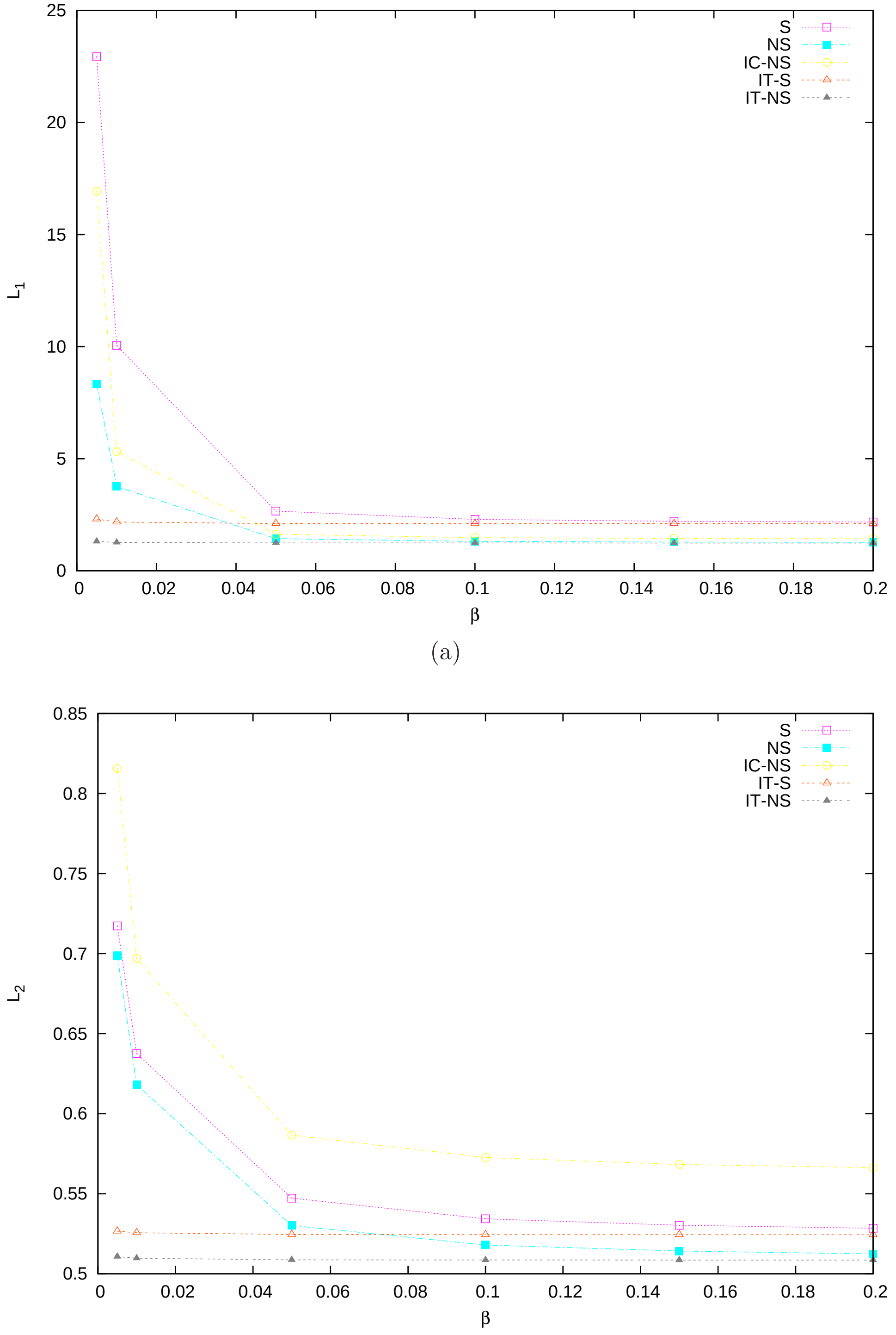
5.4 Numerical Examples and Validation

We now provide numerical examples using the analytical model explained above. In particular, we examine the variation of some performance measures with respect to changes in β, the repair rate. By S and NS we refer to *switch* and *no-switch* cases which we described above. IC denotes the "ignore catastrophic breakdown" case. IT denotes the "ignore temporary breakdown" case.

Example 5.4.1. "Single-class" case: In this example, we have deliberately chosen $\lambda_1 = \lambda_2$ and $\mu_1 = \mu_2$. By doing so, this case can be compared to the model in Chapter 4 for performance measures such as L in the class-switch case. We have both temporary and catastrophic breakdown in this example, although we will ignore one of them at a time to illustrate the effect the breakdowns have on the measures of effectiveness. The values of the parameters $\{\lambda_1,\ \mu_1,\ \lambda_2,\ \mu_2,\ \alpha,\ \beta,\ \gamma,\ \delta\ \}$ are given by $\{.15,\ .4,\ .15,\ .4,\ .001,\ \beta,\ 0.001,\ 0.01\}$. β is varied in the range $0.01 - 0.2$. The value of n considered is 2 i.e. the maximum buffer space for class 2 packets is 2. Figure 5.4 shows the variation of L_1 and L_2 with varying β. Although arrival and service rates were chosen identically, the queue-lengths are different because of (a) priority (class 2>class 1) and (b) class 2 has a finite buffer space. As expected, we have lesser queue-lengths in the NS cases because of the higher loss as compared to the corresponding S case. IC cases retain more number of tasks in the system. When β is small, the IC-S case causes a really large queue-length and for this reason they are not included in Figure 5.4. Whenever the temporary failure is ignored (i.e. IT cases), β has no impact on the corresponding measure of effectiveness as expected.

The discussion for the various cases involving W_i is exactly similar to the above (see Figure 5.5). p_T is independent of whether or not catastrophic failure occurs or not and hence it consistently falls with increasing β for the pairs of cases (S,IC-S) and (NS, IC-NS) (Figure 5.6a). Likewise, the availability R of the agent consistently increases with increasing β (Figure 5.6b). By varying the repair rate, the number of packets switching to class 1 or those that get dropped is affected as can seen in Figure 5.7. In Figure 5.8 and Figure 5.9, we show the break-up of the total loss experienced by the system. When there is no-switching, class 1 will have constant blocking probability (because class 2 will not affect it in the NS case) whether or not there is temporary failure. Hence the NS and theIT-NS cases are identical in Figure 5.8a. Because of priority, there are very few class 2 packets remaining in the queue. So when a catastrophe strikes, very few class 2 packets are lost. This difference in loss probabilities in apparant for the two classes in Figure 5.9.

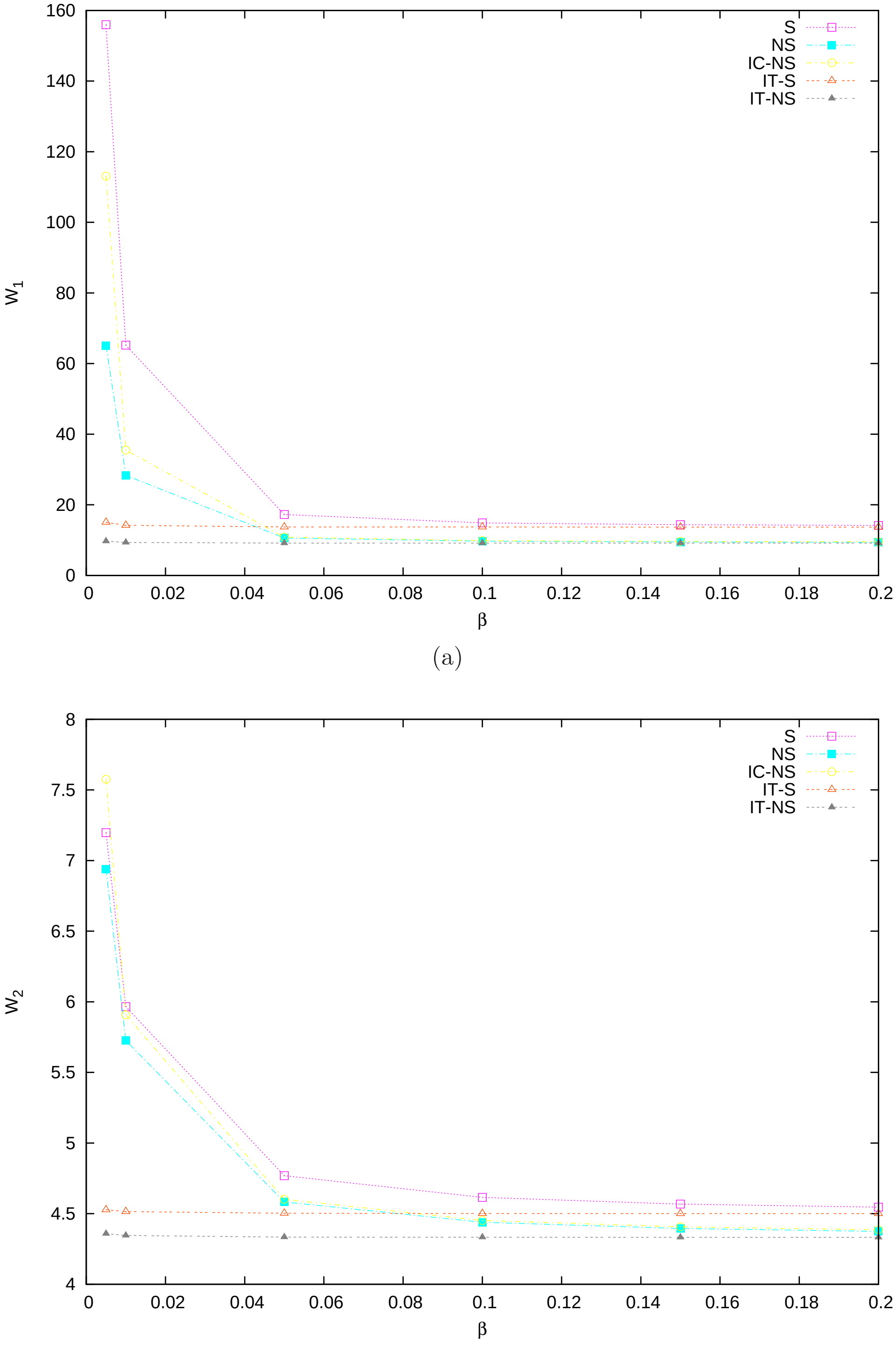
Example 5.4.2. Two-class case: For this example, the values chosen for the parameters $\{\lambda_1,\ \mu_1,\ \lambda_2,\ \mu_2,\ \alpha,\ \beta,\ \gamma,\ \delta\ \}$ are $\{.15,\ .4,\ .1,\ .2,\ .001,\ \beta,\ 0.001,\ 0.01\}$. β is varied in the range $0.01 - 0.2$. The value of n considered is 2. In Figure 5.10-Figure 5.15, we show the trends for the variation of L_i, W_i, R and the various loss probabilities as β is increased. The discussion is not repeated in the two-class case as the results are similar to Example 5.4.1.



(a)

(b)

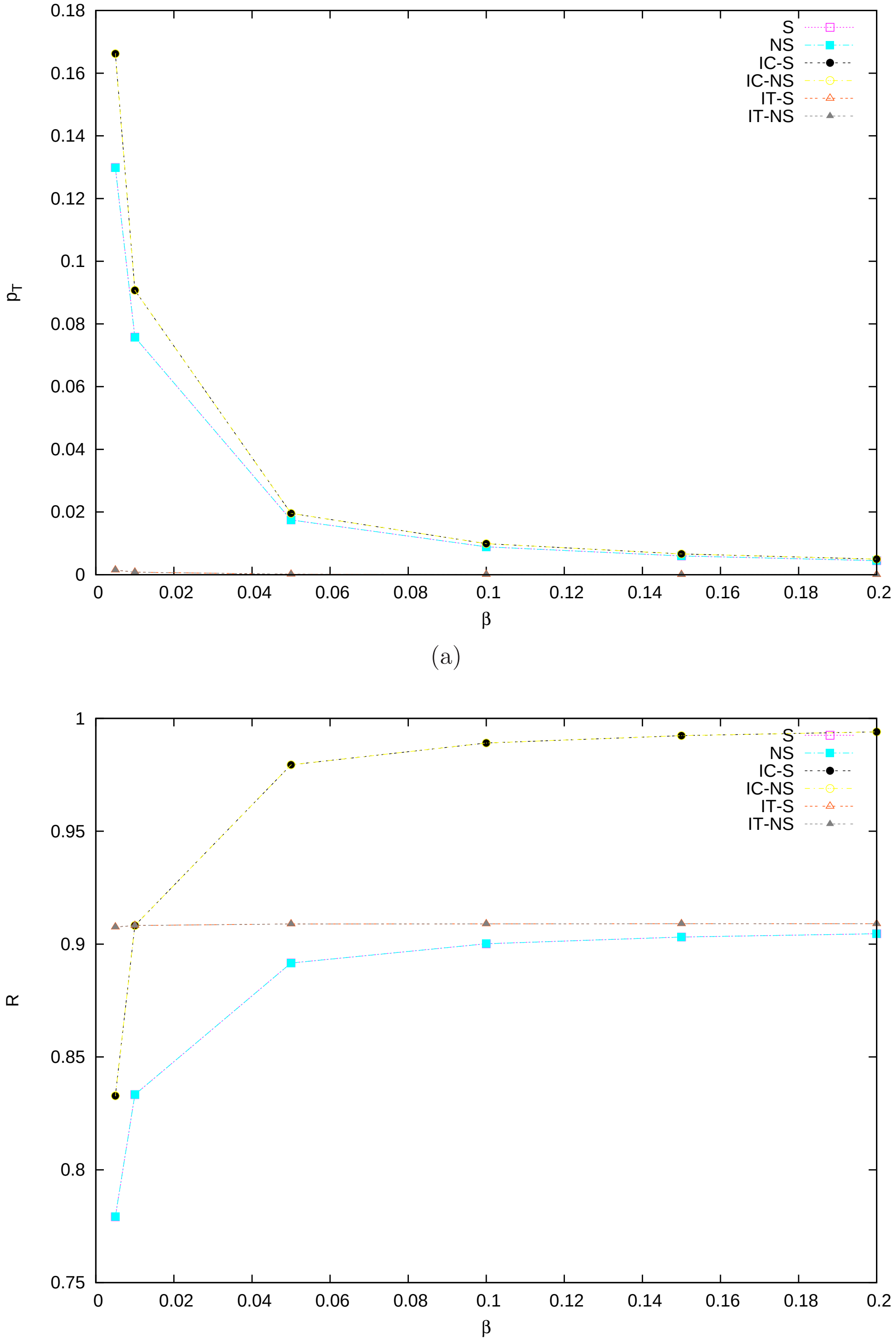
Figure 5.4: L_i versus β



(a)

(b)

Figure 5.5: W_i versus β



(a)

(b)

Figure 5.6: L_i, W_i, p_T and R versus β

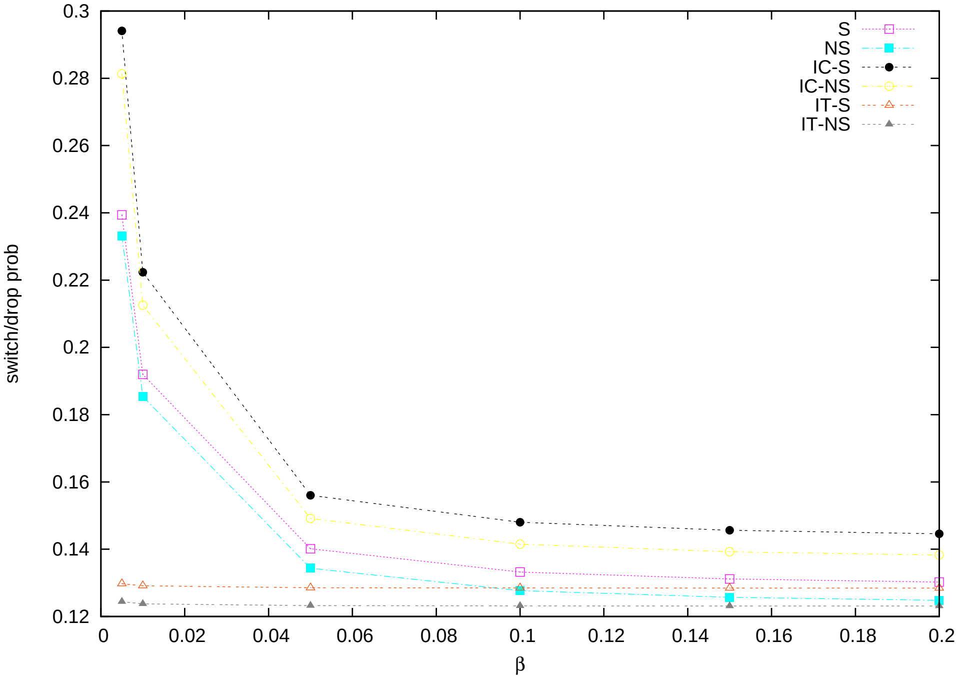


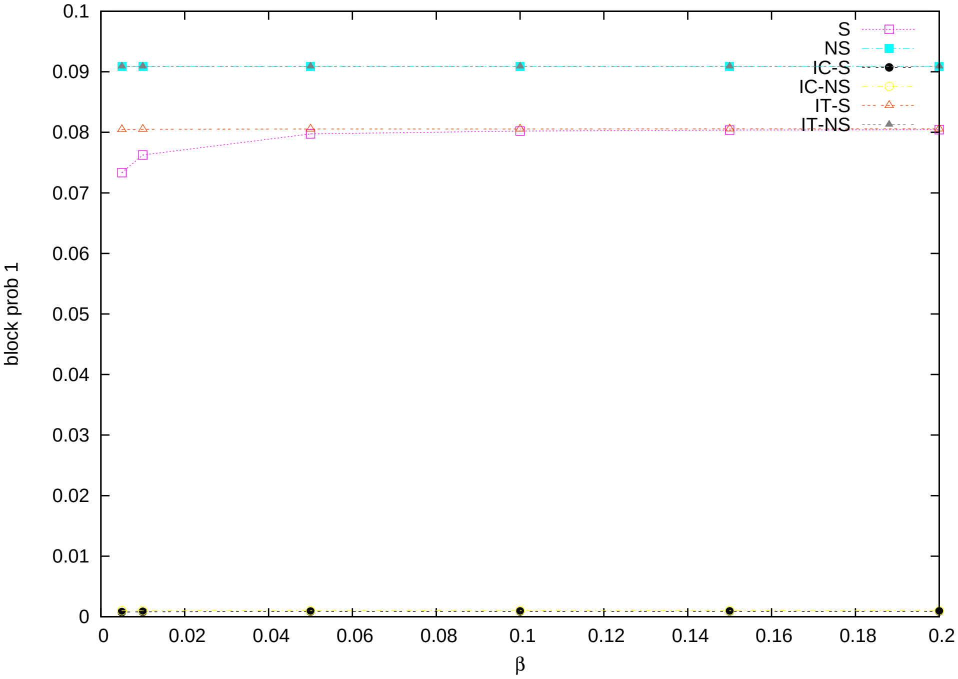
Figure 5.7: switching probabilities versus β

5.4.1 Validation of two-class model with temporary and catastrophic breakdown

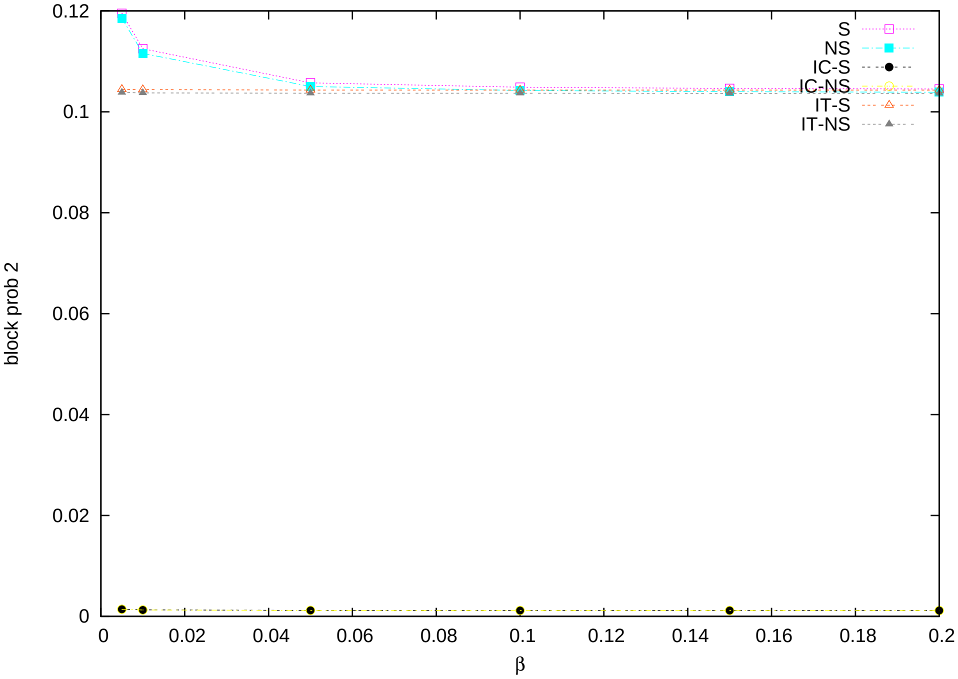
The theoretical model was validated against an Arena discrete-event simulation. The simulation files are available at the site.[3] We considered two particular cases from Example 5.4.1 and Example 5.4.2 i.e. when $\beta = 0.2$. Each simulation was run for $5 * 10^6$ seconds (1-2 million arrivals). Each simulation was replicated three times. The results for the simulation runs and their theoretical counterparts are tabulated for a specific case as shown in Table 5.2 and Table 5.3. In Table 5.2, we make $\lambda_1 = \lambda_2$ and $\mu_1 = \mu_2$and run the two-class model as a single-class model. In both Table 5.2 and Table 5.3, the values predicted by the analytical model are listed alongside those obtained from the simulation model. The simulation model does not provide $L_1$and

---

[3] http://www2.ie.psu.edu/Kumara/Research/lisq/index_files/Algorithms.htm

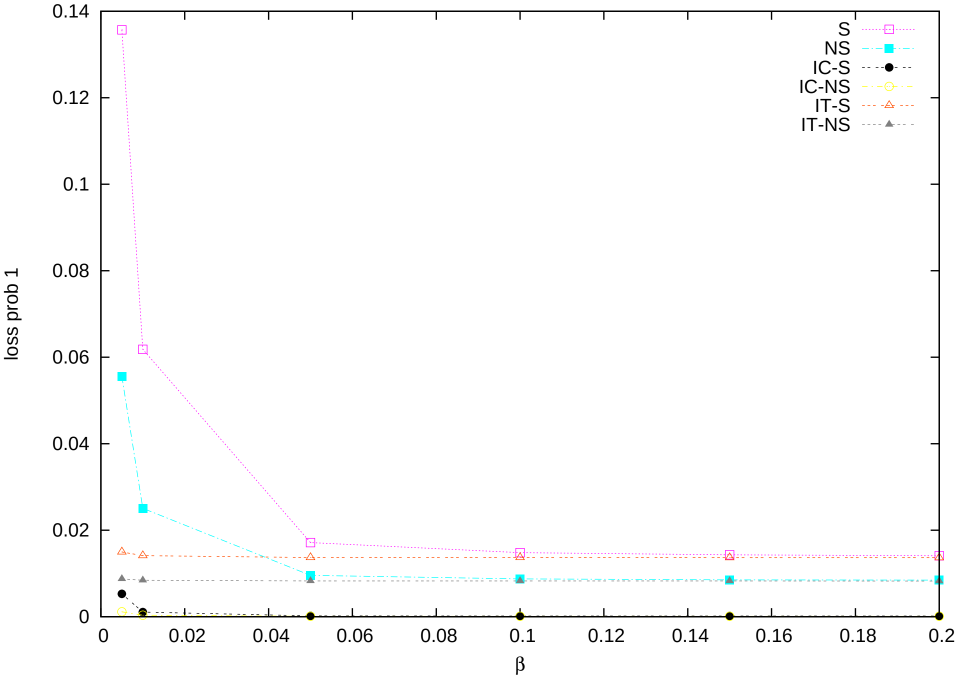


(a)

