The Pennsylvania State University

The Graduate School

College of Engineering

MARKET-BASED MODEL PREDICTIVE CONTROL FOR SURVIVABLE DISTRIBUTED INFORMATION SYSTEMS: RESOURCE ALLOCATION AND ALGORITHM SELECTION

A Thesis in

Industrial Engineering

by

Seokcheon Lee

© 2005 Seokcheon Lee

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

December 2005

The thesis of Seokcheon Lee was reviewed and approved* by the following:

Soundar R.T. Kumara
Distinguished Professor of Industrial Engineering
Thesis Co-Advisor
Chair of Committee

Natarajan Gautam
Associate Professor of Industrial Engineering
Thesis Co-Advisor

M. Jeya Chandra
Professor of Industrial Engineering

Susan H. Xu
Professor of Management Science and Supply Chain Management

Mark Greaves
Senior Research Program Manager, Vulcan Inc.
Special Signatory

Vikram Manikonda
Vice President, Intelligent Automation Inc.
Special Signatory

Richard J. Koubek
Professor of Industrial Engineering
Head of the Department of Industrial and Manufacturing Engineering

*Signatures are on file in the Graduate School

ABSTRACT

As modern networks can be easily exposed to various adverse events such as malicious attacks and accidental failures, there is a need to study their survivability. There are several important trends of modern information networks. They tend to be large-scale with distributed and component-based architectures, and the dynamic nature of operating environments leads them to utilize alternative algorithms. As a result, the behavior of such an information network can be controlled through resource allocation as well as algorithm selection. We study an information network that characterizes such trends. The service provided by the network is to produce a global solution to a given problem, which is an aggregate of partial solutions of individual tasks. Quality of service of the network is determined by the value of global solution and the time taken for generating global solution. In this thesis we design a scalable adaptive control mechanism along the lines of model predictive control to support the survivability of such networks by utilizing resource allocation and algorithm selection. To address adaptivity we model stress environment by quantifying resource availability through sensors. We build a mathematical programming model with the resource availability incorporated, which invokes optimal control actions as a function of both state and stress environment. The programming model is then decentralized through an auction market. By periodically opening the auction market, the system can achieve desirable performance adaptive to changing stress environment while assuring scalability property. We verify the designed control mechanism empirically.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to express my gratitude to my advisor Dr. Soundar Kumara for his support and encouragement throughout my Ph.D. study at the Pennsylvania State University. He is an enthusiastic researcher who is always looking for novel perspectives and motivating his students toward innovations. I have tried to learn his insightful perspectives for the last five years since I started working with him. I would also like to thank my committee members Dr. Natarajan Gautam, Dr. Jeya Chandra, Dr. Susan Xu, Dr. Mark Greaves, and Dr. Vikram Manikonda, for their invaluable guidance and suggestions during my research. In finishing my Ph.D. study, I would like to give special thanks to Dr. Myun-Woo Lee (at the Seoul National University). The work experience with him in my M.S. study indeed made a great contribution toward my finishing Ph.D. study. Also, I would like to thank my senior Dr. Yong-Han Lee who gave me meaningful advices and discussions when I was sitting with him in the lab. I am indebted to my wife Eunsuk and my sons Seungmoon and Kyoungmoon for their sacrifice. My wife has been so patient and my sons have been growing up so well despite my study in the U.S. They would never know how much I feel sorry and how much I love them. Finally, I would like to thank my parents and parents in law, from the bottom of my heart, for their prayers and confidence in me. I am really proud of them all.

Chapter 1

Introduction

Critical infrastructures in many domains are becoming increasingly dependent on networked systems for automation or organizational integration. Though such infrastructures can improve the efficiency and effectiveness, these systems can be easily exposed to various adverse events such as malicious attacks and accidental failures [**1**]. Two metrics, namely survivability and scalability, can be used to determine the efficiency and effectiveness of these systems. Survivability is defined as "the capability of a system to fulfill its mission, in a timely manner, in the presence of attacks, failure, or accidents" [**2**]. One promising way to achieve survivability is through adaptivity: changing the system behavior to achieve the system goal in response to the changing environment [**3**]. However, unpredictable adaptation can sometimes result in worse performance than without adaptation [**4**]. Scalability is defined as "the ability of a solution to work when the size of the problem increases" (From Dictionary of Computing at http://wombat.doc.ic.ac.uk). As the size of networked systems grows, scalability becomes a critical issue when developing practical software systems [**5**].

There are several trends in building software systems including distributed computing, component technology, and adaptive software. Distributed computing aims at using computing power of machines connected by a network. When a task requires intensive computation, it becomes a natural choice to achieve high performance. Component technology[1] utilizes the components so that developers can build systems needed by simply defining their specific roles and wiring them together [**6**][**7**]. In networks with component-based architecture, each component is highly specialized for specific tasks. Another emerging technology is adaptive software [**8**][**9**]. Adaptive software has alternative algorithms for the same numerical problem and a switching function for selecting the best algorithm in response to environmental changes. As modern operating environments are highly dynamic, adaptive software becomes an important tool to achieve portable high performance.

We study a large-scale information network, which is composed of distributed software components linked together through a task flow structure. A problem given to the network is

[1] A component is a reusable program element.

decomposed in terms of root tasks for some components and those tasks are propagated through a task flow structure to other components. As a problem can be decomposed with respect to space, time, or both, a component can have multiple root tasks that can be considered independent and identical in their nature. The service provided by the network is to produce a global solution to a given problem, which is an aggregate of partial solutions of individual tasks. Each component can have alternative algorithms to process a task which trade off processing time and value of partial solution. Quality of Service (QoS) of the network is determined by the value of global solution and time for generating global solution (i.e., completion time). For a given topology, the network can control its behavior by utilizing two different kinds of control actions: algorithm selection and resource allocation. While resource allocation tries to efficiently utilize limited resources, alternative algorithms can change the amount of required resources. The resource allocation we are addressing here, is allocating resources of each machine to the residing components for a given topology. Survivability of the network is the capability to provide high QoS in the presence of accidental failures and malicious attacks. In this thesis we design a scalable adaptive control mechanism to support the survivability of such networks.

1.1 Problem Domain

The networks we study represent distributed and component-based architectures for providing a solution to a given problem. When the size of a problem becomes large, the size of the network as well as the number of tasks for each component can be large. One can imagine wide range of scientific and engineering problems that can be solved with such architectures.

Cougaar (Cognitive Agent Architecture: http://www.cougaar.org) developed by DARPA (Defense Advanced Research Project Agency), is such an architecture for building large-scale multiagent systems. Recently, there have been efforts to combine the technologies of agents and components to improve building large-scale software systems [**10**][**11**][**12**]. While component technology focuses on reusability, agent technology focuses on processing complex tasks as a community. Cougaar is in line with this trend. In Cougaar a software system comprises of agents and an agent comprises of components (called plugins). The task flow structure in these systems is that of components as a combination of intra-agent and inter-agent task flows. As the agents in Cougaar can be distributed both from geographical and information content sense, the networks implemented in Cougaar have distributed and component-based architecture.

UltraLog (http://www.ultralog.net) networks are military supply chain planning systems implemented in Cougaar [**13**][**14**][**15**][**16**][**17**]. Each agent in these networks represents an organization of military supply chain and has a set of components specialized for each functionality (allocation, expansion, inventory management, etc) and class (ammunition, water, fuel, etc). The objective of an UltraLog network is to provide an appropriate logistics plan for a given military operational plan. A logistics plan is a global solution which is an aggregate of individual schedules built by components. An operational plan is decomposed into logistics requirements of each thread for each agent, and a requirement is further decomposed into root tasks (one task per day) for a designated component. As a result, a component can have hundreds of root tasks depending on the horizon of an operation and thousands of tasks to process as the root tasks are propagated. Figure **1.1** shows an example of the network with four agents residing in two machines. There are two communities in the network: Supply community (Agents 1, 2, 3) and Transport community (Agent 4). Task Generators generate logistics tasks by converting the operational plan to logistics requirements. Those tasks are sent to Expanders and expanded to multiple tasks. In the example, Expanders in Supply community expand a task into two tasks, one for Allocator and one for Aggregator. Allocators in Supply community allocate tasks to inventory assets by scheduling their demands. As tasks are allocated to inventory assets, Inventory managers generate tasks to refill their inventory assets considering asset consumption and inventory policy. Agent 4 aggregates tasks and allocates to transport assets.

An UltraLog network performs initial planning and continuous replanning to cope with logistics plan deviations or operational plan changes. Initial planning and replanning are the instances of the current research problem. QoS of these networks is determined by the quality of logistics plan (value of solution) and (plan) completion time. These two metrics directly affect the performance of an operation. As the scale of operation increases there can be thousands of agents (tens of thousands of components) working together to generate a logistics plan. Also, as the networks are working in a military environment, they are especially vulnerable to malicious attacks and accidental failures. Now, the question is how we can make this system survivable to generate high quality logistics plans in a timely manner in the presence of such adverse events.

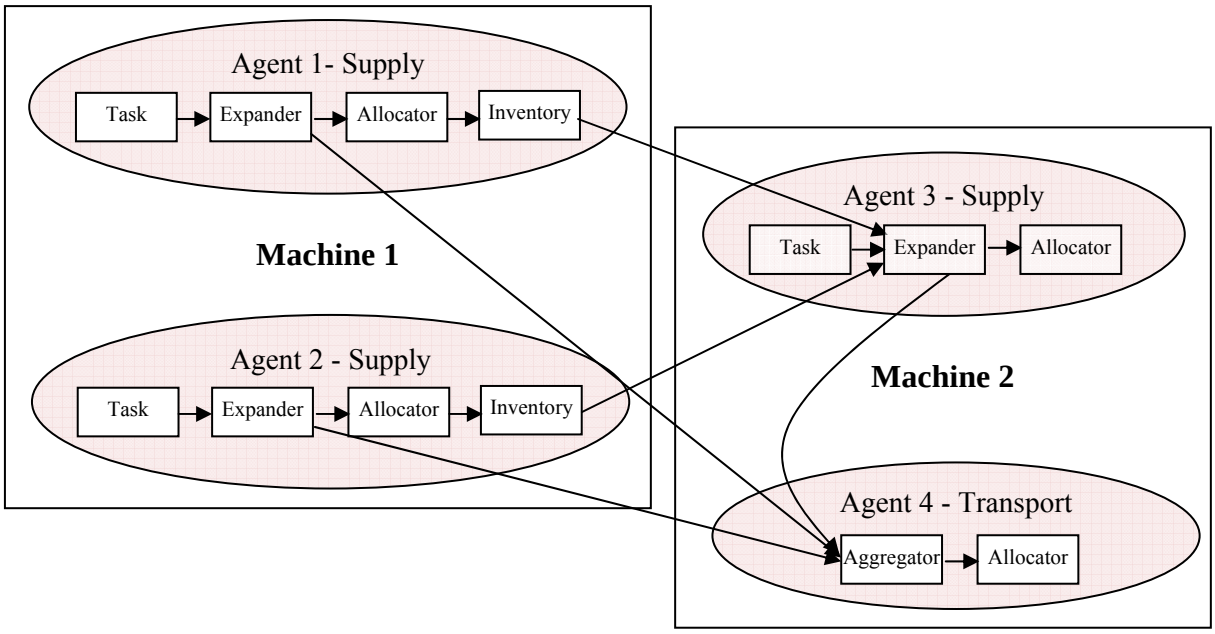


Figure **1.1**: An example UltraLog network

The problem we address in this thesis can be generalized to Grid Service environments. The Grid technology provides inexpensive access to large computational resources across institutional boundaries [**36**]. Services can be composed over the Internet via Web Service technology creating enormous opportunities for automation of business processes [**37**]. OGSA (Open Grid Services Architecture: http://www.globus.org/ogsa/) defines a Grid system architecture based on both the Grid and Web Service technologies. The Grid Service enables the integration of resources and services across distributed, heterogeneous, dynamic virtual organizations [**38**]. Cost and quality considerations may force large number of customers to look for resources and services via such an architecture to solve their own computing problems from modeling to analysis. Ubiquitous computing technology embeds computers in various objects and places for sensing and controlling environments [**39**]. As this technology is becoming more realizable it might be intractable to deal with enormous amount of complex computing problems without the use of such an architecture. In a Grid Service environment, a problem is solved by composing distributed services and resources, and, following the trends in building software systems, through a task flow structure between components as a combination of intra and inter service task flows.

1.2 Research Objectives

In this research we design a scalable adaptive control mechanism for the information networks with distributed and component-based architecture. To achieve this objective, we choose the framework of Model Predictive Control (MPC). First, to address adaptivity, we model stress environment by quantifying resource availability through sensors. Second, we build a mathematical programming model, which predicts QoS as a function of control actions (algorithm selection and resource allocation) as well as the stress environment. Third, we provide an auction market as a decentralized coordination mechanism for solving the programming model. By periodically opening the auction market, the system can achieve desirable performance adaptive to changing stress environment while assuring scalability property. Also, we study the behavior of a computational ecosystem model, which is a simplified network under the designed control mechanism. Though decentralized control is an inevitable trend in controlling networked systems, it may lead to undesirable properties with respect to stability. In order to ensure a scalable adaptive control mechanism design, we need to address the following three objectives:

- Build a mathematical programming model adaptive to changing stress environment.
- Design a decentralized coordination mechanism for solving the programming model.
- Study the stability property arising from the decentralized coordination mechanism.

1.3 Organization of the Thesis

The organization of the thesis is as follows. In Chapter 2, we formally define the problem in detail. We review previous control approaches in Chapter 3 and design overall solution methodology in Chapter 4. In Chapters 5 and 6 a mathematical programming model is built and in Chapter 7 the programming model is decentralized. Chapter 8 deals with a computational ecosystem model. After showing empirical results in Chapter 9, we discuss implications and possible extensions of this research work in Chapter 10. Figure **1.2** summarizes research road map and thesis organization.

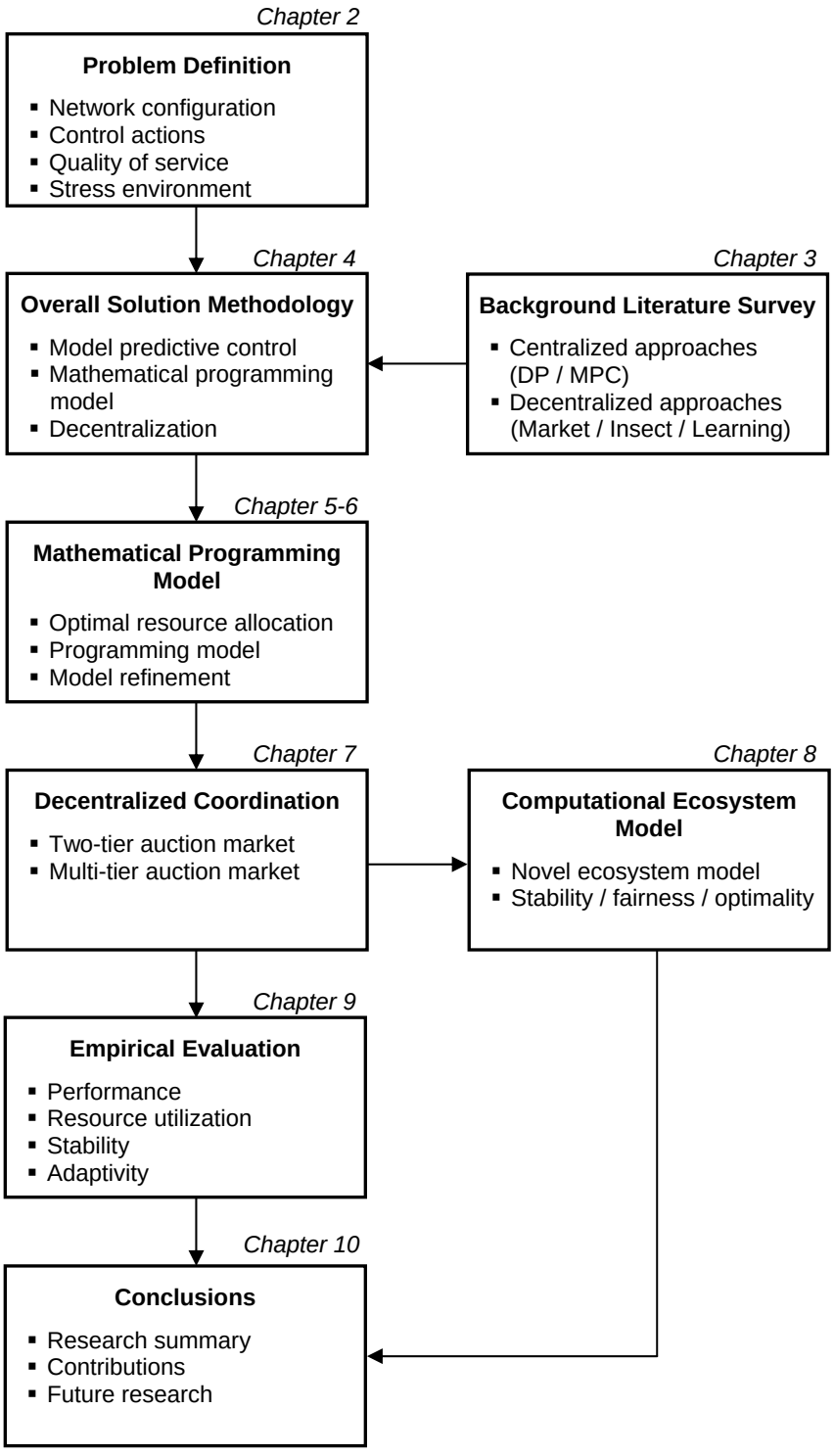


Figure **1.2**: Research road map and thesis organization

Chapter 2

Problem Definition

In this chapter we formally define the problem by detailing network configuration, control actions, quality of service, and stress environment. We focus on computational CPU resources assuming that the system is computation-bounded.

2.1 Network Configuration

A network is composed of a set of components A and a set of nodes (i.e., machines) N. K_n denotes a set of components that reside in node n sharing the node's CPU resource. Task flow structure of the network, which defines precedence relationship between components, is an arbitrary directed acyclic graph. A problem given to the network is decomposed in terms of *root tasks* for some components and those tasks are propagated through the task flow structure. Each component processes one of the tasks in its queue (which has the component's root tasks as well as the tasks from predecessor components) and then sends it to its successor components. We denote the number of root tasks of component i as rt_i.

The example network in Figure **2.1** is composed of sixteen components in five nodes. Nine components have root tasks (e.g. rt_{A1} = 1000 and rt_{A2} = 200) and those tasks are processed and propagated through the task flow structure.

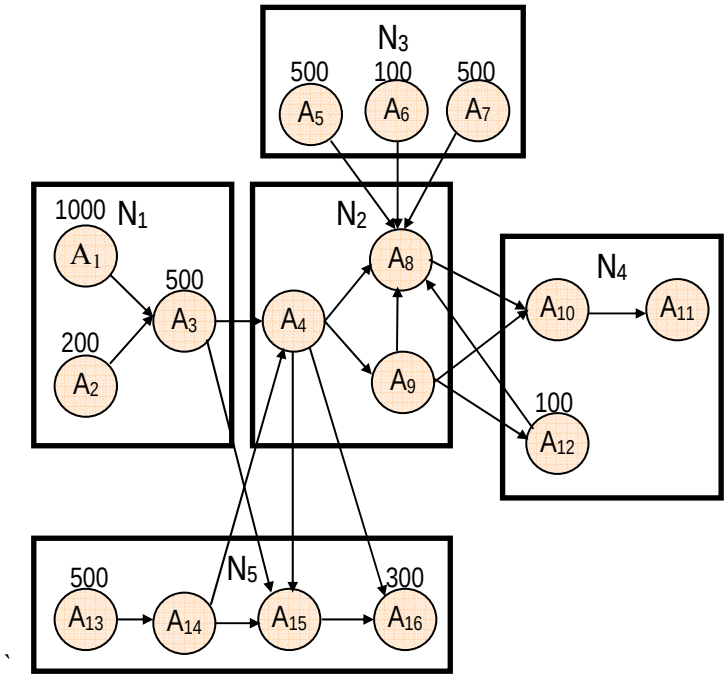


Figure **2.1**: An example network configuration

2.2 Control Actions

A network can utilize two different kinds of control actions to control its behavior: algorithm selection and resource allocation. Each component processes a task by choosing one of the alternative algorithms and utilizing allocated resources in its residing node. For example, Figure **2.2** depicts that component A_7 has two alternative algorithms which are LP (linear programming) and a heuristic, and node N_4 has three residing components (A_{10}, A_{11}, A_{12}) sharing its CPU resource.

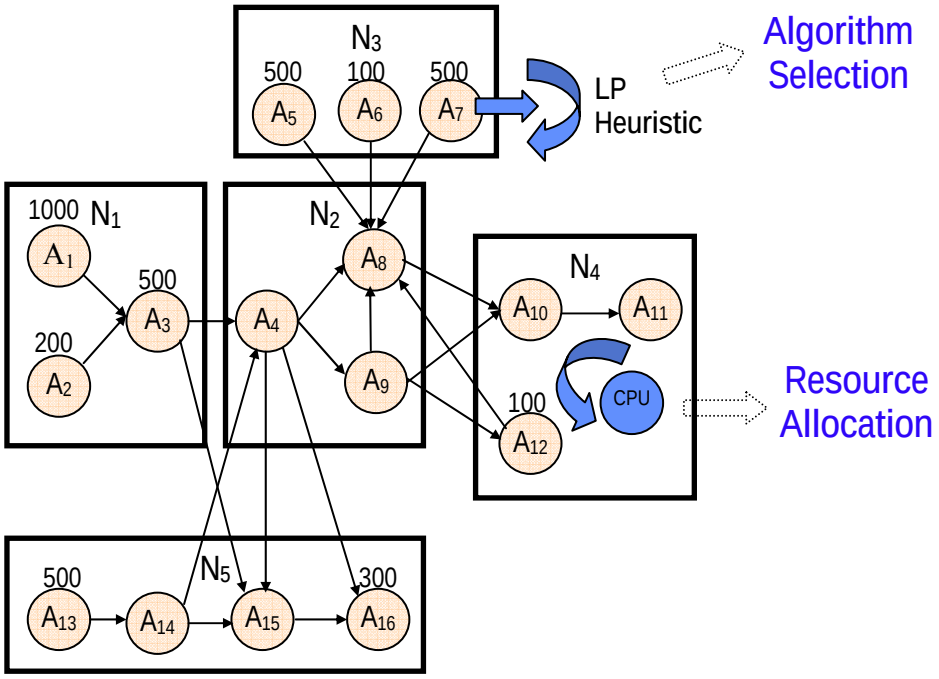


Figure **2.2**: Control actions: algorithm selection and resource allocation

2.2.1 Algorithm Selection

A component can use one of the alternative algorithms to process a task. Different alternatives trade off CPU time and value of solution with more CPU time resulting in higher expected solution value. As one can find optimal mixed alternatives, a component has a monotonically increasing convex function, say *value function*, with CPU time as a function of value. We call the value in the function as *value mode* that the component can select as its decision variable. A value function is defined with three elements as $\langle f_i(v_i), v_{i(min)}, v_{i(max)} \rangle$. This function indicates that a component i's expected CPU time[2] to process a task is $f_i(v_i)$ with a value mode v_i and $v_{i(min)} \leq v_i \leq v_{i(max)}$. Figure **2.3** shows an example value function which is piece-wise linear increasing convex function. We assume that components cannot change the mode for a task in process.

[2] The distribution of CPU time can be arbitrary though we use only expected CPU time.

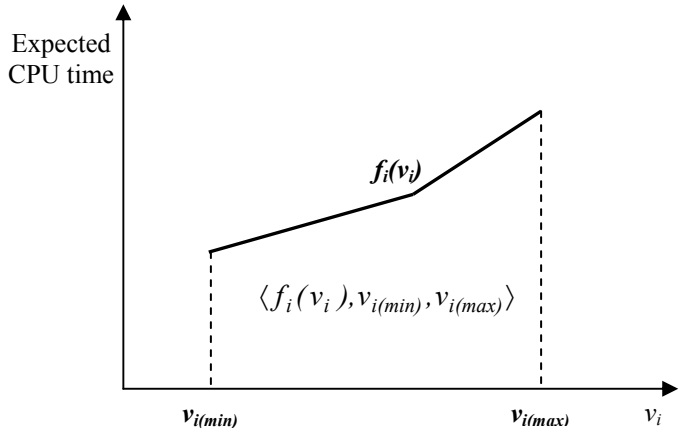


Figure **2.3**: An example value function

2.2.2 Resource Allocation

When there are multiple components in a node, the network needs to control its behavior through resource allocation. There are several CPU scheduling algorithms for allocating a CPU resource amongst multiple threads. Among the scheduling algorithms, proportional CPU share (PS) scheduling is known for its simplicity, flexibility, and fairness [**19**]. In PS scheduling threads are assigned weights and resource shares are determined proportional to the weights [**40**]. Excess CPU time from some threads is allocated fairly to other threads. There are many PS scheduling algorithms such as Weighted Round-Robin scheduling, Lottery scheduling, and Stride scheduling [**41**][**42**][**107**]. We adopt PS scheduling as resource allocation scheme because of its generality in addition to the benefits mentioned above. We define the resource allocation variable set as $\boldsymbol{w} = \{w_i(t): i \in A, t \geq 0\}$ in which $w_i(t)$ is a non-negative weight of component i at time t. If total managed weight of a node n is ω_n, the boundary condition for assigning weights over time can be described as:

$$\sum_{i \in K_n} w_i(t) = \omega_n \quad where\ w_i(t) \geq 0 \ . \tag{2.1}$$

2.3 Quality of Service

The service provided by the network is to produce a global[3] solution to a given problem, which is an aggregate of partial[4] solutions of individual tasks. QoS of the network is determined by *the value of global solution* and *the cost of completion time*. The value of global solution is the summation of partial solution values, and the cost of completion time is determined by a cost function *CCT(T)* which is a monotonically increasing function of completion time *T*. We assume that the solution values and cost are represented in a common unit[5]. Let v_i^d be the value mode used to process d^{th} task by component i and e_i the number of tasks processed by component i to the completion. Then, QoS is computed as:

$$QoS = \sum_{i \in A} \sum_{d=1}^{e_i} v_i^d - CCT(T) . \qquad (2.2)$$

2.4 Stress Environment

Survivability stresses such as accidental failures or malicious attacks, affect the system by directly consuming resources or indirectly invoking defense mechanisms as remedies. For example, "Denial of Service" attack consumes resources directly while relevant defense mechanism also consumes resources in terms of resistance, recognition, and recovery [**1**]. A defense mechanism may move components to other nodes changing the network topology dynamically. Mobile technology provides an innovative concept for managing distributed systems to adapt dynamically to changing environments though there are technical challenges such as security to fulfill its promise [**43**][**44**][**45**][**46**].