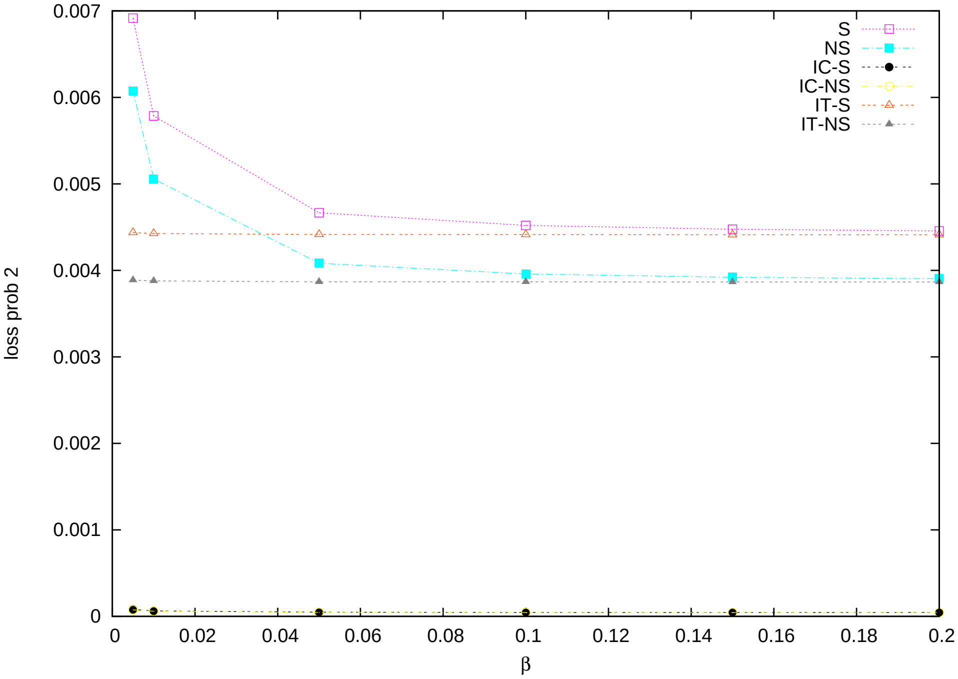


(b)

Figure 5.8: blocking probabilities versus β

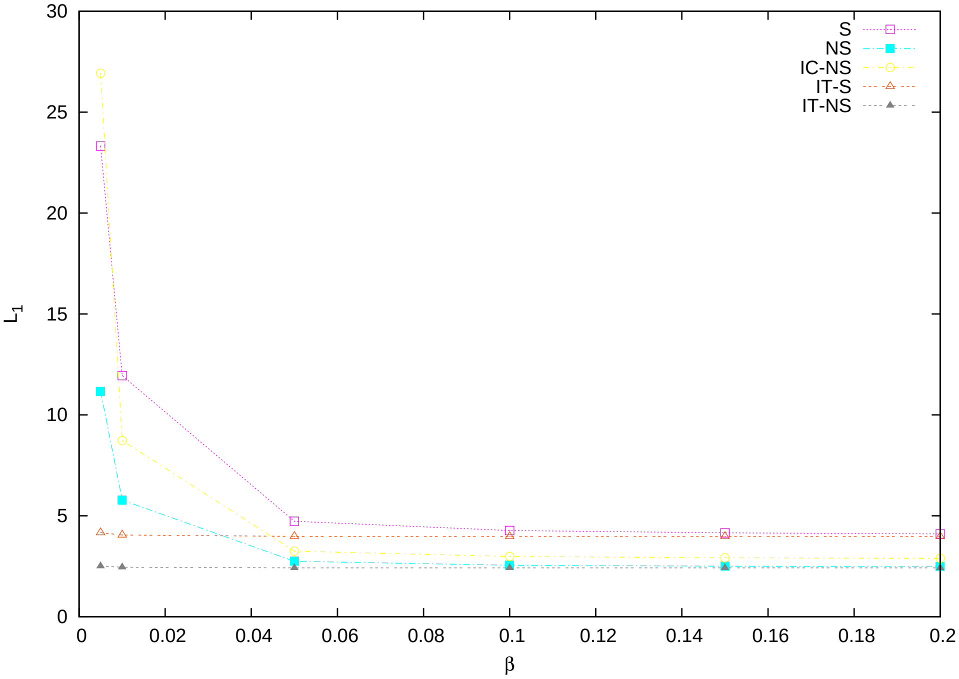


(a)

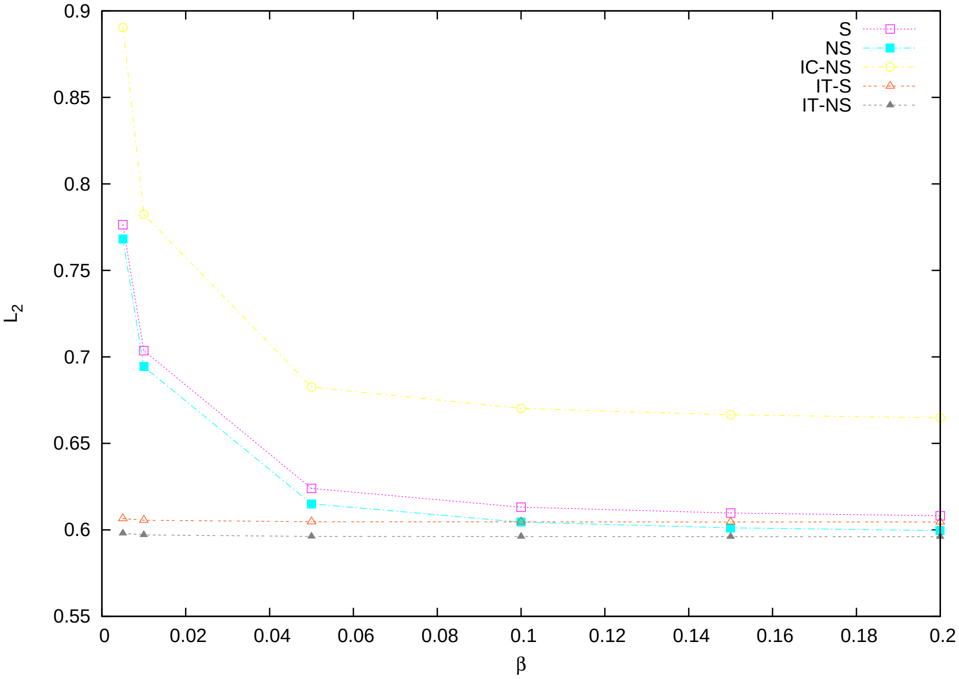


(b)

Figure 5.9: loss probabilities versus β

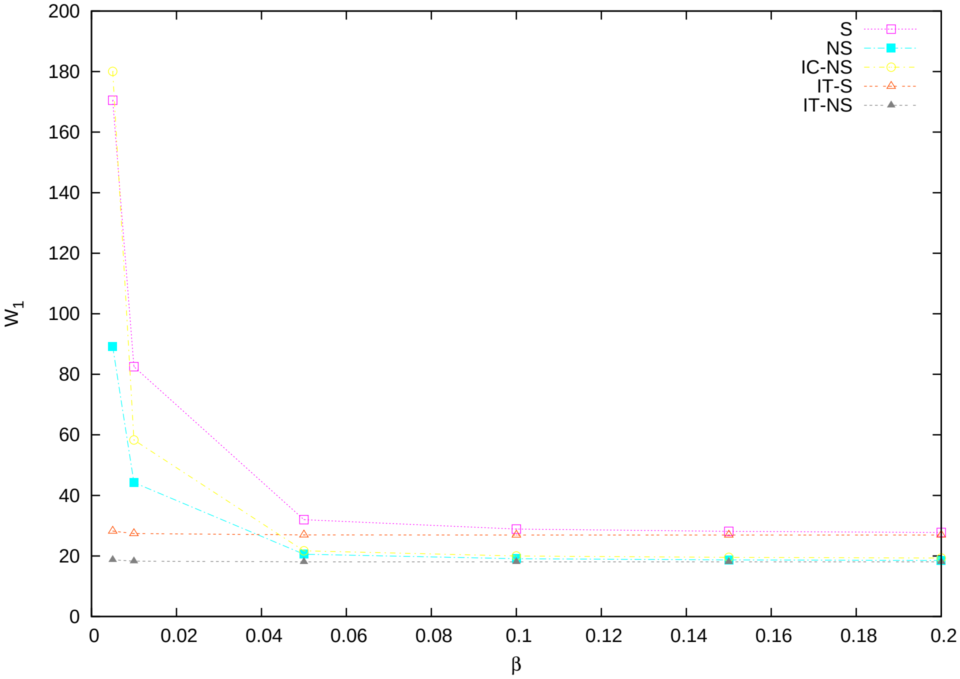


(a)



(b)

Figure 5.10: L_i versus β



(a)

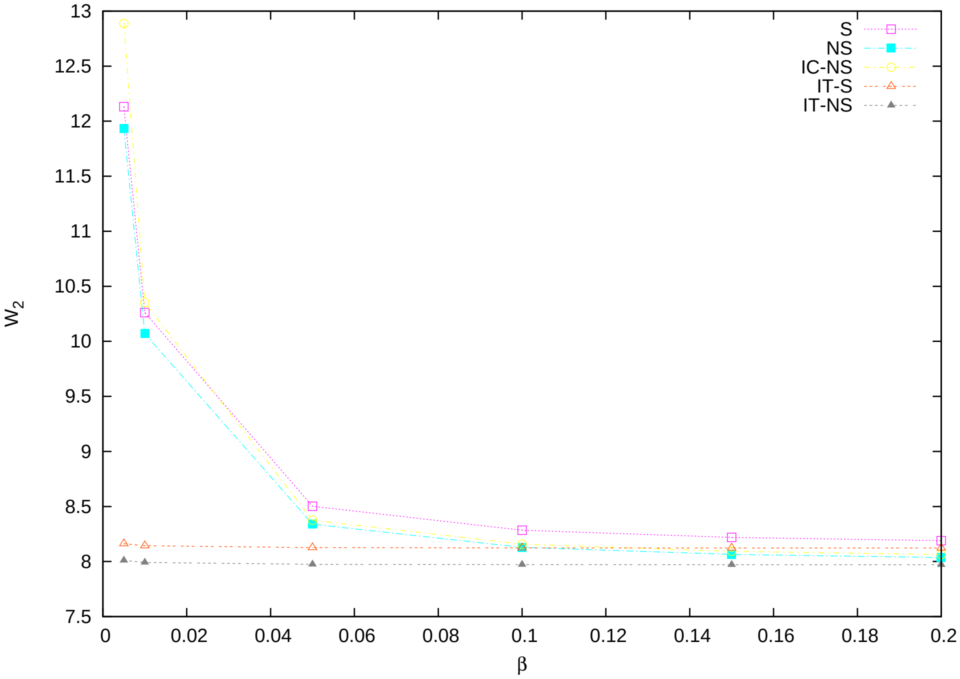


Figure 5.11: W_i versus β

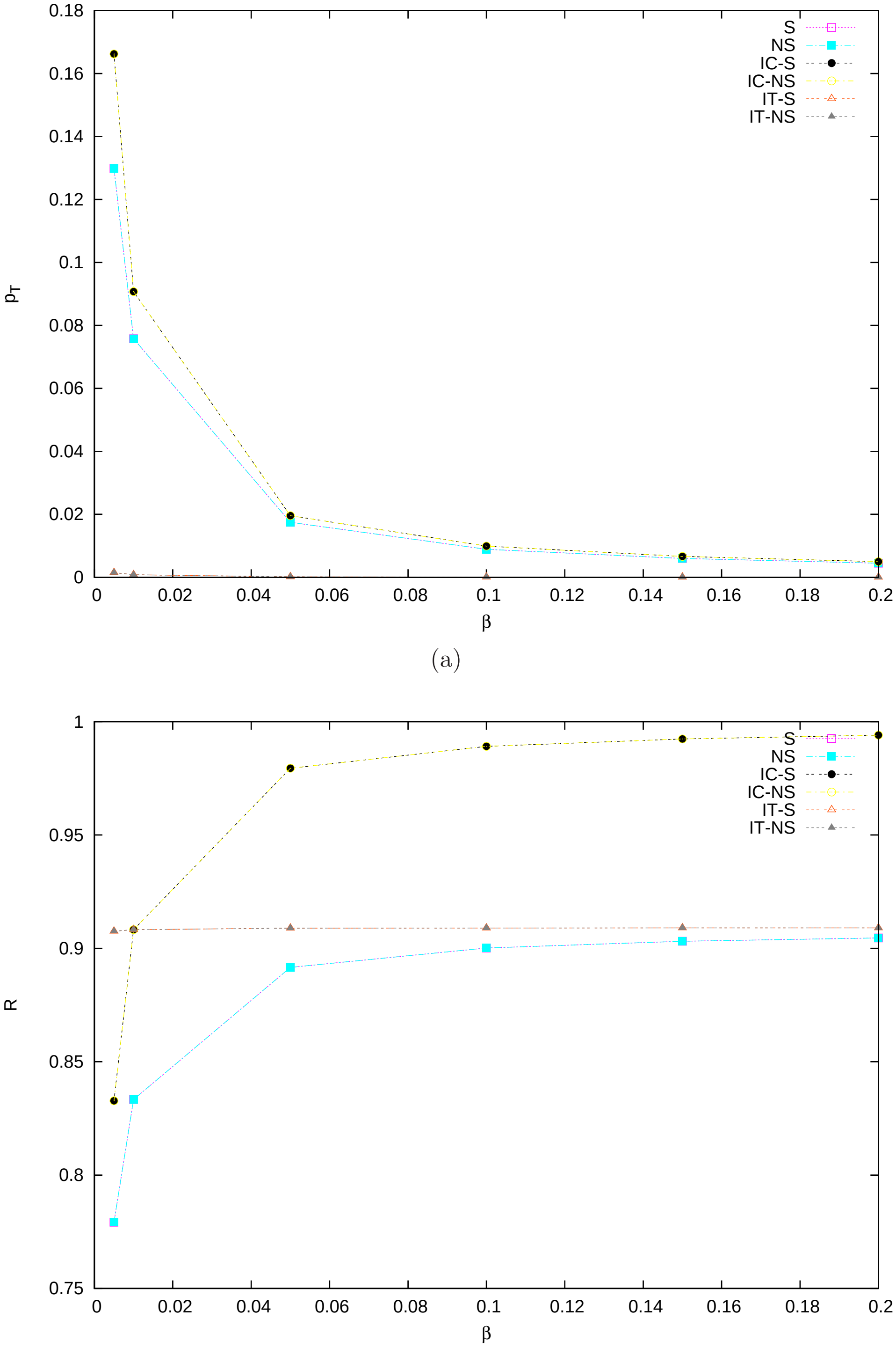


Figure 5.12: p_T and R versus β

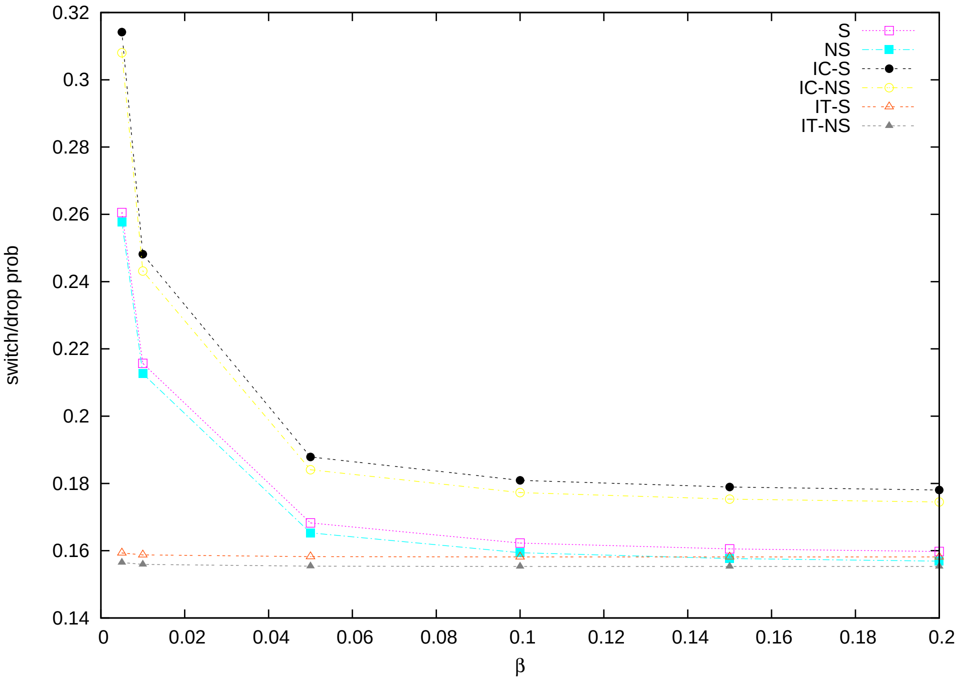
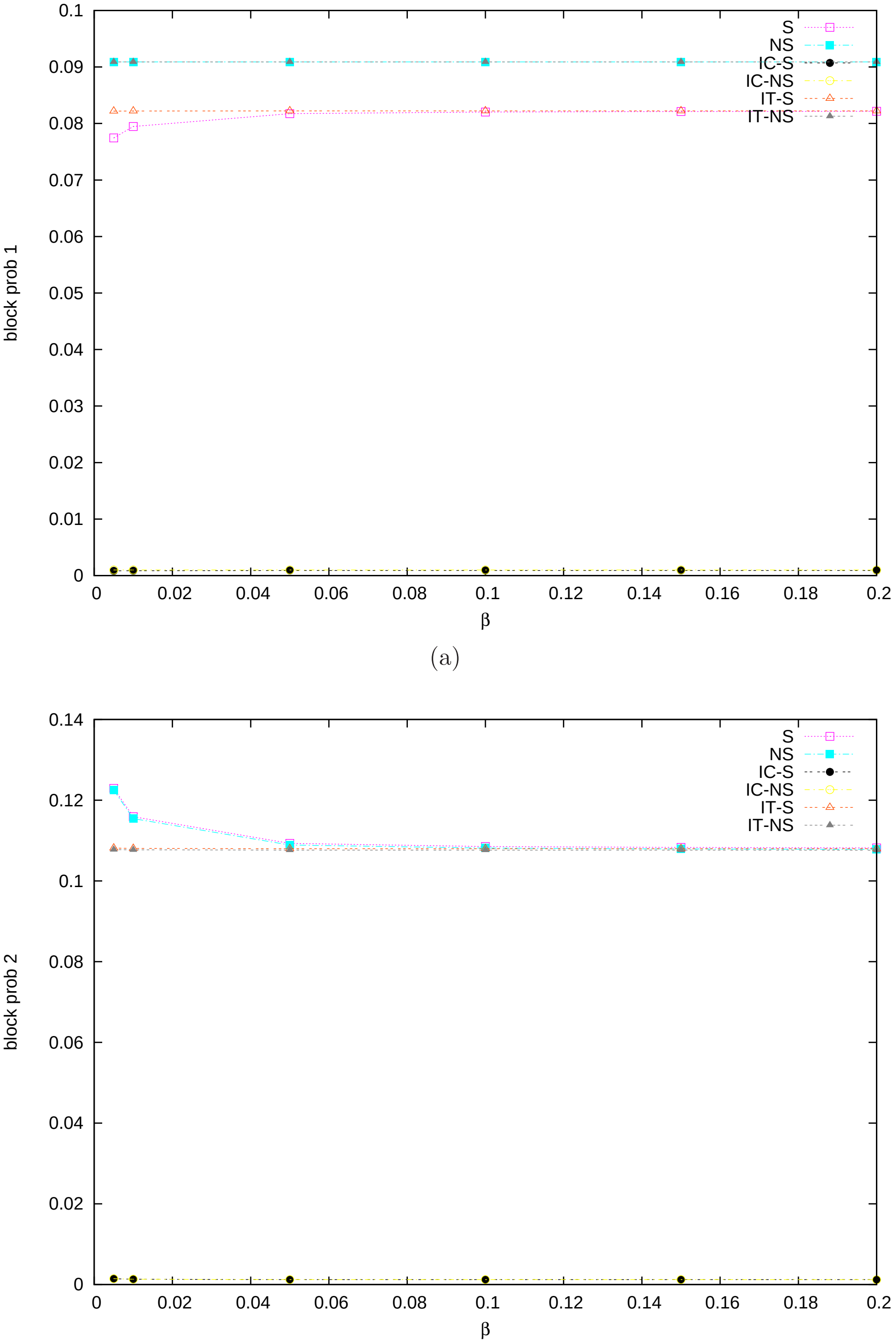


Figure 5.13: switching probabilities versus β

L_2 directly. The values reported in Table 5.2 are computed using the corresponding queue-lengths and the trafffic intensity. Across all the quantities reported, the average error is less than 5%.

5.5 Conclusions and Future Work

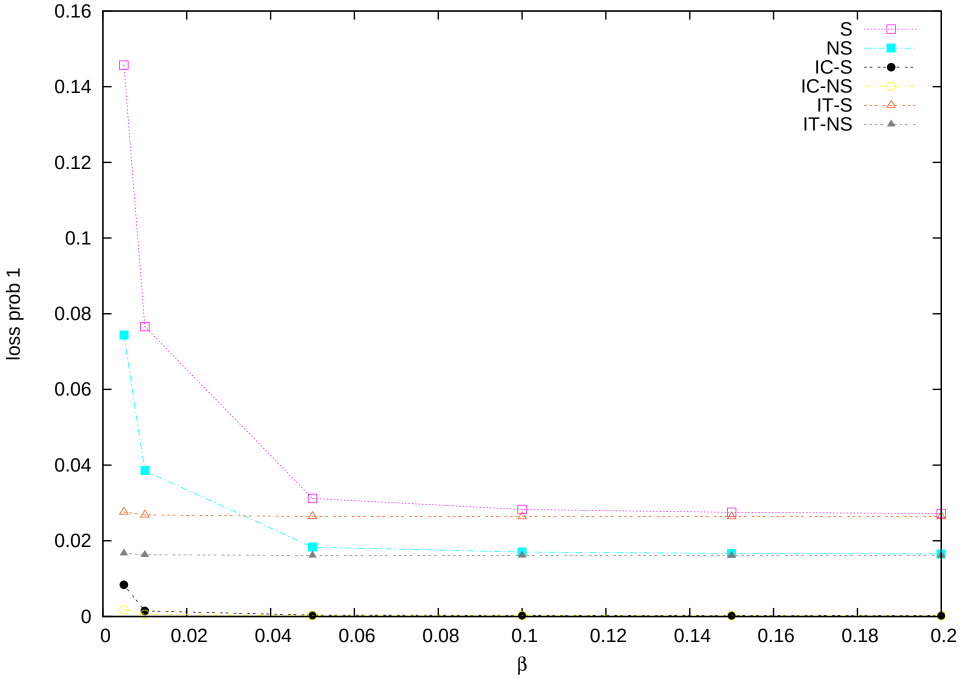
In this chapter, an agent is modeled as a 2-class non-preemptive priority queue with two kinds of failure - namely temporary and catastrophic breakdown. Within this model, we derive results for 2 cases (a) no class-switch and (b) with class-switching. Using the properties of generating functions, we obtain closed-form results for the case in which class 2 packets have a finite buffer capacity of n. By considering a finite buffer capacity for class 2, we reduce the dimensionality of the state space from



(a)

(b)

Figure 5.14: blocking probabilities versus β



(a)

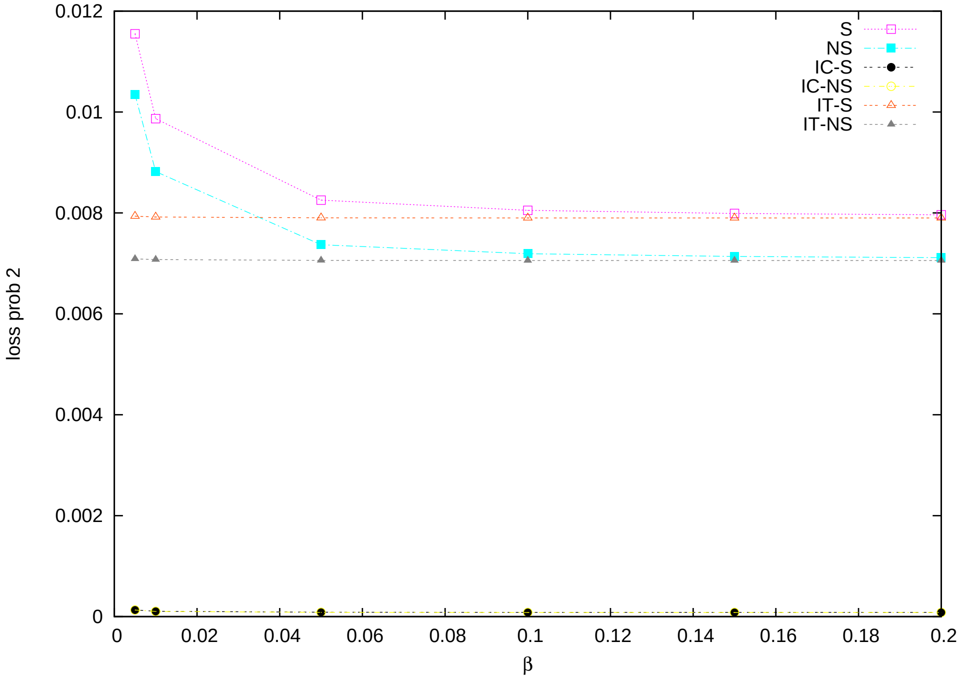
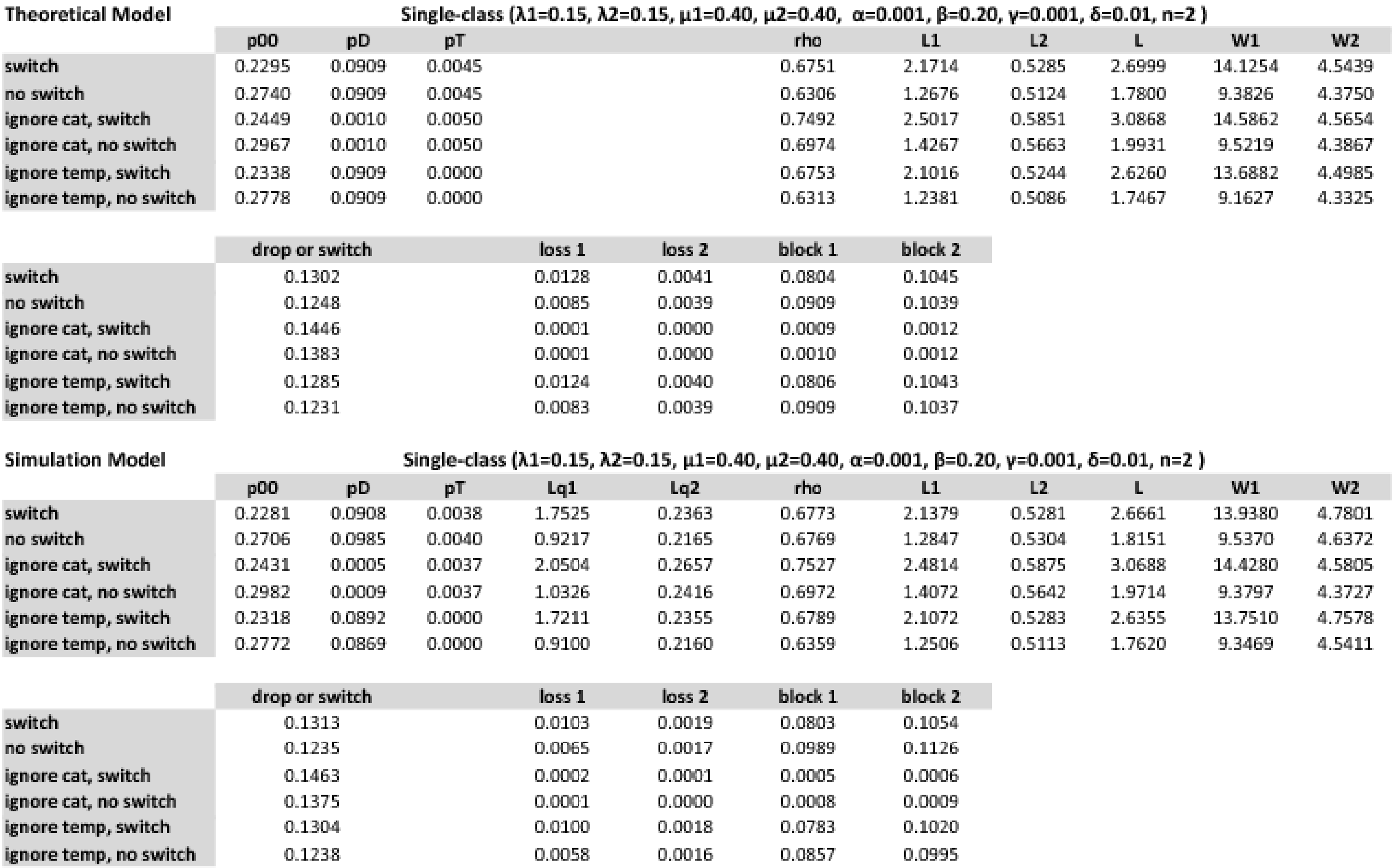


Figure 5.15: loss probabilities versus β

Table 5.2: Single class case



Theoretical Model **Single-class (λ1=0.15, λ2=0.15, μ1=0.40, μ2=0.40, α=0.001, β=0.20, γ=0.001, δ=0.01, n=2)**

	p00	pD	pT			rho	L1	L2	L	W1	W2
switch	0.2295	0.0909	0.0045			0.6751	2.1714	0.5285	2.6999	14.1254	4.5439
no switch	0.2740	0.0909	0.0045			0.6306	1.2676	0.5124	1.7800	9.3826	4.3750
ignore cat, switch	0.2449	0.0010	0.0050			0.7492	2.5017	0.5851	3.0868	14.5862	4.5654
ignore cat, no switch	0.2967	0.0010	0.0050			0.6974	1.4267	0.5663	1.9931	9.5219	4.3867
ignore temp, switch	0.2338	0.0909	0.0000			0.6753	2.1016	0.5244	2.6260	13.6882	4.4985
ignore temp, no switch	0.2778	0.0909	0.0000			0.6313	1.2381	0.5086	1.7467	9.1627	4.3325

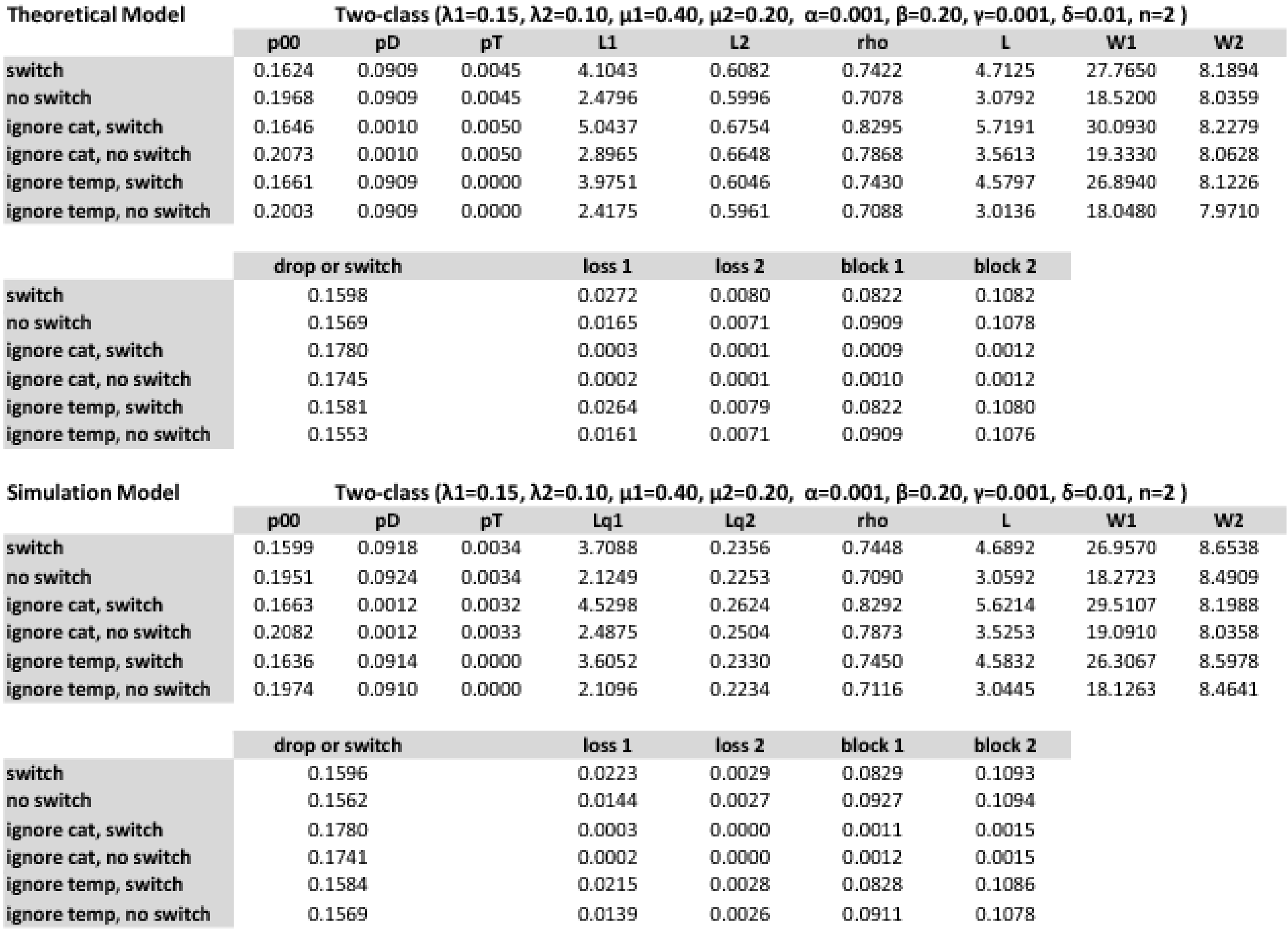
	drop or switch	loss 1	loss 2	block 1	block 2
switch	0.1302	0.0128	0.0041	0.0804	0.1045
no switch	0.1248	0.0085	0.0039	0.0909	0.1039
ignore cat, switch	0.1446	0.0001	0.0000	0.0009	0.0012
ignore cat, no switch	0.1383	0.0001	0.0000	0.0010	0.0012
ignore temp, switch	0.1285	0.0124	0.0040	0.0806	0.1043
ignore temp, no switch	0.1231	0.0083	0.0039	0.0909	0.1037

Simulation Model **Single-class (λ1=0.15, λ2=0.15, μ1=0.40, μ2=0.40, α=0.001, β=0.20, γ=0.001, δ=0.01, n=2)**

	p00	pD	pT	Lq1	Lq2	rho	L1	L2	L	W1	W2
switch	0.2281	0.0908	0.0038	1.7525	0.2363	0.6773	2.1379	0.5281	2.6661	13.9380	4.7801
no switch	0.2706	0.0985	0.0040	0.9217	0.2165	0.6769	1.2847	0.5304	1.8151	9.5370	4.6372
ignore cat, switch	0.2431	0.0005	0.0037	2.0504	0.2657	0.7527	2.4814	0.5875	3.0688	14.4280	4.5805
ignore cat, no switch	0.2982	0.0009	0.0037	1.0326	0.2416	0.6972	1.4072	0.5642	1.9714	9.3797	4.3727
ignore temp, switch	0.2318	0.0892	0.0000	1.7211	0.2355	0.6789	2.1072	0.5283	2.6355	13.7510	4.7578
ignore temp, no switch	0.2772	0.0869	0.0000	0.9100	0.2160	0.6359	1.2506	0.5113	1.7620	9.3469	4.5411

	drop or switch	loss 1	loss 2	block 1	block 2
switch	0.1313	0.0103	0.0019	0.0803	0.1054
no switch	0.1235	0.0065	0.0017	0.0989	0.1126
ignore cat, switch	0.1463	0.0002	0.0001	0.0005	0.0006
ignore cat, no switch	0.1375	0.0001	0.0000	0.0008	0.0009
ignore temp, switch	0.1304	0.0100	0.0018	0.0783	0.1020
ignore temp, no switch	0.1238	0.0058	0.0016	0.0857	0.0995

Table 5.3: Two class case



Theoretical Model Two-class (λ1=0.15, λ2=0.10, μ1=0.40, μ2=0.20, α=0.001, β=0.20, γ=0.001, δ=0.01, n=2)

	p00	pD	pT	L1	L2	rho	L	W1	W2
switch	0.1624	0.0909	0.0045	4.1043	0.6082	0.7422	4.7125	27.7650	8.1894
no switch	0.1968	0.0909	0.0045	2.4796	0.5996	0.7078	3.0792	18.5200	8.0359
ignore cat, switch	0.1646	0.0010	0.0050	5.0437	0.6754	0.8295	5.7191	30.0930	8.2279
ignore cat, no switch	0.2073	0.0010	0.0050	2.8965	0.6648	0.7868	3.5613	19.3330	8.0628
ignore temp, switch	0.1661	0.0909	0.0000	3.9751	0.6046	0.7430	4.5797	26.8940	8.1226
ignore temp, no switch	0.2003	0.0909	0.0000	2.4175	0.5961	0.7088	3.0136	18.0480	7.9710

	drop or switch	loss 1	loss 2	block 1	block 2
switch	0.1598	0.0272	0.0080	0.0822	0.1082
no switch	0.1569	0.0165	0.0071	0.0909	0.1078
ignore cat, switch	0.1780	0.0003	0.0001	0.0009	0.0012
ignore cat, no switch	0.1745	0.0002	0.0001	0.0010	0.0012
ignore temp, switch	0.1581	0.0264	0.0079	0.0822	0.1080
ignore temp, no switch	0.1553	0.0161	0.0071	0.0909	0.1076

Simulation Model Two-class (λ1=0.15, λ2=0.10, μ1=0.40, μ2=0.20, α=0.001, β=0.20, γ=0.001, δ=0.01, n=2)

	p00	pD	pT	Lq1	Lq2	rho	L	W1	W2
switch	0.1599	0.0918	0.0034	3.7088	0.2356	0.7448	4.6892	26.9570	8.6538
no switch	0.1951	0.0924	0.0034	2.1249	0.2253	0.7090	3.0592	18.2723	8.4909
ignore cat, switch	0.1663	0.0012	0.0032	4.5298	0.2624	0.8292	5.6214	29.5107	8.1988
ignore cat, no switch	0.2082	0.0012	0.0033	2.4875	0.2504	0.7873	3.5253	19.0910	8.0358
ignore temp, switch	0.1636	0.0914	0.0000	3.6052	0.2330	0.7450	4.5832	26.3067	8.5978
ignore temp, no switch	0.1974	0.0910	0.0000	2.1096	0.2234	0.7116	3.0445	18.1263	8.4641

	drop or switch	loss 1	loss 2	block 1	block 2
switch	0.1596	0.0223	0.0029	0.0829	0.1093
no switch	0.1562	0.0144	0.0027	0.0927	0.1094
ignore cat, switch	0.1780	0.0003	0.0000	0.0011	0.0015
ignore cat, no switch	0.1741	0.0002	0.0000	0.0012	0.0015
ignore temp, switch	0.1584	0.0215	0.0028	0.0828	0.1086
ignore temp, no switch	0.1569	0.0139	0.0026	0.0911	0.1078

two to one. The methodology we introduce in pretty generic in nature and can be used for other cases that involve priority and preemption. The key problem that will be encountered in this class of problems (i.e. queues with multiple failure types) is the determination of boundary probabilities. Using the matrix-based representation of the generating functions, we can make the approach proposed in this chapter generic to a class of problems - preemptive and non-preemtive queues with a vector of failures. By this generalization, it is possible to observe some structure in the stochastic process (see $A(z)\psi(z) = B(z)$) following which the properties generating functions can be suitably applied. Agents can use this analytical model efficiently to predict various measures of effectiveness (MOE). Subsequently, the computed MOEs can be used for decision-making, as is illustrated in Chapter 6 in the context of an multi-agent allocation problem.

Chapter 6

Model-Based Allocation and Pricing for a Multi-Agent Network

6.1 Introduction

In this chapter, we propose two distributed algorithms for allocating a multi-agent system using auctions and negotiation. Fundamentally, the problem that is to be solved is that of allocation - assigning n software agents to m computing entities. The Generalized Assignment Problem (GAP) as it is referred to in literature is well studied (see Chapter 2, Section 3). In fact, even finding whether the GAP is feasible is known to be NP-hard [80]. In this chapter, we will study one type of assignment problem (interchangeably called allocation problem) and the issues related with it, especially as it applies to multi-agent systems. Agents resemble software components or objects but, as pointed out in Chapter 2, they also have properties such as situatedness and autonomy. In practice, these properties translate into the capability to bid for goods and make local decisions. Furthermore, in UltraLog [3] and a COUGAAR based multi-agent system called CPE [78, 85] we build agents that have technical specifications (or TechSpecs) that detail the agents' inputs, outputs, strategies and playbooks, and resource requirements. These properties and the ability to operate as a community make the agent-based system behave as a distributed application - objects trading messages and traffic back and forth to achieve a goal.

In this context, the design of efficient allocation mechanisms is important, especially if they are sensitive to local conditions and global system survivability requirements (i.e. meeting performance constraints). Most of our work is motivated by the UltraLog scenario described in Chapter 3. From our perspective, the following aspects of the MAS allocation problem make it an interesting research topic. Firstly, the MAS being an interconnected application with end-to-end QoS requirements makes it appealing - i.e. *the application is the network* (of agents). So we are dealing not just with individual agent behavior but also collective application characteristics. Secondly, if the MAS can influence the allocation such that its QoS requirements (and that of the service provider) are met every time, it can be thought of as striving to be *survivable*[1]. So in its repertoire of strategies to adapt to its environment, the MAS has atleast one that affects allocation. Thirdly, there is a *business interaction* between the MAS and the computing entities. While the MAS needs the service provider, over-usage leads to congestion. So, the allocation mechanism will have to ascertain if the QoS requested can be paid for. Fourth, the *environment is harsh* causing failure to both the agents and the computing infrastructure. The key question, therefore, is what is QoS and how is it quantified in the presence of failures. Fifth, *who does what*? In a situation where there are interactions between agents, agents and the infrastructure,

---

[1] Survivability is the ability of a system to fulfill its mission while meeting its QoS requirements in a failure-prone environment.

and the nodes that form the infrastructure, what is private and public information, and what are the roles and responsibilities of each party? These questions motivate our problem and make our scenario rich.

6.1.1 Scope

We now briefly describe how the questions posed above will be answered in this chapter. We also define what aspects from above are within the the scope of our solution methodology. Here onwards, we will utilize terminology defined in Chapter 3 and primarily deal with the scenario through mathematical abstractions (refer Chapter 3 for background on UltraLog and MAS).

- Firstly, we model various end-to-end requirements and characteristics of the application. We allow for the MAS to reveal its desired QoS which will be accounted through constraints. For example, flow constraints specify bounds on acceptable QoS for interconnected agents. Grouping constraints, on the other hand, define which agents should not be co-located. Characteristics such as fairness of an allocation are also defined for the application as a whole.

- Secondly, the MAS will influence the allocation by bidding on QoS that it deems crucial to its survival. Paying more will result in better QoS. QoS is actually a vector of various components and agents may bid differently for components of the QoS bundle as per their individual need (and environmental conditions). An agent's bid is its own private information. There may be several strategies for survivability, we only focus on allocation.

- The bids are used by the service provider to compute a fee for the QoS delivered. This gives birth to a pricing problem in the MAS allocation context. This fee will prevent congestion if the endowments of the agents are assumed to be limited.

- Fourth, in a harsh environment (such as a battle-field) catastrophes are common. We analytically model two types of breakdown and assess their impact on performance and reliability of each node. This work is described in Chapter 4 and Chapter 5.

- Fifth, the problem of determining the ground-rules of operation and the roles and responsibilities of each participating entity in the allocation problem is that of *mechanism design*. We formally define this problem in the next section.

6.1.2 Chapter Organization

The chapter is organized as follows. We first revist the allocation problem and how it is related in the MAS context in the next section. We embark on the task of identifying an optimal allocation from the service provider's context (Section 3.1). We identify several QoS components as applicable to the distributed MAS (Section 3.2). We utilize the analytical models that capture numerous interactions (explained in Chapter 3) and to predict the impact of potential allocations on the MAS. The allocation mechanism in Section 3 (mechanism A) codifies how the information is utilized and computes the price that must be paid to the infrastructure. In order to mitigate the complexity of identifying the optimal allocation, we also identify efficient heuristics and propose decentralization strategies. In doing so, we get Mechanism B described in Section 4. This is followed by numerical examples in Section 5. In Section 6, we bridge concepts from the current and previous chapters and elucidate their overall usefulness.

6.2 Model-based allocation

Since we are dealing with a version of the Generalized Assignment Problem (GAP) [80] - allocating n agents to m nodes, we will first introduce the GAP. Subsequently we will contrast the GAP with the MAS allocation problem.

We start by recapitulating the *GAP* that can be described using terminology from the familiar Knapsack Problem. The 0-1 Multiple Knapsack Problem arises when n items have to assigned to m containers, each of given given capacity c_j $j = \{1,\ 2, \dots,\ m\}$. Let x_{ij} be the binary decision variable that is 1 when agent i is assigned to node j , 0 otherwise. It is formulated as the knapsack problem in Problem 1 where p_i and w_i are the profit and weight associated with item i. If the profit and weight associated with item i depends on the container j, the GAP arises, which is formulated similarly as Problem 2 where p_{ij}, w_{ij} and c_j are normally known in advance.

$$
\begin{aligned}
&\mathit{maximize}\ \sum_{j=1}^{m}\sum_{i=1}^{n} p_i x_{ij}\\
&\mathit{subject\ to}\\
&\qquad \sum_{i=1}^{n} w_i x_{ij} \le c_j \quad \forall\ j \in \{1, \dots,\ m\}\\
&\qquad \sum_{j=1}^{m} x_{ij} \le 1 \quad \forall\ i \in \{1, \dots,\ n\}\\
&\qquad x_{ij} = 0\ \mathit{or}\ 1 \ \ \forall\ i \in \{1, \dots,\ n\},\ \forall\ j \in \{1, \dots,\ m\}
\end{aligned}
$$

Problem 1

$$
\begin{aligned}
&maximize \ \sum_{j=1}^{m}\sum_{i=1}^{n} p_{ij}x_{ij}\\
&subject\ to\\
&\qquad \sum_{i=1}^{n} w_{ij}x_{ij} \le c_j \quad \forall\, j \in \{1, \dots, m\}\\
&\qquad \sum_{j=1}^{m} x_{ij} \le 1 \quad \forall\, i \in \{1, \dots, n\}\\
&\qquad x_{ij} = 0 \ or\ 1 \ \ \forall\, i \in \{1, \dots, n\},\ \forall\, j \in \{1, \dots, m\}.
\end{aligned}
$$

Problem 2

6.2.1 MAS Allocation Problem

If p_{ij} and w_{ij} are not known in advance and can vary depending on the allocation x, we are dealing with our MAS allocation problem. *This can happen because the profit of agent i depends not only on agent i but also on the other agents' bids or requests. If* w_{ij} *is a QoS measure, it can depend on the allocation of other agents.* Therefore the above problem can be viewed as Problem 3 where x is an $n-vector$ $(x_1,\ x_2, \dots,\ x_n)$,

$$
\begin{aligned}
&F(x) = arg\ \max_{x\in X} \ \sum_{i=1}^{n} p_i(x)x_{ij}(x)\\
&subject\ to\\
&\qquad \sum_{i=1}^{n} w_{ij}(x)x_{ij}(x) \le c_j \quad \forall\, j \in \{1, \dots, m\}\\
&\qquad \sum_{j=1}^{m} x_{ij}(x) \le 1 \quad \forall\, i \in \{1, \dots, n\}\\
&\qquad x_{ij}(x) = 0 \ or\ 1 \ \ \forall\, i \in \{1, \dots, n\},\ \forall\, j \in \{1, \dots, m\}
\end{aligned}
$$

Problem 3

an allocation, in the space of allocations X with each $x_j \in (1,\ 2, \dots,\ m)$, and p_{ij} and

w_{ij} are dependent on x, denoted as $p_{ij}(x)$ and $w_{ij}(x)$ respectively. Because of this dependence, we allocate as a group and not as individual x_{ij}s. For a given allocation x, $x_{ij}(x) = 1$ only when $j = x_i$ for $j = (1,\ 2, \ldots,\ m)$, 0 otherwise. In picking the decision variable x, we select a number of js as a group and the corresponding x_{ij}s are equal to one (rest are zero). ϕ_j represents the node-specific conditions (including stress). In the rest of the chapter we drop the notation for the dependence on x for convenience.

With the MAS allocation problem, it can be easily seen that p_{ij} and w_{ij} are dependent on x, because agents bring in different bids and parameters, causing the profit to be dependent on the overall demands of the group. Sometimes, the capacity c_j may depend on the allocation x. By capacity one could mean the number of agents in a node. To see this point, we must understand that as soon as the n agents are allocated (i.e. x is determined) they demand a QoS through a parameter called repair rate or β. Because of varying stresses and QoS requests, it is not possible to ascertain the number of agents that can be accommodated in a node. Moreover, the m nodes are only obligated to fulfill the (respective) maximum possible QoS for the demanded QoS (i.e. they honor the QoS request only if the profit from the bids accommodate the maximum QoS). These factors cause c_j to be dependent on x.

To further complicate matters in the allocation problem, w_{ij} could be non-linearly dependent on x. This could be true for p_i as well. To see that such a case arises, just consider that w_{ij} is the QoS and p_i is the profit from agent i. QoS such as response time at a node is non-linearly dependent on the input parameters (this is easily observed from the queueing models in Chapter 4 and Chapter 5). Similarly, p_i could be dependent the bids of all the agents causing it to be non-linear.