We consider both survivability stresses and remedies as stress environment from the viewpoint of the network. It would be intractable to address each possible stress environment since the space of stress environment is high-dimensional and also evolving [**18**][**20**]. But, as we concentrate on CPU resources, a stress environment can be regarded as a combination of stressors

[3] We call the solution of a given problem as 'global solution' to avoid the confusion with 'partial solution'.
[4] When a component completes a task it produces a partial solution for the task.
[5] Relative importance can be considered by scaling the functions and it results in the same function structures.

and network topology. The stressors, which are sharing resources with the components, may have admission to access resources or be stealing resources without admission. Figure **2.4** describes a stress environment of the example network. A stressor is consuming the resource of N_2 and the components of N_5 are moving to other nodes due to some catastrophic failure resulting in a different network topology.

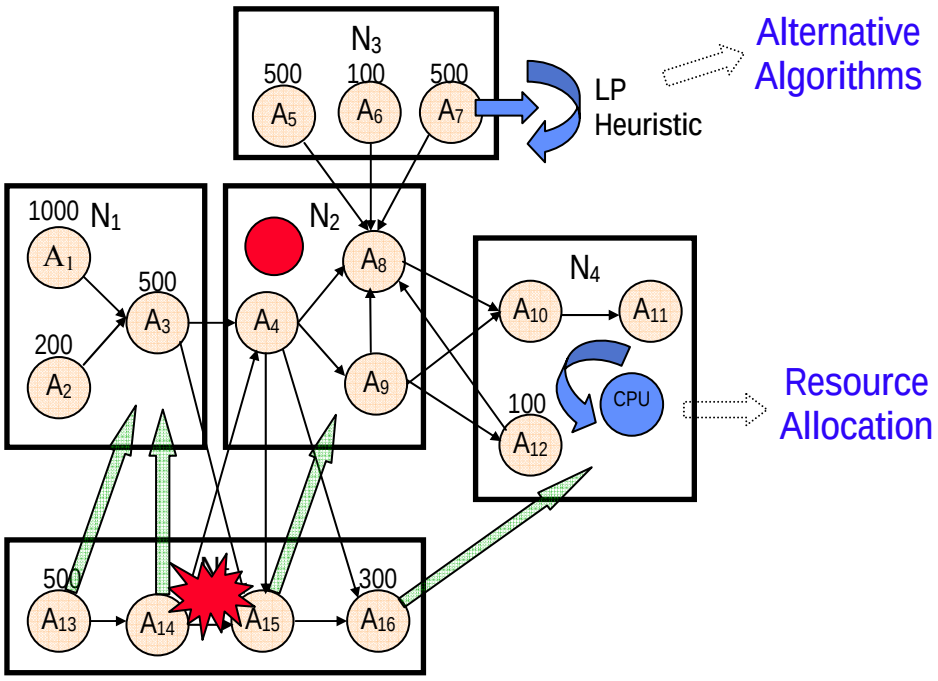


Figure **2.4**: Stress environment

2.5 Problem Definition

The objective of this research is to develop a control mechanism to maximize QoS by utilizing alternative algorithms (***v***) and resource allocation (***w***) adaptive to changing stress environment while ensuring scalability as described in (**2.3**).

$$\underset{\boldsymbol{v},w}{\operatorname{argmax}} \; QoS \tag{2.3}$$

The research problem has several characteristics that help understanding the problem and developing appropriate control mechanisms:

- **Large-scale network**: The network can be large-scale as the number of components increases with the scale of the given problem to the network. (Scalable control mechanism)
- **Unpredictable stress environment**: To predict the behavior of stress environment, which is a combination of stresses as well as remedies, is practically impossible. (Practical modeling of stress environment)
- **Finite time horizon**: The time horizon for a network to generate a global solution is finite and also it is one important element of QoS of the network. (Agile supply of control policy)
- **Indecomposable QoS**: QoS is not decomposable to individual components' or tasks' objectives because the completion time is common throughout the network. (Tight coordination)
- **Complex dynamics**: Components interact with each other and stressors through task flows as well as resource sharing. Also, these interactions are in parallel with control actions. (Tractable control mechanism)

Chapter 3

Background Literature Survey

In general, controlling a dynamic system can be centralized or decentralized. We discuss several representative approaches to help us develop appropriate control mechanisms.

3.1 Centralized Approaches

We investigate two centralized control approaches: dynamic programming (DP) and model predictive control (MPC).

3.1.1 Dynamic Programming

Dynamic programming (DP) was introduced by Bellman [**65**]. The basic idea is the Principle of Optimality, which says that in any state along an optimal trajectory, the remaining part must constitute an optimal trajectory when that state is considered as an initial state. This principle results in Bellman Optimality Equation with different forms depending on the nature of dynamic systems. DP algorithms solve the equations to produce reactive strategies in terms of optimal closed-loop feedback control policy, which is a rule specifying optimal action as a function of state.

Markovian decision problem (MDP), as a class of stochastic optimal control problems for discrete-time dynamic systems, is the simplest and the most extensively studied problem. There are off-line DP algorithms to design an optimal control policy for MDP with a complete and accurate model of the decision problem, such as synchronous dynamic programming, Gauss-Seidel dynamic programming, and asynchronous dynamic programming [**21**]. As the complexity grows exponentially with the dimension of the state space though the principle of optimality reduces the complexity significantly, it takes impractically long time to converge to a (near) optimal policy. As a result, there have been some efforts to perform DP in real-time concurrently with the actual process of control, such as in RTDP (real-time DP), Trial-based RTDP, and

LRTA* (learning-real-time A*) [**21**]. However, they take longer time to converge than off-line DP algorithms at the cost of exploitation.

When there is no complete and accurate model of the decision problem, there is a need for adaptive control methods. There are two types of methods to solve these problems: indirect and direct methods [**21**]. Indirect methods such as adaptive RTDP explicitly model the dynamic system. System identification algorithms update parameters of the current system model and control decisions are made based on the current system model. Direct methods, on the other hand, form policies without using explicit system models. These methods such as Q-learning and actor-critic, are a form of reinforcement learning, i.e., if an action is followed by a satisfactory state, then the tendency to choose that action is reinforced [**22**][**104**]. The indirect methods are very simple and powerful addressing the issue of large state space dimensionality with enormously less computation at each time step compared to conventional DP algorithms. For adaptive control methods, a central issue is the conflict between exploitation and exploration. Decreasing the exploration over time to a minimum value would be beneficial to resolve the conflict. There are a variety of reinforcement-learning techniques that work effectively on a variety of small problems, though very few of these techniques scale well to larger problems [**23**]. To make a real system work it would be necessary to utilize some knowledge corresponding to the system. A knowledge-free approach would not have achieved worthwhile performance within the finite lifetime of the systems. In principle, adaptive methods take longer time to converge than non-adaptive methods at the cost of exploitation and exploration. When the environment is non-stationary, it takes more convergence time in new environment because of the memory and system utility becomes lower as exploration must continue in order to adapt to environmental changes. There are adaptive control methods for Semi-Markov Decision Problems (SMDP) for discrete-event dynamic systems [**106**]. However, as they include more model parameters the convergence is rather slow.

3.1.2 Model Predictive Control

DP provides optimal closed closed-loop policy but the complexity in solving optimality equation grows exponentially with the dimension of state space. Model Predictive Control (MPC), also called Receding Horizon Control (RHC), is one of optimal control approaches that help overcome such a problem.

When there is a reasonably accurate process model, one way to design closed-loop policies on-line is through MPC. MPC refers to a class of control algorithms that utilize an explicit model to predict the future response of a system [**21**][**24**][**25**][**26**][**27**]. In MPC a closed-loop policy is achieved through repeated on-line design of optimal open-loop policies. An open-loop control policy is a sequence of control signals for the given initial state without using feedback information about the system's actual behavior. For each current state, an optimal open-loop policy is designed for finite-time horizon by solving a static optimization problem based on an explicit process model. After applying the first action from the optimal policy, the remainder of policy is discarded. The design process is repeated for the next observed state feedback. Through this repeated procedure MPC produces a control policy that is reactive to each current system state, a closed-loop policy. The resulting control policy from MPC is an optimal open-loop feedback control policy. In MPC the accuracy of process models is essential though feedback can overcome some effects of poor models.

Though MPC does not give optimal closed-loop policy in stochastic environments, the periodic design process alleviates the impacts of stochasticity. Also, its complexity becomes significantly reduced compared to DP, and it is easy to adapt to new contexts by explicitly handling objective function or constraints. It however requires efforts to develop accurate process models and has scalability problem when the mathematical programming model is large-scale.

3.2 Decentralized Approaches

In general, controlling distributed systems by means of a central controller has several disadvantages in terms of scalability, robustness, and information security. The controller usually needs current knowledge about the entire system, necessitating communication links from every part of the system to the controller. These centralized control mechanisms scale badly, due to the rapid increase of computation and communication overheads with increase in system size. Single point failure of the controller will often lead to failure of the complete system. Centralized controller therefore could lead to non-robust network. Information security and confidentiality is another important barrier against centralized control. The network entities may not be willing to reveal all the details of them especially when they are composed of multiple organizations with different interests.

Agent technology is a promising approach that combines the notion of local decision-making with concerns for the distributed system context [**101**][**102**]. An agent is software that is capable of flexible autonomous action in order to meet its design objectives. A multiagent system (MAS) can be defined as a loosely coupled network of agents that work together to achieve system objectives beyond individual capabilities or knowledge [**103**]. MAS limits the complexity of the problem by partitioning the complex global problem into a set of simpler local problems. As the decisions are decentralized, there is no single-point failure and agents do not need to reveal all the details.

One of the most interesting phenomena of natural systems is that highly-structured global pattern emerges over time from the interaction of simple components without any central leader [**51**][**52**][**53**]. There have been significant efforts to design MAS motivated by such self-organizing systems. Though entities act with a simple mechanism without central authority, these systems are adaptive and desirable global performance can often be realized. We survey three representative decentralized control approaches: market-based control, insect-behavioral control, and learning-based control.

3.2.1 Market-based Control

Market-based control is described as a paradigm for controlling complex systems that would otherwise be very difficult to control, maintain, or expand [**54**]. Market-based control works through the interaction of local agents in the same way as economic markets. Since markets facilitate resource allocation in human societies, one might expect them to be similarly useful in controlling distributed systems such as computer networks [**59**]. In the spirit of Adam Smith's "invisible hand" from economics, it is reasoned that collective behavior driven by self-interested agents will lead to globally desirable performance. One popular form of market-based control is auction mechanism. Auctions have been defined as "a market institution with an explicit set of rules determining resource allocation and prices on the basis of bids from the market participants" [**55**]. There is a large design space of auction mechanisms and subsequently a variety of auctions are available for designing MAS [**118**].

Enterprise and Challenger are market-based control systems proposed for minimizing mean flow time in distributed processor allocation problems [**56**][**57**]. When a task arrives at a processor it broadcasts a "request for bids" to all the agents. Agents receiving the request make

bids in terms of estimated completion time for the job as soon as it becomes idle. After an evaluation delay the originating agent selects the best bid if there are more than one bid, otherwise it selects the next bid that comes first. Another market-based algorithm for distributed processor allocation problems were designed [**58**]. When a task arrives at a processor it is allocated initial budget and should use the budget to migrate from one processor to another processor and to use CPU time in a processor. Tasks have preferences for current and its neighboring processors based on cost, service time, or their combination in limit of available current budget. When a resource is free, tasks bid it with a price of current price and some fraction of estimated remaining budget in the future. If a task's bid wins the fraction decreases, otherwise increases. A resource's price is updated with the recent transaction price.

In manufacturing environment, a market-based control approach was designed for scheduling truck painting [**60**]. A scheduling program interacts with the paint booths through an auction protocol. When a truck arrives, each booth bids for the painting job based on setup cost, current workload, etc. Also, several market-based control algorithms were proposed for controlling calls with respect to admission or routing in telecommunication networks [**61**][**62**][**64**].

3.2.2 Insect-behavioral Control

There are several algorithms inspired by effective and adaptive behavior of social insect colonies such as ants, bees, wasps, and termites [**66**]. A social insect colony can be considered as a decentralized problem solving system through simple interactions between them in a very flexible and robust way. An important and interesting behavior of ant colonies is their foraging behavior, in particular how ants can find the shortest paths between food sources and their nest. Ants deposit a substance called pheromone while moving between food sources and nest, forming a pheromone trail. Ants tend to choose paths with strong pheromone concentrations probabilistically. This simple behavior can give rise to the emergence of shortest paths between food sources and their nest when many paths are available. Inspired by such a behavior an ant algorithm was proposed to the traveling salesman problem [**67**]. Subsequently several ant algorithms have been designed for controlling dynamics systems as follows.

ABC (Ant-Based Control) is a routing algorithm for minimizing the call failure in telecommunication networks [**68**]. Each node has pheromone table that indicates the goodness of

next nodes for each destination. Nodes generate ants with random destinations to collect global information. Ants move from node to node by selecting the best next node according to the pheromone table and when arriving at destination they update the pheromone table depending on their trip time encouraging the calls to follow the shortest path. AntNet was proposed as a routing algorithm for maximizing the throughput and minimizing packet delay in packet-based communication networks [**69**][**71**]. This algorithm is similar to ABC system but ants in AntNet select the next node probabilistically according to the routing table. AC^2 (Ant Colony Control) was designed as a routing algorithm for maximizing the throughput in shop floor [**70**]. Each arriving job is assigned an ant and it selects next machine probabilistically according to pheromone trail of the job type between machines. When a job is completed, the corresponding ant increases the pheromone of the job type through all the machines it has been processed and decreases the pheromone to the machine with different job types. This encourages the same type of jobs to select the same machines to minimize the setup time.

There are additional interesting behaviors in ant colonies with respect to task allocation. An ant senses the densities of other ants and objects of interest by simply observing the intervals of encounters. The sensed densities form task demand (demand increases with less ants and more objects) and tasks are emergently allocated based on social dominance of the ants in addition to the demands. This task allocation behavior was applied to a mobile sensor network [**63**]. The sensor network tries to maximize network coverage by dynamically allocating the mobile sensors to different regions. Each mobile sensor mimics the task allocation behavior of the ant colonies. They observe the intervals of other sensors and targets in limited range and their social dominance is determined by their actual performance. Based on the demands and social dominances the sensors decide to stay or move to other regions.

Similar to ant algorithms, wasp algorithms were proposed inspired by task allocation behavior of wasp colonies. An individual wasp has a response threshold for each zone of the nest [**72**][**105**]. Based on a wasp's threshold for a given zone and the amount of stimulus from related task in that zone, a wasp may or may not become engaged in the task for that zone. When two wasps encounter each other, the wasp with the higher social rank will have a higher probability of dominating in the interaction. This task allocation behavior was used for maximizing the throughput in shop floor [**73**][**74**][**75**]. Each machine is assigned a routing wasp in charge of assigning jobs to corresponding queue. Routing wasps have different response thresholds for different job types. If a machine is processing or setting up a job type, the threshold for the job type decreases and those for other job types increase. If a machine is idle the threshold decreases

according to the idle time. Each job generates stimulus that increases as the waiting time increases. Each wasp picks up a job probabilistically depending on the strength of the stimulus and response threshold. If there are wasps more than one for a job the job is allocated to a wasp probabilistically depending on the social ranks determined by current workload.

3.2.3 Learning-based Control

Reinforcement learning can be used without prior knowledge of the system model. By making agents to learn from their experience this method can be used in a decentralized mode. Q-Routing was proposed as a routing algorithm for minimizing packet delivery time in packet-based communication networks [**77**][**78**]. Each node has a Q table that shows the estimates of delivery time (including waiting time in the node) through neighboring nodes for each destination. A packet selects a neighbor with the lowest total delivery time for the destination, and as soon as the packet is delivered to the selected node the lowest estimate from the selected node to the destination node is informed to the originating node. By adding this value to the waiting time the originating node estimates new delivery time and update the Q value using the difference between new and old estimates with a certain learning rate. Also, CDRQ (Confidence-based Dual Reinforcement Q) routing, a combination of CQ routing and DRQ routing, was designed for packet-based communication networks [**79**][**80**][**81**]. In CQ routing each Q value has a confidence value that indicates its reliability. If a Q-value is not updated for a long time, its reliability goes down. When a packet is sent to a neighbor the neighbor sends back the confidence value as well as the lowest delivery time for the destination. The confidence values of originating node and the neighbor are used to determine learning rate. In DRQ routing the system utilizes backward exploration. When a node sends a packet to a neighbor the neighbor also updates corresponding Q value.

It would be possible to bias the reward of each node such that its objective and the global objective are aligned. This concept was applied to routing in packet-based communication networks where each node learns to maximize the biased reward [**83**][**84**][**85**]. Also, a distributed reinforcement learning algorithm was proposed for distributed processor allocation problems [**82**]. The efficiency of each processor is reinforced based on its actual performance.

Chapter 4

Overall Solution Methodology

In this chapter we develop the overall control structure after discussing the surveyed control approaches with respect to the characteristics of the problem in consideration.

4.1 Discussion on the Surveyed Control Approaches

MPC gives optimal open-loop policy rather than optimal closed-loop policy and it has significant advantages with respect to computational complexity compared to DP. Adaptivity of MPC is another important benefit. It can explicitly handle objective function or constraints, which is hard to achieve with DP because DP should find relevant policy repeatedly when the system parameters change. However, MPC gives less optimal policy especially in stochastic environments and continuously requires computational resources to solve the static mathematical programming model. Therefore, the choice between the two approaches depends on the lifetime of the system, dynamic characteristics of environment, and problem size. If a system has long lifetime, its environment is stationary, and problem size is small, DP approaches would be appropriate, otherwise MPC would be more beneficial. As discussed in Chapter 2, the networks we are addressing work in finite time horizon which also needs to be minimized, the environment is non-stationary due to unpredictable or unidentifiable behavior of stress environment, and the network can be large-scale according to the scale of the given problem to the network. Therefore, our choice between the two control approaches is clearly MPC. However, we need to overcome scalability problem of MPC in dealing with large-scale mathematical programming model.

The decentralized control approaches are scalable and robust in that control is distributed to the multiple agents in the network without a central controller. Globally desirable performance emerges when agents try to achieve their local objectives through interactions between them. However, QoS as a global performance in this research problem cannot be decomposed to individual agents' objectives because the completion time (one of the two conflicting QoS elements) is common throughout the network. In addition, decentralized approaches cannot guarantee agile convergence to optimality and can take long time to adapt to changing

environment in general, due to without a centralized authority. As the networks we are addressing works in finite time horizon and the completion time is one important component of QoS, the agility is a critical consideration. Therefore, rather than adopting one of the decentralized approaches, it would be desirable to achieve their benefits by decentralizing the mathematical programming model of MPC. With successful decentralization, MPC can provide desirable optimality and agility while supporting scalability and robustness.

4.2 Solution Methodology

Our objective is to design a scalable adaptive control mechanism for the networks under consideration as stated earlier. By incorporating the discussions we have made in the previous section, we design such a mechanism in a systematic way as follows.

First, we adopt MPC as control framework considering the characteristics of the current problem. The size of the networks is large working in finite time horizon, and they need to adapt to unpredictable stress environment. *Second*, we model stress environment implicitly by quantifying resource availability of the system through sensors. To address the intractability of dealing with the stress space directly, our assumption is that the stress environment affects the system through resource availability. *Third*, under MPC framework, we build a mathematical programming model with the resource availability incorporated, which invokes optimal control actions as a function of both state and stress environment. *Fourth*, we provide a decentralized coordination mechanism for solving the programming model. Computations and communications are distributed to multiple entities through an auction market.

By periodically opening the auction market, the system can achieve desirable performance adaptive to changing stress environments while assuring scalability property. The overall control structure is described in Figure **4.1**. Components monitor their resource availability and make decisions periodically by solving a mathematical programming model through an auction market. We detail the control mechanism in Chapters 5 to 7. In chapter 5 we build a mathematical programming model and refine it in Chapter 6. Then, the programming model is decentralized in Chapter 7.

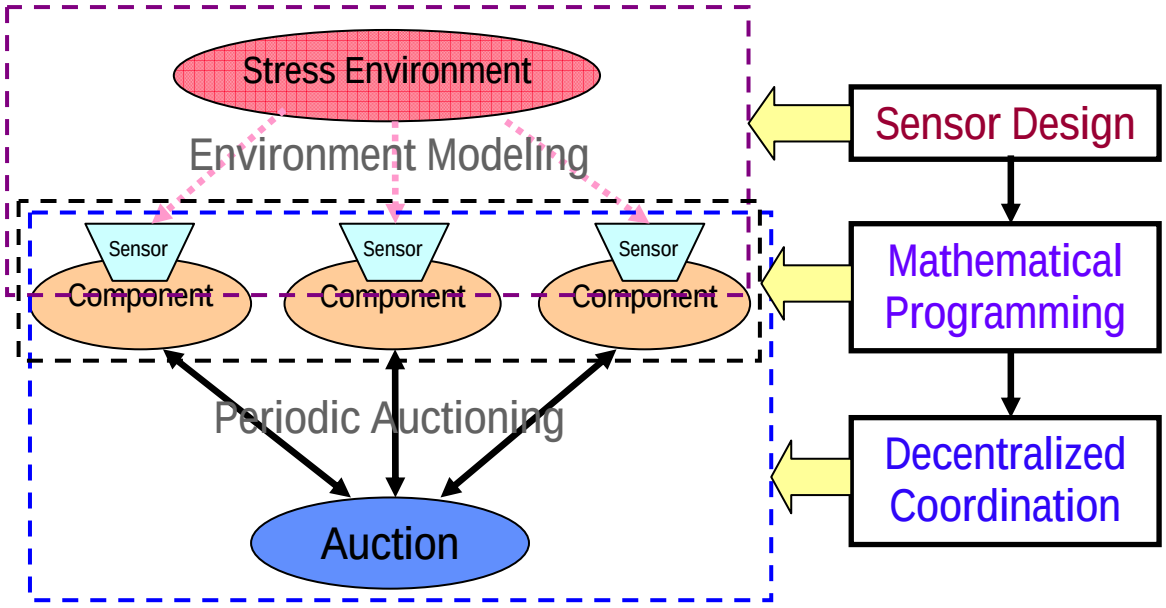


Figure **4.1**: Overall control structure

Chapter 5

Mathematical Programming Model

In this chapter, we build a mathematical programming in the framework of MPC. We construct optimal resource allocation policy after exemplifying the effects of resource allocation and build a mathematical programming model based on this policy by taking into account alternative algorithms. Then, we incorporate stress environment into the programming model by defining sensors that monitor resource availability of the system.

The mathematical programming model is essentially a scheduling problem formulation. There are a variety of formulations and algorithms available for diverse scheduling problems in the context of multiprocessor, manufacturing, and project management scenarios. In general, a scheduling problem is allocating limited resources to a set of tasks to optimize a specific objective. One widely studied objective is completion time (also called makespan in the manufacturing literature) similar to our problem. Though it is not easy to find a problem exactly same as ours, it is possible to convert our problem into one of the scheduling problems. For example, in a job shop, there are a set of jobs and a set of machines. Each job has a set of serial operations and each operation should be processed in a specific machine. A job shop scheduling problem is sequencing the operations in each machine by satisfying a set of job precedence constraints such that the completion time is minimized. When we assign a value mode to each task, our problem can be exactly transformed into such a job shop scheduling problem. However, though the job shop scheduling problem is solvable in polynomial time when there are two machines and each job has two operations, it becomes NP-hard on the number of jobs even if the number of machines or operations is more than two [**29**][**30**]. Considering that the task flow structure of our networks is arbitrary, our scheduling problem is NP-hard with respect to the number of components in general. The increase of the number of tasks and consideration of alternative algorithms make the problem even harder. Moreover, there can be large number of nodes in our networks, leading to increase in the problem complexity.

Though it may be possible to use some available heuristic algorithms from the job shop scheduling problem by taking into account alternative algorithms, our scheduling problem has a particular characteristic, i.e., the number of tasks for each component can be large. Though the increase in the number of tasks adds more complexity, it can also lead us to develop an efficient

heuristic programming model. For this purpose, we analyze the impact of the largeness and utilize it in building the mathematical programming model.

For theoretical analysis, we assume a hypothetical weighted round-robin server for CPU scheduling though it is not strictly required in practice (discussed in Section 5.2). The hypothetical server has idealized fairness as the CPU time received by each thread in a round is infinitesimal and proportional to the weight of the thread.

5.1 Effects of Resource Allocation

In this section we exemplify the effects of resource allocation. Consider a network in which each agent has only one algorithm to process tasks. We denote the expected CPU time per task of the component i as P_i. The completion time T is the time taken to process all the tasks of a network. We denote T_n as the completion time taken to process all the tasks of node n and T_i of component i. Then, the relationships as in (**5.1**) should hold.

$$T = \underset{n \in N}{Max}\, T_n = \underset{i \in A}{Max}\, T_i, \quad T_n = \underset{i \in K_n}{Max}\, T_i . \tag{5.1}$$

A component's instantaneous resource availability $RA_i(t)$ is the available fraction of a resource when the component requests the resource at time t. Service time $S_i(t)$ is the time taken to process a task at time t and has a relationship with $RA_i(t)$ as:

$$\int_t^{t+S_i(t)} RA_i(\tau)\, d\tau = P_i . \tag{5.2}$$

When $RA_i(t)$ remains constant $S_i(t)$ becomes:

$$S_i(t) = \frac{P_i}{RA_i(t)} . \tag{5.3}$$

Now, consider an example network in Figure **5.1**, in which the number of root tasks and expected CPU time per task are represented as $[rt_i, P_i]$. In the example network only N_1 has the chance to allocate its resource as it has two residing components. T_{N1} is invariant to resource allocation and equal to 300 (=100*1+100*2). But, T_{A1} and T_{A2} can vary depending on the resource

allocation of N_1. To observe the effects of resource allocation we trace the behavior of system under different resource allocation strategies.