The main question that arises in our problem is - how does one solve for x when parameters such as w_{ij} are not known? We answer this question in the next paragraph.

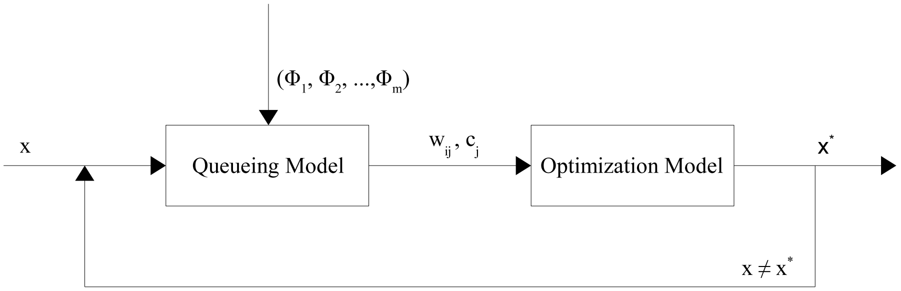


Figure 6.1: Model-Based Allocation

6.2.2 Model-Based Allocation

The answer to the allocation problem lies in a model (in our case, a queueing model such as the one in Chapter 4 or Chapter 5) that will assist in the computation of all these parameters (such as w_{ij}) for a given allocation x so that the optimization problem can be solved. In this section, we presented a simplified version of the *GAP* as it applied to the MAS allocation. The actual problem which we will consider in the following sections considers additional interactions such as *environmental* factors (i.e stresses or failure). A model that is chosen to assist in MAS allocation has to be able to deal with the aforementioned problem characteristics as well as capture environmental interactions. Since we integrate the model in the allocation process (i.e. model is used for probing the state space), we refer to this procedure as *Model-Based Allocation* (Figure 6.1). Hence this procedure is similar to model-predictive control [72].

6.3 Design of Mechanism A

In this section, we present the first distributed algorithm for allocating the agents on the computing infrastructure. The algorithm is an auction-based mechanism which simultaneously allocates the agents and charges them for the QoS requested. Figure 6.2 represents the mechanism we describe in this section.

The software agents are the consumers, i.e. each of them demand a bundle of an information good consisting of a performance and a reliability component. The infrastructure nodes supply the requested information goods to the agents by adjusting parameters within their control. There are n agents and m infrastructure nodes. However, the m nodes of the infrastructure are owned by the same Principal. The proportion in which the bundle is sold gives rise to a range of QoS values.

Agents bid on how much they are willing to pay for the bundle. In return, the Principal allocates the agents on one of its m nodes, announces the price for the QoS provided and satisfies any constraints that the agents may reveal to the provider. Constraints, for example, may refer to special conditions with respect to the QoS received by one agent and that received by other agents. Some examples of constraints on the part of the agents is provided in Section 6.3.1. Agents are aware of the differentiated service but they disclose their value of the service only to the provider. Agents do not share their individual valuations with other peers.

Upon receiving bids from all agents, the service provider decides what QoS each of them is entitled to. The first step in this decision process to calculate the QoS bundles that can be delivered depending on the environmental conditions. These conditions are possibly different at each of the m nodes. The second step is to utilize the sorted bids to partition the agents into the different QoS levels. The third step is to allocate

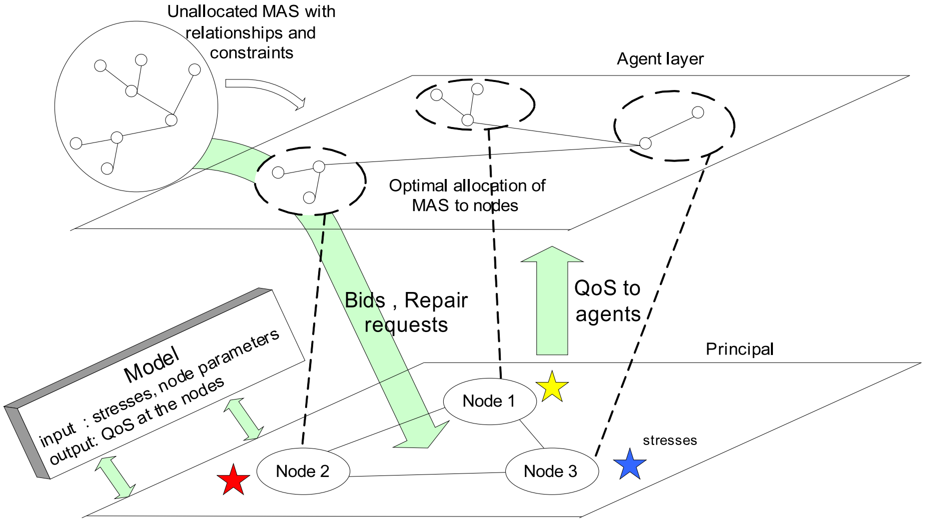

Unallocated MAS with relationships and constraints
Agent layer
Optimal allocation of MAS to nodes
Bids , Repair requests
QoS to agents
Principal
Model
input : stresses, node parameters
output: QoS at the nodes
Node 1
Node 2
Node 3
stresses

Figure 6.2: Schematic of Model-Based Allocation Mechanism

them to one of the infrastructure nodes after satisfying the agents' constraints.

Every agent is admitted at one of the QoS levels. Agents generally pay more in order to receive better QoS. The price is computed using a variant of the Vickery-Clarke-Groves (VCG) mechanism [7]. The fee charged is dependent on the usage rather than a flat service fee. Through the bidding process, the agents easily partition themselves into QoS levels they will ultimately receive.

6.3.1 Formal definitions

A mechanism is a game with players, outcomes, players' strategies, outcome function and players' payoff functions. We now define the aforementioned aspects and rules concerning our mechanism.

1. Players: Although there are m infrastructural nodes, these belong to the same Principal. So along with the Principal and n agents we have $n + 1$ players in the game. The agents are denoted as player $i = 1, ..., n$ and the Principal as

player $i = 0$.

2. Profile of values: A profile is a collection of values for a given variable (say α) with one value for each player. Now $\alpha = (\alpha_1, ..., \alpha_n)$. Borrowing from game theory, if for $i \in [1,\ n]$, α_i corresponds to the value for the i^{th} player, α_{-i} is the collection of values for all players except i, i.e. $\alpha_{-I} = (\alpha_1,\ ...,\ \alpha_{i-1},\ \alpha_{i+1}, ...,\ \alpha_n)$.

3. Service and QoS: The Principal computes the service levels that can be provided by adjusting some internal parameters. In this case, the Principal has a model $M_j(\beta_j,\ \phi_j)$ using which it will estimate the QoS that can be provided (at node j, β_j is the parameter that can be controlled while ϕ_j constitutes the set of parameters that are fixed and/or not controllable). By adjusting β_j for $j \in (1, \ldots,\ m)$, the Principal controls the number of levels of service (L) that are provided. A maximum of m levels of service are possible because there are m nodes forming the infrastructure, i.e. $L \leq m$. The QoS bundle at node j is denoted by d_j and let $D = (d_1,\ d_2,\ ...,\ d_L)$. Each d_j consists of performance and reliability components. Both L and D are not known ahead of time, they are computed during allocation as something that would optimize the Principal's objective.

4. Agents' Actions: The Principal expects each agent to individually disclose its desired β_i^a. Let $\beta^a = (\beta_1^a,\ ...,\ \beta_i^a,\ ...\ ,\ \beta_n^a)$. Based on their estimates β^a, the agents announce their individual values for the service (which consists of several components) to be provided. Depending on how all the agents value the service, the Principal prices the QoS. Let these *bids* by the agents be $b = (b_1, ...,\ b_i, ...\ , b_n)$ where b_i is the bid for agent i. The true values or *types* are denoted by $\theta = (\theta_1, ...\ , \theta_i,\ ...\ ,\ \theta_n)$. Let B_i denote the space of allowable

bids and Θ_i denote the space of agent type for agent i, $i \in \{1, ..., n\}$. Let $b = b_{(1)}, b_{(2)}, \dots, b_{(n)}$ denote the order statistics corresponding to $b_1, b_2, \dots, b_n$. The agents pay the Principal a *usage fee* upon receiving the service. Agents collectively express their constraints to the Principal which they expect the Principal to honor. These constraints may be regarded as general characteristics of the agents (i.e. the MAS as a whole) that remain relatively constant unlike parameters such as b or β . A few constraints are listed below.

(a) A *flow constraint* could set bounds on the worst-case total mean delay of a set of agents (w_{ij} denotes delay at agent i if it has been allocated to node j), i.e.

$$\sum_{j=1}^{m} \sum_{i \in \{F_z\}} w_{ij} A_{ij} \leq \Delta_{F_z} \ \ \forall\ F_z \in F$$

where F_z is a flow $\{f_1, \ ..., \ f_i, \ \dots, \ f_t\}$ such that each $f_i \in \{1, ..., n\}$ and Δ_{F_z} is the maximum tolerable worst-case delay for flow F_z. Let $F = \{F_1, \ ..., \ F_{N^f}\}$ denote the set of all flows where N^f is the total number of flows that the MAS needs.

(b) A *grouping constraint* forbids certain agents from residing on the same node j i.e.

$$\sum_{i \in G_z} A_{ij} = 1 \ \ \forall\ j \in \{1, \ ..., \ m\}, \ G_z \in \ G$$

where G_z is a group $\{g_1, \ ..., \ g_i, \ \dots, \ g_y\}$ such that each $g_i \in \{1, \ ..., \ n\}$ and $g_d \neq g_e$ for $\forall\ g_d, \ g_e \in G_z$. Let $G = \{G_1, \ ..., \ G_{N^g}\}$ denote the set of all groups where N^g is the maximum number of (non-permissible) groups the in the MAS. In order for the Principal to take these constraints into account, the agents collectively reveal G, F and $\Delta = \{\Delta_1, \ldots, \Delta_{N_f}\}$.

(c) Agents may have a *fairness constraint* which states that

$$b_i \geq b_{i+1} \implies x_i \geq x_{i+1} \quad \forall\, i \;\in\; (1, \ldots,\; n-1). \tag{6.1}$$

This constraint can also be expressed as

$$(x_{(i)} - x_{(i+1)})(b_{(i)} - b_{(i+1)}) \geq 0 \quad \forall\, i \;\in\; (1, \ldots,\; n-1).$$

Since $b_{(i)}$are order statistics, $(b_{(i)} - b_{(i+1)}) \geq 0$ for adjacent agents. Let $x_{(i)}$ denote the QoS received by the i^{th} highest bidder. The above constraint is satisfied only if the higher bidding agent among any two adjacent agents (as per order statistics) receives greater or equal service relative to its counterpart.

(d) Let $\beta^a = (\beta^a_1,\; \ldots,\; \beta^a_i,\; \ldots,\; \beta^a_n)$ indicate the repair rates requested by the n agents. Not all agents may get the β^a_i they request. Some of them may get less than their desired rate and others more, but they are all guaranteed something from β^a if the allocation is successful. Let$\beta^a_{max} = max(\beta^a)$.

5. Principal's Action: The Principal must allocate the agents, decide the QoS to be provided and subsequently price the QoS based on the bids b. The Principal computes the level of QoS each agent should be served at. In other words, it should compute $x = (x_1,\; ...,\; x_i,\; ...\;,\; x_n)$ where x_i is the QoS bundle that agent i buys. Since D is not known in advance, the Principal implements an algorithm that determines D while trying to find the optimal allocation $\hat{a}^*$. As part of

the algorithm, a set of repair rates β^p has to found in response to the requested repair rates β^a. Let $\beta^p = (\beta_1^p,\ \ldots,\ \beta_j^p,\ \ldots,\ \beta_m^p)$, where each β_j^p could be one among the set of requested repair rates for node j - the set for node j being $\beta_{set-j}^p = \{\beta_i^a \mid A_{ij} = 1\}$. Let B be the space of all possible repair rates chosen at the m nodes. For this algorithm, the Principal uses the model M_j and hence we refer to this procedure as *model-based allocation*. To contrast, first assume all the QoS levels (i.e. the set D) are known in advance. Then Principal tries to find the solution

$$x^*(b,\ \beta^a) = arg \max_{[x_i,\ \hat{x_i}]\in D} \sum_{i=1}^{n}(b_i x_i - \hat{p}_i \hat{x_i})$$

where $[x_i,\ \hat{x_i}]$ is the QoS delivered to the agent i, $\hat{p}_i$ is the penalty suffered per agent i, and b_i is the bid of agent i. For now, we can think of $\hat{p}_i\hat{x_i}$ as the cost incurred by the infrastructure to provide the service to agent i. Later on, we will see that this cost can be used to prevent the infrastructure from overcharging the agents. Since D is known, the above is an admission control problem for the Principal where the first highest bidders are admitted into level m at $QoS\ x_i \geq d_m$, the second highest bidders are admitted into level $m-1$ at QoS $d_{m-1} \leq x_i \leq d_m$ and so on until all agents are admitted. But the MAS allocation problem is different. The number of levels of QoS L or the delivery rates D are not known until allocation is completed. This is because d_k at node k depends on the agents allocated to node k and interactions thereof. The solution to this problem is presented in Section 6.3.3. The Principal has a few constraints of its own.

(a) Each agent i is served at only one node at QoS level x_i, i.e.

$$\sum_{j=1}^{m} A_{ij} = 1 \quad \forall\, i \in \{1, \dots,\ n\}.$$

(b) The *universal service coverage constraint* (every agent is admitted) has to be satisfied also, i.e. $\sum_{j=1}^{m} \sum_{i=1}^{n} A_{ij} \geq n$.

(c) From a stability standpoint, each node j must satisfy the condition $\frac{\lambda_j}{\mu_j} < 1$ (*traffic intensify constraint*), where λ_j is the total arrival rate of tasks at node j and μ_j is the processing speed of node j.

6. Sorting Bids for Bundles: Since each bundle consists of more than one QoS component, the agents can announce their bids with a certain amount apportioned for each QoS component. For example, if an information good consists of two QoS components i.e. $x = [x_P,\ x_R]$, a bid b may be of the form $b = [b_P,\ b_R]$ where P and R denote different types of QoS. One way to sort the bids is to just sort it by total bid value by summing the individual portions. This causes a problem if the sum of the bids are too close to each other or are tied. To avoid such problems, the Principal could discern what bid b means by taking the ratio $\chi = b_P/b_R$. If $\chi > 1$, then x_p is valued more than x_R and vice versa. Extending this argument to the general case, the Principal can determine which component of QoS comes first in the *QOS lexicon* so that the bids can be lexicographically sorted. This is useful if the agents prefer different QoS quantities at different periods as determined by the operational tempo.

7. Usage Price: Based on the agents' bids and the allocation, the Principal computes the usage price for each level of QoS i.e. for each bundle. Let $p =$

$(p_1,\ p_2,\ ....,\ p_m)$ denote the prices for the m levels. If penalty $\hat{p}_i = 0$ ($\forall\ i$), the price for level 1 traffic (the lowest level) is set at a constant zero,

$$p_1(b) \equiv 0$$

and the price for level k is

$$p_k(b) = p_{k-1}(b) + (d_k - d_{k-1}) b_{(n - \sum_{i=1}^{n} \sum_{j=k}^{m} A_{ij})}$$

where $b_{(n - \sum_{i=1}^{n} \sum_{j=k}^{m} A_{ij})}$ is the highest bid of all agents that were rejected at level k or higher. It is to be noted that level 1 QoS does not necessarily indicate the QoS offered on node 1. If $\hat{p} \neq 0$, then $p_1(b) \equiv \hat{p}_1 \hat{x}_1 / x_1$ where x_1 and $\hat{x}_1$ are the QoS at the lowest level and $\hat{p_1}$ is the penalty at the lowest level.

8. Agents' Utility: The agents have their private utility functions

$$u_i(b,\ \theta_i) = \theta_i x_i - \sum_{j=1}^{m} p_j A_{ij} \quad \forall\ i \in \{1,\ ...,\ n\} \tag{6.2}$$

where $\sum_{j=1}^{m} p_j A_{ij}$ is payment of agent i.

From the above definitions, summarizes the aspects controlled by agents and the

Table 6.1: Parameters controlled by the Application and the Infrastructure for Mechanism A

Controlling Entity	Controlled Quantities
Application / MAS	$\{b,\ \beta^a\}$ i.e. the bids and the associated repair rates
Infrastructure	$\{\ p,\ \ \beta^p,\ \hat{a}\}$ i.e. the infrastructure controls the prices, the assigned repair rates and the allocation

Principal for Mechanism A.

6.3.2 Queueing Model

In order to assist in the allocation, an analytical model $M_j(\beta_j,\ \phi_j)$ will be utilized by the Principal to evaluate the effect of allocating an agent to a node j on various QoS components. We will now describe the QoS components and how these can be obtained from the model.

First, the analytical model M_j comprises of six parameters, $\phi_j = (\lambda_j,\ \mu_j,\ \alpha_j,\ \gamma_j,\ \delta_j)$ and β_j. We divide the parameters into those that are controllable (β_j) and those that are inherent to the node or environment dependent (ϕ_j). β_j is controlled by the Principal and is used to vary the QoS provided. Some parameters in ϕ_j are measured while others could be inherent to the node or operating environment (more interpretation in Table 6.2, formal definitions of the parameters are in Chapter 4).

Second, the information good (or QoS) x_i received by agent i is a bundle of a performance and a reliability component. As x_i depends on the allocation, the goal is to compute it utilizing $M_j(\beta_j,\ \phi_j)$. M_j will capture several interactions from a performance and reliability standpoint, such as those between the agents in node j, the agents and the harsh environment, and the agents and the node.

6.3.2.1 QoS Components

For agents operating in harsh environments, one component of QoS may be valued more than the other depending on the environmental operating conditions (also called *optempo*). The performance component relates the mean service time requirement $1/\mu_{ij}$, where μ_{ij} is the service rate of agent i on node j. Considering a single model,

Table 6.2: Description of Model Parameters as Applicable to Military Logistics

Parameters (exponentially distributed)	Description
repair rate β_j	controlled by node (for example, by making *soft* changes - deadlock resolution mechanism, resolving thread contentions, killing processes, changing algorithm or scheduling policy)
arrival rate λ_j	total arrival rate at node j depends on the number of agents allocated to node j. measurements from history
processing speed μ_j	inherent/fixed at every node j
stress rate α_j	environmental factor, measured by node j by monitoring operating environment
attrition rate γ_j	relates to the health factor of a node j, inherent to a node and the effect the environment has on the node
re-hydration rate δ_j	inherent to infrastructure, indicates how fast an agent can be re-spawned

the subscript i can be dropped (all agents on node j will experience the same service rate). $1/\mu_j$ is the performance component of the QoS bundle for agents on node j. It is assumed that agents are willing to pay more for a better $1/\mu_j$. The reliability component of QoS considered is availability. Let R_j denote the availability or the fraction of time the infrastructure node is available for service. Recalling results from Chapter 4 for the single class model and setting $\delta_2 = 0$ and $\delta_1 = \delta$,

$$R_j = \psi(z) \mid_{z=1} .$$

In order to satisfy the constraints of the flows, the Principal needs to compute the waiting time

$$W_j \quad = \quad \frac{L}{\lambda_j(1-p_D)^2}$$

where $\lambda_j = \sum_i \lambda_{ij} A_{ij}$,

$$p_D \quad = \quad \frac{\gamma}{\delta+\gamma}. \tag{6.3}$$

and

$$L = \frac{p_D\delta(\beta+B_0) + p_{000}B_0\mu - 2\lambda p_D\delta - (\lambda+\alpha)p_{000}\mu}{A_1} + A_2\frac{p_D\delta(\beta+B_0) - \lambda p_D\delta}{(A_1)^2}.$$

Here $A_1 \equiv AB_0 - \lambda(A_0+B_0) + \lambda^2 - \mu B_0 + \mu\lambda - \alpha\beta$, $A_2 \equiv A_0B_0 - 2\lambda(A_0+B_0) + 3\lambda^2 + \mu\lambda - \alpha\beta$, $A_0 \equiv (\gamma+\alpha+\lambda+\mu)$ and $B_0 \equiv (\gamma+\beta+\lambda)$. Also,

$$p_{000} \quad = \quad \frac{p_{001}\delta(\lambda z^* - B_0)}{(\mu B_0(1-1/z^*) + \lambda\mu(1-z^*))}$$

where $z^* \in [0,1)$ and $(A_0z^* - \lambda(z^*)^2 - \mu)(B_0 - \lambda z^*) - \alpha\beta z^* = 0$. (The detailed derivations for this model are in Chapter 4).

6.3.2.2 Normalization

Let $P = (p_1, \ \ldots, \ p_i, \ldots \ , \ p_n)$ denote the performance component of all agents such that $p_i = \mu_j$ if $A_{ij} = 1$. Let $R = (r_1, \ \ldots, \ r_i, \ldots \ , \ r_n)$ denote the reliability component of all agents where $r_i = \psi_j(1)$ if $A_{ij} = 1$. Similarly, the absolute waiting times of the agents are defined as $W = (w_1, \ \ldots, \ w_i, \ldots \ , \ w_n)$. In order to obtain relative measures of performance, we normalize the QoS components. We list a few examples (i and j are the indices for the agent and node respectively):

1. Relative Performance:

$$p_i^{rel} \equiv \frac{\mu_j}{\mu_{max}} \; if \; A_{ij} = 1 \tag{6.4}$$

   where p_i^{rel} denotes the relative processing speed and $\mu_{max} = max(\mu_1, \ldots, \mu_i, \ldots, \mu_m)$.

2. Relative Waiting Time:

$$w_i^{rel} \equiv \frac{w_j}{(\sum_{j=1}^{m} w_j)/m} \; if \; A_{ij} = 1 \tag{6.5}$$

   where w_i^{rel}denotes the relative waiting time of agent i.

3. Relative Repair Rate:

$$\beta_i^{rel} \equiv \frac{\beta_j^p}{\beta_{max}} \; if \; A_{ij} = 1 \tag{6.6}$$

   where β_{max} is the maximum repair rate that the Principal can provide.

4. Availability: By definition, availability (R_j) is a probability, directly giving a relative QoS measure. Note that $\psi_j(1) \in [0, 1]$. $R_i \equiv R_j$ if $A_{ij} = 1$.

All relative QoS components are unit-less physical quantities in the range $[0, 1]$. This ensures that quantities in the bundle can be compared easily. μ_{max} and β_{max} are assumed to be known to the Principal. The model not required in Equation (6.6) and Equation (6.4). It is required in Equation (6.5) and for computing availability R_i.

6.3.3 Solution to the Principal's problem

Let $\hat{a} = (a_1,\ a_2,\ \ldots,\ a_n)$ denote one possible allocation of the n agents to the m nodes. From $\hat{a}$, we know that the decision variable corresponding to agent i, $A_{ia_i} = 1$. Let A denote the space of all allocations $\hat{a}$. Also, during the allocation a β^p has to be found in response to the requested repair rates β^a. Both $\hat{a}$ and β^p will affect the QoS at each node. Once $\hat{a}$ and β^p are chosen, the resulting QoS d_j on node j can be computed using model M_j. If $A_{ij} = 1$, $d_j = [x_i,\ \hat{x_i}]$ where x_i and $\hat{x_i}$ are the QoS components received by agent i. Collecting all the pieces of the problem, the complete optimization problem faced by the Principal is

$$\hat{a}^*(b,\ \beta^a) = arg \max_{\hat{a} \in A,\ \beta^p \in B} \sum_{i=1}^{n} (b_i x_i - \hat{p}_i \hat{x}_i) \tag{6.7}$$

subject to

$$\sum_{j=1}^{m} A_{ij} = 1 \quad \forall\, i \in \{1, \ldots,\ n\} \tag{6.8}$$

$$\sum_{j=1}^{m} \sum_{i \in \{F_z\}} w_{ij} A_{ij} \leq \Delta_{F_z} \quad \forall\ F_z \in F \tag{6.9}$$

$$\sum_{i \in G_z} A_{ij} \ = 1 \quad \forall\ j \in \{1,\ \ldots,\ m\},\ G_z \in\ G \tag{6.10}$$

$$(x_{(i)} - x_{(i+1)})(b_{(i)} - b_{(i+1)}) \geq 0 \quad \forall\ i\ \in\ (1, \ldots,\ n-1) \tag{6.11}$$

$$\sum_{j=1}^{m} \sum_{i=1}^{n} A_{ij} \geq n \tag{6.12}$$

$$\frac{1}{\mu_j} \sum_{i=1}^{n} \lambda_i A_{ij} < 1 \quad \forall\ j \in \{1,\ \ldots,\ m\} \tag{6.13}$$

As defined earlier, Equation (6.9), Equation (6.10) and Equation (6.11) are the agents' constraints. The rest of the constraints belong to the Principal. Generally,

x_i, $\hat{x_i}$ and w_{ij} are non-linearly dependent on the allocation $\hat{a}$ and the repair rate β^p. For convenience, we do not show this dependence in the formulation. x_i and $\hat{x_i}$ are relative QoS measures. w_{ij} (and correspondingly ΔF_z) are absolute QoS values computed from the model or known to the application from the TechSpecs. $\hat{a}^*$ is the allocation $(\hat{a},\ \beta^p)$ that optimizes the objective function.

6.3.3.1 Exhaustive search

The optimization problem described above is non-linear and discrete in the objective function and constraints. Therefore, exhaustive enumeration is a possibility. But as the problem size increases, the search quickly becomes intractable. Within exhaustive search, we need to specify how the aforementioned queueing model is used to compute the QoS x. This along with inputs, outputs, and roles of the parties involved is provided in Algorithm 1.

6.3.3.2 Greedy allocation heuristic

The following greedy algorithm is used to allocate the agents to the infrastructure nodes. As mentioned earlier, if the QoS levels are known prior to beginning the allocation, the Principal would have to admit a certain number of agents at every level subject to the capacity constraints at that level. Here we get to know of the QoS levels as we allocate. For the allocation process we utilize an analytical model $M_j(\beta_j,\ \phi_j)$ primarily to sort the available machines by attainable QoS. upgrade(A) : called if the forward pass completes without being able to allocate all the agents.

6.3.3.3 Algorithm Complexity

Exhaustive search

The exhaustive search for $\hat{a}^*$ is an exponential complexity algorithm. Generating all possibilities for $\hat{a}$ alone has m^n total iterations. Furthermore, β^p has to be chosen also. Assuming each agent i requests a unique β_i^a, a total of m values have to be chosen out of the n-valued tuple β^a. The number of combinations for β^p is governed by the allocation $\hat{a}$. If there are $n_1$agents in node 1, $n_2$agents in node 2 and so on, the total number of combinations for β^p would be $\Pi_{j=1,\ n_j \neq 0}^{m} n_j$ (discounting nodes with no agents). In terms of $\hat{a}$, $n_j = \sum_{i=1}^{n} A_{ij}$. Therefore, the overall complexity is $O(m^n * \Pi_{j=1,\ n_j \neq 0}^{m} n_j)$.

Heuristic

Sorting the bids can be accomplished in $O(n\ log\ n)$ time complexity. We now evaluate the cost of allocating one agent.

- Model costs: When each agent i is to be allocated, the model M_j is called m times to evaluate the *best* node for the agent to be allocated. Assume a constant cost C per call to the analytical model. Since there are n_j agents in node j already, a total of $(n_j + 1)$ calls to the model are made - one call per value of repair rate of agents in j. For all the m nodes, the total cost for calling the model is $\Pi_{j=1}^{m}(n_j + 1) * C$ or $C^{'}O(n^m)$.

- Cost from evaluating constraints: The cost of evaluating all the flow constraints in Equation (6.9) is $N^f O(n^2)$. The cost of evaluating the fairness constraint (Equation (6.11)) is $O(n^2)$. Other constraints are of lesser or equal time complexity.

Since the algorithm is an iterative procedure, the model and constraint evaluating costs are incurred per agent. Therefore the total cost of allocating the agents (given the bids are sorted) is $O(n*(N^f n^2 + n^2 + C^{'} n^m))$. Including the sorting costs, we have a time complexity of $O(n\ log\ n + c_1 n^3 + c_2 n^{m+1})$ where c_1 and c_2 are new constants that replace $(N^f + 1)$and $C^{'}$. If $n >> m$, the n^{m+1} term will dominate making the total complexity $O(n^m)$.

Now, consider the costs due to the upgrade routine. If a single upgrade has to be carried out from node j to node $j-1$, then $(n_j - 1)(n_{j-1} + 1)$ new repair rates have to be checked using the model. That makes it $O(n^2)$ time complexity for the model evaluation. Adding the constraints, would still keep the complexity $O(n^2)$. If only one upgrade is permitted per pair of nodes to accommodate one agent, then a maximum $m - 1$ upgrades will occur per agent. That makes it $O(n^2 m)$. Adding the costs for all agents that require upgrades, the overall upgrade cost rises up to $O(n^3 m)$ complexity.

Choice of algorithms

In light of the fact that usually $m << n$, decisions can be made about which algorithm to employ. The exhaustive search is very costly. If $n = 20$ and $m = 3$, this algorithm can take in excess of 10^9 iterations. The greedy heuristic is slightly better. For this example, it will take of the order of 8000 iterations to arrive at a good solution. If need be, the quality of the solution can be made better using the upgrade routine. This will add another 2400 iterations, taking the total to around 10^4 iterations.

6.4 Design of Mechanism B

The motivation for a designing an alternate mechanism stems from the very high time complexity of Mechanism A (in particular the allocation problem within mechanism A). Once the bids are received, the Principal is responsible for finding the optimal allocation (Algorithm 1) or a good solution (Algorithm 2). The second motivation is a matter of trust - why should the MAS trust the infrastructure with all of its details and constraints? Mechanism B treats more of the agents' information as private and uses the Principal for the purpose of pricing. Allocation is carried out through a decentralized negotiation mechanism amongst neighbors and peers - this is the main focus of this section. Thirdly, we quantify the parameters that are negotiated and formalize the roles of the negotiating parties, primarily as applied to the MAS allocation problem. Figure 6.3 represents the mechanism we describe in this section.

In Figure 6.3, there are two allocations shown in the agent layer. The one on top is the *initial allocation*. After several rounds of negotiation, this allocation morphs into what is labeled the *final allocation*. These initial rounds are called the T_1 negotiation phase. The next negotiation phase is called the T_2 negotiation. This occurs between the two layers marked agent layer and the Principal. The purpose is to decide favorable repair rates for the MAS. Bids are submitted by the agents at each node. Repair rates are negotiated between each node and the agents that reside on that node. The *stars* denote varying stress levels experienced by the nodes. At the conclusion of several iterations of T_1 and T_2, the agents experience service at the contracted levels.

6.4.1 Negotiation Mechanism Formalisms

We now define the players, outcomes, players' strategies, outcome function and players' payoff functions as applicable to the negotiation mechanism (aspects that are identical to Mechanism A in Section 6.3.1 will be omitted).

1. Players: Negotiation occurs between pairs of players. In other words, we consider only bilateral negotiation. The first type of negotiation, denoted T_1, is

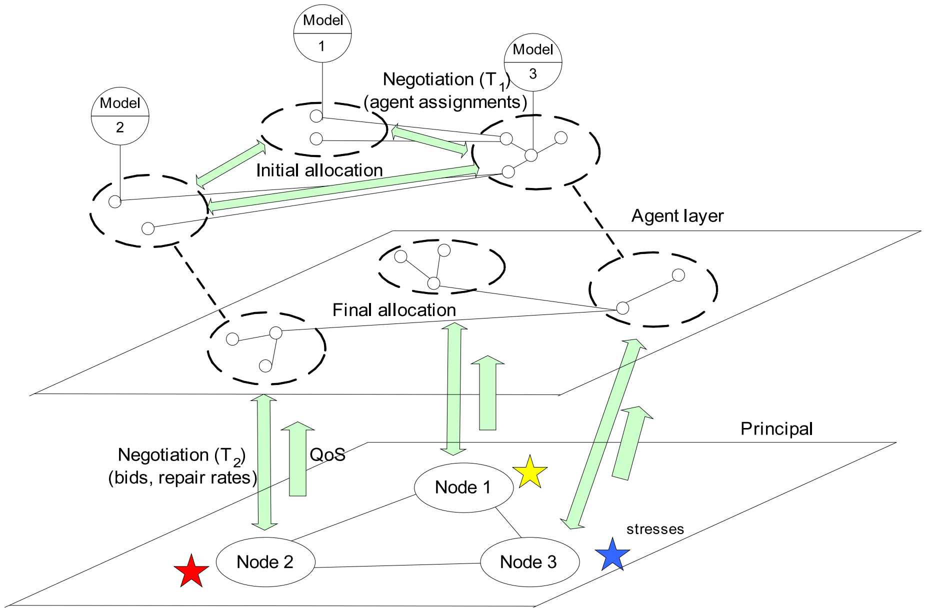


Figure 6.3: Schematic of Model-Based Negotiation Mechanism

between two groups of agents. The groups are identified with the indices r and s, i.e. the agents that are currently located on node r of the infrastructure and those on s. The set of agents on the two nodes are denoted A_r and A_s, and $\mid A_j \mid \equiv n_j$ for $j = r,\ s$. Likewise, let β_r^a and β_s^a denote the set of requested repair rates at the nodes. Although there are two groups, only two representative agents, $\bar{a}_s$ and $\bar{a}_r$ communicate with each other in the T_1 phase. The second type of negotiation, denoted T_2, is one that occurs between a set of agents on a node j and the node itself. Agents on node j are once again represented and this arbitrarily elected agent is denoted $\bar{a}_j$. The players in the T_2 negotiation are $\bar{a}_j$ and node j.

2. Service and QoS: The Principal and the agent are both responsible for collecting the information relating to model parameters (i.e. ϕ_j for every node j). The model M_j is assumed to be located at a neutral third party. Both agents and

the Principal can avail of the model by making a *system call.* This arrangement increases the confidence in the output of the model as it avoids the conflicts of interest. Each node individually selects its repair rate , denoted β_j^p, from among the requests of agents residing on node j (as opposed to the Principal choosing the globally optimal value).

3. Agents Actions: The mechanism starts with a random allocation of all the agents to the nodes. This allocation is denoted $\hat{a}_{init}$. For $\hat{a}_{init}$, some of the constraints (6.9)-(6.13) may not be satisfied. Let the set of agents assigned to node j be denoted by A_j. Each agent in A_j announces its bid individually to the node j. The agents in set A_j share their repair requests with each other[2]. The representative agent $\bar{a}_j$ maintains a sorted list L (in increasing order) of repair rates requested on node j. On the one hand, $\bar{a}_s$ and $\bar{a}_r$ may negotiate to potentially change the assignment of agents to nodes. This is the goal of the T_1 *negotiation.* On the other hand, the goal of the T_2 *negotiation* is to cause a potential change in repair rate. In both negotiations, agents will make a call the model M_j (the queueing model for node j). Agents on the same node are referred to as peers while others on nodes reachable in one hop are called neighbors. Agents do have the same constraints as in Mechanism A as the application characteristics have not changed. But agents do *not* express their constraints to the Principal. Whenever the agents engage in T_1, they check the constraints on a bilateral basis. In the T_1 negotiation phase, agents solve the

---

[2] The data in TechSpecs can be composed to get the desired repair rates of the agents.

problem

$$\hat{a}^{*}(\hat{a}_{init}) = arg \max_{\hat{a} \in \bar{A}} \sum_{i=1}^{n} x_i \tag{6.14}$$

subject to

$$\sum_{j \in \{r,\ s\}} A_{ij} = 1 \quad \forall\, i \in \{1, \dots,\ n\} \tag{6.15}$$

$$\sum_{j \in \{r,\ s\}} \sum_{i \in \{\bar{F_z}\}} w_{ij} A_{ij} \leq \Delta_{\bar{F_z}} \quad \forall\, \bar{F}_z \in F \tag{6.16}$$

$$\sum_{i \in \bar{G_z}} A_{ij} \ = 1 \quad \forall\, j \in \{r,\ s\},\ \bar{G_z} \in\ G \tag{6.17}$$

$$\sum_{j \in \{r,\ s\}} \sum_{i=1}^{n} A_{ij} \geq \bar{n} \tag{6.18}$$

$$\frac{1}{\mu_j} \sum_{i=1}^{n} \lambda_i A_{ij} < 1 \quad \forall\, j \in \{r,\ s\} \tag{6.19}$$

where $\bar{A}$ is the set of all allocations of the agents in $j \in \{r,\ s\}$ and $\hat{a}^{*}$ is the optimal allocation (no β^p here unlike Mechanism A). The fairness constraint cannot be checked for two reasons (a) the agents have no knowledge of each other's bids; (b) even if they do, it is hard to ensure global fairness (Equation (6.1)) through bilateral negotiation.[3] $\bar{n}$ is the total number of agents in nodes s and r ($\bar{n} = n_s + n_r$). $\bar{F}_z$ and $\bar{G}_z$ are respectively the flow and group constraints for the bilateral case. For the T_2 negotiation, the only job of the agents (in particular $\bar{a}_s$ and $\bar{a}_r$) is to check if the constraints (6.16)-(6.19) are satisfied. If any constraint fails, the agents may migrate to another node in the next T_1 iteration. In Algorithm 4 and Algorithm 5 details of the sequence of actions in

---

[3] By relaxing the fairness constraint, it seems as though we are degrading the *quality* of the allocation. But in reality, we must not forget that agents know about each other from TechSpecs as well as belong to the same application. Hence, in reality they might be fair to each other after all.

T_1 and T_2 are given.

4. Principal's Action: The role of the Principal is limited to an "*accept/reject*" answer based on certain conditions. During T_1, the Principal does not take any action.[4] Let B_j denote the set of bids of agents belonging to node j (b_i s.t. $A_{ij} = 1$). During T_2, each node j belonging to the Principal will individually check the following condition on its revenue rev_j

$$\{rev_j(\hat{a}) = \sum_{i=1}^{n}(b_i x_i - \hat{p}_i \hat{x}_i)A_{ij}\} \geq 0. \tag{6.20}$$

   This is the condition that each node will at least break-even in the fully decentralized case, i.e. when the infrastructure's nodes do not communicate. Alternatively, to model a Principal whose nodes actually communicate (eg. data centers and grids), this condition in (6.20) is modified to $\sum_j rev_j \geq \Omega$ where Ω is the absolute profit (note that this equation can equivalently be stated in terms of acceptable profit margin in %). If (6.20) is not satisfied, the node will reject the requested repair rate. in that case the negotiation proceeds to the next round. In some cases, the Principal may impose a limit on the total number of rounds possible which is denoted ζ. The sequence of actions in phases T_1 and T_2 is listed in Algorithm 4 and Algorithm 5.

5. Usage price: The price is computed in much the same way as Mechanism A. The difference is that the Principal does not know all the bids. Therefore, upon completion of the T_2 negotiation algorithm, it requests the nodes to reveal the highest bid in each node. We denote these high bids as $(\bar{b}_1, \bar{b}_2, \ldots, \ \bar{b}_m)$.

---

[4] In reality, the Principal has to facilitate the migration of the mobile agents that we consider in our COUGAAR/UltraLog scenario. We neglect the cost and burden due to the migration of these light-weight (low memory footprint) COUGAAR agents.

Likewise, the QoS delivery rate d_k, computed by each node by invoking the model M_k, is also centrally compiled.[5][6] Let $p = (p_1,\ p_2,\ ....,\ p_m)$ denote the prices for the m levels. If penalties $\hat{p}$ are zero, the price for level 1 traffic (the lowest level) is set at a constant zero,

$$p_1(b) \equiv 0$$

and the price for level k is

$$p_k(b) = p_{k-1}(b) + (d_k - d_{k-1})\bar{b}_{k-1}.$$

If $\hat{p} > 0$, the $p_1(b)$ is defined as the break-even price of the lowest QoS level (see Section 6.3.1).

From the above definitions, we summarize the aspects controlled by the agents and the Principal in Table 6.3. .

Table 6.3: Parameters controlled by the Application and the Infrastructure for Mechanism B

Controlling Entity	Controlled Quantities
Application / MAS	$\{b,\ \beta^a,\ \hat{a}\}$ i.e. the bids and the associated repair rates individually at the nodes.
Infrastructure	$\{\ p,\ \beta^p\}$ i.e. the infrastructure controls the prices, the assigned repair rates and the allocation

---

[5] Since d_k is only known at this stage, compiled list D of delivery rates may not necessarily be in increasing order. Without loss of generality, we assume the d_k is ordered. Notice that the bids $\bar{b}_j$ may be reordered and renamed to reflect the order in D.

[6] The only central computation is the price because it is relative. The goal of decentralization i.e. the reduction of computational burden from the allocation problem, has already been achieved.

6.4.2 Queueing Model

For the sake of comparison, the queueing model used is same one used in Mechanism A. The only thing to add to here is that the models are used primarily by the agents during both the negotiation phases. The nodes (and hence the Principal) use the model sparingly in the T_2 phase. Since agents base their decisions on the output from the queueing model, it is required that the estimates be trustworthy. The Principal uses the model for revenue computation and once again cannot trust the agents to provide the estimates even if the agents the models built-in. In this situation, a few alternatives are present:

1. The Principal and the agents use their own models. This is a viable alternative because it does not create dependencies.
2. The Principal and the agents can use a trusted third party (similar to certification authorities on IP networks) for generating the QoS estimates. Both the parties can utilize their own data and measurements in conjunction with the third party.

6.4.3 Solution to the Principal's and the Agents' problems

We have two negotiation phases T_1 and T_2 which are used together to negotiate QoS contracts. The solution to the Principal's and the agents' problems is found by making alternating the calls to the T_1 and T_2 stages of negotiation until the parameters converge. Once the parameters converge, the agents make payments periodically (e.g. at the end of the operational period; every t seconds etc.). It is possible that the parameters do not converge, perhaps due to the initial allocation. Under such circumstances, the agents can either change their bids or restart with a different configuration.

6.4.4 Algorithm Complexity

Since the agents participate in the allocation we compute the complexity as seen by each participant.

T_1 Phase: For each pair of agents, the pair combinations of allocations are $2^{n_s+n_r}$. Computing w_{ij} utilizing the model is $O(1)$ for each node - both nodes participate in this evaluation. Evaluation of the constraints is $O((n_s+n_r)^2)$. So, if the total number of agents is $\tilde{n} \equiv n + n_r$, then the complexity of the T_1 phase is $O(2^{\tilde{n}} * \tilde{n}^2)$.

T_2 Phase: Since this phase just involves running through the list of βs at every node j, this will be completed in $O(n_j)$ time at every node. At every stage in the T_2 phase, the Principal has to compute the value V - this is done in $O(n)$ time. Combining these two steps the complexity of this step is $O(n^2)$ (because the maximum value of n_j is n).

6.5 Numerical Examples

In this section we provide a few numerical examples for the three aforementioned algorithms (*alg* 1, *alg* 2 and *alg* 3). We considered a problem with $n = 7$ agents and $m = 3$ nodes. The bids by the agents (in sorted-order) were

$$b = \left\{ \begin{array}{ccccccc} 1.00 & 0.5 & 0.25 & 0.125 & 0.06 & 0.05 & 0.03 \\ 1.75 & 1.0 & 0.75 & 0.500 & 0.40 & 0.25 & 0.15 \end{array} \right\}.$$

The arrival rates λ_i for $i = (1\ ,\ 2\ , \ldots ,\ 7)$ were $\lambda = \{0.1,\ 0.15,\ 0.11,\ 0.12,\ 0.05,\ 0.14,\ 0.013\}$ and these were assumed to be computed by the Principal from the previous period. The requested repair rates $\beta_a = \{0.1,\ 0.2,\ 0.15,\ 0.05,\ 0.12,\ 0.07,\ 0.09\}$. The constraints 6.8-6.13 are to be respected by algorithms *alg* 1 and *alg* 2. Recall that for these algorithms the Principal is responsible for computing the allocation, after the agents announce their bids. For *alg* 3, the agents negotiate bilaterally as described above and are responsible for checking the constraints 6.15-6.19 in a distributed fashion. Two flow constraints $F_1 = \{1,\ 2\}$ and $F_2 = \{1,\ 0\}$ are considered with the respective $\Delta = \{48,\ 60\}$ seconds. The nodes are under three different stress conditions (optempo): *low*, *medium* and *high*. These conditions are described in Table 6.4. The nodes could handle tasks at the rate of 0.4, 0.3 and 0.2 task/second respectively. For the conditions described, it can be seen for some residual load (say $\lambda = 0.001$) that the *power* of the nodes is $node_0 > node_1 > node_2$. For other conditions (such as randomly varying γ, δ, λ, β_a) the queueing model can dynamically determine the most powerful node and hence the order. We also assume that the maximum repair rate $\beta_{max} = 0.25$ and the penalty charge $\hat{p}$ is 0.1 dollars/unit of QoS uniformly across all agents i.e $\hat{p} = \hat{p_i}\ \forall i = (1,\ 2, \ldots ,\ n)$. First, we describe the results from *alg* 1, which being the solution from the enumerative algorithm will be used as the baseline. We then describe results from *alg* 2 which is an iterative heuristic. Lastly, we describe results from *alg* 3 which is the negotiation algorithm.

Table 6.4: Optempo experienced by the nodes

Node (optempo)	Environmental Parameters (ϕ_j) $j = \{0,\ 1,\ 2\}$
Node 0 (low)	$\{\alpha = 0.001,\ \gamma = 0.001,\ \delta = 0.01\}$
Node 1 (medium - 5×)	$\{\alpha = 0.005,\ \gamma = 0.001,\ \delta = 0.01\}$
Node 2 (high - 10×)	$\{\alpha = 0.010,\ \gamma = 0.001,\ \delta = 0.01\}$

The QoS components considered for this example are $x_i = (p_i^{rel},\ R_i)$ and $\hat{x}_i = (w_i^{rel},\ \beta_i^{rel})$ for $i = (1,\ 2, \ldots,\ n)$. These quantities have been defined in Section 6.3.2.2. The objective function $F(x)$ the Principal seeks to maximize (for *alg* 1 and *alg* 2) is therefore

$$F(x) = \max_{\hat{a} \in A,\ \beta^p \in B} \sum_{i=1}^{n} (b_i x_i - \hat{p}\hat{x}_i).$$

where $x(\hat{a},\ \beta^p)$ is $\{(x_1,\ \hat{x_1}),\ (x_2,\ \hat{x_2}), \ldots,\ (x_n,\ \hat{x_n})\}$. We do not repeat the constraints here for brevity. The QoS delivery rate is $d = (d_1,\ d_2,\ \ldots,\ d_m)$ can be obtained from $x^*(\hat{a},\ \beta^p)$ by observation. The space of allocations A, has 2187 combinations (3^7). The space of repair rates B will be 343 (7^3) if there are no restrictions on how β on a node is picked.[7]

Algorithm 1 (*alg* 1) Results

The total dollar value potentially earned by the Principal for different allocations is shown in Figure 6.4. This is the sum of the values at the individual nodes as shown in Figure 6.5. Since the problem is actually discrete, the raw QoS for different allocations (locations of the agents and Principal-assigned repair rates) is shown in Figure 6.4b. The smoothed graphs allow for better comparison and are hence used to present *alg* 1 results. Also, these graphs allow for identifying *regions* of operation, where for example, the revenue may be high. In Figure 6.4b, we can see the regions where the Principal will earn lowest revenue (purple) for the bid profile b at a particular time instant when the all the other parameters (such as ϕ_j) are known and are held

---

[7] The repair rate at a node is usually picked from among requests in that node (instead of all requests from the agents). If there are no agents on node it is set to a default value.

constant. The x axis (*AllocID*) shows the order of the best allocation to worst allocation of agents from left to right. The right most allocation is the worst because puts all the agents on node 2 which is under highest stress and has relatively low processing *power*. Going from low to high β_p (y axis) causes the assigned repair rates to worsen. Since the Principal experiences a penalty for providing really high repair rates, it tries to supply a comfortable β - one that maximizes its objective function (i.e. Equation (6.7) or the z axis) while satisfying constraints. In the purple region (high AllocID, low β_p), the Principal pays high penalty to supplying top β while allowing the agents to physically locate in a high optempo area. Here the other QoS (such as availability) is also low bringing in lower value to the Principal. Therefore, the revenue is lower (inspite of high β_p in this region). A reverse argument is made for the diametrically opposite region. The Principal makes the decision of which *AllocID* and β_p to provide according to the combination that maximizes the total value as well as one that satisfies the constraints.

The differentiation in terms of the QoS provided to the agents is shown in Figure 6.6 and Figure 6.7. In this case, the Principal can identify 3 levels of QoS possible and the impact the environment has on the allocation at the different nodes (as illustrated by the regions). In other cases, the Principal could potentially provide lesser number of QoS levels, if the differentiation in value allowed for by environmental conditions is too *close* to consider different.

For given stress conditions, Figure 6.8 shows how requests by agents are satisfied by the Principal on an average (across the nodes). In a way, this is a measure of how close the Principal is able to get to requested repair rates. Since the Principal selects from a set of discrete β values requested by the agents on respective node, it can seldom satisfy all the agents on a node in terms of β. Note that this does not mean it cannot satisfy, for example, the flow (or the waiting time) constraints. When agents are not given their requested β values repeatedly they may over time have reason to migrate (a desired condition in *alg* 3 where agents move around in search of better β). We do not explicitly model the rules that govern migration or how agents update the requisition strategies.