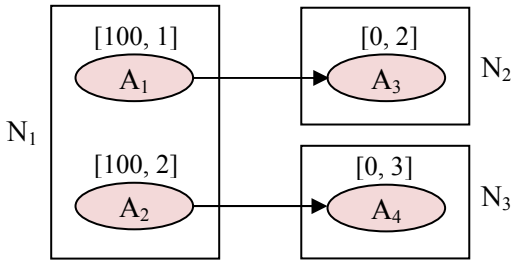
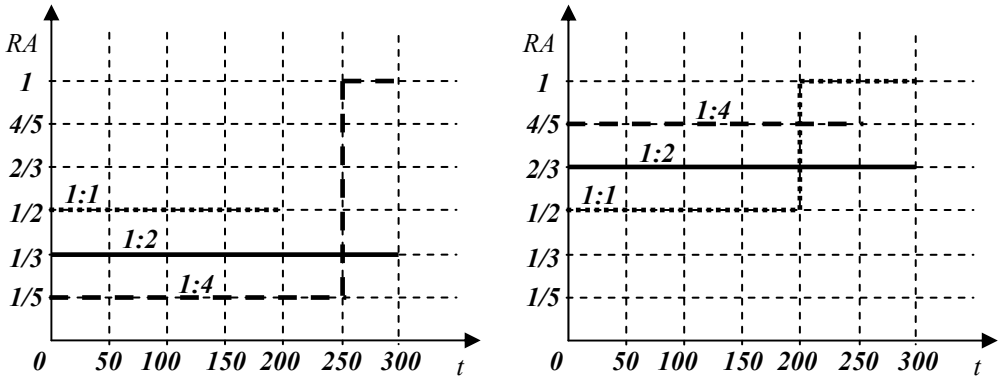


Figure **5.1**: An example network for the effects of resource allocation

When the resource is allocated equally to the components, both $RA_{A1}(t)$ and $RA_{A2}(t)$ are equal to 0.5 initially. As A_1 completes at t=200 (=100*1/0.5), A_2 starts utilizing the resource fully from then, i.e. $RA_{A2}(t)$=1 for $t \geq 200$. So, A_2 completes 50 tasks at t=200 (=50*2/0.5) and remaining 50 tasks at t=300 (=200+50*2/1). A_3 completes at t=202 (=200+1*2/1) because task inter-arrival time from A_1 is equal to its service time. As A_4's service time is less than task inter-arrival time (=4) for $t \leq 200$, A_4 completes 49 tasks at t=200 with one task in queue arriving at t=200. From t=200 task inter-arrival time from A_2 becomes reduced to 2 which is less than A_4's service time. So, tasks become accumulated till t=300 and A_4 completes at t=353 (=200+51*3/1). In this way we trace exact system behavior under three resource allocation strategies as shown in Figure **5.2** and Table **5.1**.



(a) Resource availability of A_1 *(b) Resource availability of A_2*

Figure **5.2**: Effects of resource allocation on resource availability

Table **5.1**: Effects of resource allocation on completion time

	$w_{A1} : w_{A2}$		
	1 : 1	1 : 2	1 : 4
T_{A1}	200	300	300
T_{A2}	300	300	250
T_{A3}	202	302	352
T_{A4}	353	303	302.5
T	353	303	352

The network cannot complete at less than t=300 because each of N_1 and N_3 requires 300 CPU time units. When the resource is allocated with 1:2 ratio, the completion time T is minimal, close to 300. The ratio is proportional to each component's total required CPU time, i.e., 1:2 ≡ 100*1:100*2. One interesting question is whether the proportional allocation can give the best performance even if the successors have different parameters. The answer is yes. If a component A_1 is allocated more resource than the proportional allocation, T_{A3} is dominated by the maximal of T_{A1} and A_3's total CPU time. But, the first quantity is less than T_{N1} and the second quantity is an invariant. So, allocating more resource than the proportional allocation cannot help reducing the completion time of the network. However, if a component is allocated less resource than the proportional allocation, its successor's task inter-arrival time is stepwise decreasing. As a result, the successor underutilizes resource and can complete later than under the proportional allocation. Therefore, the proportional allocation leads the network to efficiently utilize distributed resources and consequently helps minimizing the completion time of the network, though it is localized independent of the successors' parameters.

5.2 Optimal Resource Allocation

Let current time equal to 0 and assume that each component uses a value mode common to all the tasks (i.e. pure strategy). We will discuss the optimality of the pure strategy later in this section. We define *Load Index* LI_i which represents component i's total CPU time required to process its tasks. As a component needs to process its own root tasks as well as incoming tasks from its predecessors, its number of tasks L_i is identified as in (**5.4**), where $\underline{i}$ denotes the immediate predecessors of component i. Then, LI_i is computed as in (**5.5**).

$$L_i = rt_i + \sum_{a \in \underline{i}} L_a \qquad (5.4)$$

$$LI_i = L_i f_i(v_i) \qquad (5.5)$$

To provide theoretical foundation of optimal resource allocation policy, we convert a network into a network with tasks having infinitesimal processing times. Each root task is divided into r infinitesimal tasks and each $f_i(v_i)$ is replaced with $f_i(v_i)/r$. Then, the load index of each component is the same as the original network but tasks are infinitesimal. We denote the completion time of the network with infinitesimal tasks as T'. Also, we define a term called *task availability* as an indicator of relative preference for task arrival patterns. An arrival pattern gives higher task availability than another if cumulative number of arrived tasks is larger or equal over time. A component prefers a task arrival pattern with higher task availability as it can utilize more resource. Consider a network and reconfigure it such that all components have their tasks in their queues at t=0. Each component has maximal task availability in the reconfigured network and the completion time of the reconfigured network forms the lower bound T^{LB} of a network's completion time T given by:

$$T^{LB} = \underset{n \in N}{Max} \sum_{i \in K_n} LI_i \ . \qquad (5.6)$$

Theorem 1. *T' equals to T^{LB} when each node allocates its resource proportional to its residing components' load indices as:*

$$w_i(t) = w_i = \frac{LI_i}{\sum_{p \in K_{n(i)}} LI_p} \omega_{n(i)} \quad \text{for all } i \in A \text{ and } t \geq 0 \ , \qquad (5.7)$$

where n(i) denotes a node in which component i resides.

Proof. $RA_i(t)$ is more than or equal to assigned weight proportion as:

$$RA_i(t) \geq \frac{w_i(t)}{\omega_{n(i)}} \quad for\ t \geq 0\ . \tag{5.8}$$

Suppose a component i receives its tasks at a constant interval of T^{LB}/L_i. Then, under proportional allocation, $S_i(t)$ is less than or equal to T^{LB}/L_i over time as shown in (**5.9**).

$$\begin{aligned} f_i(v_i) &= \int_t^{t+S_i(t)} RA_i(\tau)d\tau \geq \int_t^{t+S_i(t)} \frac{w_i(t)}{\omega_{n(i)}} d\tau = \frac{w_i}{\omega_{n(i)}} S_i(t) \\ &= \frac{LI_i}{\sum_{p \in K_{n(i)}} LI_p} S_i(t) \geq \frac{LI_i}{T^{LB}} S_i(t) \quad \Rightarrow \frac{T^{LB}}{L_i} \geq S_i(t) \quad for\ t \geq 0 \end{aligned} \tag{5.9}$$

So, any component can complete by T^{LB} and generate tasks at a constant interval of T^{LB}/L_i from $t=T^{LB}/L_i$ (first task generation time) under proportional allocation when it receives tasks at a constant interval of T^{LB}/L_i from $t=0$ (first task arrival time). As tasks are infinitesimal and root tasks increase task availability, each component can receive infinitesimal tasks at a constant interval in $0 \leq t \leq T^{LB}$ or more preferably, and complete at less than or equal to T^{LB}. So, the network completes at T^{LB} under proportional allocation.

From Theorem 1 we can conjecture that a network can achieve a performance close to T^{LB} under proportional allocation in the limit of large number of tasks. If nodes do not follow the proportional allocation policy, some components can receive their tasks less preferably than constant interval resulting in underutilization and consequently increased completion time. Also, it is optimal for each component to use a pure strategy. Each component's optimal strategy in the network with maximal task availability is a pure strategy due to the convexity of value functions, and a network can achieve the optimal performance under proportional allocation. Though we have assumed a hypothetical weighted round-robin server which is difficult to realize in practice, our arguments do not seem to be invalid because they are based on worst-case analysis and quantum size is relatively infinitesimal compared to working horizon in reality.

The proportional resource allocation policy has several emergent properties as discussed in detail in Appendix A. Though it is localized requiring almost no computation, it realizes desirable global performance adaptive to changing environments. Such emergent properties hold in the limit of large number of tasks and to clarify the obscurity of "large" we provide a

quantitative criterion in Appendix A. Also, Theorem 1 can be used to determine network topology as shown in Appendix B. The completion time of a network is a function of network topology and resource allocation. The fact that the components in a network can be considered independent under proportional allocation, leads to developing an efficient method for solving the network topology problem.

5.3 Mathematical Programming Model

In this section, we build a mathematical programming model based on the optimal resource allocation policy by taking into account alternative algorithms. Then, we convert the programming model into an adaptive one by incorporating stress environment.

5.3.1 Non-adaptive Programming Model

As discussed, each component's optimal strategy is a pure strategy and the completion time T is close to T^{LB} under proportional resource allocation in the limit of large number of tasks. Now, consider current time as t. To update load index as the system moves on, we slightly modify it to represent the total CPU time for the remaining tasks as:

$$LI_i(t) = R_i(t) + L_i(t) f_i(v_i) , \qquad (\mathbf{5.10})$$

in which $R_i(t)$ denotes remaining CPU time for a task in process and $L_i(t)$ the number of remaining tasks excluding a task in process. After identifying initial number of tasks $L_i(0)=L_i$, each component updates it by counting down as they process tasks.

Then, under proportional resource allocation, the completion time T can be estimated as:

$$T - t \approx \underset{n \in N}{Max} \sum_{i \in K_n} [\, R_i(t) + L_i(t) f_i(v_i) \,] . \qquad (\mathbf{5.11})$$

The estimation leads to building a programming model in a straightforward way. Given completion time T it is optimal for a node n to select a mode by the following:

$$Max \sum_{i \in K_n} L_i(t) v_i \qquad (5.12)$$

subject to:

$$\sum_{i \in K_n} [R_i(t) + L_i(t) f_i(v_i)] \le T - t \,. \qquad (5.13)$$

Consequently, the programming model can be formulated with two sub-models: optimization model as in (**5.14**) and resource allocation model as in (**5.15**). The optimization model maximizes QoS by trading off between the value of solution and the cost of completion time, and the resource allocation model allocates resources proportional to the load indices of residing components based on the solution of (**5.14**). The optimal QoS from (**5.14**) with t=0 forms a QoS upper bound QoS^{UB} and a network can achieve a performance close to QoS^{UB} in the limit of large number of tasks. The programming model is efficient in terms of complexity because the two different kinds of control actions are completely separated.

- **Non-adaptive programming model**

$$\begin{aligned} & Max \sum_{i \in A} L_i(t) v_i - CCT(T) \\ & s.t. \quad \sum_{i \in K_n} [R_i(t) + L_i(t) f_i(v_i)] \le T - t \qquad for\ all\ n \in N \\ & \qquad v_{i(min)} \le v_i \le v_{i(max)} \qquad for\ all\ i \in A \end{aligned} \qquad (5.14)$$

$$w_i^* = \frac{R_i(t) + L_i(t) f_i(v_i^*)}{\sum_{p \in K_{n(i)}} [R_p(t) + L_p(t) f_p(v_p^*)]} \omega_{n(i)} \qquad (5.15)$$

5.3.2 Adaptive Programming Model

Each component monitors its operating environment through a sensor. The sensor measures resource availability $MRA_i(t)$, which is defined as available fraction of a resource when

a component *i* requests that resource in the last control period at control point *t*. There are two quantities to extract this measurement, which are request time and execution time. Request time is the duration for which the component requests resource or equivalently queue length (including a task in process) is more than zero. Execution time is the duration for which the component utilizes the resource. If control period is SW, resource sensor calculates $MRA_i(t)$ as:

$$MRA_i(t) = \frac{execution\ time\ in\ [t - SW, t]}{request\ time\ in\ [t - SW, t]}. \qquad (\mathbf{5.16})$$

Based on the measured resource availability a component extracts normalized resource availability $URA_i(t)$ which is the resource availability per weight in the last control period at decision point *t*. By denoting the weight of the last control period as w_i^p, $URA_i(t)$ is calculated as:

$$URA_i(t) = \frac{MRA_i(t)}{w_i^p}. \qquad (\mathbf{5.17})$$

The constraints of (**5.14**) and (**5.15**) can be converted into the constraints as in (**5.18**). By incorporating these constraints we can build an adaptive programming model as in (**5.19**) and (**5.20**). The first formulation aims at maximizing QoS by trading off between the value of solution and the cost of completion time based on the modeled stress environment. The second one reallocates resources proportional to the actual workload[6] because the components can have slack resources in the solution of (**5.19**).

$$\begin{aligned} w_i\, URA_i(t) = MRA_i(t) &\geq \frac{w_i}{\omega_{n(i)}} = \frac{R_i(t) + L_i(t) f_i(v_i)}{\sum_{p \in K_{n(i)}} [R_p(t) + L_p(t) f_p(v_p)]} \\ &\geq \frac{R_i(t) + L_i(t) f_i(v_i)}{T - t} \qquad (\mathbf{5.18}) \\ \Rightarrow \frac{R_i(t) + L_i(t) f_i(v_i)}{w_i\, URA_i(t)} &\leq T - t \qquad for\ all\ i \in A \end{aligned}$$

[6] Though it would be possible to reallocate resources without considering $URA_i(t)$, it makes it easier to decentralize the programming model as will be shown in Chapter 7.

- **Adaptive programming model**

$$
\begin{aligned}
&Max \sum_{i \in A} L_i(t) v_i - CCT(T) \\
&s.t. \quad \frac{R_i(t) + L_i(t) f_i(v_i)}{w_i URA_i(t)} \le T - t \qquad for\ all\ i \in A \\
&\qquad v_{i(min)} \le v_i \le v_{i(max)} \qquad for\ all\ i \in A \\
&\qquad \sum_{i \in K_n} w_i = \omega_n \qquad for\ all\ n \in N
\end{aligned}
\tag{5.19}
$$

$$
w_i^* = \frac{[R_i(t) + L_i(t) f_i(v_i^*)] / URA_i(t)}{\sum_{p \in K_{n(i)}} [R_p(t) + L_p(t) f_p(v_p^*)] / URA_p(t)} \omega_{n(i)} \tag{5.20}
$$

When there is no stress, $URA_i(t) \ge 1/\omega_{n(i)}$ for all $i \in A$. As the adaptive programming model is used repeatedly, $URA_i(t)s$ tend to decrease because of the efforts to utilize more resources. This leads to an equilibrium where all the components of at least one node measure $URA_i(t)=1/\omega_{n(i)}$ (equality of the constraints in (**5.14**)) and those with $URA_i(t)>1/\omega_{n(i)}$ (inequality of the constraint in (**5.14**)) use their maximal modes. So, in the equilibrium the optimal solution of the adaptive programming model satisfies the constraints of (**5.14**). Also, for any given T the value of solution from adaptive model cannot be less than the one from non-adaptive model because (**5.19**) has larger solution space than (**5.14**) (both solution spaces are equivalent when $URA_i(t)=1/\omega_{n(i)}$ for all $i \in A$). Therefore, both programming models give equivalent optimal solution in the equilibrium. However, optimal weights may be different. The bottleneck nodes with $URA_i(t)=1/\omega_{n(i)}$ will have equivalent weights to the non-adaptive model while others may not. Non-bottleneck nodes have slack resources and (**5.20**) will allocate resources depending on the components' observed resource availability. However, note that such a difference of allocating slack resources does not affect the performance. We will discuss on the convergence of the adaptive model in detail in Chapter 8. Despite the equivalence the adaptive programming model leads the system to make adaptive decisions to the changing stress environments.

Chapter 6

Model Refinement

In this chapter we analyze system behavior under the designed control mechanism and refine it to eliminate undesirable behavioral properties.

6.1 System Behavior under Adaptive Programming Model

To analyze the system behavior under the adaptive programming model, we made experimentation using a discrete-event simulator. There are five components in the system linked serially as in Figure **6.1**. Component A_1 in the lowest position is assigned 100 root tasks. Components have a common deterministic value function and the cost of completion time is a linear increasing function as indicated in Figure **6.1**. There is no stress in the system and components measure $MRA_i(t)$ equal to 1 all the time. The system makes decision every 100 time units (i.e., SW=100) by solving the adaptive programming model.

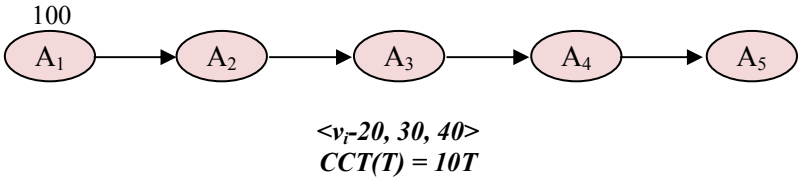


Figure **6.1**: An example network for behavior analysis

Figures **6.2** and **6.3** show the resultant behavior of the system, in which the decisions T^* and v_i^* are divergent. The divergent behavior indicates that there is inefficiency in the programming model and system performance can be improved if we eliminate this inefficiency.

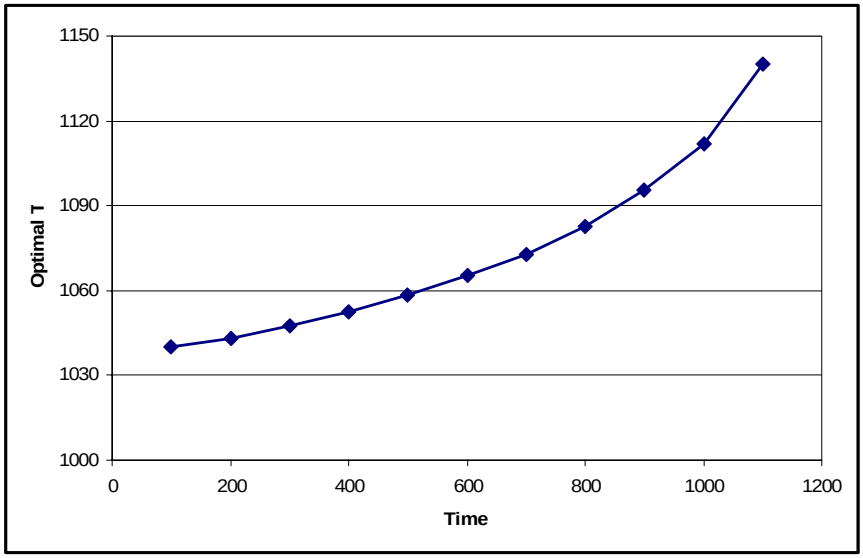


Figure **6.2**: Behavior of T^* under adaptive programming model

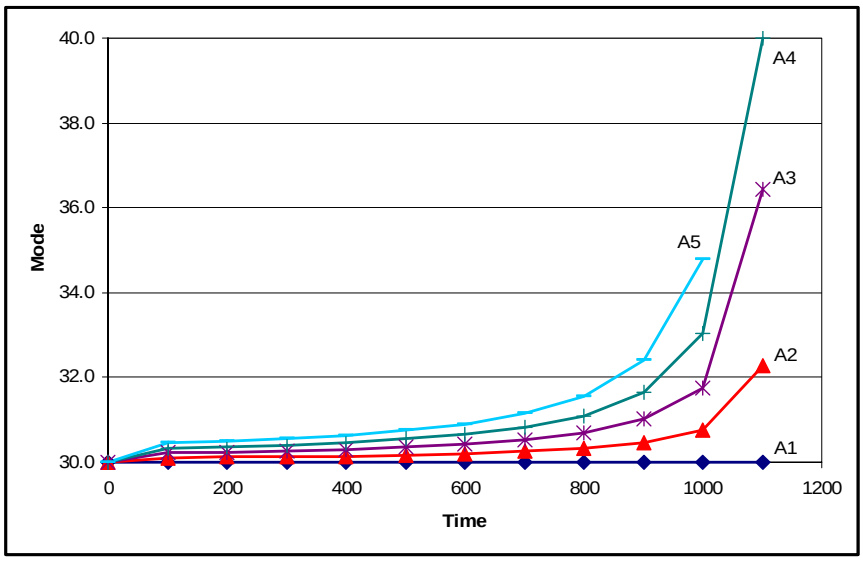


Figure **6.3**: Behavior of v_i^*s under adaptive programming model

The divergent behavior is due to the inaccurate prediction of the programming model. In the example network the system (or A_5) can complete at T^* when A_4 completes before T^*. As each component is trying to complete at T^* without considering its position in the task flow structure, the components cannot receive tasks in time from their predecessors. This inaccuracy leads to changing the decisions in the subsequent decision points resulting in the divergent behavior.

6.2 Model Refinement

To stabilize the system behavior we need to reinforce the adaptive programming model by taking into account the components' positions in the task flow structure. For this purpose, we define *Depth* $D_i(t)$ as a quantitative representation of the component's position. $D_i(t)$ is the required time gap between the system's and the component's completion times at time t. Each component needs to complete at less than or equal to $T\text{-}D_i(t)$ to keep the completion time T. Components without successors have depth equal to 0 but components with successors have positive depths. A component a can keep its depth if its predecessors' depth is $D_a(t)$ plus its total service time for the last arriving tasks in the worst case. So, a component i's depth to keep the depths of its all successors is the maximal of the required depths from its successors represented as:

$$D_i(t) = \max_{a \in \bar{i}} [\, D_a(t) + \sum_{b \in \underline{a}} f_a(v_a) / (w_i URA_a(t)) \,] , \qquad (\mathbf{6.1})$$

in which $\bar{i}$ denotes successors of component i and $\underline{a}$ predecessors of component a.

Though it would be possible to refine the adaptive programming model by incorporating the depths as variables, the model complexity increases because each component's constraint will be intertwined with the decision variables of all connected components. So, we simply estimate components' depths through the decisions used at the last control point. At each control point each successor informs its predecessors of required depths and each predecessor chooses the maximal one as its depth. As a result, we can consider the depth as constant rather than variable so that the refined model has no increase in complexity. We call the refined model in (**6.2**) and (**6.3**) as *stable adaptive programming model*. If we don't consider the depth, i.e., $D_i(t)$=0, the model becomes equivalent to the adaptive programming model.

- **Stable adaptive programming model**

$$
\begin{aligned}
&Max \sum_{i \in A} L_i(t) v_i - CCT(T) \\
&s.t. \quad \frac{R_i(t) + L_i(t) f_i(v_i)}{w_i URA_i(t)} \le T - t - D_i(t) \qquad for\ all\ i \in A \\
&\qquad v_{i(min)} \le v_i \le v_{i(max)} \qquad for\ all\ i \in A \\
&\qquad \sum_{i \in K_n} w_i = \omega_n \qquad for\ all\ n \in N
\end{aligned}
\tag{6.2}
$$

$$
w_i^* = \frac{[R_i(t) + L_i(t) f_i(v_i^*)] / URA_i(t)}{\sum_{p \in K_{n(i)}} [R_p(t) + L_p(t) f_p(v_p^*)] / URA_p(t)} \omega_{n(i)} \tag{6.3}
$$

6.3 System Behavior under Stable Adaptive Programming Model

We experimented using the stable adaptive programming model with the example network described in Figure **6.1**. Figures **6.4** and **6.5** show the resultant behavior of the system, in which the decisions T^* and v_i^*s behave stable. The stability indicates that the inefficiency of adaptive programming model is removed as a result of improved prediction accuracy.

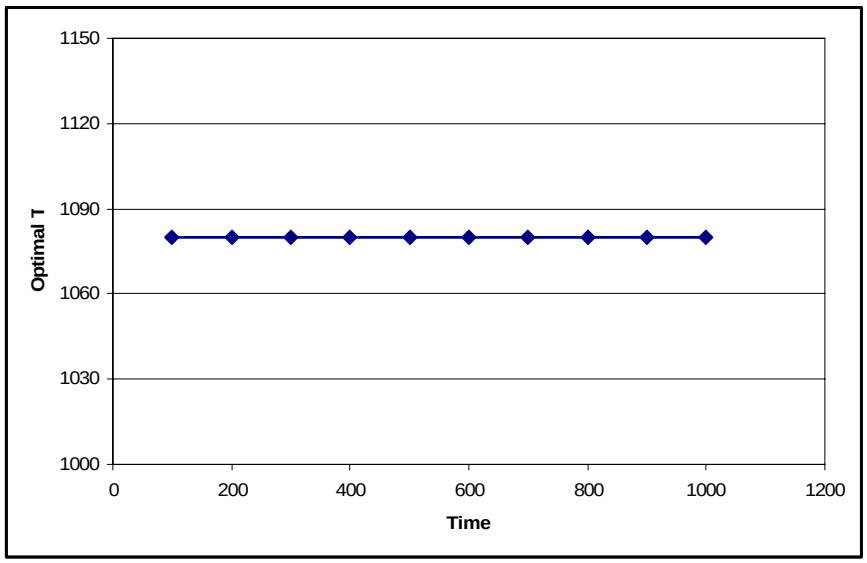


Figure **6.4**: Behavior of T^* under stable programming model

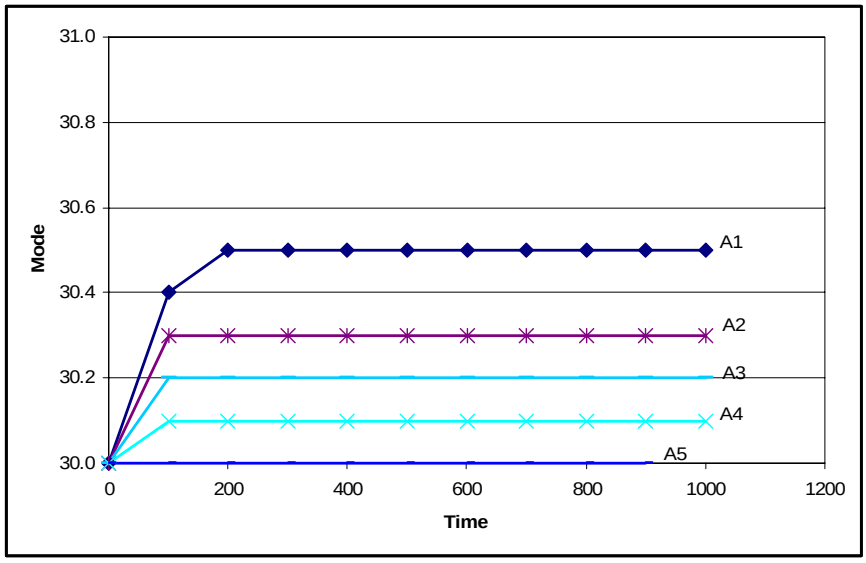


Figure **6.5**: Behavior of v_i^*s under stable programming model

The effects of the stability on performance are shown in Table **6.1**. QoS is improved significantly when using the stable programming model. Improved prediction accuracy made the system behaving stable and consequently performing better.

Table **6.1**: The effect of stability on performance

Programming model					
Adaptive			Stable		
T	*V*	*QoS*	*T*	*V*	*QoS*
1171	15289	3583	1082	15104	4282

T: Completion time
V: Value of solution

Chapter 7

Decentralized Coordination

The next question is how to decentralize the designed mathematical programming model. Centralized problem solving has problems with respect to scalability and robustness. Decentralization can address these issues by distributing the computation and communication overheads to multiple entities in the system. In addition to these properties, decentralization will give a byproduct, *information security*. As we discussed earlier our effort is to support survivability. If information is revealed to others directly information security will be in question. So, decentralization will also help survivability with respect to information security. In this chapter we decentralize the stable adaptive programming model through an auction market after briefly surveying decentralization algorithms in literature.

7.1 Decentralization Algorithms

One branch of distributed control approaches is that of decentralizing structured mathematical programming models. In this branch there are two popular approaches: decomposition methods and auction algorithms. However, note that the boundary between these two methods is not clear and we categorize them as they are called in literature.

Decomposition methods are mainly used for scalability when mathematical programming model becomes too large or too complicated. In decomposition methods, the original problem is decomposed into smaller sub-problems and they are coordinated by a master problem through the information exchange between them [**86**]. Decomposition methods for the problems with continuous variables have been proposed for different problem structures: Dantzig-Wolfe's decomposition for block angular structure [**87**] and primal partitioning decomposition for block diagonal structure [**108**][**109**]. For the problems with mixed-integer variables Benders' decomposition and generalized Benders' decomposition methods are proposed [**88**][**89**]. Decomposition methods decompose the original problem to sub-problems by constraint-partitioning or variable-partitioning. In Dantzig-Wolfe's decomposition method, which uses constraint-partitioning principle, each sub-problem optimizes its subsystem to determine the

amount of resource using its resource constraints given the price of the resource. The master problem changes the resource prices based on the offers from the sub-problems. In contrast, the other decomposition methods adopt variable-partitioning principle, in which each sub-problem optimizes its subsystem by fixing the values of the complicating variables given from the master problem. For example, in Benders decomposition method sub-problems solve LP or NLP problems by fixing integer part from master problem. The master problem solves all-integer problem by considering the primal and dual solutions of the sub-problems.

In auction algorithms agents solve problems by competitively maximizing their own utility. Those algorithms can have additional benefits in competitive environments with respect to incentive compatibility. An auction mechanism is incentive compatible if true revelation is the bidders' dominant strategy [**100**]. If the bidders in an auction market bid with true valuation, the auction market solves a mathematical programming model in a decentralized mode. So, when agents are self-interested behaving strategically, an incentive compatible mechanism such as Vickrey auction should be facilitated. In Vickrey auction, where the winner with the highest bid price pays only the second bid price, the bidders' dominant strategy is to bid with their true valuation [**91**]. If the problem is too complicated to solve in a single bid, it can be solved through an iterative process between participants. Such iterative algorithms have been proposed in several problem domains. Assignment problems can be solved optimally through an English auction algorithm which can be also applied to network flow and shortest path problems [**92**][**93**][**94**][**110**]. When interconnected markets are in a perfect balance of supply and demand with respect to utility-maximizing behaviors of self-interested agents, the economy is in general equilibrium where the solution is Pareto-optimal. The equilibrium can be reached through an iterative bidding process, so-called tatonnement process (or Walrasian auction). This process was applied to network flow, scheduling, and configuration design problems [**111**][**112**][**113**][**114**]. Also, Lagrangean relaxation can be used to decompose a (primal) problem to multiple Lagrangean dual sub-problems [**115**]. By iteratively changing prices (Lagrangean multipliers) based on the bids from sub-problems, the primal problem can be (approximately) solved. This approach, so called Lagrangean auction, was applied to plant and supply chain planning problems [**116**][**117**].

7.2 Two-tier Auction Market Design

Considering the compatible structure of the programming model, we decentralize it through a non-iterative auction mechanism[7], so called *multiple-unit auction with variable supp*ly [**28**]. In the programming model we have built, all nodes and components are coupled with each other. However, it has a typical structure, where objective function and constraints are *separable* to each node if one variable T is fixed. This characteristic makes it possible to solve the model through an auctioning process for T. The completion time T is an unbounded resource and the supply can be adjusted as a function of bidding.

To design the auction market we define two different types of participants in addition to the components: Seller and Resource Manager. There is one seller in the system which determines T^* based on the bids from resource managers. A resource manager of each node manages the resource of the node and arbitrates between its components and the seller. The overall structure of the auction market is described in Figure **7.1**. We call this auction market as *Two-tier auctioning model*.

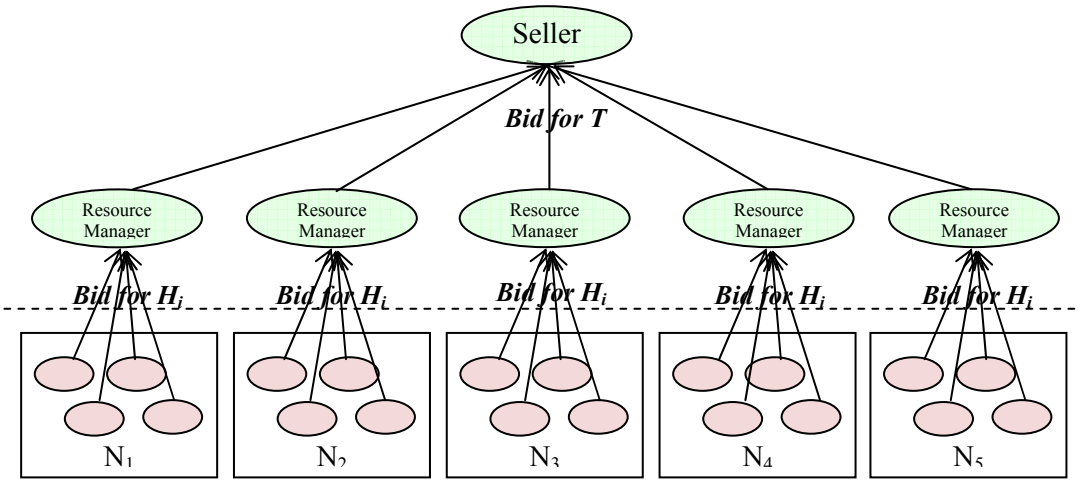


Figure **7.1**: Two-tier auctioning model

[7] We don't consider incentive compatibility assuming that all the agents bid with their true valuations. But, through using auction mechanism there is a good chance to extend the mechanism to have the property referring to the rich related literature.

For simplicity we define a hybrid resource that combines $T\text{-}t\text{-}D_i(t)$ and w_i as:

$$H_i = [T - t - D_i(t)]w_i \,. \tag{7.1}$$

We define H_i as available resource of component i which is required minimally to the amount of $H_{i(min)}$ as in (**7.2**) and maximally $H_{i(max)}$ as in (**7.3**).

$$H_{i(min)} = [R_i(t) + L_i(t) f_i(v_{i(min)})] / URA_i(t) \tag{7.2}$$

$$H_{i(max)} = [R_i(t) + L_i(t) f_i(v_{i(max)})] / URA_i(t) \tag{7.3}$$

Using the hybrid resource definition the programming model becomes:

$$\begin{aligned} Max \sum_{i \in A} & L_i(t) v_i - CCT(T) \\ s.t. \quad & [R_i(t) + L_i(t) f_i(v_i)] / URA_i(t) \le H_i \qquad for\ all\ i \in A \\ & v_{i(min)} \le v_i \le v_{i(max)} \qquad for\ all\ i \in A \\ & \sum_{i \in K_n} H_i / [T - t - D_i(t)] = \omega_n \qquad for\ all\ n \in N \end{aligned} \tag{7.4}$$

$$w_i^* = \frac{max(H_i^*, H_{i(max)})}{\sum_{p \in K_{n(i)}} max(H_p^*, H_{p(max)})} \omega_{n(i)} \tag{7.5}$$

Then, the programming model can be solved through an auctioning process as follows. A component i bids to its resource manager with maximal value as a function of H_i as in (**7.6**). The resource manager bids to the seller with maximal total value of its components as a function of T based on the bids from its components as in (**7.7**). The seller decides T^* based on the bids from resource managers by taking into account the cost of T as in (**7.8**). After the seller broadcasts T^*, each resource manager decides H_i^* and w_i^* as in (**7.9**) and (**7.10**). Finally, each component selects optimal value mode in the limit of H_i^* as in (**7.11**). Though this auctioning process gives an equivalent solution to the programming model, it gives more benefits as computations and communications are distributed to multiple market participants.

- **Two-tier auctioning model**

Component's bid

$$\begin{aligned} b_i(H_i) &= -\infty && \text{if } H_i < H_{i(min)} \\ &= L_i(t)v_{i(max)} && \text{if } H_i > H_{i(max)} \\ &= L_i(t) f_i^{-1}\left(\frac{(H_i URA_i(t) - R_i(t)}{L_i(t)}\right) && \text{else} \end{aligned} \quad \textbf{(7.6)}$$

Resource manager's bid

$$b_n(T) = Max\{\sum_{i\in K_n} b_i(H_i) : \sum_{i\in K_n} H_i /[T - t - D_i(t)] = \omega_n\} \quad \textbf{(7.7)}$$

Seller's decision

$$T^* = \underset{T}{\text{argmax}} \sum_{n\in N} b_n(T) - CCT(T) \quad \textbf{(7.8)}$$

Resource manager's decision

$$\{H_i^* : i \in K_n\} = \underset{\{H_i : i\in K_n\}}{\text{argmax}} \{\sum_{i\in K_n} b_i(H_i) : \sum_{i\in K_n} H_i /[T^* - t - D_i(t)] = \omega_n\} \quad \textbf{(7.9)}$$

$$w_i^* = \frac{max(H_i^*, H_{i(max)})}{\sum_{p\in K_{n(i)}} max(H_p^*, H_{p(max)})}\omega_{n(i)} \quad \textbf{(7.10)}$$

Component's decision

$$\begin{aligned} v_i^* &= L_i(t)v_{i(max)} && \text{if } H_i^* > H_{i(max)} \\ &= f_i^{-1}\left(\frac{(H_i^* URA_i(t) - R_i(t)}{L_i(t)}\right) && \text{else} \end{aligned} \quad \textbf{(7.11)}$$

7.3 Multi-tier Auction Market Design

Though the designed auction market is decentralized it incorporates a centralized seller. The seller needs to coordinate all the resource managers. As the centralized auction can still exhibit problems in terms of scalability and robustness we introduce a multi-tier auction market through Theorem 2.

Theorem 2. *Suppose there are two node groups a and b with $a \subset b$, and denote S_a as a set of optimal completion time solutions of group a and S_b of group b. Then, the maximal of S_b is greater than or equal to the maximal of S_a as in* (**7.12**).

$$max\, S_a \le max\, S_b \quad if\ a \subset b \tag{7.12}$$

Proof. Suppose it is not true, that is, $T^a = max\, S_a > T^b = max\, S_b$. Then, for group b,

$$\begin{aligned}
&\sum_{i\in b} b_i(T^b) - CCT(T^b) > \sum_{i\in b} b_i(T^a) - CCT(T^a) \\
&\equiv \sum_{i\in a} b_i(T^b) + \sum_{i\notin a} b_i(T^b) - CCT(T^b) \\
&\quad > \sum_{i\in a} b_i(T^a) + \sum_{i\notin a} b_i(T^a) - CCT(T^a) \\
&\equiv [\sum_{i\in a} b_i(T^b) - CCT(T^b)] - [\sum_{i\in a} b_i(T^a) - CCT(T^a)] \\
&\quad > \sum_{i\notin a} b_i(T^a) - \sum_{i\notin a} b_i(T^b)
\end{aligned} \tag{7.13}$$

.

And, for group a,

$$\sum_{i\in a} b_i(T^b) - CCT(T^b) \le \sum_{i\in a} b_i(T^a) - CCT(T^a)\ . \tag{7.14}$$

So, the inequality in (**7.15**) should hold.

$$\sum_{i\notin a} b_i(T^a) < \sum_{i\notin a} b_i(T^b) \tag{7.15}$$

However, this inequality is not possible because $b_i(T)$ is an increasing function with T.

Through this property the two-tier auctioning model can be transformed into a multi-tier auctioning model, in which there are multiple brokers arbitrating resource managers and the seller as shown in Figure **7.2**. A resource manager or broker bids to its superior for $T \geq T^{s(m)}$ as in (**7.16**) and (**7.17**), in which $T^{s(m)}$ denotes the optimal completion time of $s(m)$ which are the subordinates of resource manager or broker m. In this way, the search space becomes reduced as the bidding process goes to the superior. In this multi-tier auctioning model computations and communications are more distributed through the brokers overcoming the problems of the two-tier auctioning model.

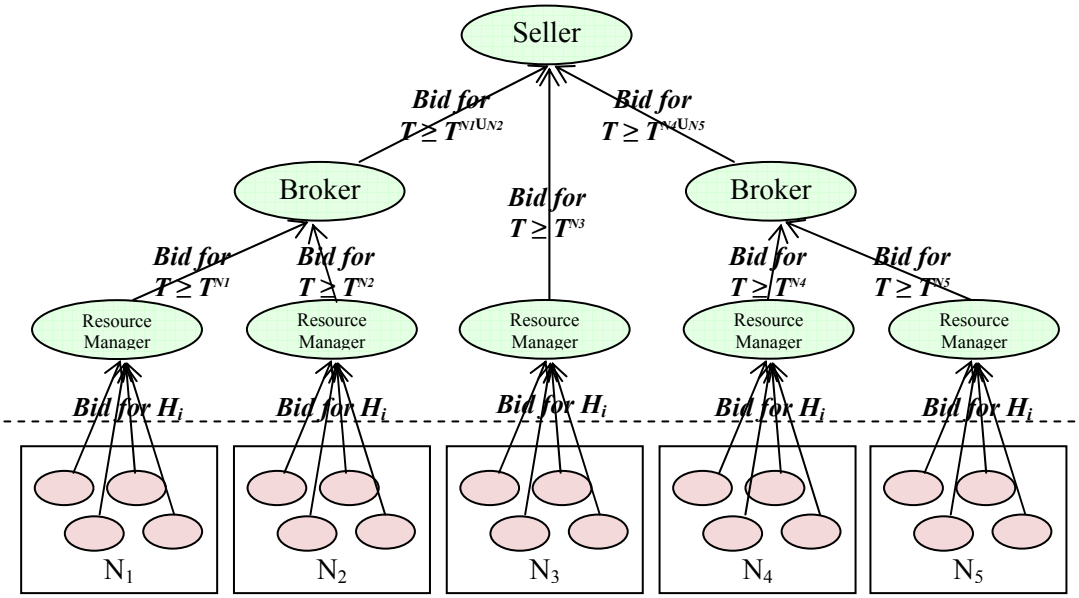


Figure **7.2**: Multi-tier auctioning model

- **Multi-tier auctioning model**

Resource manager's bid

$$b_n(T) = -\infty \quad if\ T < \arg\max_T \{ \sum_{i \in K_n} b_i(H_i) - CCT(T) : \sum_{i \in K_n} H_i / [T - t - D_i(t)] = \omega_n \}$$
$$= Max \{ \sum_{i \in K_n} b_i(H_i) : \sum_{i \in K_n} H_i / [T - t - D_i(t)] = \omega_n \} \quad else \qquad \textbf{(7.16)}$$

Broker's bid

$$b_m(T) = -\infty \qquad if\ T < \arg\max_T \sum_{a \in s(m)} b_a(T) - CCT(T)$$
$$= \sum_{a \in s(m)} b_a(T) \qquad else \qquad \textbf{(7.17)}$$

Chapter 8

Computational Ecosystem Model

Though decentralized control is an inevitable choice in controlling large-scale networks, it may be undesirable from a stability point of view. As individual agents make decisions with local perspectives, their expectations for the future may not come true due to collective action. Consequently, the system can behave unstably even though their operating environment is static and the instability can degrade the performance in general. So, it is important to provide MAS with appropriate coordination mechanism that ensures the stability property.

In this chapter, we design a novel computational ecosystem model characterized by resource sharing and algorithm selection, and study collective behavior of the model. A set of agents share a computational CPU resource to process their tasks. They contend for the resource trying to choose value-maximizing algorithms based on their observed resource availability, in a social way through an auction market. As excess resource for some of agents can be utilized by others, the system can exhibit interesting collective dynamics. As a result of increasing sharing of resources both within and across applications, the designed model is worthwhile to investigate in depth. In specific, we study how fairness of resource allocation affects stability and consequently optimality. This study concludes that the fairness, allocating resources proportional to the required resources, is a leading factor to stability and consequently optimality.

The designed computational ecosystem model is a simplified version of the networks we study under the designed control mechanism, which captures main characteristics with respect to the interactions from resource sharing. The conclusion from the study of the computational ecosystem model implies that proportional resource allocation will lead the system stable and optimal. Fortunately, we use the proportional resource allocation policy in the designed control mechanism without any discrepancy. It would be problematic if we did not utilize the proportional resource allocation in controlling the networks.

8.1 Related Researches

As resources cannot be ever affluent, collective dynamics of MAS with resource contention is a crucial area to study for efficiently allocating resources. Computational ecosystem model was introduced to study the collective behavior of computational agents in which individual agents choose among available resources with local perspectives on the basis of payoffs depending on the fraction of agents using the resources [**31**][**32**][**33**][**34**][**35**]. It was found that the resulting behavior ranges from simple fixed points to nonlinear oscillations and chaos with performance degradation of the system. Diversity of agents in this model, which can be implemented by diverse payoff functions or diverse methods to estimate the current state of the system, was defined as a leading factor to stabilize the system behavior because diversity makes the agents to have different preferences. As the level of diversity depends on the situations the right level can be found through a learning-based scheme. The collective behavior was also described as a distributed root-finding algorithm [**90**][**95**]. Load balancing problems in computer or communication networks can be considered as practical examples of this model.

Another models driven by dynamic equations was studied [**96**][**97**][**98**][**99**]. Agents can choose the coefficients of the dynamic equations and the system dynamics varies depending on the selected coefficients. Q-learning was used to choose the coefficients that stabilize the system dynamics.

8.2 Model Statement

In this section we formally describe the overall structure of a computational ecosystem model as a framework to study collective behavior of MAS characterized by resource sharing and algorithm selection. Detail decision processes and resultant behaviors follow in the subsequent sections.

The model is a simplified version of the networks we study focusing on a node rather than whole network. There are three types of agents in the model as shown in Figure **8.1**: customer agents A, supplier agents S, and a resource manager. Suppliers have finite number of root tasks initially and send them to corresponding customers after processing. Customers process tasks by sharing a CPU resource with others. Customers and suppliers have alternative algorithms to process a task and their solution values depend on the algorithms used to process tasks. A

customer tries to maximize its own value primarily and its supplier's secondarily. The system completes when all customers complete the tasks from suppliers and the customers are responsible for the cost of resource which is a function of the completion time. Resource manager makes resource decisions such as the completion time and resource allocation by coordinating customers.

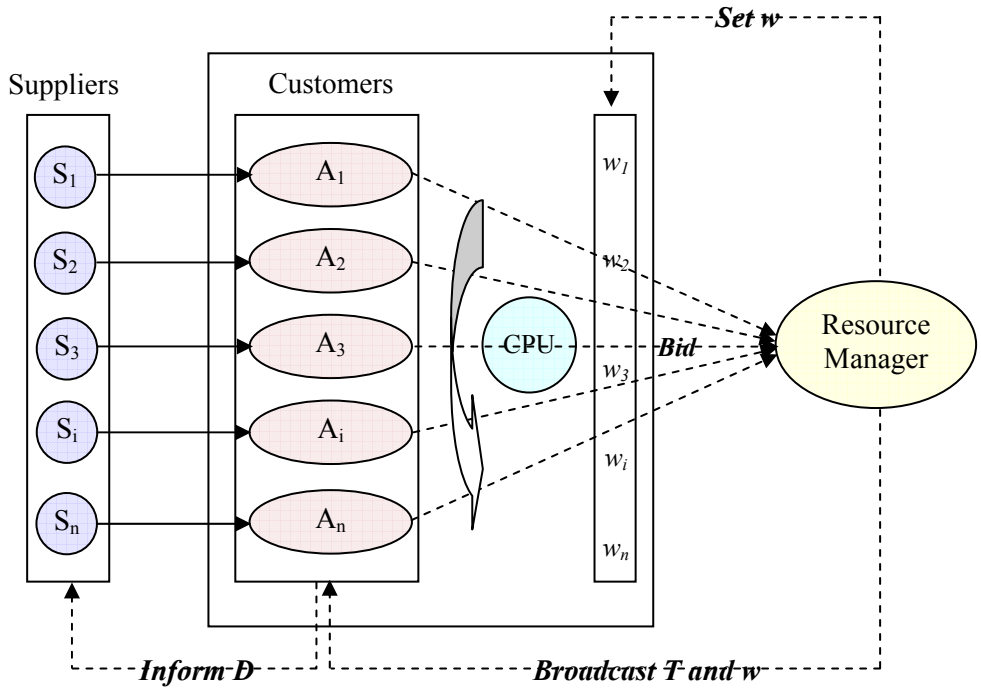


Figure **8.1**: The architecture of a computational ecosystem model

Periodically, decisions are made through an auction market. Each customer monitors its operating environment and makes bid for resource. Based on bids resource manager decides the completion time T and resource allocation $\boldsymbol{w}$ taking into account the cost of completion time. After resource manager broadcasts the resource decisions, customers select value-maximizing algorithms in the limit of resource constraints and inform suppliers of due dates $\boldsymbol{DD}$ for task generation. Then, suppliers select value-maximizing algorithms in the limit of due dates. To simplify the model we assume that suppliers have infinite value modes so that they can keep the due dates from customers.

We describe each agent's decision process in following subsections, similar to the auctioning model designed in Chapter 7. We analyze varying behavior of system depending on resource allocation policy of resource manager. In naïve ecosystem model resource manager

determines only completion time T fixing resource allocation $\boldsymbol{w}$ over time while in the subsequent model resource allocation also.

8.3 Naïve Ecosystem Model

In naïve ecosystem model resource manager decides only completion time fixing resource allocation over time. There can be many reasons of fixing resource allocation. Resource manager may just follow predetermined scheduling algorithms such as round-robin scheduling or resource may be reserved beforehand. In this section we model agents' rational decision process in this context and analyze system behavior.

8.3.1 Decision Process of Naïve Ecosystem Model

As resource allocation is fixed over time, each customer bids only for completion time based on measured resource availability without taking resource allocation into account. Given measured resource availability $MRA_i(t)$, T is required minimally to the amount of $T_{i(min)}$ as in (**8.1**) and maximally $T_{i(max)}$ as in (**8.2**).

$$T_{i(min)} = [\, R_i(t) + L_i(t) f_i(v_{i(min)})] \,/\, MRA_i(t) + t \qquad (\mathbf{8.1})$$

$$T_{i(max)} = [\, R_i(t) + L_i(t) f_i(v_{i(max)})] \,/\, MRA_i(t) + t \qquad (\mathbf{8.2})$$

A customer i bids to the resource manager with maximal value as a function of T as in (**8.3**). A customer's optimal mode is a pure mode common to all tasks because of the convexity of value function. The seller decides T^* based on the bids from the customers by taking into account the cost of T as in (**8.4**). After the resource manager broadcasts T^*, each customer decides v_i^* in the limit of T^* as in (**8.5**). As it is beneficial for suppliers to have longer due dates, customers inform suppliers of maximal due dates as long as they can complete by T^*. The optimal due date DD_i^* of customer i's supplier is equal to T^* minus the supplier's depth as in (**8.6**). After the suppliers are informed their due dates, they decide their optimal value modes in the limit of DD_i^*. We represent supplier's decision not by value mode but by task generation interval because supplier's value is not main concern of the model. As a supplier's optimal mode is a pure mode,

optimal task generation interval G_i^* of customer i's supplier is as in (**8.7**). We denote $L_i^s(t)$ as the number of tasks and $R_i^s(t)$ as remaining time for a task in process of customer i's supplier at time t.

- **Naïve ecosystem model**

Customer's bid

$$\begin{aligned} b_i(T) &= -\infty && \text{if } T < T_{i(min)} \\ &= L_i(t)v_{i(max)} && \text{if } T > T_{i(max)} \\ &= L_i(t) f_i^{-1}\left(\frac{(T-t)MRA_i(t) - R_i(t)}{L_i(t)}\right) && \text{else} \end{aligned} \quad \textbf{(8.3)}$$

Resource manager's decision

$$T^* = \arg\max_T \sum_{i \in A} b_i(T) - CCT(T) \quad \textbf{(8.4)}$$

Customer's decision

$$\begin{aligned} v_i^* &= L_i(t)v_{i(max)} && \text{if } T^* > T_{i(max)} \\ &= f_i^{-1}\left(\frac{(T^*-t)MRA_i(t) - R_i(t)}{L_i(t)}\right) && \text{else} \end{aligned} \quad \textbf{(8.5)}$$

$$DD_i^* = T^* - [\, f_i(v_i^*) / MRA_i \,] \quad \textbf{(8.6)}$$

Supplier's decision

$$G_i^* = (DD_i^* - R_i^s(t) - t) / L_i^s(t) \quad \textbf{(8.7)}$$

8.3.2 Analysis

To observe system behavior of naïve ecosystem model we made experimentation using a computational ecosystem simulator. There are three customers and three suppliers in the system

as in Figure **8.2**. Suppliers are assigned different number of root tasks, customers have a common value function, and the cost of completion time is a linear increasing function, as indicated in the figure. The distributions of CPU time in the value functions are considered as deterministic. In this experimentation w_i equals to 0.1 for all customers over time with ω=0.3. The customers' initial modes are (3, 3, 3) and the suppliers' initial task generation intervals are (40, 30, 30). The auction market opens every 200 time units.