For the example provided above, the optimal allocation

$$\hat{a}^* = \{(0, 0, 0, 1, 1, 2, 2),\ (0.20,\ 0.05,\ 0.07)\}$$

which provided the following QoS levels $d = (d_0,\ d_1,\ d_2)$ as $d_0 = (1,\ 0.9046,\ 0.8259,\ 0.80)$, $d_1 = (0.75,\ 0.8279,\ 0.5264,\ 0.20)$ and $d_2 = (0.5,\ 0.7969,\ 1.6477,\ 0.28)$. These QoS levels were not known prior to allocation, rather computed during allocation using the queueing model. All constraints, including fairness and flow constraints are satisfied for this allocation. The penalties $\hat{p}$ are assumed to be exogenously obtained (e.g. market research, random sampling / probing). If $\hat{p}$ is zero or nearly zero, the price that is computed for the optimal allocation using the mechanism is given in Table 6.5. d_0 is the lowest level of QoS (given that node 2 has the lowest availability (high stresses) and processing power). If $\hat{p} = 0$, the price for d_0 is zero. d_1 and d_2 are respectively the middle and high QoS tiers (provided on node 1 and node 0 respectively). . Since $\hat{p} \neq 0$, we compute p_0 or the price for the lowest QoS level as

Table 6.5: Prices for optimal QoS

QoS level d_j QoS of $(+p_{rel}, +R)$ over d_{j-1}	Usage Prices p_j per unit of QoS per second $j = (0,\ 1,\ 2)$ $[p_j^p,\ p_j^R]$
Level d_0 (base)	0
Level d_1 $(+0.25,\ +0.03)$	0.02 $[0.0125,\ 0.0078]$
Level d_2 $(+0.25,\ +0.08)$	0.09 $[0.0438,\ 0.0461]$

$$p_0 = \frac{\hat{p_2}\ \hat{x_2}}{x_2}$$

because we have to break-even at the base QoS which is provided on node 2 (x_2 and $\hat{x_2}$ being the QoS on node 2, $\hat{p_2}$ being the penalty at node 2). After computing p_0 and taking the prices for the individual components as the ratio of the QoS (since x_2 is a QoS matrix which is normalized), the prices that are obtained is tabulated in Table 6.6.

Table 6.6: Prices for optimal QoS

QoS level d_j QoS of $(+p_{rel}, +R)$ over d_{j-1}	Usage Prices p_j per unit of QoS per second $j = (0,\ 1,\ 2)$ $[p^p_j,\ p^R_j]$
Level d_0 (base)	0.2972 [0.1146, 0.1827]
Level d_1 (+0.25, +0.03)	0.3176 [0.1271, 0.1905]
Level d_2 (+0.25, +0.08)	0.3872 [0.1584, 0.2288]

This price means that the period that the allocation remains as above, the agents would pay at the rate of 30, 32 and 39 cents/task at QoS levels d_0, d_1 and d_2 respectively. Therefore, at the given λ, that would translate into an expenditure rate of 23 cents/second for the entire MAS. The Principal pockets about 12 cents/second after penalties are adjusted for.

Algorithm 2 (*alg* 2) Results

This algorithm is an iterative algorithm which again is controlled by the Principal completely after bids are obtained. As mentioned earlier, *alg* 2 is somewhat of a greedy algorithm. At every stage it tries to allocate the current agent best available node subject to constraints. If at some stage, it cannot fill all the agents into the nodes, it upgrades previously allocated nodes to higher QoS before allocating the current agent. In this case the total value obtained by the Principal rapidly approaches the optimal value attainable for the given stress conditions and bids.

Figure 6.10 shows the QoS attained at every iteration of *alg* 2. The QoS approaches the optimal value in 6 iterations. Running time is drastically reduced in this case.These show the quality of the allocation and the number of satisfied beta. Since we reach the optimal allocation, the price computation is exactly as shown above. But since we are not guaranteed to reach this solution, the price may be different than the optimal value shown for *alg* 1.

Algorithm 3 (*alg* 3) Results

alg 3 attempts to solve an optimization problem that is not identical to the one considered by *alg* 1 and *alg* 2. This is because it does not consider the fairness constraint - it actually cannot being a distributed algorithm. So in general, the solution may be inferior to the ones provided above in a strict mathematical sense. But *alg* 3 is computationally less intensive and truly distributed. It is a negotiation based solution and does not require the MAS to reveal a lot of information (e.g. several constraints).[8] In this case, we see that the solution approaches (almost) the optimal value as can be seen in Figure 6.12 albeit with a few oscillations.

The QoS at the nodes generally does not exactly reach the optimal point in this example (see Figure 6.13). This is because in the absence of a central authority, each node makes decisions for itself - i.e. attempts to obtain an non-negative value for itself. For the same reason, the agents are better satisfied with the QoS provided

---

[8] It can be observed that $alg-3$ allows the MAS to have better qualitative properties such as (a) preserving privacy in the sense that it reveals lesser information; (b) more autonomy in the sense that the MAS decides when to migrate (the autonomic computing literature would refer to this as a self-configuring application); (c) at any point in time, there is an interim solution (an anytime allocation); and (d) the MAS can be charged a *set-up* fee for computing the allocation by the Principal (i.e. decreased computational burden and a potential revenue stream for the Principal). In our modeling, we do not get into these aspects in greater detail.

(and after a point stop migrating) but from a global sense the solution is sub-optimal (although to a very small degree). Prior to concluding the negotiation, the repair rates at the individual nodes have to be rounded down to the closest requested β at that given node if the current rate is not from among the requested rates. When the value curves stabilize, the parties decide to conclude the negotiation. The final allocation $\hat{a}^{neg} = \{(0,0,0,1,1,2,2),(0.20,\ 0.05,\ 0.09)\}$. The prices can be similarly computed (see Table 6.7). From the Principal's perspective, the allocation is not optimal (although pretty close). The revenue from the packets was slightly lesser because the differentiation in QoS between levels d_0 and d_1 was not sufficient. With penalties being almost unchanged from *alg* 1, the Principal pockets a lesser amount (although the difference in the Principal's profits between the two cases is very small in absolute value and is under 14% on a relative basis). This is apparently good for the agent because it pays less for better QoS in this case. It must also be noted that for this example the fairness constraint is met.

Table 6.7: Prices for negotiation algorithm ($alg - 3$)

QoS level d_j QoS of $(+p_{rel}, +R)$ over d_{j-1}	Usage Prices p_j per unit of QoS per second $j = (0,\ 1,\ 2)$ $[p_j^p,\ p_j^R]$
Level d_0 (base)	0.2821 [0.1048, 0.1733]
Level d_1 (+0.25, +0.01)	0.2967 [0.1212, 0.1755]
Level d_2 (+0.25, +0.07)	0.3663 [0.1525, 0.2138]

6.6 Conclusions

In this chapter, we provide two mechanisms for devising service level agreements between multi-agent systems and the computing infrastructure. Both mechanisms are distributed, but vary in the degree of distribution. In Mechanism A, once the bids are obtained the Principal compute the complete allocation. Mechanism B enables the agents to truly exercise their autonomy by allowing to bilaterally negotiate. In return it reaps the benefit of lesser computational burden as far as collocation is concerned.

We conclude by first pointing out a subtle difference in the way an SLA can be represented. Subsequently,we highlight the salient aspects of using the mechanisms in the context of an application - a MAS for military logistics. In this way, we recapitulate how the pieces, namely the allocation problem, quality of service (for software agents) and pricing, fit together.

6.6.1 Service Contract

The service level agreement SLA, between the agents and the infrastructure for the operational period (which is the time between two successive negotiations) can be written as a tuple $\{AP,\ IP\}$ whereAP stands for the agents' parameters and IP for the infrastructure's parameters. For example, we write the SLA we obtain for mechanism A and B as follows:

$$SLA_B \;=\; \{(\hat{a}),\ (\beta^p,\ p)\} \tag{6.21}$$

and

$$SLA_A \;=\; \{(\Phi),\ (\hat{a},\ \beta^p,\ p)\}. \tag{6.22}$$

The notations are essentially saying the same thing: given all the inputs, the agents and the nodes are agreeing to the agent assignments $\hat{a}$, the repair rates β^p and the prices p. Every other parameter or condition is just used within the mechanism to compute the SLA. However, there is a slight difference in notation for SLA_A and SLA_B. This is because of the mechanism used. For SLA_B, the agents compute $\hat{a}$ and the Principal computes β^p and p. In the case of the SLA_A, the Principal decides all the parameters.

6.6.2 An Application: Automatic deployment and configuration for software agents

The aforementioned mechanisms can be utilized to perform automatic (agent-based) application deployment and (re)configuration as per the QoS requirements of the service consumer. Since manually deploying and tuning large-scale distributed software applications is cumbersome, we can use intelligent software agents (as the application) to reduce the management burden. The idea is to embed information about utilities, QoS models, inputs and outputs (which we called TechSpecs) and suitable mechanisms to make the agents *smart*. Then the agents and the infrastructure can engage in a dialog that can tune the settings and/or requests of the participating entities to ultimately match each others' needs.

Figure 6.15 depicts the complete picture of using the predictive models and the mechanism together so that applications can be deployed and allocated with very little human intervention, if anything at all. The application we will mostly dwell upon is agent-based logistics and emphasize the benefits that this method offers. Many points in this section are a reiteration of some motivating ideas which we used for this chapter as well as Chapter 4 and Chapter 5.

6.6.2.1 The need for a model

When the infrastructure does not know how the agents' demands and constraints are going to order the physical resources at its disposal, especially in the presence of interactions with the environment (namely various breakdowns), a suitable model may be useful. In particular, we find queueing models useful because they give insights about performance and reliability and, under some conditions, provide fast analytical QoS estimation tools. When considering military logistics, the physical resources are *computers* in the battle-field. Interactions exist between catastrophic failures and performance of a distributed agent-based application. Reliability suffers when repairs (in software or hardware) cannot be made in time. Moreover, as pointed out in Section 6.2, the variables in the agents' (and Principal's) optimization problems are all interdependent - making a micro-model very valuable for allocating the MAS. While sensors provide up-to-date information, agents utilize micro-models to measure the impact of failures on their QoS.

6.6.2.2 The need for a distributed mechanism

Although auction based mechanisms (such as Mechanism A) provide incentives to agents to reveal the the true value of the goods (information goods or QoS in our case), ultimately, the Principal faces the burden of a huge computation. The need for truly distributed solutions is further amplified when there are thousands of agents because the optimal allocations cannot be computed fast enough in that case. Moreover, distributed mechanisms must also have interim solutions, if there is a sudden need to stop the allocation in favor of more pressing battle-field tasks. In this chapter, we propose one such mechanism (Mechanism B).

6.6.2.3 Self-Management

In the literature, agents and autonomy are usually inseparable. That means that agents should be capable of governing aspects relating to their functionality on their own. In this chapter, we relate the allocation problem to this aspect of agents capable of deploying (choosing $\hat{a}$) and configuring (choosing β) themselves. In point of fact, when we consider a sizable problem with several interactions, mechanisms, protocols and fast internal micro-models may be a viable alternative.

6.6.2.4 The need for prices

In mission critical conditions, especially when there distributed self-interested agents (even if they are just software agents) involved the danger of the free-rider problem and resource starvation is imminent. Even software agents have gradations in capability and endowments, enabling the most powerful agents to spend more for better service in times of need. This means that a *Battalion* or a *Brigade* agent can pay more and expect better QoS for completing a task. Simultaneously, limited endowments ensure that this power is not abused by any single agent. Even if pricing is just a mechanism for controlling congestion as is envisioned for military logistics, the idea is suitable for the commercial realm as well. Hence in this chapter, we tie the revenue maximization problem of the Principal with market-controlled usage prices that agents pay for experiencing QoS.

6.6.3 Using the mechanisms

An application consisting of agents arrives at the periphery of the infrastructure, at once, requesting service. In reality, being software, an operating system call may just invoke the MAS. The agents have many characteristics as explained before, all of which we call TechSpecs. These agents want the best service that their money can buy. The infrastructure's nodes may incidentally be experiencing varying stresses at different points within its boundary.

Since there are dependencies (for example, giving one agent resources affects other agent), models are invoked to quantify the impact of potential allocations and system configurations on QoS. In fact, at that point it will be known whether or not a configuration is feasible. Whatever mechanism is followed, the agents must recognize the rules and play the game to finally experience the QoS they desire. Given the bids of other agents and available resources, the agents are allocated.

Once the agents are allocated, the agree to pay a usage price. That means that for every packet that experiences the contracted QoS a micro-payment must be made. To make things convenient, the Principal may device billing periods and collect payments from agents at the end of the period. The agents experience the service until the next disruption or contract violation. Disruptions may come in the form of catastrophic failure. On the other hand, contract violations may stem from agents sending, on an average, more tasks than the prior time-period. Usually under such circumstances, the allocation (and reconfiguration) may be re-triggered by either party - the agent or the infrastructure. In some situations, these disruptions may be tolerable upto a point. Node may actually have room to accommodate a few more tasks. The Principal will be capable of providing service until all required constraints (such as traffic intensity) are met. The overloading is still mitigated by the fact that there are usage prices in place to ultimately curb the agents' desire to compute and reverse the trend.

Algorithm 1 Model-Driven Exhaustive Search By Principal

```
Input
agents - $n$, $b$, $\beta^a$; nodes - $m$, $\phi_j$; shared information - $\Delta$, $F$, $G$, $\hat{p_i}$
Output
$A^*_{ij}$, $D^* \; \forall \; i, \; j$
Notation
$i$: agent index, $j$: node index

1. Let the tuple $A = \{A_1, \; \dots, \; A_k, \; \dots, \; A_n\}$ denote a single allocation of the $n$ agents where $A_k \in \{1, \; \dots, \; m\}$. $A$ defines one set of non-zero $A_{ij}$s. Compute all $m^n$ possibilities for $A$ and add them to the set of possibilities set $S$.
2. Let $t = (t_1, \dots, \; t_j, \dots, \; t_m)$. Initialize $t = 1$.
3. Pick an allocation $A \in S$, compute the QoS for each of the $j$ nodes - compute $x_j$ and $\hat{x_j}$ using $M_j(\beta_j, \; \phi_j)$. Using $\phi_j$ and the current allocation $A$, compute $\lambda_j = \sum_i \lambda_i A_{ij}$. $\beta^p_j = min_{t_j}(subset_j(\beta^a))$ for every $j \in \{1, \dots, \; m\}$, where $subset_j()$ picks all $\beta^a_i$s from $\beta^a$ if $A_{ij} = 1$ and $min_{t_t}()$ picks the $t^{th}$ minimum from a set of $\beta^a_i$s.
4. Evaluate all the constraints, now that $x_j$, $\hat{x_j}$ , $A_{ij}$ are known $\forall \; i, \;\; j$.
5. If all the constraints are satisfied, evaluate the objective function and store the value.
6. In $t$, set one $t_j = t_j + 1$ i.e. change $t_j$ in one node at a time until all possible combinations of requested $\beta^a_i$s are exhausted. When one $t_j$ is changed, recompute $x_j$ and $\hat{x_j}$ for every node $j$ using the updated $\beta^p_j$. Check the constraints. If they satisfied, compute and store the objective function.
7. Now, pick the next allocation from $S$. Reset $t = 1$. Repeat steps 3-6 until all allocations in $S$ are exhausted.
8. Pick the optimal allocation $a^*$ and $t^* = (t_1, \dots, \; t_m)$ (i.e. the one that offers the maximum value to the Principal) and output $A_{ij}$ and $D$. If the value to the Principal is negative, return $\Phi$.
```

Algorithm 2 Model-Based Greedy Allocation Heuristic Implemented By Principal

Input

agents - n, b, β^a; nodes - m, ϕ_j; shared information - Δ, F, G, $\hat{p}_i$

Output

A_{ij}, D (possibly A^*_{ij}, D^*) $\forall\ i,\ j$

Notation

i: agent index, j: node index

1. Initialize the agent allocation $A = \{-1,\ \ldots,\ -1,\ \ldots,\ -1\}$.
2. Using the order statistics for the bids b of the agents, pick the highest bidder, say agent i. Now λ_i and β^a_i are known.
3. Use the model M_j for every node j to compute the QoS $x = (x_1, \ldots,\ x_j, \ldots,\ x_m)$ if agent i is allocated to node j.
4. Allocate the agent i to node j, if node j offers the highest non-negative value to Principal. Check constraints. If satisfied, update $\lambda_j = \lambda_i$ and $\beta^p_j = \beta^a_i$. Update A.
5. Pick the next highest bidder. Reset i as current agent. Use M_j to recompute the QoS x. While computing the effect of allocating the current agent to a non-empty node, the new $\lambda_j = \lambda_j + \lambda_i$ and the new $\beta^p_j = min(\beta^p_j,\ \beta^a_i)$. Check constraints. For candidate allocations that satisfy the constraints, compute the objective function and store it.
6. If there are multiple values for β^a_i, generate all possible combinations forβ^p at the various nodes. Compute constraints. For candidate allocations that satisfy the constraints, compute the objective function and store it.
7. From all the stored candidate allocations, pick the one that maximizes the value to the Principal. Allocate the agent and update A. Update β_j and λ_j for every node j. If the list of stored candidate allocations is empty (constraints must have been violated in steps 5 and/or 6), goto step 9.
8. If there are more agents to be allocated, goto step 5.
9. Call the routine upgrade(A). The routine returns a modified allocation $A^{'}$ if it can upgrade some agents by executing a backwards pass, or else it returns A. If result is $A^{'}$, the heuristic outputs $A^{'}$else it outputs Φ.

Algorithm 3 upgrade(A) routine

Input

agents - n, b, β^a; nodes - m, ϕ_j; shared information - Δ, F, G, $\hat{p}_i$; current allocation A

Output

A_{ij}, D (possibly A^*_{ij}, D^*) $\forall\ i,\ j$

Notation

i: agent index, j: node index

1. Pick the candidate node where the unallocated agent i can be inserted without violating, in particular, the fairness constraint. From this node, one or more agents are to be upgraded to a higher QoS node to create space for the new one.

2. Pick the highest bidder from the candidate node. And upgrade it another node j such that it does not violate any constraints.

3. If there are multiple nodes that can accommodate the 'upgraded' agent, then the best upgrade should be picked amongst the different choices using the model to compute the new QoS upon upgrade. The value to Principal is computed and stored, if constraints are not violated.

4. If there are multiple combinations of β^p feasible, they must be checked and the objective function must be computed and stored.

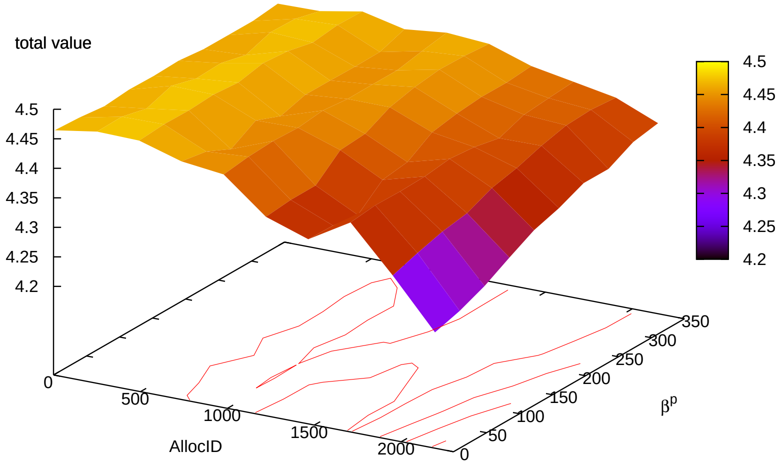
5. Pick the allocation that provides maximum value to Principal. Update A. If there are more agents to be allocated goto step 1, else return A. If the set of candidate upgrade-new allocation pairs is empty, return ϕ.

Algorithm 4 T_1 Negotiation Phase (between agents $\bar{a}_s$ and $\bar{a}_r$)

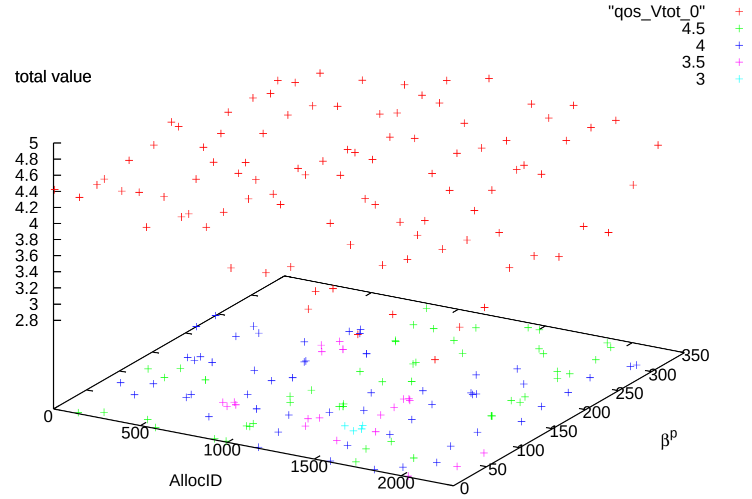
```
Input
agents - $\beta_s^a$, $\beta_r^a$ $\Delta_{\bar{F}_z}$, $\bar{F}_z$, $\bar{G}_z$, $n_s$, $n_r$ shared information - $\phi_r$ and $\phi_s$
Output
$\hat{a}_{rs}^*$ or $\{\Phi\}$
Initial Conditions
current allocation $\hat{a}_{curr}$ ($A_{ij}^{curr}$ defined accordingly)
Notation
$i$: agent index, $j$: node index
  (a) Generate the set of possible allocations among the sets of agents represented by $\bar{a}_s$ and $\bar{a}_r$.
  (b) For each set, pick the highest possible value of $\beta$ from among the requested values of $\beta$ for that node. Call them $\beta_s$ and $\beta_r$.
  (c) Compute QoS using $M_s$ and $M_r$ (and $\phi_s$ and $\phi_r$). Find the highest value of the objective function given in Equation (6.14) that obeys the constraints by iteratively carrying out steps 2-3.
  (d) Make the agent transfers, if the optimization problem results in a solution. If the optimization problem is infeasible, no transfer occurs. Return $\{\Phi\}$.
```

Algorithm 5 T_2 Negotiation Phase (between $\bar{a}_j$ and node j)

```
Input
agents - $B_j$, $\beta_j^a$, $\Delta_{\bar{F}_z}$, $\bar{F}_z$, $\bar{G}_z$, $n_j$; nodes - $\hat{p}_i$, ; shared information - $\phi_j$
Output
$\beta_{neg}$ or $\Phi$
Initial Conditions
Some random allocation or the result from $T_1$ negotiation - called $\hat{a}_{init}$ ($A_{ij}^{init}$ defined accordingly)
Notation
$i$: agent index, $j$: node index
  (a) Agents on node $j$ announce their bids $b_i$ to the node (for $\forall i \quad s.t. \ A_{ij} = 1$). The agents elect a representative $\bar{a}_j$ (arbitrary choice) and disclose the set $\beta_j^a$ to the $\bar{a}_j$.
  (b) $\bar{a}_j$ orders the set $\beta_j^a$ increasing order (cheapest to costliest $\beta$), denotes it by $L$. $\bar{a}_j$ picks the first submitted value from $L$ and presents it to the Principal.
  (c) The node computes the value $V = \sum_i (b_i x_i - \hat{p}_i \hat{x}_i) A_{ij}$. If $V > 0$, node $j$ accepts, goto step 5. Else, it rejects. Goto step 4.
  (d) The agent picks the next unsubmitted bid from $L$ and presents it to the node. Goto step 3. If there are no-more values in $L$, return false.
  (e) If the agent's constraints 6.15-6.19 are satisfied, agents also accept. Return current $\beta$ as $\beta_{neg}$. If constraints are not satisfied, return $\Phi$.
```