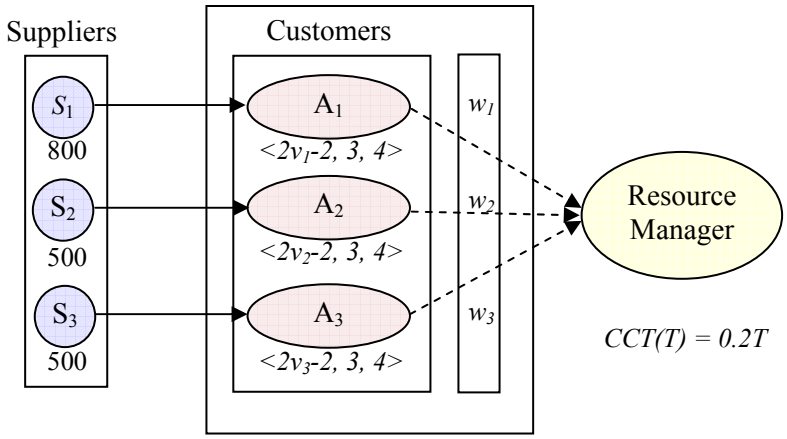


Figure **8.2**: Experimental ecosystem configuration

Figures **8.3** to **8.5** shows the resultant behavior of the system, in which all decision variables, T^*, v_i^*s, and G_i^*s, are oscillating. Changing decisions from time to time indicates that there is inefficiency in the naïve ecosystem model that we need to identify to stabilize the system behavior.

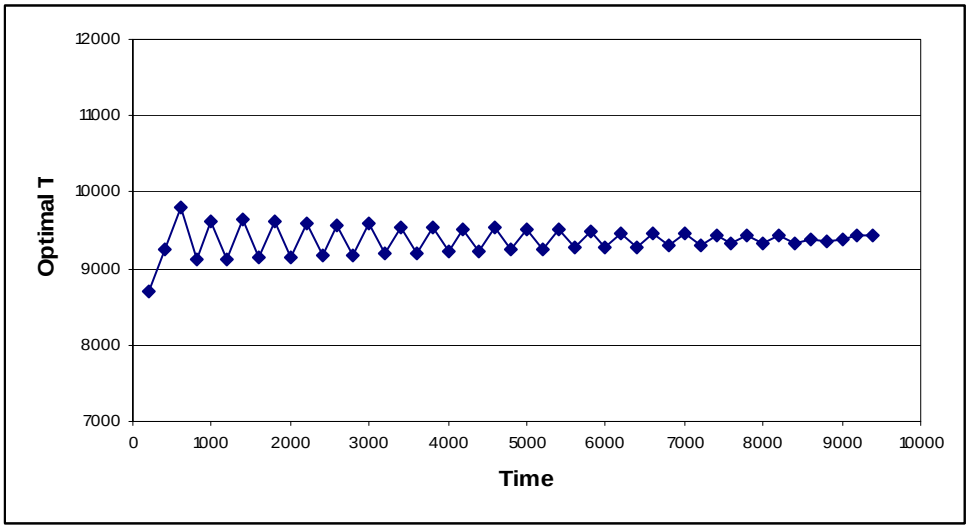


Figure **8.3**: Behavior of T^* under naïve ecosystem model

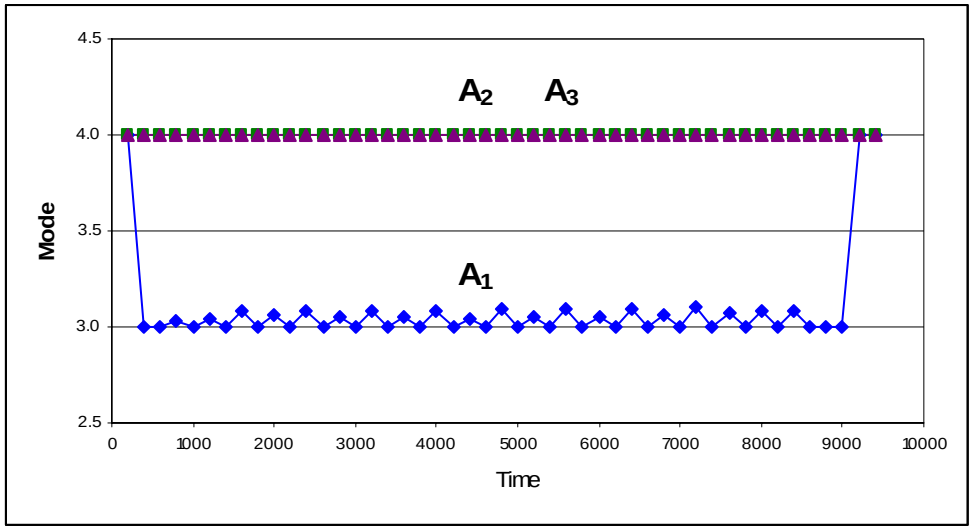


Figure **8.4**: Behavior of v_i^*s under naïve ecosystem model

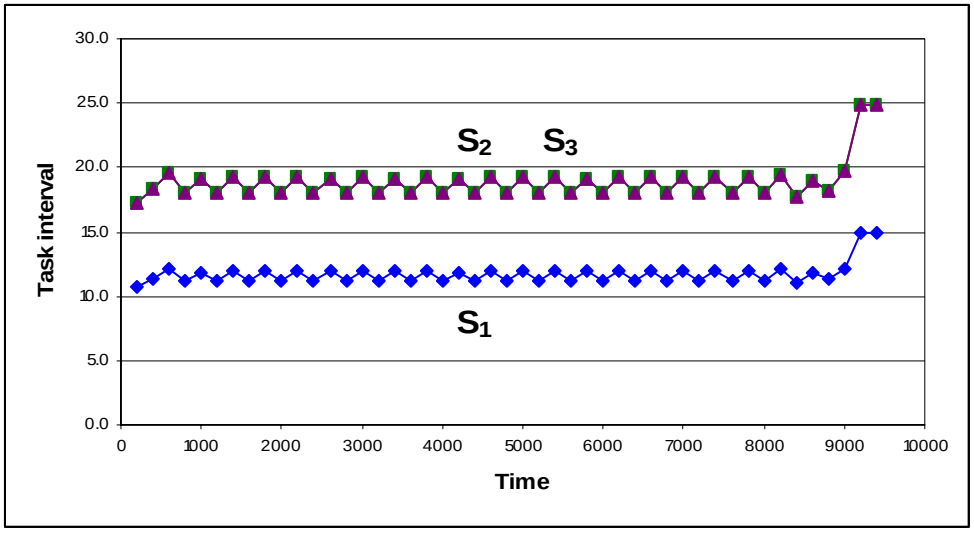


Figure **8.5**: Behavior of G_i^*s under naïve ecosystem model

To conjecture the source of instability we trace measured resource availability on which the decisions are based. The behavior of $MRA_i(t)$ is shown in Figure **8.6**. One of customers, A_1, observes oscillating resource availability and as noticed the oscillation of decision variables corresponds to that. The oscillation is driven from the interaction between customers from resource sharing. If they have independent resources there is no reason for the oscillation as they can observe consistent resource availability equal to one. As customers and suppliers change their value modes simultaneously, a customer observes different resource availability in the next period as a result of the change of resource request patterns of others. This process repeats resulting in oscillating behavior.

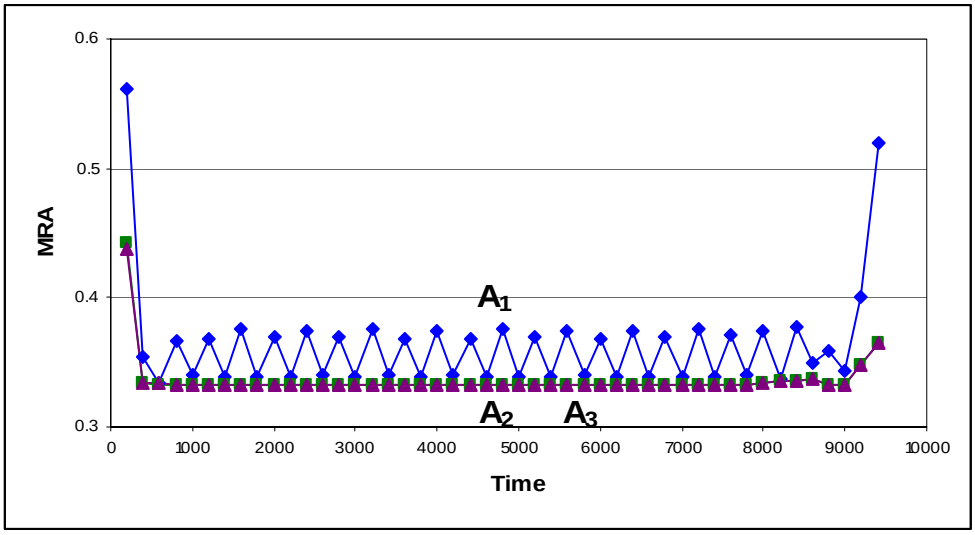


Figure **8.6**: Behavior of $MRA_i(t)$s under naïve ecosystem model

Under a hypothetical weighted round-robin server with idealized fairness, customers will measure resource availability such that:

$$MRA_i(t) \geq \frac{w_i}{\omega} \quad for\ all\ i\ . \tag{8.8}$$

In the experimentation lower bound of observed resource availability corresponds to 0.1/0.3=1/3 for all customers. Though it is not exact due to impossibility to implement the hypothetical weighted round-robin server, each customer observes approximately more than or equal to 1/3. When some of customers do not utilize their allocated resources the remaining customers utilize the excess resources and observe more than 1/3. As shown in Figure **8.7**, A_2 and A_3 have empty queues frequently indicating that they have excess resources. As a result, A_1 tend to observe resource availability more than 1/3 triggering the oscillation of decision variables.

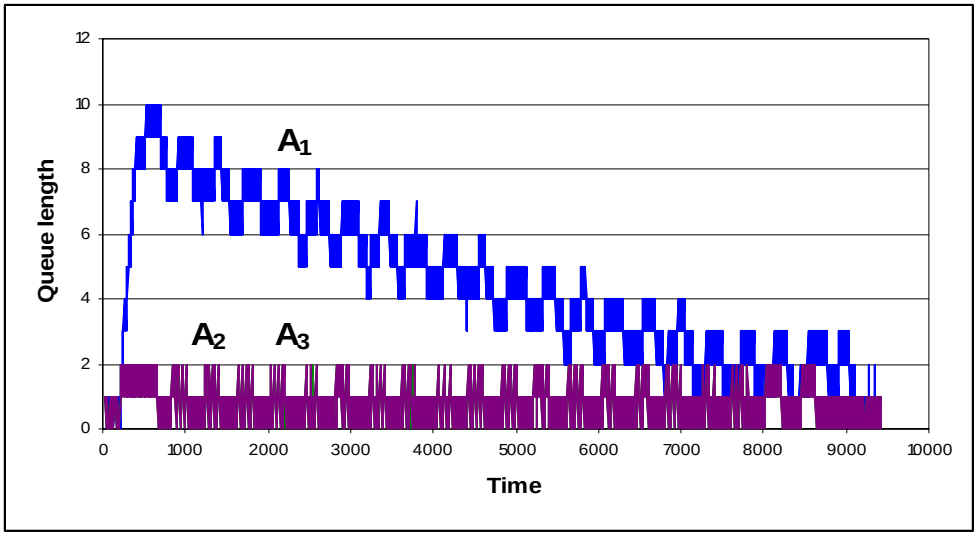


Figure **8.7**: Behavior of queue length under naïve ecosystem model

The source of instability is excess resource caused by the lack of flexibility in allocating resource. A customer's resource can become affluent when T^* is determined to be more than required with maximal value mode. That is, it is possible for a customer to have allocated resource $(T^*-t)w_i/\omega$ more than required resource $R_i(t)+L_i(t)f_i(v_i^*)$ under naïve ecosystem model as in (**8.9**).

$$R_i(t)+L_i(t)f_i(v_i^*)\le(T^*-t)MRA_i(t)\ge\frac{w_i}{\omega}(T^*-t) \qquad \textbf{(8.9)}$$

Now, the next question is how to stabilize the system behavior. To stabilize the system behavior it is essential to stabilize the resource availability and it can be achieved when each customer keeps its queue non-empty over time. As empty queue is caused by excess resource we need to eliminate the excess resource. In the next section we modify the model simply by taking resource allocation into account and analyze the resultant system behavior.

8.4 Stable Ecosystem Model

In this section we change the resource manager's resource allocation policy. The resource manager determines not only completion time T but also resource allocation $\boldsymbol{w}$. We model agents'

rational decision process in this context, forming so-called stable ecosystem model, and analyze the resultant system behavior.

8.4.1 Decision Process of Stable Ecosystem Model

As customers know that resource manager decides not only T but also $\boldsymbol{w}$, they make bids by taking resource allocation into account. The decision process is similar to the auctioning model described in Chapter 7. A customer i bids to the resource manager with maximal value as a function of H_i as in (**8.10**). The resource manager decides T^*, H_i^* and w_i^* based on the bids from the customers by taking into account the cost of T as in (**8.11**) and (**8.12**). After the resource manager broadcasts the resource decisions, each customer decides their modes v_i^* and due date DD_i^* in the limit of available resources as in (**8.13**) and (**8.14**). After the suppliers are informed their due dates, they decide their optimal task generation interval G_i^* in the limit of DD_i^* as in (**8.15**).

- **Stable ecosystem model**

Customer's bid

$$\begin{aligned} b_i(H_i) &= -\infty && \text{if } H_i < H_{i(min)} \\ &= L_i(t)v_{i(max)} && \text{if } H_i > H_{i(max)} \\ &= L_i(t) f_i^{-1}\left(\frac{H_i URA_i(t) - R_i(t)}{L_i(t)}\right) && \text{else} \end{aligned} \qquad \textbf{(8.10)}$$

Resource manager's decision

$$T^*, \{H_i^* : i \in A\} = \underset{T, \{H_i : i \in A\}}{\arg\max} \{ \sum_{i \in A} b_i(H_i) - CCT(T) : \sum_{i \in A} H_i / (T - t) = \omega \} \qquad \textbf{(8.11)}$$

$$w_i^* = \frac{H_i^*}{T^* - t} \qquad \textbf{(8.12)}$$

Customer's decision

$$v_i^* = v_{i(max)} \qquad if\ H_i^* > H_{i(max)}$$
$$= f_i^{-1}\left(\frac{H_i^* URA_i(t) - R_i(t)}{L_i(t)}\right) \qquad else \tag{8.13}$$

$$DD_i^* = T^* - f_i(v_i^*)/(w_i^* URA_i(t)) \tag{8.14}$$

Supplier's decision

$$G_i^* = (DD_i^* - R_i^s(t) - t)/L_i^s(t) \tag{8.15}$$

8.4.2 Analysis

As discussed, the basis to stabilize the system behavior is the elimination of excess resource. In naïve ecosystem model, it is possible for some customers to have allocated resources more than the required resources. However, in stable ecosystem model, there cannot be any excess resource for any customer. First of all, resource decision H_i^* cannot be more than required with maximal value mode for any customer because otherwise there can be better solution with less completion time. That is:

$$[R_i(t) + L_i(t) f_i(v_i^*)] / URA_i(t) = H_i^* \quad for\ all\ i\,. \tag{8.16}$$

Consequently, each customer's required resource becomes more than or equal to allocated resource for any observed $URA_i(t)$ ($\geq 1/\omega$) as in (**8.17**).

$$R_i(t) + L_i(t) f_i(v_i^*) = H_i^* URA_i(t)$$
$$= w_i^*(T^* - t) URA_i(t) \geq \frac{w_i^*}{\omega}(T^* - t) \quad for\ all\ i \tag{8.17}$$

As there is no excess resource, all customers become having non-empty queues over time and measure a common $URA_i(t)$ equal to $1/\omega$. With the same experimental setting as in previous section, we made experimentation with resource allocation incorporated in decision process. Customers are observing a common $URA_i(t)$ approximately equal to $1/\omega$=1/0.3 as shown in Figure **8.8** as all of them have non-empty queues over time as shown in Figure **8.9**.

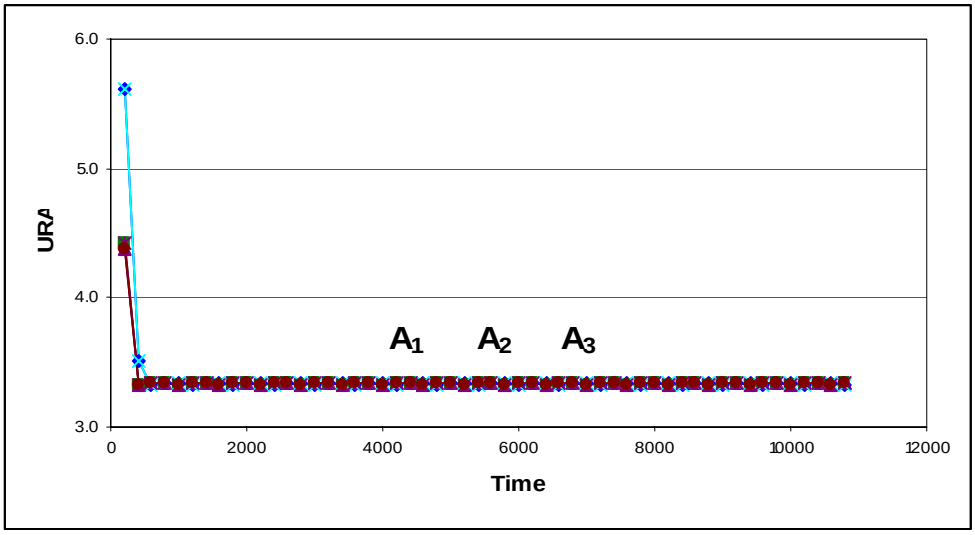


Figure **8.8**: Behavior of $URA_i(t)$s under stable ecosystem model

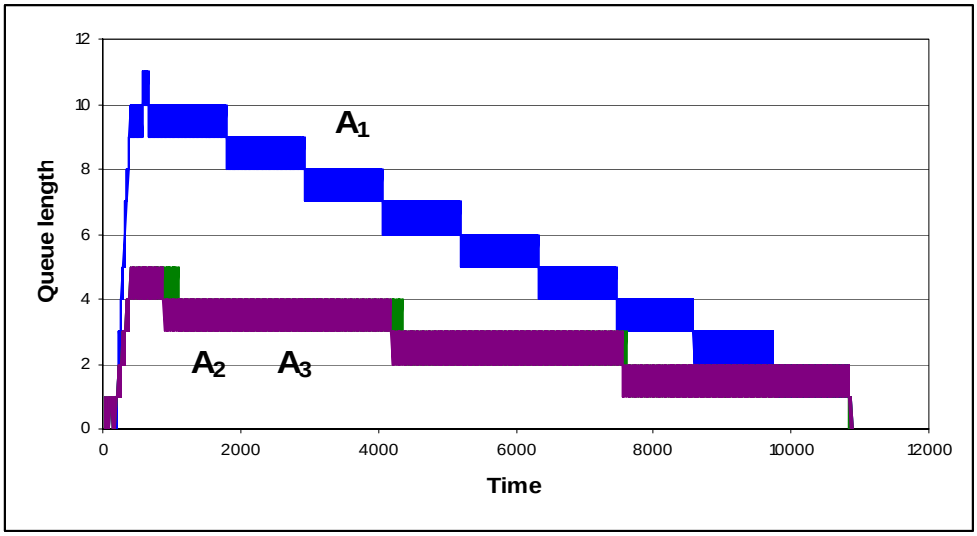


Figure **8.9**: Behavior of queue length under stable ecosystem model

Though $URA_i(t)$s are stable it is possible for decision variables to oscillate if the decisions are non-optimal. But, all decision variables are stable as shown in Figures **8.10** to **8.12**. It is because the decisions are optimal once all customers observe $URA_i(t)$ equal to $1/\omega$.

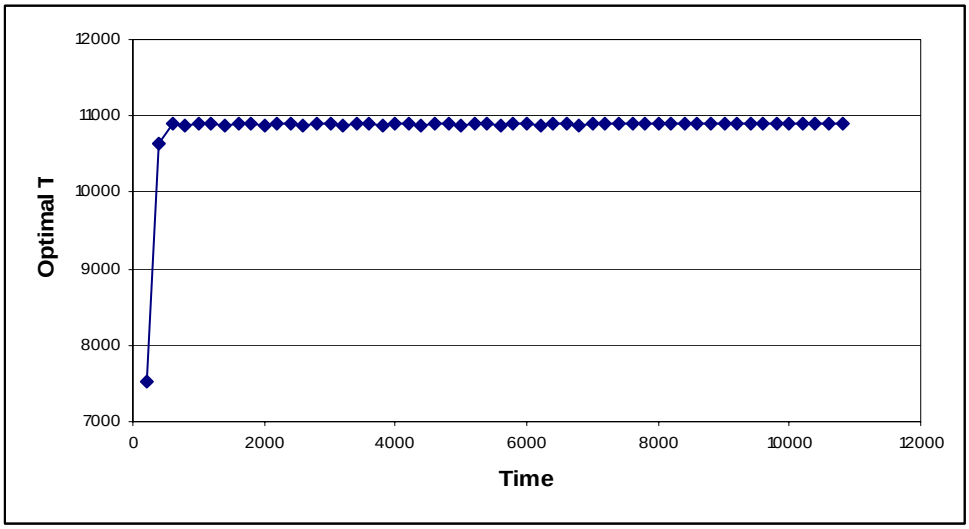


Figure **8.10**: Behavior of T^* under stable ecosystem model

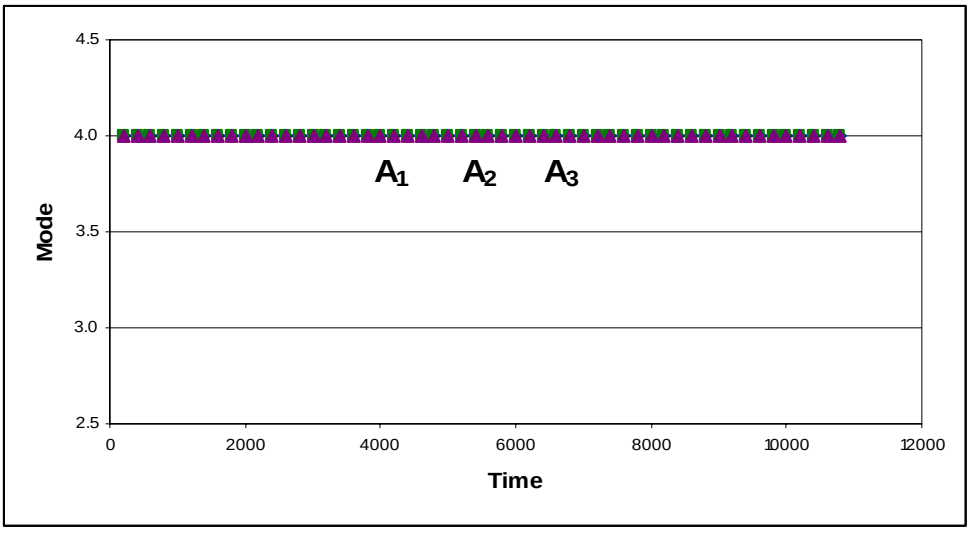


Figure **8.11**: Behavior of v_i^*s under stable ecosystem model

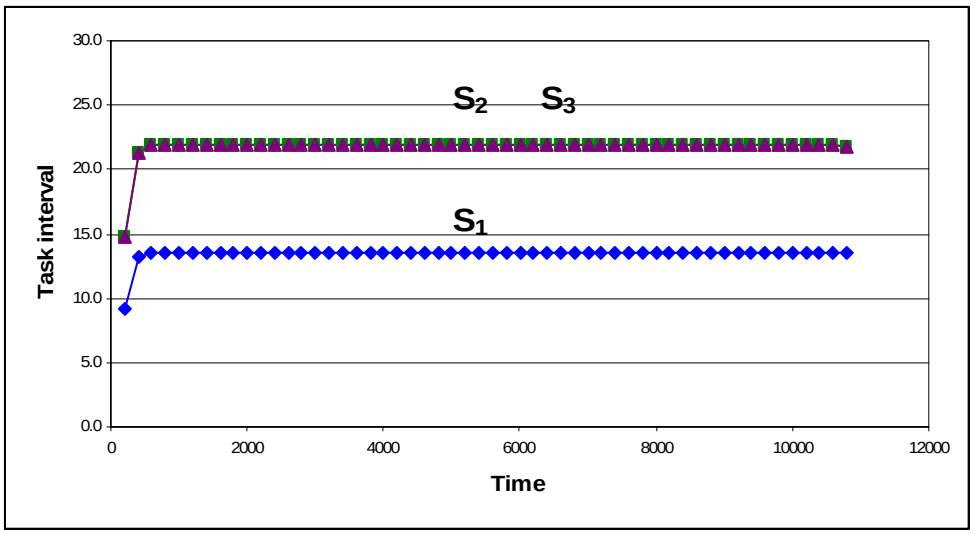


Figure **8.12**: Behavior of G_i^*s under stable ecosystem model

In a centralized point of view, the optimal decision of the node can be formulated as an optimization problem given by:

$$\begin{aligned} &Max \sum_{i \in A} L_i(t) v_i - CCT(T) \\ &s.t. \quad \sum_{i \in A} R_i(t) + L_i(t) f_i(v_i) = T - t \\ &\qquad v_{i(min)} \le v_i \le v_{i(max)} \qquad\qquad for\ all\ i \in A \end{aligned} \tag{8.18}$$

, and it becomes equivalent to resource manager's decision problem when all $URA_i(t)$s are equal to $1/\omega$.

The desirable properties of stable ecosystem model are driven by simply incorporating resource allocation in the decision process. The resource allocation in the stable ecosystem model is equivalent to:

$$w_i^* = \frac{[R_i(t) + L_i(t) f_i(v_i^*)] / URA_i(t)}{\sum_{p \in K_{n(i)}} [R_p(t) + L_p(t) f_p(v_p^*)] / URA_p(t)} \omega \,. \tag{8.19}$$

As the customers become to observe a common $URA_i(t)$ the resource allocation becomes fair in that the resource is allocated proportional to the customers' required resources. The fairness eliminates excess resource so that customers can keep their queues non-empty. As a result, customers observe a common $URA_i(t)$ equal to $1/\omega$ and consequently make optimal decisions over time. The behavior of this fair allocation is shown in Figure **8.13**, in which resource manager allocates resource proportionally to required resources. The fairness leads the system stable and consequently optimal.

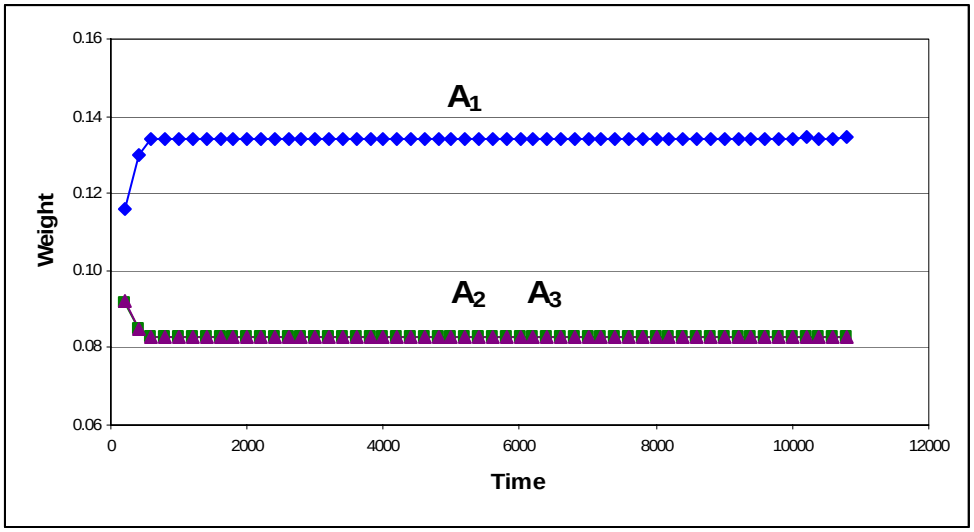


Figure **8.13**: Behavior of w_i^*s under stable ecosystem model

8.5 System Behavior in Stochastic Environments

To observe the system behavior in stochastic environments, we used exponential value functions[8] for customers in the same experimental framework. Figures **8.14** and **8.15** show the behavior of $URA_i(t)$ and T^* respectively in the stochastic environment. $URA_i(t)$s, which are representative stability indicators, behave more stable under stable ecosystem model (SDM) than under naïve ecosystem model (NDM) even in stochastic environment. Consequently, the behavior of T^*, which is a representative decision, follows the same pattern.

[8] The CPU time taken to process a task with a value mode is exponentially distributed.

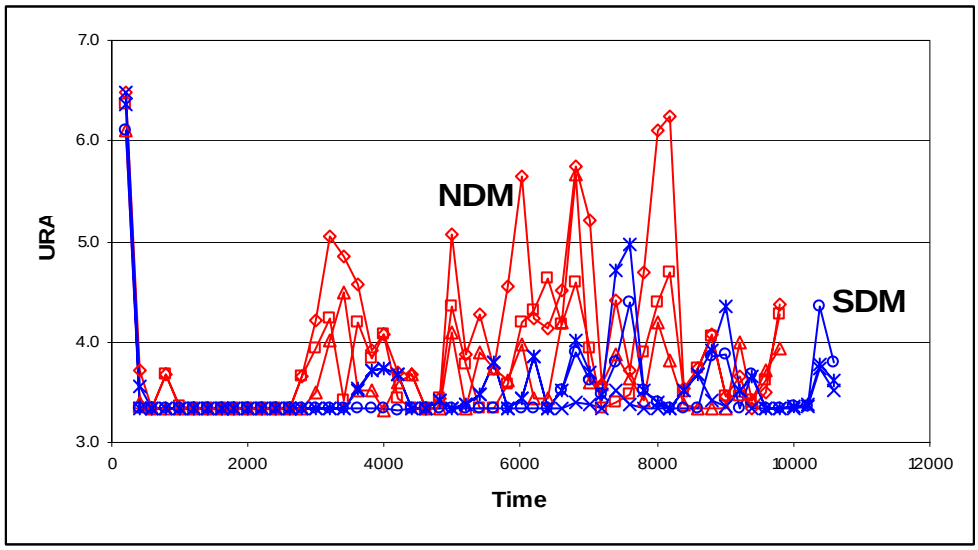


Figure **8.14**: Behavior of $URA_i(t)$ in stochastic environment

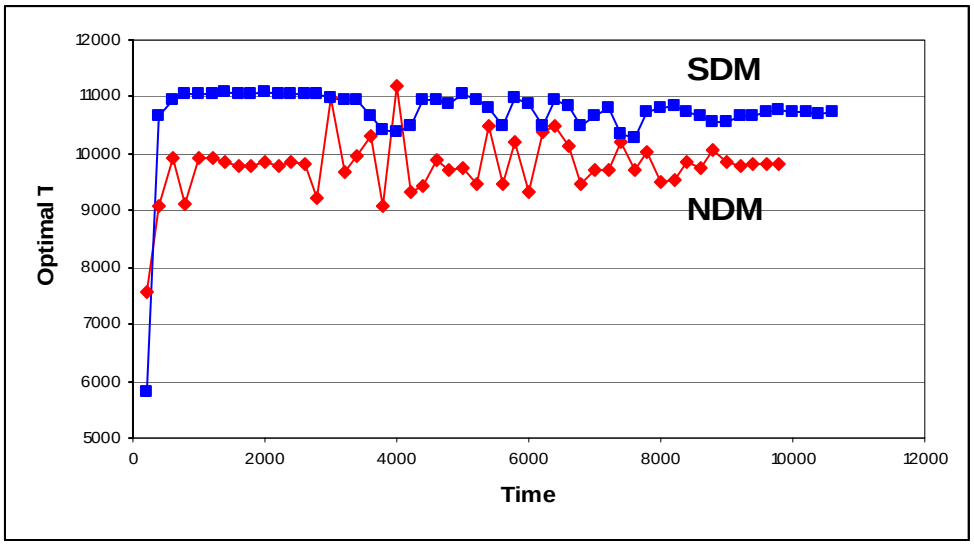


Figure **8.15**: Behavior of T^* in stochastic environment

8.6 Performance Evaluation

The performance of the ecosystem models is summarized in Table **8.1**. In both deterministic and stochastic environments, stable ecosystem model outperforms naïve ecosystem model by about 10%. If all customers are homogeneous with respect to value function and workload, stable ecosystem model is equivalent to naïve ecosystem model. But, in general, they are heterogeneous and the two models behave differently.

Table **8.1**: Ecosystem performance evaluation

	NDM			SDM		
	T	*V*	*QoS*	*T*	*V*	*QoS*
Deterministic	9430	6436	4550	10889	7181	5003
Stochastic	9866	6687	4713	10683	7181	5044

NDM: Naïve ecosystem model, SDM: Stable ecosystem model
T: Completion time, *V*: Value of solution

Chapter 9

Empirical Evaluation

We ran several experiments using discrete-event simulation to validate the designed control mechanism. Due to the equivalence between centralized and decentralized models we do not discriminate them in this experimentation. Though we use a small network in the experimentation for validation purposes, the decentralized model can handle much larger networks.

9.1 Experimental Design

We set up twelve experimental conditions and test seventeen different control policies in each condition.

9.1.1 Experimental Conditions

The experimental network is composed of sixteen components in seven nodes as shown in Figure **9.1**. Each component in the lowest position has root tasks as indicated in the figure. The value function is defined as $<v_i+1, 2, 5>$ for A_7 and A_8, and $<v_i, 2, 5>$ for others. In addition, ω_n is 1 for all $n \in N$ and CPU is allocated using a weighted round-robin scheduling in which CPU time received by each component in a round is equal to its assigned weight.

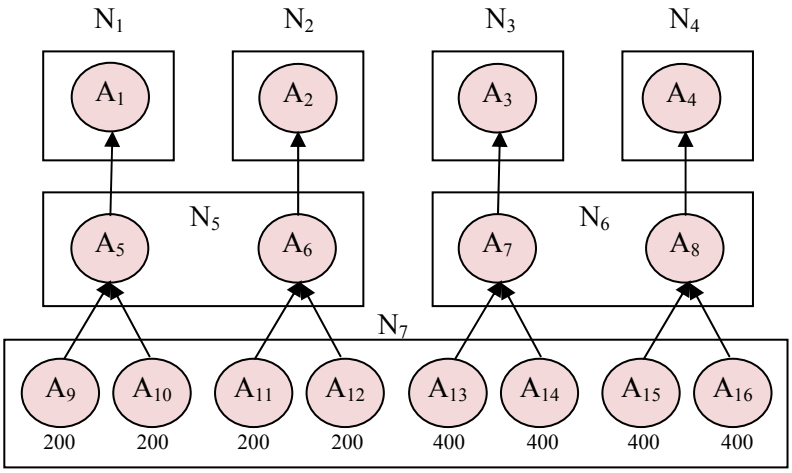


Figure **9.1**: Experimental network configuration

We set up twelve different experimental conditions as shown in Table **9.1**. We use three functions for the cost of completion time and the distribution of CPU time can be deterministic or stochastic. While using stochastic value functions we repeat 5 experiments. There can also be stressors which share resources with the components. We assign weight w_n^s to a stressor residing in node n. A stressor, which has infinite work (continuously requiring resource), can impose different levels of stress on the component directly by changing w_i^s. When it is zero there is no stress, and as it increases the stress level increases. In specific, we assign 1.5 to w_{N6}^s for $t \geq 2000$ in stress environments. QoS^{UB}, which is the QoS upper bound, is computed from (**5.14**) with t=0 for each condition as shown in the table. We evaluate each control policy by computing its relative performance to the performance bound.

Table **9.1**: Experimental conditions

Condition	*CCT(T)*	$f_i(v_i)$	*Stress*	QoS^{UB}
Con1-1	*0.5T*	Deterministic	No	30000
Con1-2	*0.5T*	Exponential	No	30000
Con2-1	*1.5T*	Deterministic	No	19200
Con2-2	*1.5T*	Exponential	No	19200
Con3-1	*2.5T*	Deterministic	No	12000
Con3-2	*2.5T*	Exponential	No	12000
Con4-1	*0.5T*	Deterministic	w_{N6}^S=1.5 in t≥2000	30000
Con4-2	*0.5T*	Exponential	w_{N6}^S=1.5 in t≥2000	30000
Con5-1	*1.5T*	Deterministic	w_{N6}^S=1.5 in t≥2000	19200
Con5-2	*1.5T*	Exponential	w_{N6}^S=1.5 in t≥2000	19200
Con6-1	*2.5T*	Deterministic	w_{N6}^S=1.5 in t≥2000	12000
Con6-2	*2.5T*	Exponential	w_{N6}^S=1.5 in t≥2000	12000

9.1.2 Control Policies

We use seventeen different control policies for each experimental condition as shown in Table **9.2**. First twelve control policies are fixed mode policies that use fixed value modes over time. The policies of F*X*-R have no control as the resources are allocated by round-robin resource allocation where the components in each node are assigned equal weights over time. In comparison, the policies of F*X*-O allocate resources proportional to the components' load indices only at t=0 while those of F*X*-C periodically as in (**5.15**). The last five policies are predictive control policies that select value modes by solving a programming model. PO solves the non-adaptive programming model in (**5.14**) and (**5.15**) only at t=0 while PCNN periodically. PCYN solves periodically the adaptive programming model in (**5.19**) and (**5.20**) and PCYY the stable adaptive programming model in (**6.2**) and (**6.3**). RCYY uses the stable adaptive programming model except that weights are assigned equally over time. PCYY is the control policy we are proposing in this research. The system makes decision every 100 time units[9] (i.e. SW=100).

Table **9.2**: Control policies used for experimentation

Control policy	Mode	Resource allocation	Feedback	Monitoring	Depth
F2-R	$v_i = 2$ for all i	Round-Robin	--	--	--
F2-PO	$v_i = 2$ for all i	Proportional	Open	--	--
F2-PC	$v_i = 2$ for all i	Proportional	Closed	--	--
F3-R	$v_i = 3$ for all i	Round-Robin	--	--	--
F3-PO	$v_i = 3$ for all i	Proportional	Open	--	--
F3-PC	$v_i = 3$ for all i	Proportional	Closed	--	--
F4-R	$v_i = 4$ for all i	Round-Robin	--	--	--
F4-PO	$v_i = 4$ for all i	Proportional	Open	--	--
F4-PC	$v_i = 4$ for all i	Proportional	Closed	--	--
F5-R	$v_i = 5$ for all i	Round-Robin	--	--	--
F5-PO	$v_i = 5$ for all i	Proportional	Open	--	--
F5-PC	$v_i = 5$ for all i	Proportional	Closed	--	--
RCYY	--	Round-Robin	Closed	Yes	Yes
PO	--	Proportional	Open	--	--
PCNN	--	Proportional	Closed	No	No
PCYN	--	Proportional	Closed	Yes	No
PCYY	--	Proportional	Closed	Yes	Yes

[9] The decision period is determined arbitrary. However, it would be desirable to choose it based on the burden of control as well as dynamic nature of the system.

9.2 Numerical Results

We analyze the numerical results for the completion time of fixed mode policies and QoS of all control policies.

9.2.1 Completion Time of Fixed Mode Policies

The lower bound performances T^{LB} of completion time under fixed mode policies are summarized in Table **9.3**, which are based on (**5.6**). The actual performance of the fixed mode policies are shown in Table **9.4**. In the environments with no stress, both proportional allocation policies of F*X*-PO and F*X*-PC give the performance close to lower bound performance with significant advantages compared to round-robin allocation policy in all different fixed modes. Though both policies perform equivalently in deterministic environments F*X*-PC tends to perform better in stochastic environments. The periodic design process alleviates the impacts of stochasticity. Similar arguments are also valid even in stressed environments though the overall performance is reduced. These observations support the optimality of the periodical proportional resource allocation and consequently the validity of the subsequent programming models we have built.

Table **9.3**: Lower bound performance of completion time

Mode	T^{LB}
$v_i = 2$ for all i	4800
$v_i = 3$ for all i	7200
$v_i = 4$ for all i	9600
$v_i = 5$ for all i	12000

Table **9.4**: Completion time of fixed mode policies

		Control Policy								
		F2-R	F2-PO	F2-PC	F3-R	F3-PO	F3-PC	F4-R	F4-PO	F4-PC
Con1-1	*T*	5614	4814	4814	8019	7219	7219	10423	9624	9624
	%	85.50	99.71	99.71	89.79	99.74	99.74	92.10	99.75	99.75
Con1-2	*T*	5592	4881	4993	8093	7351	7114	10356	9668	9593
	%	85.83	98.35	96.13	88.96	97.95	101.21	92.70	99.30	100.07
Con4-1	*T*	10278	9029	9030	14699	13596	13596	18919	17882	17882
	%	46.70	53.16	53.15	48.98	52.96	52.96	50.74	53.69	53.69
Con4-2	*T*	10294	9459	9026	14799	13945	13412	19382	17802	17811
	%	46.63	50.75	53.18	48.65	51.63	53.68	49.53	53.93	53.90
		F5-R	F5-PO	F5-PC						
Con1-1	*T*	12828	12028	12028						
	%	93.55	99.77	99.77						
Con1-2	*T*	12846	12011	11885						
	%	93.42	99.91	100.97						
Con4-1	*T*	23061	22055	22056						
	%	52.04	54.41	54.41						
Con4-2	*T*	23295	22240	21836						
	%	51.51	53.96	54.96						

T: Actual completion time, %: $100*T^{LB}/T$

9.2.2 QoS

QoS under the control policies are shown in Tables **9.5** to **9.8**. As the cost of completion time increases the system under predictive control policies completes earlier as a result of trading off between the value of solution and the cost of completion time. In unstressed environments, the last four policies (PO, PCNN, PCYN, PCYY) give the best performance close to QoS^{UB} in all different test conditions. In stressed environments the benefits of the last two policies (PCYN, PCYY) become apparent. The performance degradation of these two adaptive policies is significantly smaller than non-adaptive policies. These observations support the effectiveness of the adaptive programming models we have designed. However, it is not clear if the depth consideration gives better performance. However, we can say that PCYY is a robust policy to keep the system from behaving divergently and degrading performance significantly as have shown in the previous analysis in Chapter 6.

Table **9.5**: Numerical results: Deterministic environments with no stress

		Control Policy								
		F2-R	F2-PO	F2-PC	F3-R	F3-PO	F3-PC	F4-R	F4-PO	F4-PC
Con1-1	*T*	5614	4814	4814	8019	7219	7219	10423	9624	9624
	V	14400	14400	14400	21610	21610	21610	28800	28800	28800
	QoS	11593	11993	11993	17601	18001	18001	23589	23988	23988
	%	38.64	39.98	39.98	58.67	60.00	60.00	78.63	79.96	79.96
Con2-1	*T*	5614	4814	4814	8019	7219	7219	10423	9624	9624
	V	14400	14400	14400	21610	21610	21610	28800	28800	28800
	QoS	5979	7179	7179	9582	10782	10782	13166	14364	14364
	%	31.14	37.39	37.39	49.90	56.15	56.15	68.57	74.81	74.81
Con3-1	*T*	5614	4814	4814	8019	7219	7219	10423	9624	9624
	V	14400	14400	14400	21610	21610	21610	28800	28800	28800
	QoS	365	2365	2365	1563	3563	3563	2743	4740	4740
	%	3.04	19.71	19.71	13.02	29.69	29.69	22.85	39.50	39.50
		F5-R	F5-PO	F5-PC	RCYY	PO	PCNN	PCYN	PCYY	
Con1-1	*T*	12828	12028	12028	12779	12028	12028	12028	12025	
	V	36000	36000	36000	35959	36000	35999	35999	35997	
	QoS	29586	29986	29986	29570	29986	29985	29985	29985	
	%	98.62	99.95	99.95	98.57	99.95	99.95	99.95	99.95	
Con2-1	*T*	12828	12028	12028	9285	9629	9870	9691	9661	
	V	36000	36000	36000	32918	33600	33832	33661	33630	
	QoS	16758	17958	17958	18991	19156	19027	19124	19139	
	%	87.28	93.53	93.53	98.91	99.77	99.10	99.61	99.68	
Con3-1	*T*	12828	12028	12028	6421	4824	4884	4865	4834	
	V	36000	36000	36000	27199	24000	24064	24061	24019	
	QoS	3930	5930	5930	11146	11940	11854	11898	11934	
	%	32.75	49.42	49.42	92.88	99.50	98.78	99.15	99.45	

T: Completion time, *V*: Value of solution, %: $100*QoS/QoS^{UB}$

Table **9.6**: Numerical results: Stochastic with no stress

		Control Policy								
		F2-R	F2-PO	F2-PC	F3-R	F3-PO	F3-PC	F4-R	F4-PO	F4-PC
Con1-2	*T*	5592	4881	4993	8093	7351	7114	10356	9668	9593
	V	14400	14400	14400	21610	21610	21610	28800	28800	28800
	QoS	11604	11960	11903	17563	17935	18053	23622	23966	24004
	%	38.68	39.87	39.68	58.54	59.78	60.18	78.74	79.89	80.01
Con2-2	*T*	5592	4881	4993	8093	7351	7114	10356	9668	9593
	V	14400	14400	14400	21610	21610	21610	28800	28800	28800
	QoS	6011	7079	6910	9470	10584	10939	13266	14299	14411
	%	31.31	36.87	35.99	49.32	55.12	56.97	69.09	74.47	75.06
Con3-2	*T*	5592	4881	4993	8093	7351	7114	10356	9668	9593
	V	14400	14400	14400	21610	21610	21610	28800	28800	28800
	QoS	419	2199	1917	1377	3233	3825	2910	4631	4818
	%	3.49	18.32	15.97	11.47	26.94	31.88	24.25	38.59	40.15
		F5-R	F5-PO	F5-PC	RCYY	PO	PCNN	PCYN	PCYY	
Con1-2	*T*	12846	12011	11885	12722	12011	12005	11912	12139	
	V	36000	36000	36000	35784	36000	35999	35996	35997	
	QoS	29577	29995	30058	29424	29995	29996	30040	29927	
	%	98.59	99.98	100.19	98.08	99.98	99.99	100.13	99.76	
Con2-2	*T*	12846	12011	11885	9036	9807	10149	9726	9674	
	V	36000	36000	36000	32440	33600	34125	33589	33510	
	QoS	16731	17984	18173	18887	18890	18902	19000	18999	
	%	87.14	93.67	94.65	98.37	98.39	98.45	98.96	98.95	
Con3-2	*T*	12846	12011	11885	7165	5007	5043	5118	5067	
	V	36000	36000	36000	28298	24000	24205	24484	24395	
	QoS	3886	5973	6289	10387	11482	11598	11688	11727	
	%	32.38	49.78	52.40	86.56	95.68	96.65	97.40	97.73	

T: Completion time, *V*: Value of solution, %: $100*QoS/QoS^{UB}$

Table **9.7**: Numerical results: Deterministic with stress

		Control Policy								
		F2-R	F2-PO	F2-PC	F3-R	F3-PO	F3-PC	F4-R	F4-PO	F4-PC
Con4-1	*T*	10278	9029	9030	14699	13596	13596	18919	17882	17882
	V	14400	14400	14400	21610	21610	21610	28800	28800	28800
	QoS	9261	9886	9885	14261	14812	14812	19341	19859	19859
	%	30.87	32.95	32.95	47.54	49.37	49.37	64.47	66.20	66.20
Con5-1	*T*	10278	9029	9030	14699	13596	13596	18919	17882	17882
	V	14400	14400	14400	21610	21610	21610	28800	28800	28800
	QoS	-1017	857	854	-439	1216	1216	422	1977	1977
	%	-5.30	4.46	4.45	-2.28	6.33	6.33	2.20	10.30	10.30
Con6-1	*T*	10278	9029	9030	14699	13596	13596	18919	17882	17882
	V	14400	14400	14400	21610	21610	21610	28800	28800	28800
	QoS	-11295	-8171	-8176	-15138	-12380	-12380	-18498	-15905	-15905
	%	-94.13	-68.09	-68.13	-126.15	-103.17	-103.17	-154.15	-132.54	-132.54
		F5-R	F5-PO	F5-PC	RCYY	PO	PCNN	PCYN	PCYY	
Con4-1	*T*	23061	22055	22056	14019	22054	22054	12091	12096	
	V	36000	36000	36000	32391	36000	35999	32014	32016	
	QoS	24470	24973	24972	25382	24973	24972	25969	25968	
	%	81.57	83.24	83.24	84.61	83.24	83.24	86.56	86.56	
Con5-1	*T*	23061	22055	22056	11596	21063	14384	11596	11601	
	V	36000	36000	36000	30527	33600	32531	31664	31657	
	QoS	1409	2918	2916	13133	2006	10955	14270	14256	
	%	7.34	15.20	15.19	68.40	10.45	57.06	74.32	74.25	
Con6-1	*T*	23061	22055	22056	10313	9032	9070	9048	9044	
	V	36000	36000	36000	29011	24000	25934	28055	28049	
	QoS	-21653	-19138	-19140	3229	1421	3258	5434	5440	
	%	-180.44	-159.48	-159.50	26.90	11.84	27.15	45.29	45.33	

T: Completion time, *V*: Value of solution, %: $100*QoS/QoS^{UB}$

Table **9.8**: Numerical results: Stochastic with stress

		Control Policy								
		F2-R	F2-PO	F2-PC	F3-R	F3-PO	F3-PC	F4-R	F4-PO	F4-PC
Con4-2	*T*	10294	9459	9026	14799	13945	13412	19382	17802	17811
	V	14400	14400	14400	21610	21610	21610	28800	28800	28800
	QoS	9253	9671	9887	14211	14637	14904	19109	19899	19895
	%	30.84	32.24	32.96	47.37	48.79	49.68	63.70	66.33	66.32
Con5-2	*T*	10294	9459	9026	14799	13945	13412	19382	17802	17811
	V	14400	14400	14400	21610	21610	21610	28800	28800	28800
	QoS	-1042	212	862	-588	692	1492	-273	2097	2084
	%	-5.42	1.10	4.49	-3.06	3.60	7.77	-1.42	10.92	10.85
Con6-2	*T*	10294	9459	9026	14799	13945	13412	19382	17802	17811
	V	14400	14400	14400	21610	21610	21610	28800	28800	28800
	QoS	-11336	-9247	-8164	-15387	-13254	-11920	-19655	-15706	-15727
	%	-94.47	-77.06	-68.03	-128.23	-110.45	-99.33	-163.79	-130.88	-131.06
		F5-R	F5-PO	F5-PC	RCYY	PO	PCNN	PCYN	PCYY	
Con4-2	*T*	23295	22240	21836	14200	22240	21963	12176	12292	
	V	36000	36000	36000	32549	36000	35999	32064	32052	
	QoS	24352	24880	25082	25449	24880	25017	25976	25906	
	%	81.17	82.93	83.61	84.83	82.93	83.39	86.59	86.35	
Con5-2	*T*	23295	22240	21836	12118	21450	14416	11892	11635	
	V	36000	36000	36000	30767	33600	32542	31583	31556	
	QoS	1057	2640	3246	12589	1425	10918	13745	14104	
	%	5.51	13.75	16.91	65.57	7.42	56.86	71.59	73.46	
Con6-2	*T*	23295	22240	21836	10823	9355	9342	9603	9267	
	V	36000	36000	36000	29510	24000	26051	28293	28093	
	QoS	-22238	-19600	-18590	2451	612	2695	4285	4925	
	%	-185.32	-163.33	-154.92	20.43	5.10	22.46	35.71	41.04	

T: Completion time, *V*: Value of solution, %: $100*QoS/QoS^{UB}$

9.3 Behavior

We analyze the system behavior by studying resource utilization, stability, and adaptivity.

9.3.1 Resource Utilization

The performance differences can be reasoned from resource utilization as discussed earlier. In the experimental network the resource allocation of node N_7 affects the task arrival patterns for the components of N_5 and N_6. In fixed mode conditions, the task arrival patterns to N_6 can affect the overall performance because the node has large number of tasks compared to other nodes. So, we analyze the resource utilization of N_6 depending on the resource allocation policies. Resource utilization profiles of N_6 are shown in Figures **9.2** to **9.5** for no stress environments and Figures **9.6** to **9.9** for stress environments, in which a data point corresponds to the amount of resource utilized during a control period (100 time units).

In no stress environments, N_6 utilizes resource consistently to the completion under proportional resource allocation. In contrast, N_6 utilizes less resource in the first stage and more in the second stage under round robin allocation. This is due to the task arrival patterns depending on the resource allocation of N_7. Though tasks are generated at a constant rate under proportional allocation they are generated at an increasing rate under round robin resource allocation. Note that the time to generate all the tasks from N_7 is the same for both resource allocation policies. As the increasing task arrival pattern is not preferable due to the possibility of underutilization, the completion time of the N_6 can increase resulting in the increased completion time of the network as have shown in Table **9.4**. These arguments also hold in stressed environments. When stressed, N_6 utilizes its assigned resource fully in all different conditions. As more resources are utilized in the initial stage, the completion time under proportional allocation is shorter than under round robin allocation. These observations support the argument that proportional allocation leads to maximize the utilization of distributed resources.