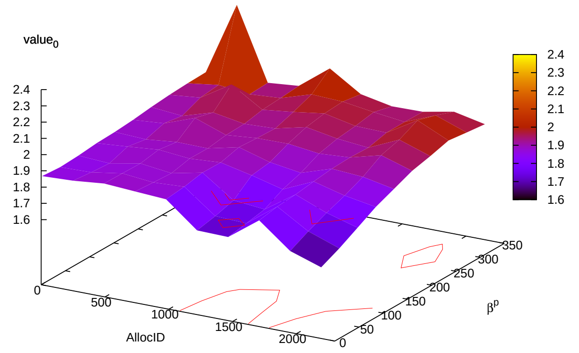


(a) smoothed

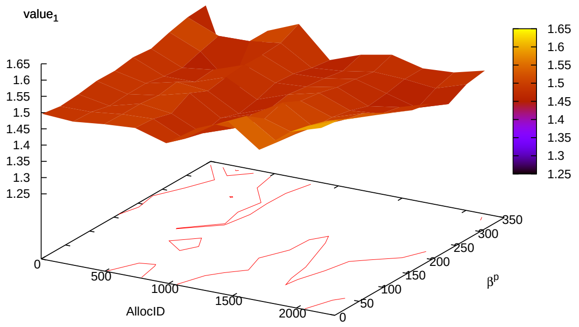


(b) raw data

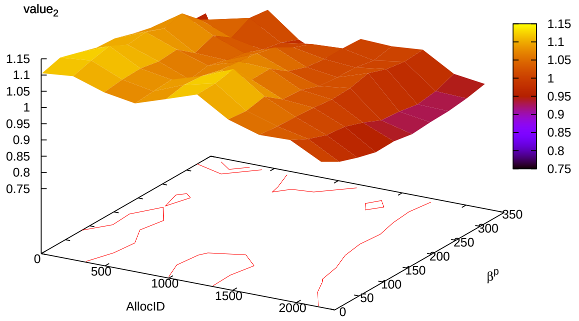
Figure 6.4: Total Value to Principal



(a) node 0

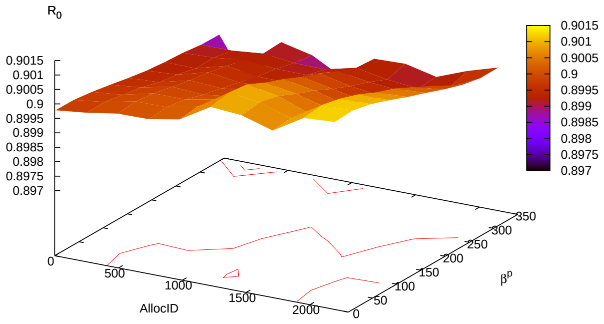


(b) node 1

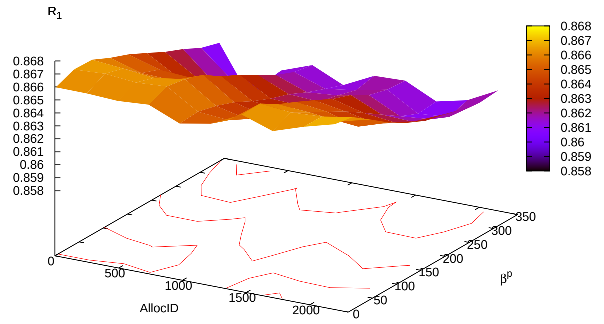


(c) node 2

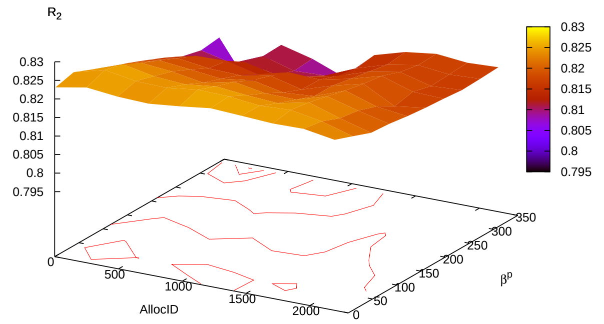
Figure 6.5: Value to Principal (broken down by nodes)



(a) node 0

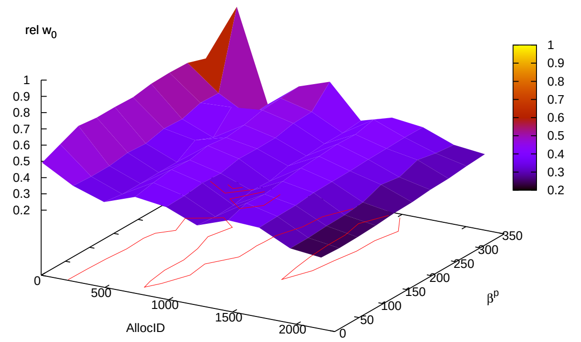


(b) node 1

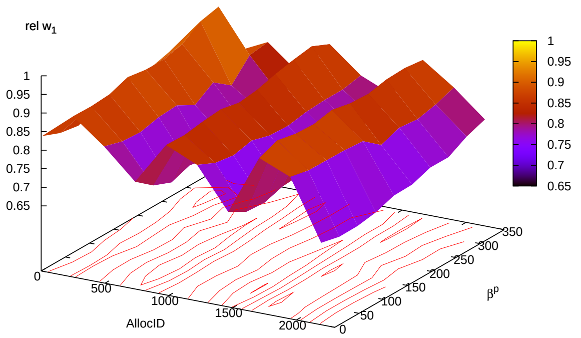


(c) node 2

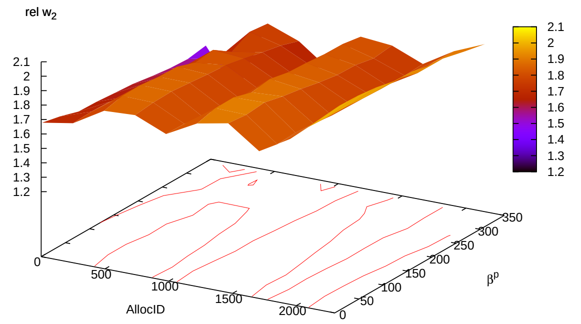
Figure 6.6: Availability at nodes



(a) node 0

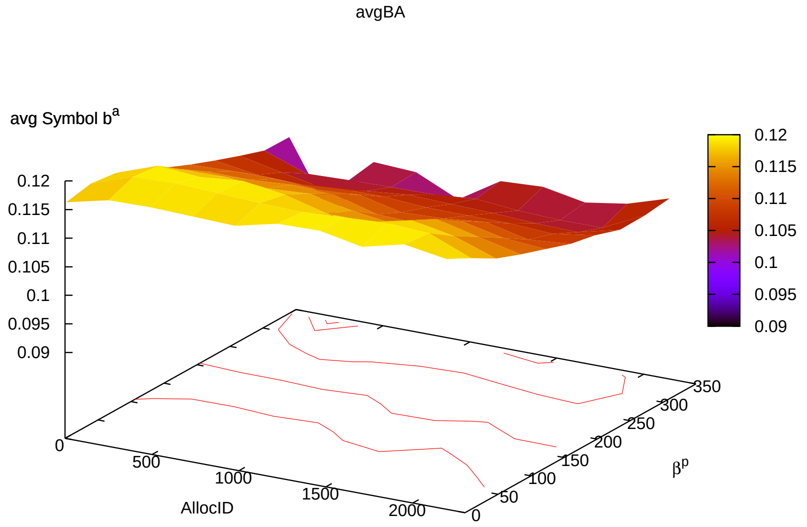


(b) node 1

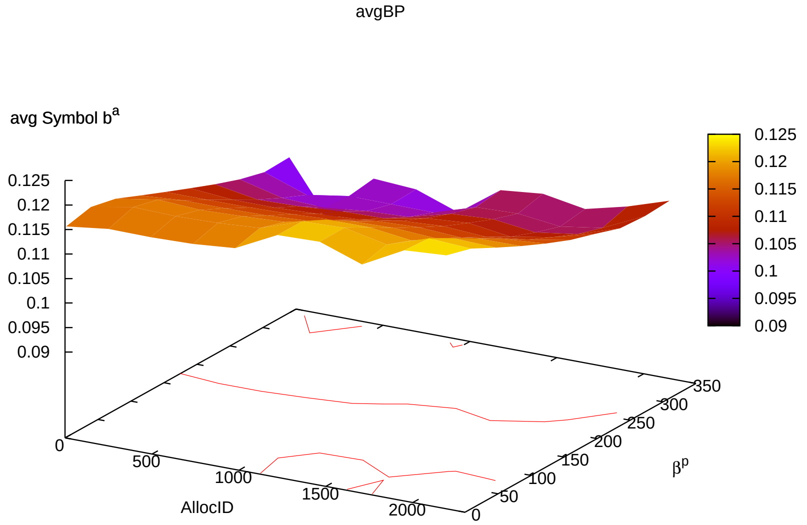


(c) node 2

Figure 6.7: Relative waiting time at nodes



(a) agents



(b) principal

Figure 6.8: Average β experienced by the agents and provided by the Principal

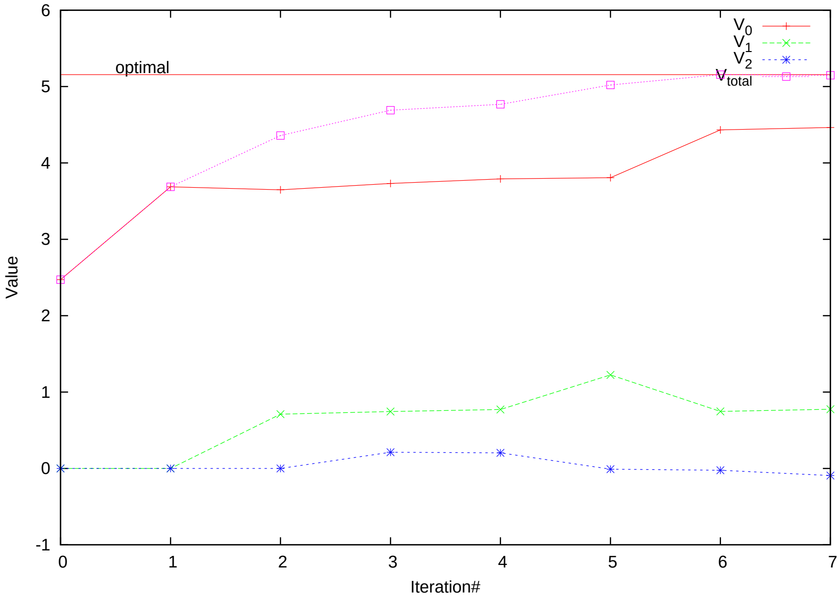
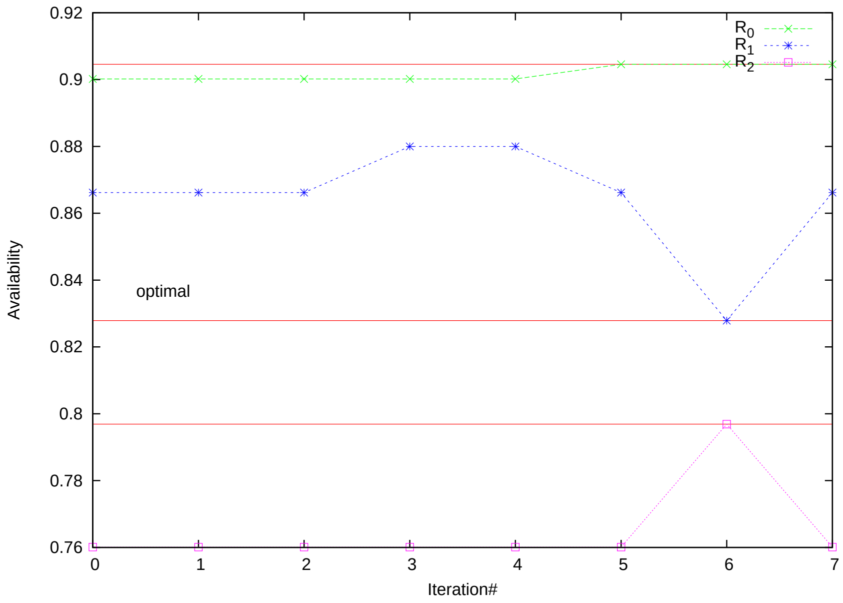
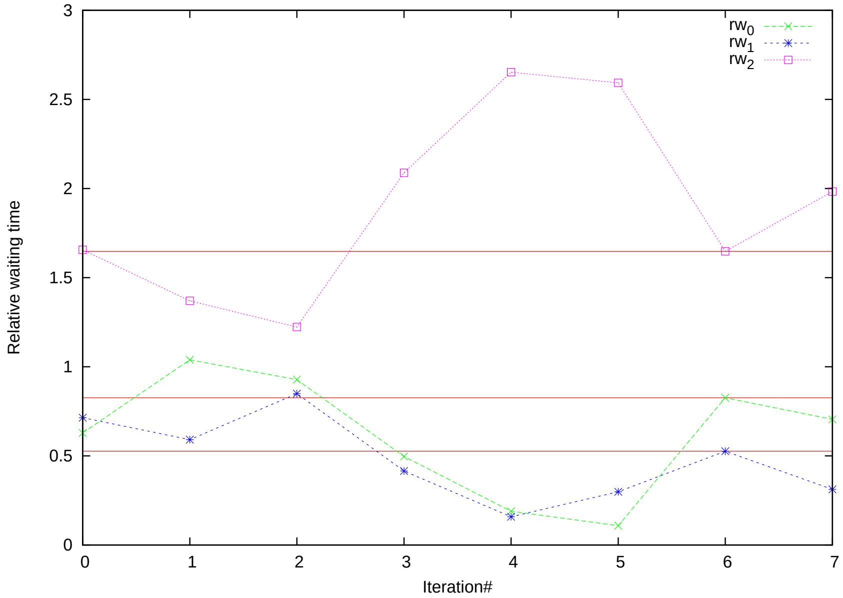


Figure 6.9: Value to Principal (*alg* 2)

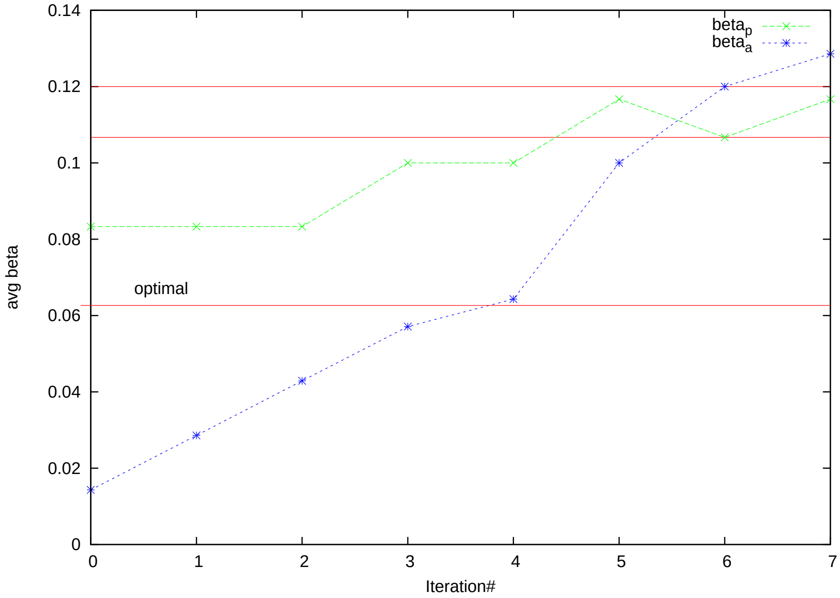


(a) availability

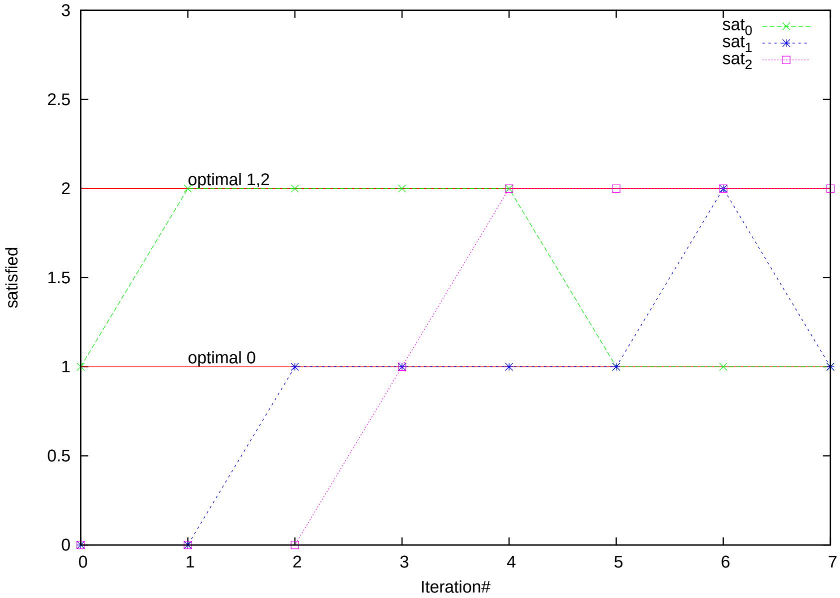


(b) relative waiting time

Figure 6.10: QoS at the nodes (*alg* 2)



(a) average repair rate



(b) number of agents satisfied with repair rate

Figure 6.11: Quality of Allocation at the nodes (*alg* 2)

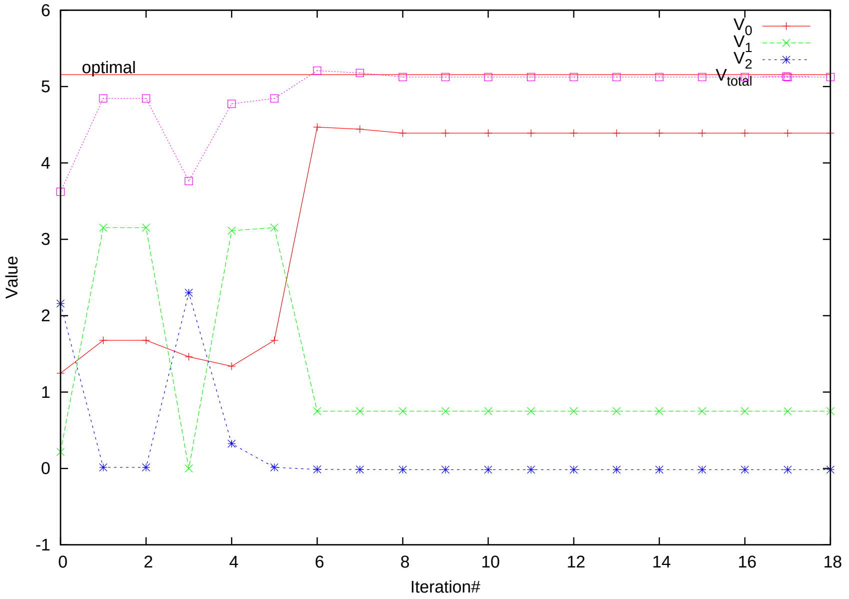
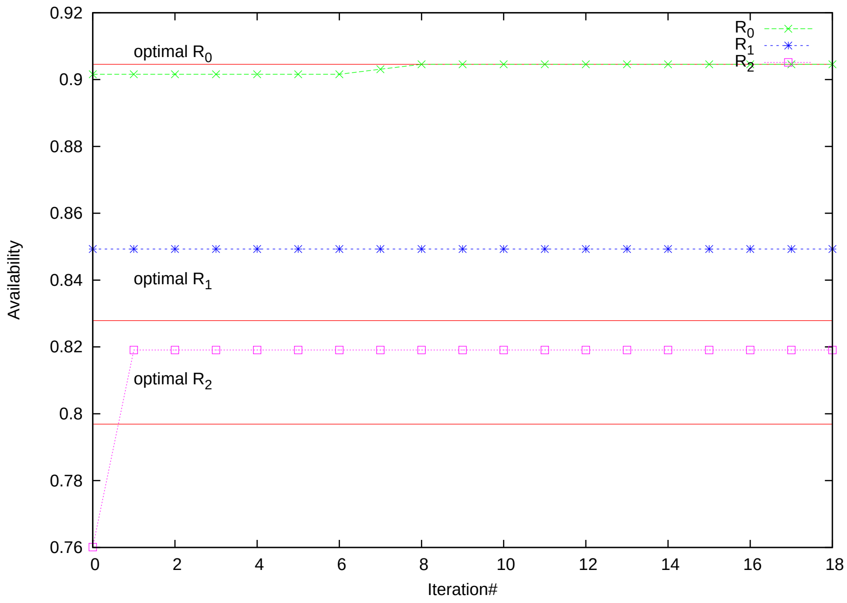
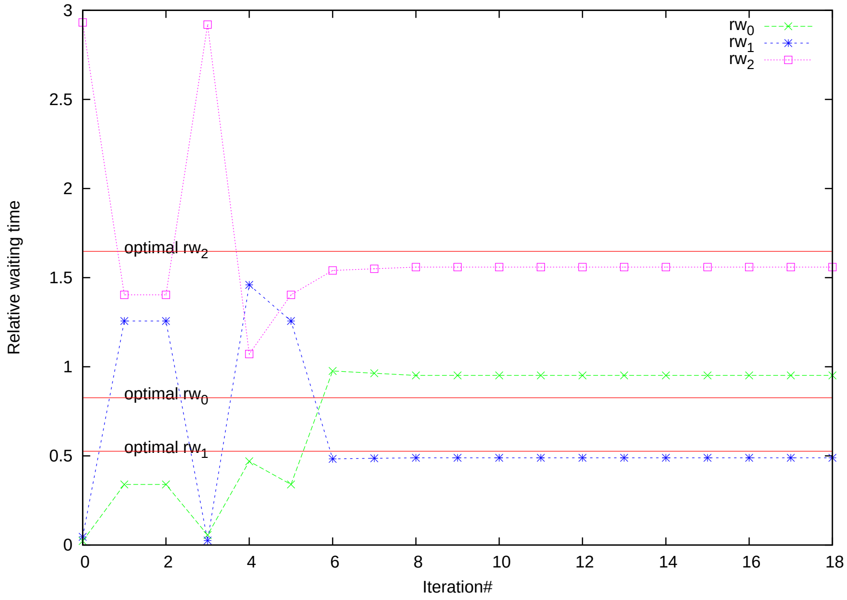


Figure 6.12: Value to Principal (*alg* 3)

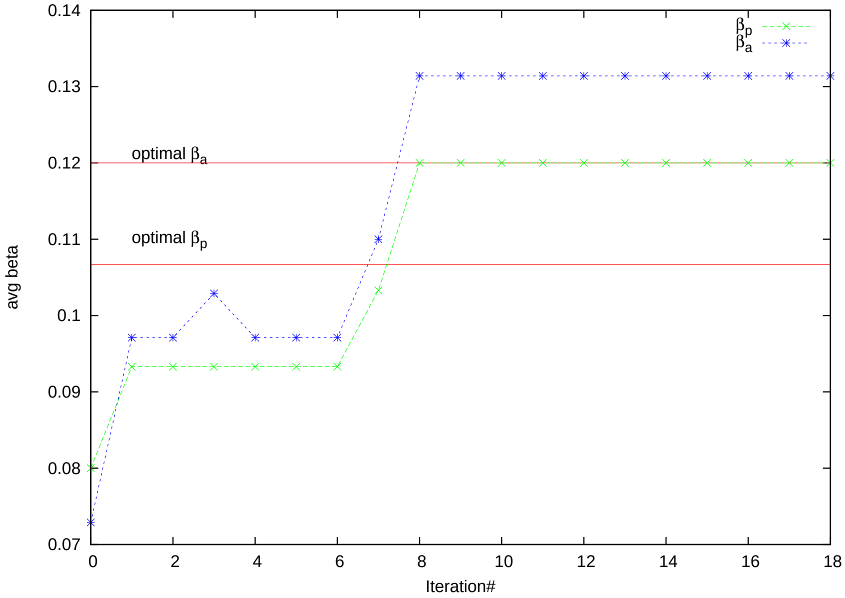


(a) availability

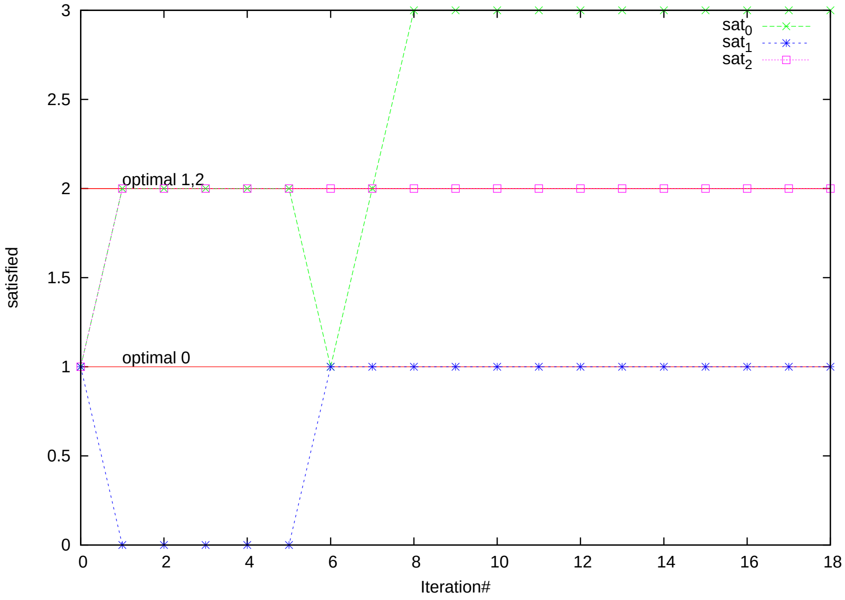


(b) relative waiting time

Figure 6.13: QoS at the nodes (*alg* 3)



(a) average repair rate



(b) number of agents satisfied with repair rate

Figure 6.14: Quality of Allocation at the nodes (*alg* 3)

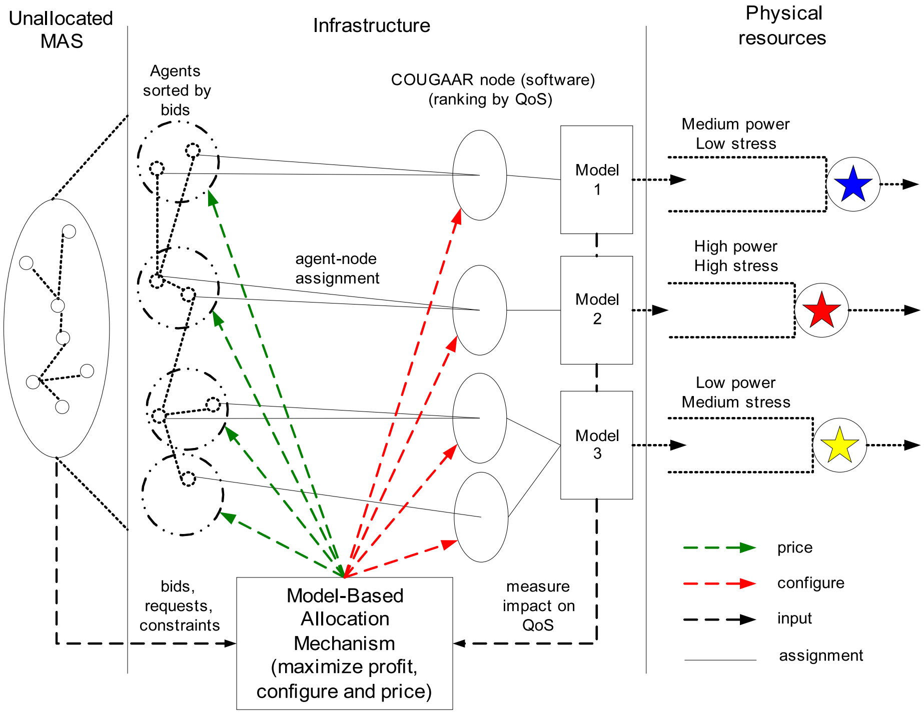


Figure 6.15: Mechanism-driven application deployment and pricing

Chapter 7

Conclusions and Future Research

The models and mechanisms proposed in this research enable a MAS to function effectively in stressful environmental conditions. In this thesis, we mainly studied two problems. In the first problem, we developed micro-models for performance of agent systems with different kinds of failure. This involved analyzing a class of queueing models with catastrophic and temporary breakdowns that has not received much attention in the literature. We consider both single- and multi-class queueing models and derive performance and reliability measures analytically. This work is discussed in Chapter 4 and Chapter 5. In the second problem, we developed two allocation mechanisms for a multi-agent application to configure itself on a distributed computing infrastructure. By self-selecting an optimal structure and configuration parameters, agents can maximize their expected steady state QoS while the infrastructure can ascertain an efficient allocation. By computing and charging usage prices using the agents' bids, the infrastructure controls congestion and provides good QoS.