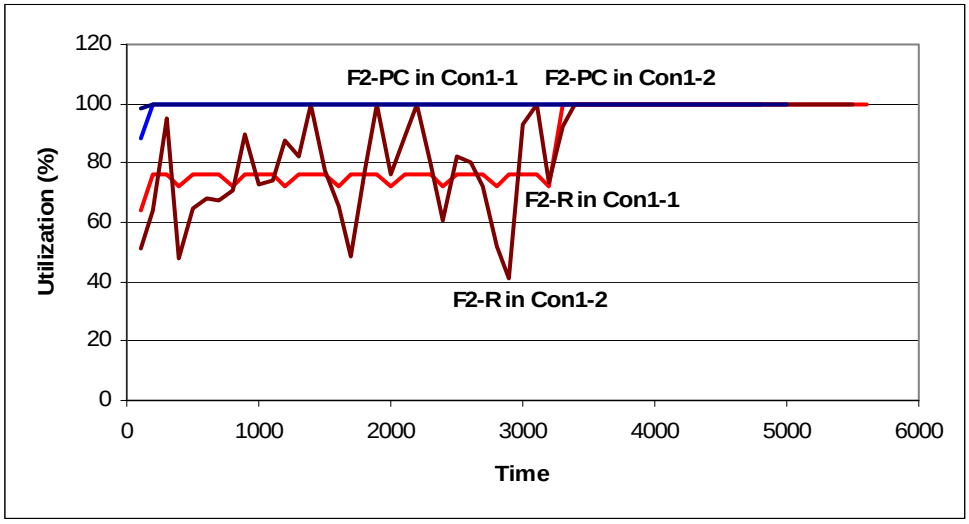


Figure **9.2**: Resource utilization under F2 control policies in no stress environments

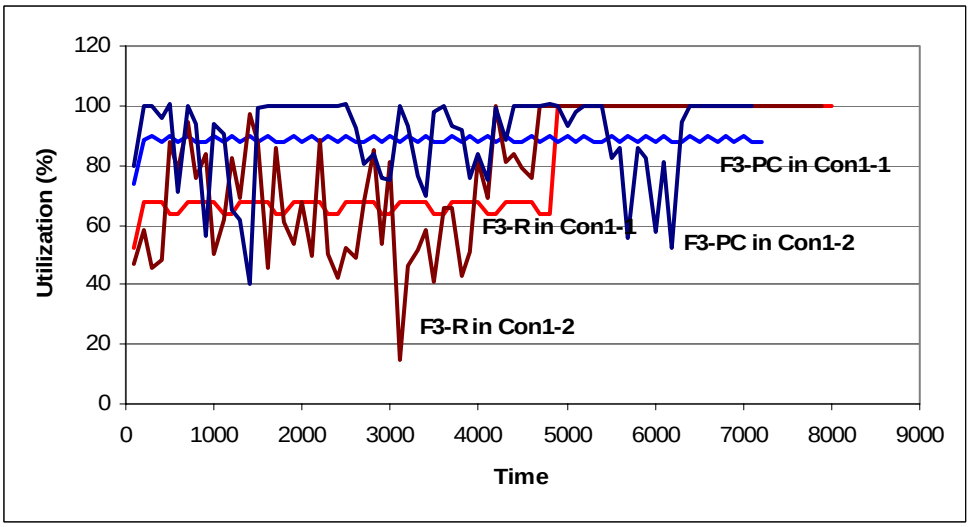


Figure **9.3**: Resource utilization under F3 control policies in no stress environments

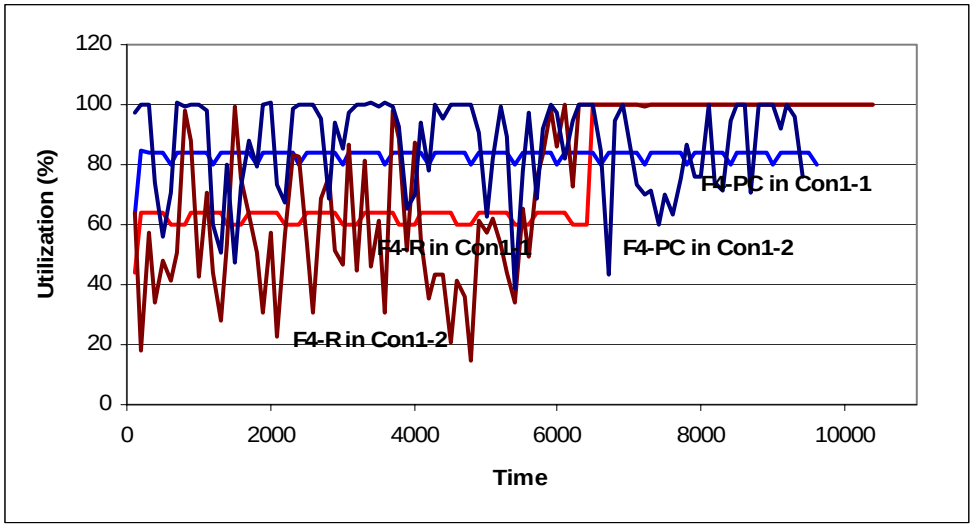


Figure **9.4**: Resource utilization under F4 control policies in no stress environments

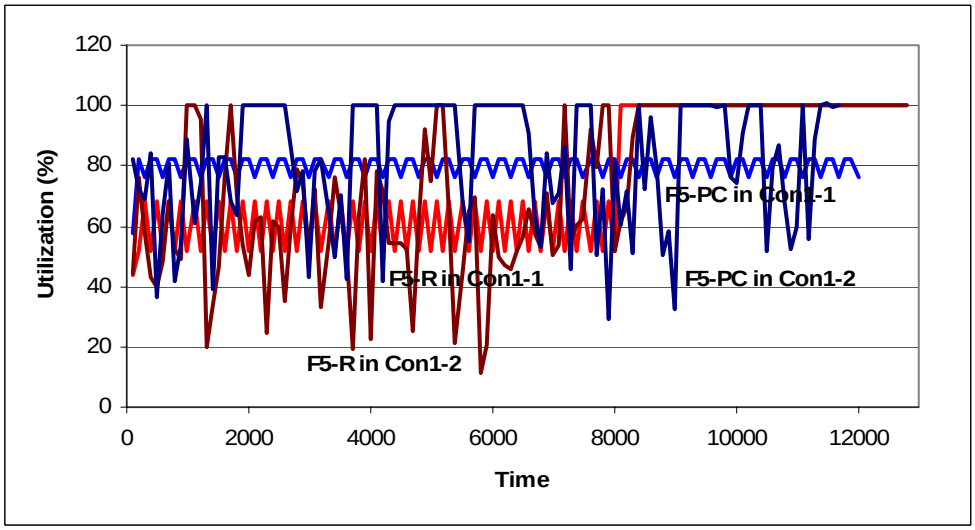


Figure **9.5**: Resource utilization under F5 control policies in no stress environments

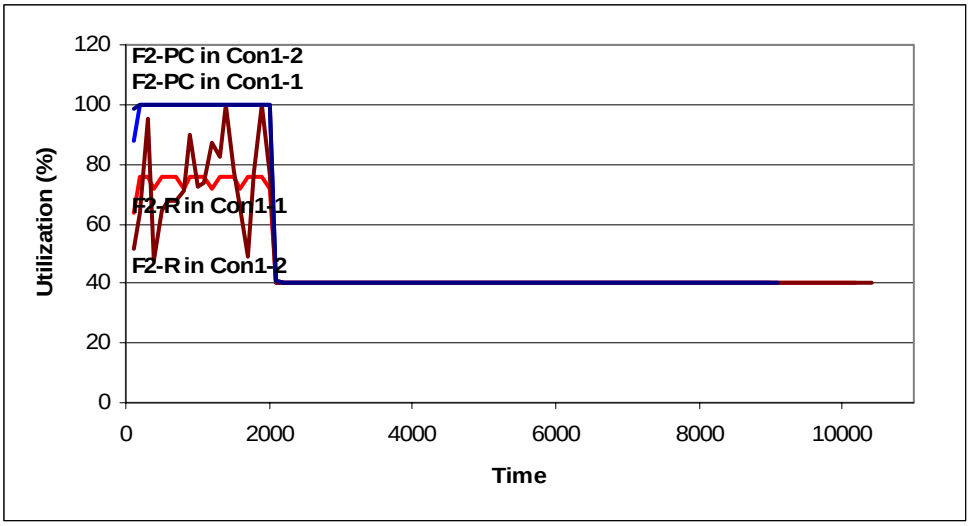


Figure **9.6**: Resource utilization under F2 control policies in stress environments

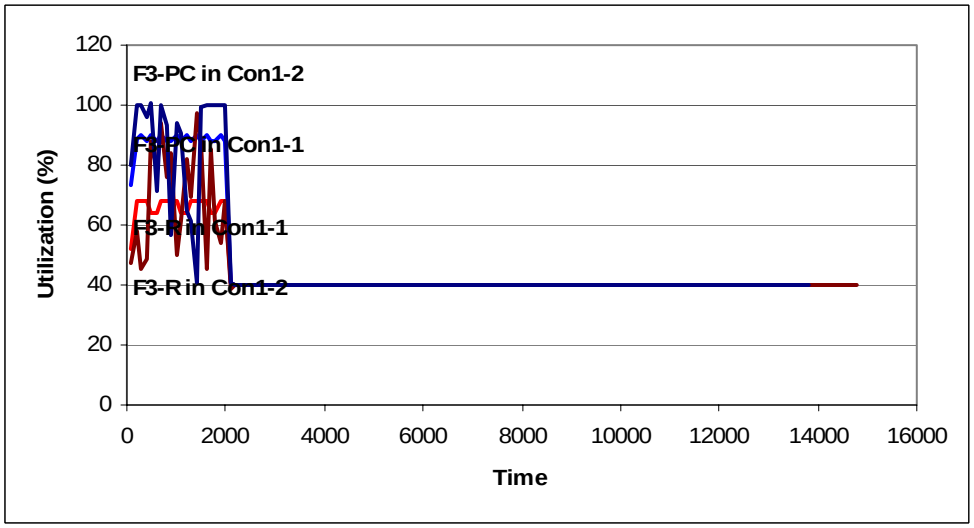


Figure **9.7**: Resource utilization under F3 control policies in stress environments

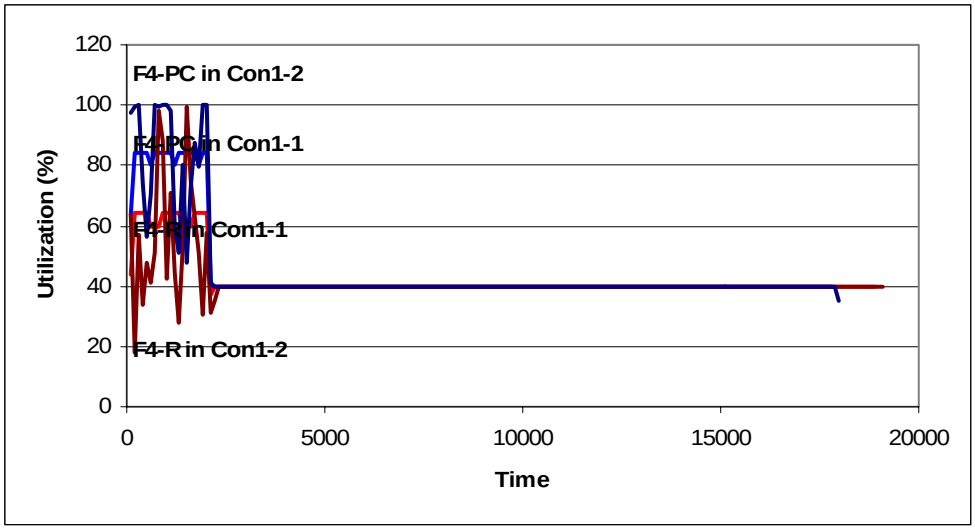


Figure **9.8**: Resource utilization under F4 control policies in stress environments

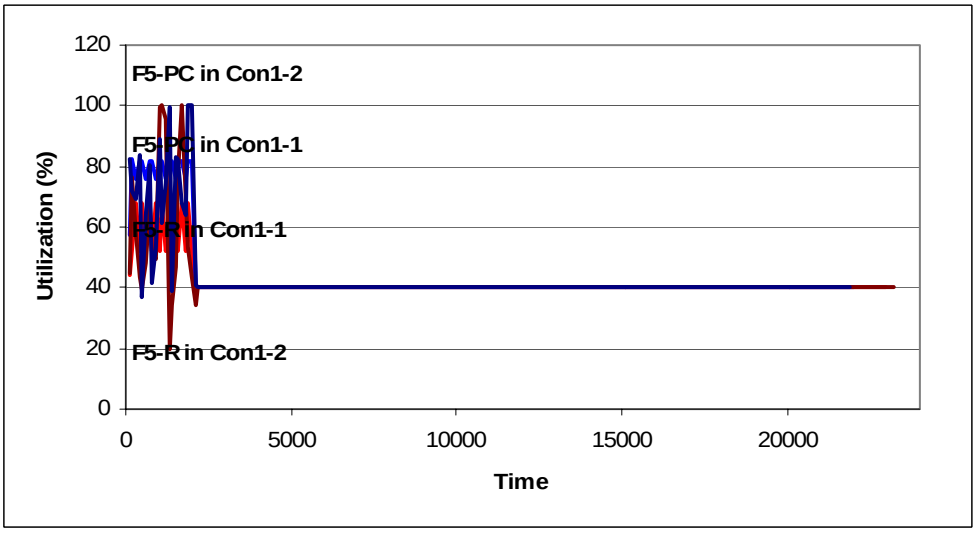


Figure **9.9**: Resource utilization under F5 control policies in stress environments

9.3.2 Stability

Figures **9.10** to **9.15** show the behaviors of T^* under the last four closed-loop control policies (RCYY, PCNN, PCYN, PCYY) in no stress environments. Note that the behavior of T^* represents the system behavior and one can observe that the behavior of v_i^* in Appendix C follows the pattern of T^*.

The system behaves stable in overall making consistent decisions over time except RCYY. When *CCT(T)=0.5T* (Figures **9.10** and **9.11**), RCYY overestimates the completion time in the initial state because it underestimates the resource availability in the future. As the components complete their tasks the remaining components become utilizing more resources. When *CCT(T)=1.5T* (Figures **9.12** and **9.13**), T^* fluctuates even in deterministic environments. This is due to the existence of equivalent solutions. As there are multiple optimal solutions the decision changes from time to time depending on the slight changes of measured parameters such as *URA'(t)*.

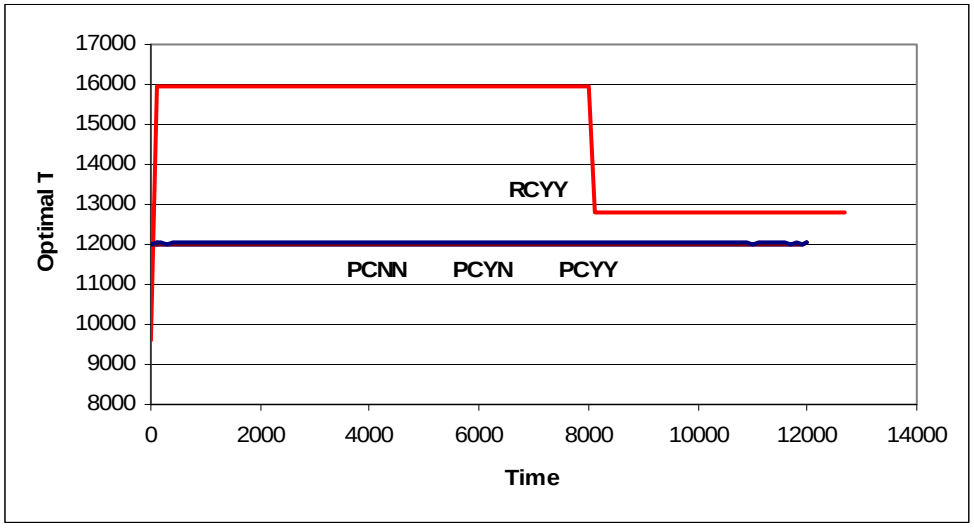


Figure **9.10**: Behavior of T^* in deterministic environment with no stress (Con1-1)

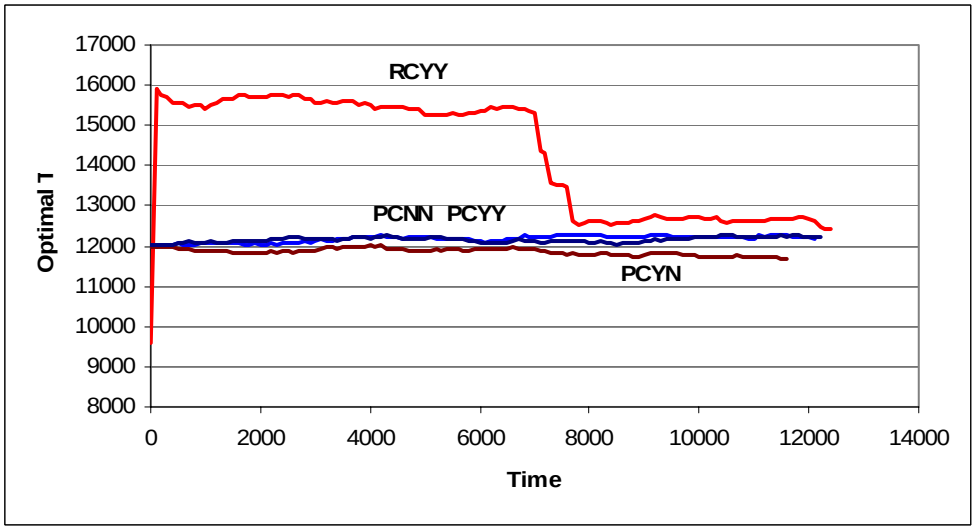


Figure **9.11**: Behavior of T^* in stochastic environment with no stress (Con1-2)

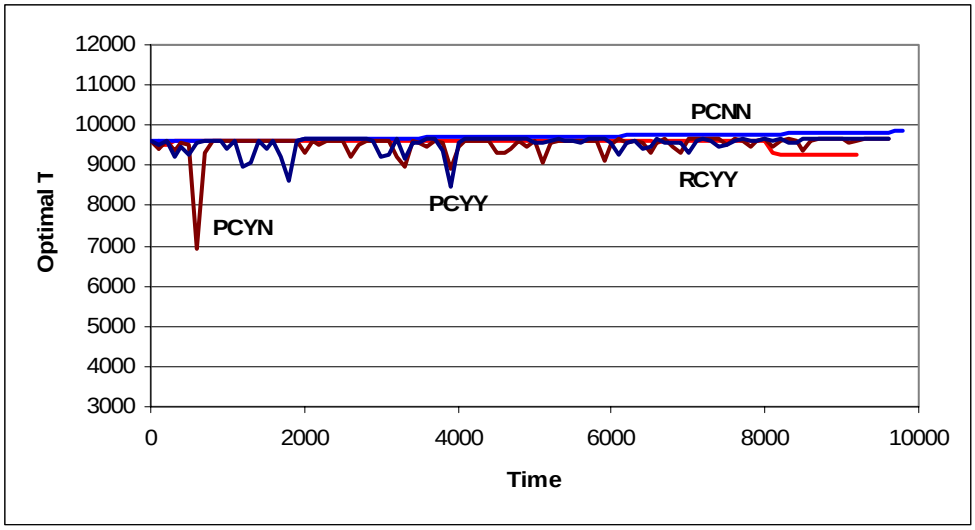


Figure **9.12**: Behavior of T^* in deterministic environment with no stress (Con2-1)

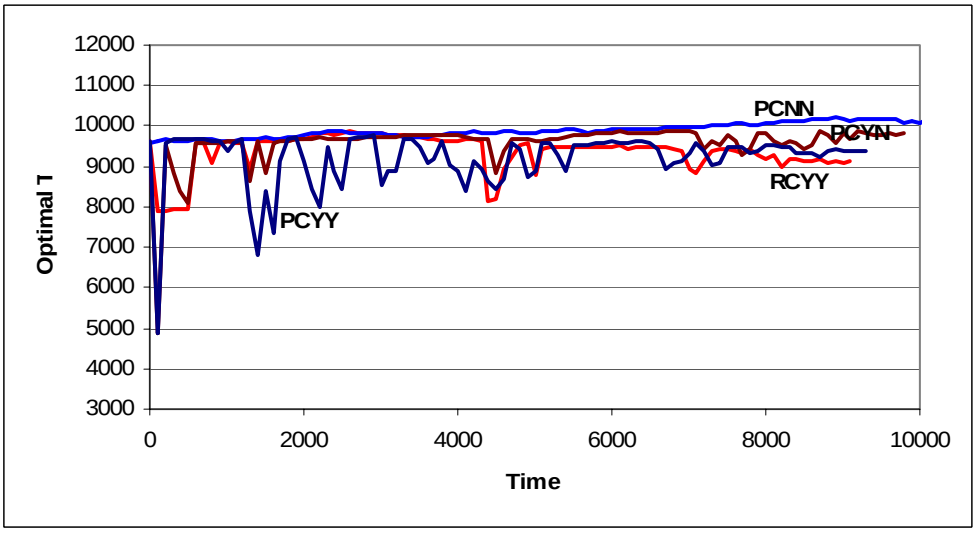


Figure **9.13**: Behavior of T^* in stochastic environment with no stress (Con2-2)

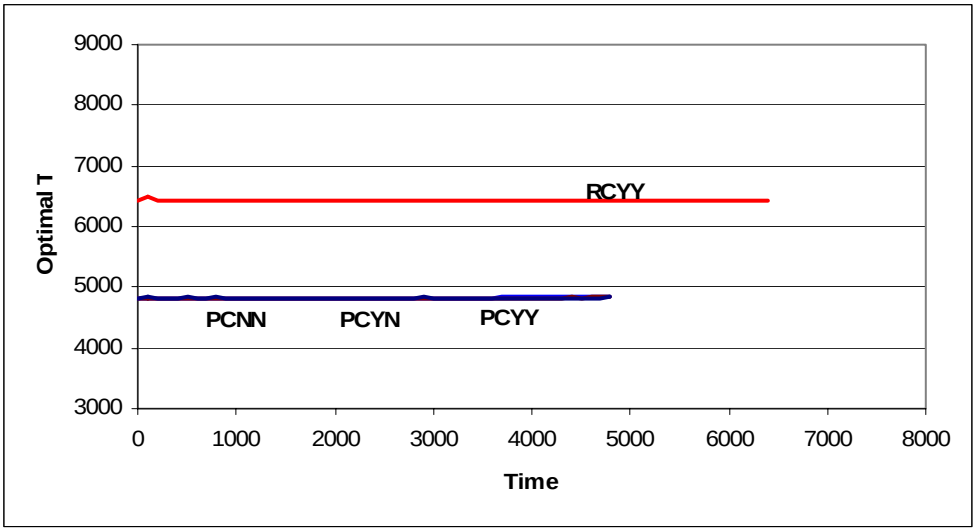


Figure **9.14**: Behavior of T^* in deterministic environment with no stress (Con3-1)

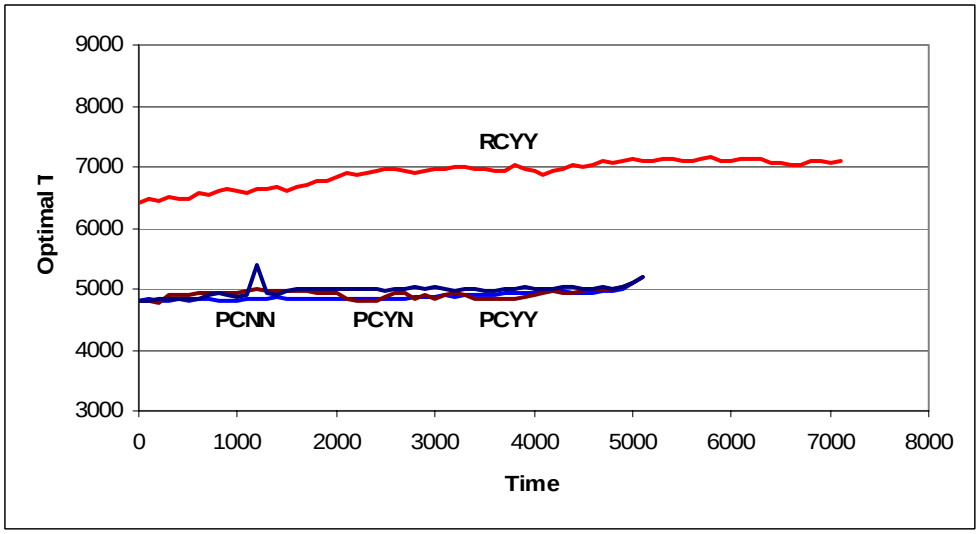


Figure **9.15**: Behavior of T^* in stochastic environment with no stress (Con3-2)

9.3.3 Adaptivity

The system controlled by adaptive control policies (RCYY, PCYN, PCYY) is naturally adaptive to changing environments as components monitor their environments and incorporate them into the decision process. As shown in Figures **9.16** to **9.21** the behaviors of T^* under the adaptive control policies change in response to the stressor. However, T^* under the non-adaptive policy PCNN tends to increase as a result of neglecting the stress environments, i.e. the system cannot process tasks as planned. One can observe the behaviors of v_i^* in Appendix D.

The adaptivity arises as a result of monitoring stress environments through resource sensors which quantify $URA_i(t)$. Figures **9.22** and **9.23** show the behaviors of these measurements for deterministic and stochastic environments respectively. When stress is imposed on node N_6, the component A_7 in this node observes reduced resource availability.