7.1 Contributions

7.1.1 Summary

To the best of our knowledge, the use of quantitative performance and reliability models in multi-agent systems is novel to this work. In particular, the paradigm of class-switching within the two-class queueing model effectively captures the opportunistic level-switching observed in the UltraLog scenario. Additionally, the queueing models can be utilized to gain insights about the minimum number of agents and infrastructural nodes that may be required for the MAS to function within the required QoS limits for a given stress profile. Secondly, we provide two decentralized model-based allocation mechanisms that enable the MAS to autonomously provision the available resources. We believe that model-based resource allocation and the assignment of roles using a designed mechanism is not well-studied in the context of multi-agent negotiation.

7.1.2 Description of major contributions

Some of the major contributions of this research are listed below:

1. We develop a methodology to analyze queues with two kinds of failure and multiple classes. We consider both single- and two-class models and analytically obtain performance measures using generating functions. In the single-class scenario, closed-form results have been obtained. In the two-class case, we first reduce the dimensionality of the state space from two to one. In that process, we develop a class-conversion model for graceful and controllable performance degradation. We derived analytical results for performance and availability met-

rics that are applicable to a class of queues with breakdown. The methodology we introduce in pretty generic in nature and takes advantage of the properties of generating functions. We classify the problems and show that these classes of problems z^*, the root that solves the denominator of the generating functions lies in $(0,\ 1)$. This result helps in uniquely identifying the boundary probabilities.

2. We introduce the concept of model-based allocation when the parameters of the generalized assignment problem (w_{ij} and p_{ij}) are not independent. The micro-models developed are utilized in model-based allocation because they offer fast analytical solutions with which candidate allocations can be tested for feasibility. Using this concept we develop three distributed algorithms for allocation - $alg-1$, $alg-2$ and $alg-3$. In $alg-1$, the bids are obtained in a decentralized fashion (using mechanism A) and the principal computes the efficient allocation. In $alg-2$, the principal iteratively allocates the agents in a greedy fashion and subsequently improves the solution. In $alg-3$, the agents allocate themselves and negotiate with the principal for other parameters. The three proposed algorithms consider different degrees of privacy requirements on the part of the MAS.

3. Finally, we design two (auction-based) mechanisms - mechanism A and mechanism B which are essentially games in which the agents' can participate to reveal their preferences and self-select the desired QoS. Mechanism A is a variant of the Vickory-Clarke-Groves mechanism [7]. Mechanism B trades off optimality for total decentralization (hence agents' autonomy) and more privacy. In fact, it utilized a negotiation scheme for (re)configuration is an anytime algorithm -

which helps mitigate the set-up time. In the end, we compute usage prices for the agents.

Through these contributions, we formalized and automated the process of negotiating quantitative QoS contracts for MASs. While negotiation protocols, strategies and argumentation methods are widely studied in the MAS domain, autonomous negotiation of quantitative QoS contracts using internal models is novel to this work. While game theory traditionally looks at strategies and queueing theory at performance models, we have attempted to bridge the two.

7.2 Future Work

The following research aspects may be natural extensions of this work.

1. The performance modeling work for the multi-class model can be extended by choosing different assumptions while reducing the dimensionality of the state-space. The use of phase-type distributions enables the estimation of the state space beyond a particular value of n_2. This assumption will increase the accuracy of the solution provided in cases where the traffic intensity (ρ) is very high.

2. Both the single and multi-class performance models can be extended to a case where we have more than the two types of failures we considered. In other words, instead of one type of temporary failure, one could potentially have k levels with different levels of processing capability. This assumption will provide much more resolution on the QoS of systems with multiple levels of failure. Perhaps, it can be used to model multi-levels of survivability of systems.

3. The two-class queueing problem with class-switching can be extended to a case where there are three or more classes. This extention would attempt to generalize the performance and reliability models for a multi-class setting. This is interesting because the associated Markov chain becomes multi-dimensional. Hence the main problem that one would attempt is that of dimensionality reduction to obtain analytical solutions.

4. In the MAS allocation problem, an extended theoretical treatment is possible. By adjusting a few assumptions, one can try to establish theoretical proofs relating to the strategy-proofness of the mechanisms proposed. One important strategy in this context is how much the agents should bid, for example, when they have quasi-linear utility functions. Subsequently, one can examine other relevant types of utility functions. Another theoretical aspect worth pursuing is to model the negotiation schemes. As the first step, one can establish theoretical models for bilateral negotiation and follow it up with multi-lateral (for example, 3-way) negotiation.

5. Resource allocation for the MAS can be formulated as a multi-criteria optimization problem. The QoS x_i of agent i as seen in Chapter 6 can be a vector of QoS metrics with various weights associated with it. In other words, if $x_i = \{x_{i1},\ x_{i2},\ \ldots,\ x_{ip}\}$ for p QoS components then the objective function is formulated as

$$\max_{x \in A} \sum_{i=1}^{n} \sum_{j=1}^{p} b_{ij} x_{ij}(x)$$

   where b_{ij} is the bid of the i^{th} agent for the p^{th} component. Since x is from the space of QoS allocations A for all agents i.e. $i \in \{1, \ldots, N\}$the objective function is non-linear. Moreover, since the QoS components x_{ij} can be conflicting (say response time and loss probability), the bids b_{ij} have to be chosen appropriately. From the perspective of the agent, this is an multi-criteria optimization problem. From a practical MAS standpoint, the operational tempo will be pivotal is deciding the weights (or the agents' bids) associated with the QoS components. Within military logistics, the performance component of QoS may weighted significantly more than availability at a higher optempo, and vice-versa. Hence the trade-off between performance and reliability could become be the focus of a multi-objective optimization problem while factoring the distributed nature of the agents.

7.3 Practical Significances

It is envisioned that work will have the following practical significances: (1) Since it not possible to manually set-up one *global* (often inefficient) contract in agent-based computing environments ridden with complexity stemming from their numbers, mutual interactions and unpredictability, an automated procedure is necessary. The autonomously negotiated SLAs that this thesis proposes will form the *micro-contracts* at the lower levels or components of a bigger system whose benefits could accrue throughout the system. (2) The usage prices that are used may help control the congestion of the MAS and prevents resource starvation of other mission-critical software components in the network. (3) One aspect of the negotiation is the allocation of agents (or applications) to the computing infrastructure - a task presently achieved mostly by manual or semi-automated partitioning of infrastructure resources into clusters. By using the proposed technique, partitioning is not required.

Bibliography

[1] K. P. Sycara, “Multi-agent systems,” *AAAI Artificial Intelligence Magazine*, 1998.

[2] N. R. Jennings and M. Wooldridge, *Handbook of Agent Technology*, ch. Agent-Oriented Software Engineering. AAAI/MIT Press, 2000.

[3] “Ultralog program site,” *http://dtsn.darpa.mil/ixo/*. DARPA.

[4] T. Y. Choi, K. J. Dooley, and M. Rungtusanatham, “Supply networks and complex adaptive systems: control versus emergence,” *Journal of Operations Management*, vol. 19, pp. 351–366, 2001.

[5] G. Tesauro, D. M. Chess, W. E. Walsh, R. Das, I. Whalley, J. O. Kephart, and S. R. White, “A multi-agent systems approach to autonomic computing,” *Autonomous Agents and Multi-Agent Systems*, pp. 464–471, 2004.

[6] V. Lesser, C. Oritz, and M. Tambe, eds., *Distributed Sensor Networks: A Multiagent Perspective*, vol. 9. Kluwer Academic Publishers, Boston, May 2003.

[7] D. Fudenberg and J. Tirole, *Game Theory.* The MIT Press, Cambridge Massachusetts, 1991.

[8] E. Rasmusen, *Games and Information.* Blackwell Publishing, 2004.

[9] J. B. Rawlings, "Tutorial overview of model predictive control," *IEEE Control Systems*, vol. 20, no. 3, pp. 38–52, 2000.

[10] M. Morari and J. H. Lee, "Model predictive control: past, present and future," *Computers and Chemical Engineering*, vol. 23, no. 4, pp. 667–682, 1999.

[11] W. Li, D. Shi, and X. Chao, "Reliabilty analysis of m/g/1 queueing systems with server breakdowns and vacations," *Journal of Applied Probability*, vol. 34, pp. 546–555, 1997.

[12] X. Chao, "A queueing network model with catastrophies and product form solutions," *Operations Research Letters*, vol. 18, pp. 75–79, 1995.

[13] N. Gautam, "Pricing issues in web hosting services," *Journal of Revenue and Pricing Management*, vol. 4, no. 1, pp. 7–23, 2004.

[14] B. Kumar and D. Arivudainambi, "Transient solution of an m/m/1 queue with catastrophes," *Computers and MAthematics with Applications*, vol. 40, no. 12331240, 2000.

[15] X. Chao and Y. Zheng, "Transient analysis of immigration birth-death processes with total catastrophes," *Probability in the Engineering and Informational Sciences*, vol. 17, pp. 83–106, 2003.

[16] M. F. Neuts, *Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach.* The Johns Hopkins University Press, 1981.

[17] L. Green, "A queueing system with general-use and limited-use servers," *Operations Research*, vol. 33, no. 1, pp. 168–182, 1985.

[18] D. A. Stanford and W. K. Grassmann, *Ananlysis of Communication Networks: Call Centres, Traffic and Performance*, ch. Bilingual Server Call Centres, pp. 31–47. Fields Institute Communications, 2000.

[19] T. Osogami, *Analysis of multi-server systems via dimensionality reduction of Markov chains*. PhD thesis, Carnegie Mellon University, 2005.

[20] J. W. Cohen, *Quantitative Methods in Parallel Systems*, ch. Two-dimensional nearest-neighbor queueing models, a review and an example, pp. 141–152. Springer-Verlag, 1995.

[21] J. W. Cohen and O. J. Boxma, *Boundary Value Problems in Queueing System Analysis*. North-Holland Publ. Cy., 1983.

[22] S. K. Lando, *Lectures on Generating Functions*, vol. 23. American Mathematical Society, 2003.

[23] I. P. Goulden and D. M. Jackson, *Combinatorial Enumeration*. Jon Wiley & Sons, Inc., New York, 1983.

[24] V. Ramaswami, "Algorithmic analysis of stochastic models: The changing face of mathematics," *Ramanujam Endowmnet Lecture at Anna university, Chennai, India*, 2000.

[25] A. Riska and E. Smirni, "Mamsolver: A matric analytical method tool," *International Conference on Modeling Techniques and Tools for COmputer Communication Systems and their Applications*, vol. 2324, pp. 205–211, 2002.

[26] "Future combat systems (brigade combat team)," *http://www.army.mil/fcs/.*

[27] A. W. Services, “Amazon elastic compute cloud (amazon ec2) - limited beta,” 2007.

[28] K. W. Ross and D. H. K. Tsang, “The stochatic knapsack problem,” *IEEE Transactions on Communications*, vol. 37, pp. 740–747, 1989.

[29] D. P. Morton and R. K. Wood, *Advances in Computational and Stochatic Optimization, Logic Programming and Heuristic Search*, ch. On stochastic knapsack problems and generalizations. 1998.

[30] J. D. Papastavrou, S. Rajagopalan, and A. J. Kleywegt, “The dynamic and stochatic knapsack problem with deadlines,” *Management Science*, 1996.

[31] M. Sniedovich, “Preference order stochastic knapsack problems; methodological issues,” *The Journal of the Operations Research Society*, vol. 31, pp. 1025–1032, 1980.

[32] L. L. Lu, S. Y. Chiu, and L. A. C. Jr, “Optimal project selection: Stochatic knapsack with finite time horizon,” *The Journal of the Operations Research Society*, vol. 50, pp. 645–650, 1999.

[33] C. J. Parris, S. Keshav, and D. Ferrari, “A framework for the study of pricing in integrated networks,” no. TR-92-016, 1992.

[34] R. Cocchi, S. Shenker, D. Estrin, and L. Zhang, “Pricing in computer networks: motivation, formulation, and example,” *IEEE/ACM Transactions on Networking*, vol. 1, no. 6, pp. 614–627, 1993.

[35] J. K. Mackie-Mason and H. R. Varian, *Pricing the Internet.* MIT Press, 1995.

[36] S. Shenker, D. Clark, D. Estrin, and S. Herzog, "Pricing in computer networks: Reshaping the research agenda," *ACM Computer Communication Review*, vol. 26, pp. 19–43, April 1996.

[37] B. Tuffin, "Charging the internet without bandwidth reservation: An overview and bibliography of mathematical approches," *Journal of Information Science and Engineering*, vol. 19, pp. 765–786, 2003.

[38] C. Courcoubetis and R. Weber, *Pricing Communication Networks: Economics, Technology and Modelling.* Wiley, 2003.

[39] G. Davies, M. Hardt, and F. Kelly, "Come the recolution - network dimensioning, service costing and pricing in a packet switched environment," *Telecommunications Policy*, vol. 28, pp. 391–412, 2004.

[40] M. C. Caesar, S. Balaraman, and D. Ghosal, "A comparative study of pricing strategies for ip telepony," *IEEE GLOBECOMM*, pp. 344–349, 2000.

[41] M. Yacoubi, M. Emelianenko, and N. Gautam, "Pricing in next generation networks: a queueing model to guarantee qos," *Performance Evaluation*, 2002.

[42] Y. Chen, A. Das, N. Gautam, Q. Wang, and A. Sivasubramaniam, "Pricing-based strategies for autonomic control of web servers for time-varying request arrivals," *Journal of Engineering Application of Artificial Intelligence*, vol. 17, no. 7, pp. 841–854, 2004.

[43] T. Henderson, J. Crowcroft, and S. Bhatti, "Congestion pricing: Paying the way in communication networks," *IEEE Internet Computing*, vol. 5, no. 5, pp. 85–89, 2001.

[44] I. C. Paschalidis and J. N. Sitsiklis, "Congestion dependent pricing of network services," *IEEE/ACM Transactions on Networking*, pp. 171–184, 2000.

[45] X. Lin and N. B. Shroff, "Pricng based control of large networks," *LCNS*, no. 2170, pp. 212–231, 2001.

[46] T. W. Malone, R. Fikes, K.R.Grant, and M.T.Howard, *Enterprise: A Market-like Task Scheduler for Distributed Computing Environments*. Holland: Elsevier, 1988.

[47] A. Chavaz, A. Moukas, and P. Maes, "Challenger: A multi-agent systems for distributed resource allocation," *Agents*, 1997.

[48] D. Ferguson, Y. Yemini, and C. Nikolaou, "Microeconomic algorithms for load balancing in distributed computer systems," *Proceedings of the International Conference on Distributed Systems*, 1988.

[49] J. Bredin, D. Kotz, and D. Rus, "Market-based resource control for mobile agents," *Autonomous Agents*, 1998.

[50] M. P. Wellman, "A computational market model for distributed configuration design," *AI EDAM*, 1995.

[51] M. P. Wellman, *Market Based Control - A Paradigm for Distributed Resource Allocation*, ch. Market Oriented Programming: Some Early Lessons. World Scientific, 1996.

[52] H. R. Varian, *Microeconomic Analysis*. W W Norton and Company, 1992.

[53] S. Lee, S. Kumara, and N. Gautam, “Efficient scheduling algorithm for component-based networks,” *Future Generation Computer Systems*, vol. 23, pp. 558–568, 2007.

[54] G. Zlotkin and J. S. Rosenschein, “Negotiation fn task sharing among autonomous agents in cooperative domains,” *IJCAI*, pp. 912–917, 1989.

[55] S. Kraus, K. Sycara, and A. Evenchik, “Reaching agreements through argumentation: a logical model and implementation,” *Artificial Intelligence Journal*, vol. 104, no. 1-2, pp. 1–69, 1998.

[56] L.-K. Soh and C. Tsatsoulis, “A real-time negotiation model and a multi-agent sensor network implementation,” *Autonomous Agents and Multi-Agent Systems*, 2005.

[57] A. Rao and M. Georgeff, “Decision procedures fro bdi logics,” *Journal of Logic and Computation*, vol. 8, no. 3, pp. 293–342, 1998.

[58] A. Rao and M. Georgeff, “Bdi agents: from theory to practice,” *ICMAS*, pp. 312–319, 1995.

[59] P. Stone and M. Veloso, “Using decision tree confidence factors for multi-agent control,” *Autonomous Agents*, 1998.

[60] L. Chen, K. Bechkoum, and G. Clapworthy, “A logical approach to high-level agent control,” *Agents*, 2001.

[61] T. Chao, F. Shan, and S. X. Yang, “Modeling and design monitor using layered control architecture,” *Autonomous Agents and Multi-Agent Systems*, 2002.

[62] T. Vu, J. Go, G. Kaminka, M. Velosa, and B. Browning, “Monad: A flexible architecture for multi-agent control,” *Autonomous Agents and Multi-Agent Systems*, 2003.

[63] M. M. Kokar, K. Baclawski, and Y. A. Eracar, “Control theory-based foundations of self-controlling software,” *IEEE Intelligent Systems*, pp. 37–45, May/June 1999.

[64] R. Sanz and K.-E. Arzen, “Trends in software and control,” *IEEE Control Systems Magazine*, June 2003.

[65] K. Kleinmann, R. Lazarus, and R. Tomlinson, “An infrastructure for adaptive control of multi-agent systems,” *IEEE Conference on Knowledge-Intensive Multi-Agent Systems*, 2003.

[66] M. N. Bennani and D. A. Menasce, “Assessing the robustness of self-managing computer systems under highly variable workloads,” *International Conference on Autonomic Computing*, 2004.

[67] A. E. Fallah-Seghrouchni, I. Degirmenciyan-Cartault, and F. Marc, “Modelling, control and validation of multi-agent plans in dynamic context,” *Autonomous Agents and Multi-Agent Systems*, 2004.

[68] K. C. Lee, W. H. Mansfield, and A. P. Sheth, “A framework for controlling cooperative agents,” *IEEE Computer*, 1993.

[69] R. Vincent, B. Horling, V. Lesser, and T. Wagner, “Implementing soft real-time agent control,” *Agents*, 2001.

[70] A. Raja, V. Lesser, and T. Wagner, “Toward robust agent control in open environments,” *Agents*, 2000.

[71] I. Soto, M. Garijo, C. A. Iglesias, and M. Ramos, “An agent architecture to fulfill real-time requirement,” *Agents*, 2000.

[72] A. G. Barto, S. J. Bradtke, and S. Singh, “Learning to act using real-time dynamic programming,” *Artificial Intelligence*, vol. 72, pp. 81–138, 1995.

[73] R. S. Sutton, A. G. Barto, and R. J. Williams, “Reinforcement learning is direct adaptive optimal control,” *IEEE Control Systems*, vol. 12, no. 2, pp. 19–22, 1992.

[74] L. P. Kaelbling, M. L. Littman, and A. Moore, “Reinforcement learning: A survey,” *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[75] A. Helsinger, R. Lazarus, W. Wright, and J. Zinky, “Tools and techniques for performance measurement of large distributed multi-agent systems,” *Autonomous Agents and Multi-Agent Systems*, 2003.

[76] “Cougaar open source site,” *http://www.cougaar.org*. DARPA.

[77] A. Helsinger, K. Kleinmann, and M. Brinn, “A framework to control emergent survivability of multi agent systems,” *Autonomous Agents and Multi-Agent Systems*, 2004.

[78] N. Gnanasambandam, S. Lee, N. Gautam, S. R. T. Kumara, W. Peng, V. Manikonda, M. Brinn, and M. Greaves, “Reliable mas performance prediction using queueing models,” *IEEE Multi-agent Security and Survivabilty Symposium*, 2004.

[79] F. Sheikh, J. Rolia, P. Garg, S. Frolund, and A. Shepard, "Performance evaluation of a large scale distributed application design," *World Congress on Systems Simulation*, 1997.

[80] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. Wiley, 1990.

[81] M. Brinn and M. Greaves, "Leveraging agent properties to assure survivability of distributed multi-agent systems," *Proceedings of the Second Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2003.

[82] A. Cassandra, D. Wells, M. Nodine, and P. Pazandak, "Techspecs: Content, issues and nomenclature," *Technical Report, Telcordia Inc. and OBJS Inc.*, 2003.

[83] G. Bolch, S. Greiner, H. de Meer, and K. S.Trivedi, *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. John Wiley and Sons, Inc., 1998.

[84] E. W. Weisstein, "Cubic formula," *Mathworld – A Wolfram Web Resource*.

[85] N. Gnanasambandam, S. Lee, and S. R. T. Kumara, "An autonomous performance control framework for distributed multi-agent systems: A queueing theory based approach," *Autonomous Agents and Multi-Agent Systems Conference*, 2005.

Vita

Shanmuga-Nathan Gnanasambandam

Nathan Gnanasambandam holds a Bachlor of Engineering in Electronics and Instrumentation Engineering from the Birla Institute of Technology and Science (BITS) in Pilani, India. He also concurrently pursued a Master of Science degree in Physics from the same university. Upon completion of his undergraduate degree and Master's, he pursued a Master's degree in Industrial Engineering and Master's degree in Computer Science both at the State University of New York at Binghamton. He was involved in research consortium headed by Universal Instruments Corporation while he was working on his Master's degree. He then was a coop at IBM, Endicott for a year and 4 months. Nathan came to Penn State to pursue a Ph.D. in Industrial Engineering. While being a graduate student at Penn State, Nathan worked on the DARPA-UltraLog project, a research project that was awarded to his advisor. He subsequently interned at Xerox Research at Webster, NY during a summer. Nathan worked with his advisor and Xerox Research to co-organize a tutorial at a international IEEE conference. Upon completion of his Ph.D., Nathan will join Xerox Research at Webster, NY. He will be involved in performance analysis and algorithm design for large-scale software systems.