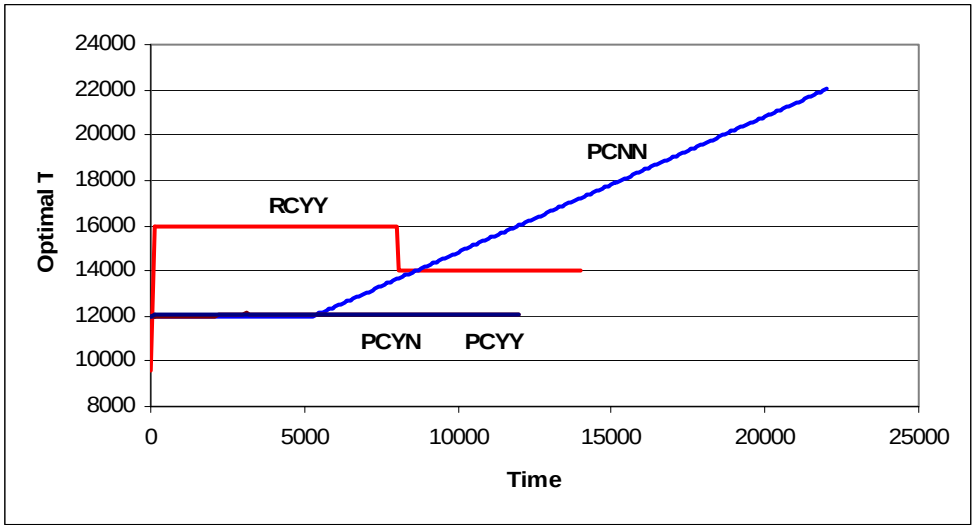


Figure **9.16**: Behavior of T^* in deterministic environment with stress (Con4-1)

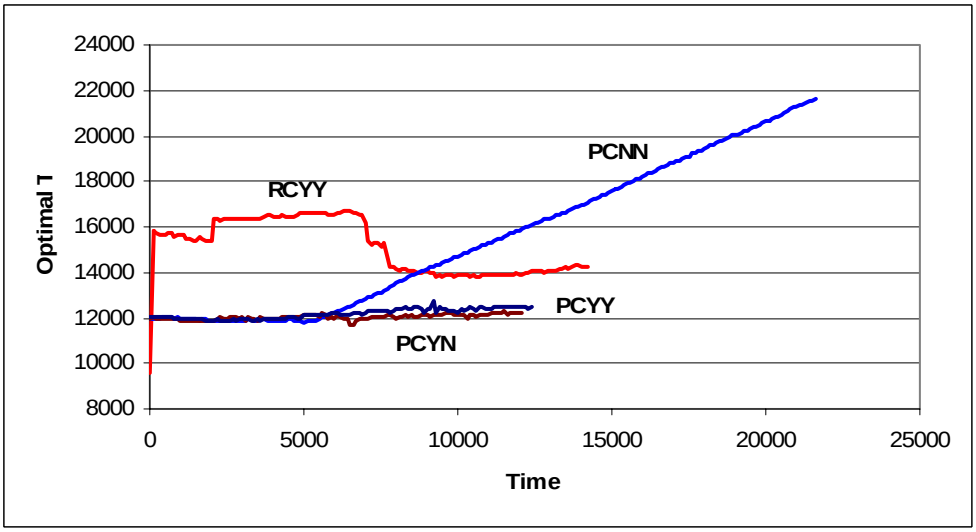


Figure **9.17**: Behavior of T^* in stochastic environment with stress (Con4-2)

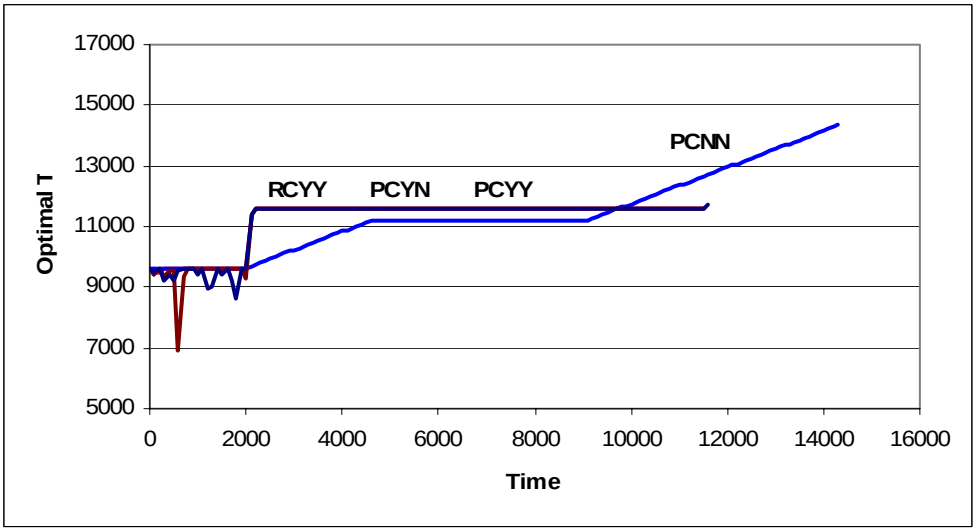


Figure **9.18**: Behavior of T^* in deterministic environment with stress (Con5-1)

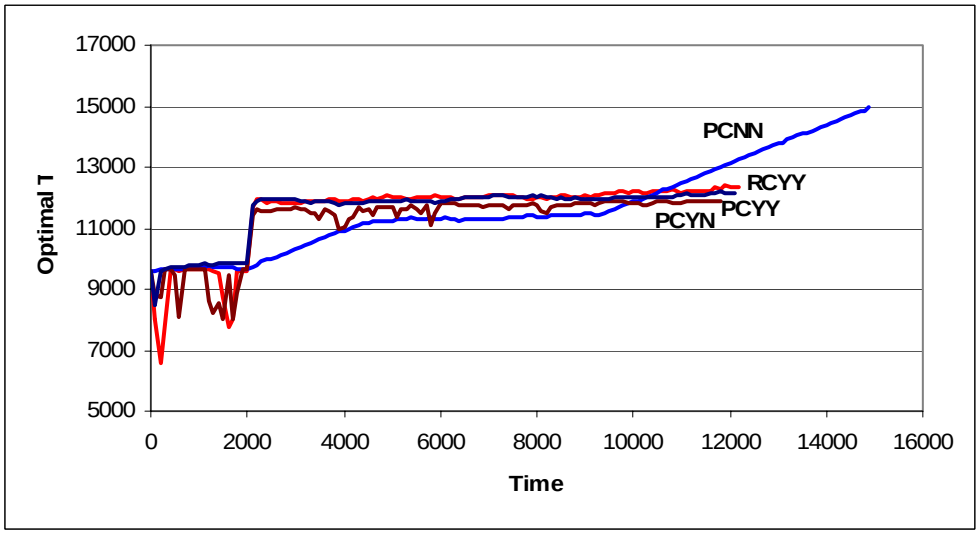


Figure **9.19**: Behavior of T^* in stochastic environment with stress (Con5-2)

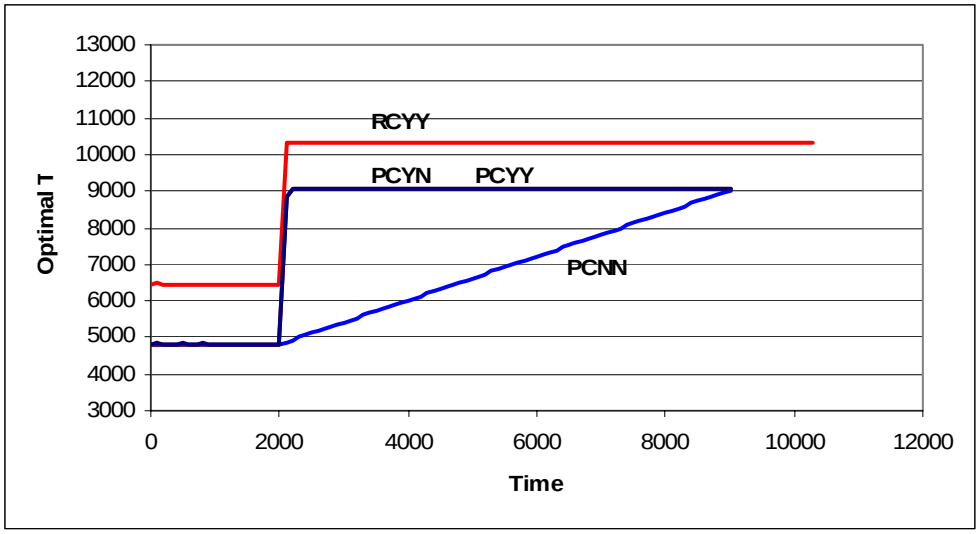


Figure **9.20**: Behavior of T^* in deterministic environment with stress (Con6-1)

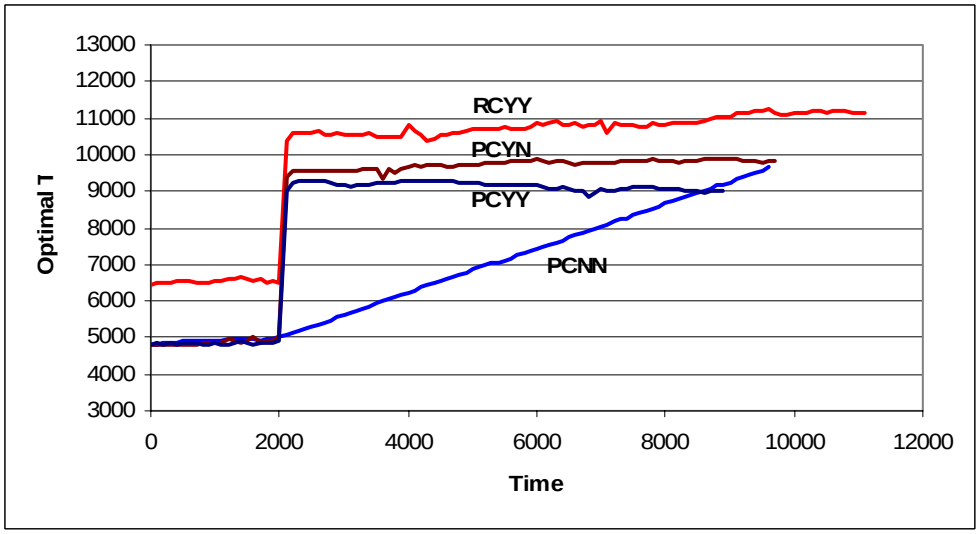


Figure **9.21**: Behavior of T^* in stochastic environment with stress (Con6-2)

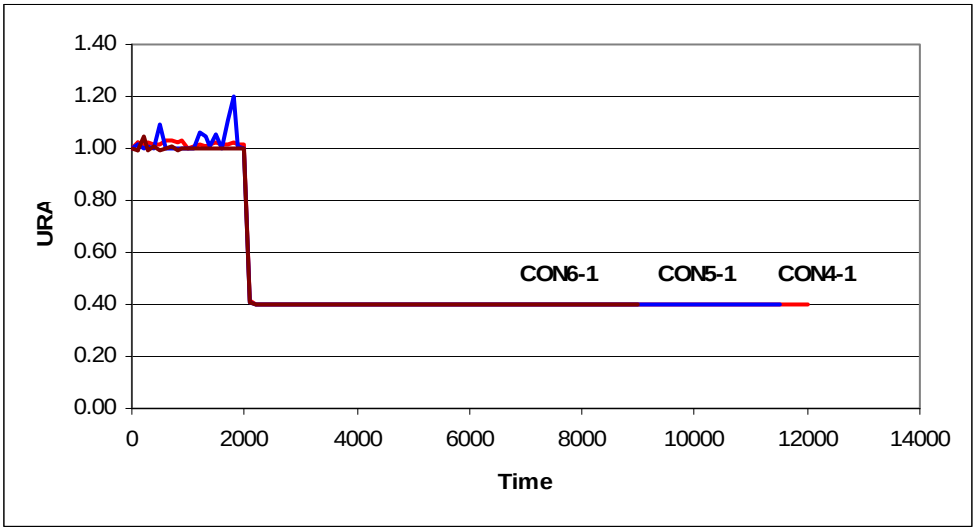


Figure **9.22**: Behavior of URA_{A7} under PCYY in deterministic environments with stress

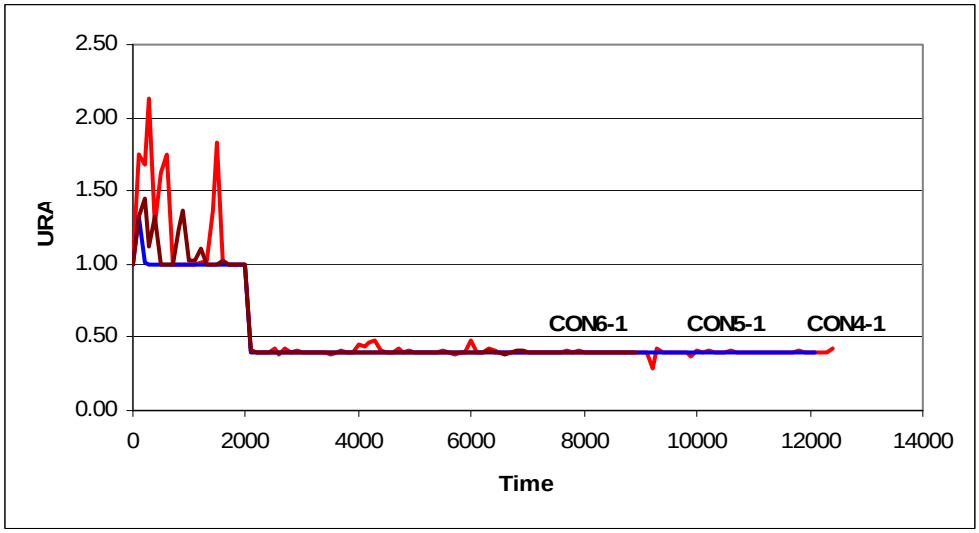


Figure **9.23**: Behavior of URA_{A7} under PCYY in stochastic environments with stress

9.4 Summary of Empirical Study

The empirical study demonstrates the superiority of the designed control mechanism (PCYY) to other alternative control policies. Table **9.9** summarizes the observations from the empirical study, which shows average QoS (%) for no stress and stressed environments, and the behavioral properties of stability and adaptivity. PCYY gives the best performance in all different conditions, close to upper bound performance in no stress environments and degraded minimally in stressed environments. In addition, the system under the designed control mechanism is stable making consistent decisions over time and gracefully adapts to new environments. Though the control policy PCYN, which does not consider the depth, is comparable to PCYY in this experimental network, it can exhibit divergent behavior in some other networks resulting in significant performance degradation. Therefore, this empirical study validates the effectiveness of the designed control mechanism. Also, the multi-tier auctioning process we have designed can address the scalability issue. We aimed at developing a scalable adaptive control mechanism and we have achieved it.

Table **9.9**: Summary of empirical study

Control policy	QoS (%) (No stress)	QoS (%) (Stress)	Stability	Adaptivity
F2-R	24.38	-22.93	--	--
F2-PO	32.02	-12.40	--	--
F2-PC	31.45	-10.22	--	--
F3-R	40.16	-27.47	--	--
F3-PO	47.95	-17.59	--	--
F3-PC	49.14	-14.89	--	--
F4-R	57.02	-31.50	--	--
F4-PO	64.54	-18.28	--	--
F4-PC	64.91	-18.32	--	--
F5-R	72.79	-31.70	--	--
F5-PO	81.05	-21.28	--	--
F5-PC	81.69	-19.25	--	--
RCYY	95.56	58.46	No	Yes
PO	98.88	33.50	--	No
PCNN	98.82	55.03	Yes	No
PCYN	99.20	66.67	Yes	Yes
PCYY	99.25	67.83	Yes	Yes

Chapter 10

Conclusions and Future Research

The increasing complexity and vulnerability to adverse events of modern software systems give rise to the need for more sophisticated but scalable control mechanisms. In this thesis, we designed such a control mechanism for an emerging information network. The network we study is large-scale with distributed and component-based architecture, and quality of service of the network is determined by the value of global solution and the time for generating global solution. There are two different kinds of control actions facilitated to control the behavior of the network: algorithm selection and resource allocation. In the control mechanism designed, an auction market coordinates the components of a network to produce optimal decisions. Each component bids based on its measured resource availability and optimal decisions are made through a multi-tier auctioning process. By periodically opening the auction market, the system can achieve desirable performance adaptive to changing stress environment while assuring scalability property. The simulation results demonstrate that the proposed approach works effectively. In the following sections, we review the major contributions from this research and suggest future research.

10.1 Contributions

Some of the major contributions of this research are as follows:

1. **Defined a novel control problem**: In this thesis we defined a novel control problem in line with the trends of modern software systems. They are large-scale, distributed, component-based, and capable of switching between algorithms. The control objective is how to make these systems to survive in the presence of malicious attacks or accidental failures which are unpredictable and unidentifiable.

2. **Designed a self-organizing resource allocation mechanism**: We designed proportional resource allocation policy for minimizing completion time. The proportional

resource allocation policy has several emergent properties. Though it is localized requiring almost no computation, it realizes desirable global performance adaptive to changing environments. Such emergent properties can be found in many self-organizing systems such as social or biological systems. Though entities act with a simple mechanism without central authority, desirable global performance can often be realized. A large-scale network working in a dynamic environment under the designed control mechanism, is a representation of a self-organizing system.

3. **Formulated an adaptive mathematical programming model**: We formulated the control problem as an adaptive mathematical programming model in the framework of model predictive control. Each component models its stress environment implicitly by quantifying resource availability through a sensor and the programming model incorporates these modeled stress environments.

4. **Developed mechanisms to decentralize of the mathematical programming model**: The mathematical programming model we have built was decentralized through a multi-tier auction market for the purpose of improving scalability and robustness. Though the designed market gives an equivalent solution to the centralized programming model, it gives more benefits as computations and communications are distributed among multiple market participants. In the market, there are four different types of market participants: components, resource managers, brokers, and a seller.

5. **Developed a novel computational ecosystem model**: We designed a novel computational ecosystem model characterized by resource sharing and algorithm selection, and studied collective behavior of the model. The computational ecosystem model designed is a simplified version of the networks we study under the designed control mechanism, which captures main characteristics with respect to the interactions from resource sharing. As excess resource for some of agents can be utilized by others, the system can exhibit interesting collective dynamics. The fairness, allocating resources proportional to the required resources, was identified as a leading factor to stability and consequently optimality.

10.2 Future Research

Some of the possible future research extensions are as follows:

1. **Generalization of the problem**: In this research we restricted the methodology to the cases where the number of tasks for each component is large. When it is not that case, the method we have developed may not be effective. As we mentioned, the problem is NP-hard in general and there should be some efficient heuristic algorithm to address this generalized problem. That is, the mathematical programming model should be adjusted to take into account such cases in an efficient way.

2. **Network topology determination**: Though we addressed the topology determination problem in Appendix B, it is limited to the cases where there is no alternative algorithm. So, the methodology needs to be extended by taking into account the alternative algorithms also. It would be promising to design an auction market for determining the network topology in a similar way to auctioning for assignment problems.

3. **Consideration of multiple applications**: We considered a control problem for one application. However, there can be multiple applications under consideration. The applications may arrive in batch or in intervals with a certain distribution. In such cases, we need to devise higher-level resource allocation mechanism. The mathematical programming model we have built can be used as a basis to support such decision problems.

4. **Extension of the computational ecosystem model**: In the previous computational ecosystem model, diversity of agents was identified as a leading factor in controlling distributed systems. In our model framework fairness plays a crucial role. As the models are quite different it is natural to make different arguments. However, there is a chance to integrate the models so as to study wider range of collective dynamics. Individual agents choose not only among available resources but also among available algorithms. Through the integrated model framework we can study the roles of diversity and fairness in conjunction.

Bibliography

1 S. Jha and J. M. Wing, "Survivability analysis of networked systems," in *Proc. 23rd International Conference on Software engineering*, 2001, pp. 307-317.

2. R. Ellison, D. Fisher, H. Lipson, T. Longstaff, and N. Mead, "Survivable network systems: An emerging discipline," Software Engineering Institute, Carnegie Mellon University, Pittsburg, PA, Tech. Rep. CMU/SEI-97-153, 1997.

3. J. E. Eggleston, S. Jamin, T. P. Kelly, J. K. MacKie-Mason, W. E. Walsh, and M. P. Wellman, "Survivability through market-based adaptivity: The MARX project," in *Proc. DARPA Information Survivability Conference and Exposition*, 2000, pp. 145-156.

4. S. Bowers, L. Delcambre, D. Maier, C. Cowan, P. Wagle, D. McNamee, A. L. Meur, and H. Hinton, "Applying adaptation spaces to support quality of service and survivability," in *Proc. DARPA Information Survivability Conference and Exposition*, 2000, pp. 271-283.

5 O. F. Rana and K. Stout, "What is scalability in multi-agent systems?," in *Proc. 4th International Conference on Autonomous Agents*, 2000, pp. 56-63.

6. B. Meyer, "On to components," *IEEE Computer*, vol. 32, no. 1, pp. 139-140, 1999.

7. P. Clements, "From subroutine to subsystems: Component-based software development," in *Component Based Software Engineering*, A. W. Brown, Ed. IEEE Computer Society Press, pp. 3-6, 1996.

8. M. O. McCracken, A. Snavely, and A. Malony, "Performance modeling for dynamic algorithm selection," in *Proc. International Conference on Computational Science*, 2003, pp. 749-758.

9. P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, and A. L. Wolf, "An architecture-based approach to self-adaptive software," *IEEE Intelligent Systems*, vol. 14, no. 3, pp. 54-62, 1999.

10. F. M. T. Brazier, C. M. Jonker, and J. Treur, "Principles of component-based design of intelligent agents," *Data and Knowledge Engineering*, vol. 41, no. 1, pp. 1-28, 2002.

11. H. J. Goradia and J. M. Vidal, "Building blocks for agent design," in *Proc. 4th International Workshop on Agent-Oriented Software Engineering*, 2003, pp. 17-30.

12. R. Krutisch, P. Meier, and M. Wirsing, "The AgentComponent approach, combining agents and components," in *Proc. 1st German Conference on Multiagent System Technologies*, 2003, pp. 1-12.

13. D. Moore, W. Wright, and R. Kilmer, "Control surfaces for Cougaar," in *Proc. First Open Cougaar Conference*, 2004, pp. 37-44.

14. W. Peng, V. Manikonda, and S. Kumara, "Understanding agent societies using distributed monitoring and profiling," in *Proc. First Open Cougaar Conference*, 2004, pp. 53-60.

15. H. Gupta, Y. Hong, H. P. Thadakamalla, V. Manikonda, S. Kumara, and W. Peng, "Using predictors to improve the robustness of multi-agent systems: Design and implementation in Cougaar," in *Proc. First Open Cougaar Conference*, 2004, pp. 81-88.

16. D. Moore, A. Helsinger, and D. Wells, "Deconfliction in ultra-large MAS: Issues and a potential architecture," in *Proc. First Open Cougaar Conference*, 2004, pp. 125-133.

17. R. D. Snyder and D. C. Mackenzie, "Cougaar agent communities," in *Proc. First Open Cougaar Conference*, 2004, pp. 143-147.

18. A. P. Moore, R. J. Ellison, and R. C. Linger, "Attack modeling for information security and survivability," Software Engineering Institute, Carnegie Mellon University, Pittsburg, PA, Tech. Note CMU/SEI-2001-TN-001, 2001.

19. J. Regehr, "Some guidelines for proportional share CPU scheduling in general-purpose operating systems," Presented as a work in progress at 22nd IEEE Real-Time Systems Symposium, 2001.

20. F. Moberg, "Security analysis of an information system using an attack tree-based methodology," M.S. thesis, Automation Engineering Program, Chalmers University of Technology, Sweden, 2000.

21. G. Barto, S. J. Bradtke, and S. P. Singh, "Learning to act using real-time dynamic programming," *Artificial Intelligence*, vol. 72, no. 1-2, pp. 81-138, 1995.

22. 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.

23. L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237-285, 1996.

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

25. 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.

26. M. Nikolaou, "Model predictive controllers: A critical synthesis of theory and industrial needs," in *Advances in Chemical Engineering Series*, Academic Press, 2001.

27. S. J. Qin and T. A. Badgwell, "A survey of industrial model predictive technology," *Control Engineering Practice*, vol. 11, pp. 733-764, 2003.

28. Y. Lengwiler, "The multiple unit auction with variable supply," *Economic Theory*, vol. 14, no. 2, pp. 373-392, 1999.

29. T. Gonzalez and S. Sahni, "Flowshop and jobshop schedules: Complexity and approximation," *Operations Research*, vol. 26, pp. 36-52, 1978.

30. J. Lenstra, A. R. Kan, and P. Brucker, "Complexity of machine scheduling problems," *Annals of Discrete Mathematics*, vol. 1, pp. 343-362, 1977.

31. B. A. Huberman and T. Hogg, "The behavior of computational ecologies," in *The Ecology of Computation*, B. A. Huberman, Ed. Amsterdam: North-Holland, 1988, pp. 77-115.

32. J. O. Kephart, T. Hogg, and B. A. Huberman, "Dynamics of computational ecosystems," *Physical Review A*, vol. 40, no. 1, pp. 404-421, 1989.

33. T. Hogg and B. A. Huberman, "Controlling chaos in distributed systems," *IEEE Trans. Systems, Man and Cybernetics*, vol. 21, no. 6, pp. 1325-1332, 1991.

34. B. A. Huberman and T. Hogg, "The emergence of computational ecologies," in *1992 Lectures in Complex Systems*, L. Nadel and D. L. Stein, Eds. Addison-Wesley, 1993, pp. 185-205.

35. N. Glance, T. Hogg, and B. A. Huberman, "Computational ecosystems in a changing environment," *International Journal of Modern Physics*, vol. 2, no. 3, pp. 735-753, 1991.

36. I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*. San Francisco: Morgan Kaufmann Publishers, 1999.

37. R. Hamadi and B. Benatallah, "A Petri net-based model for web service composition," in *Proc. 14th Australasian Database Conference on Database technologies*, 2003, pp. 191-200.

38. I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "Grid services for distributed system integration," *IEEE Computer*, vol. 35, no. 6, pp. 37-46, 2002.

39. M. Weiser, "The computer for the 21st century," *Scientific American*, vol. 265, no. 3, pp. 94-104, 1991.

40. I. Stoica, H. Abdel-Wahab, J. Gehrke, K. Jeffay, S. K. Baruah, and C. G. Plexton, "A proportional share resource allocation algorithm for real-time, time-shared systems," in *Proc. 17th IEEE Real-Time Systems Symposium*, 1996, pp. 288-299.

41. C. A. Waldspurger and W. E. Weihl, "Lottery scheduling: Flexible proportional-share resource management," in *Proc. First Symposium on Operating System Design and Implementation*, 1994, pp. 1-11.

42. C. Waldspurger and W. Weihl, "Stride scheduling: Deterministic proportional-share resource management," Lab. for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, Tech. Rep. MIT/LCS/TM-528, 1995.

43. D. B. Lange, "Mobile objects and mobile agents: The future of distributed computing?," in *Proc. 12th European Conference on Object-Oriented Programming*, 1998, pp. 1-12.

44. D. Schoder and T. Eymann, "The real challenges of mobile agents," *Communications of the ACM*, vol. 43, no. 6, 2000, pp.111-112.

45. D. B. Lange and M. Oshima, "Seven good reasons for mobile agents," *Communications of the ACM*, vol. 42, no. 3, 1999, pp. 88-89.

46. D. Chess, C. Harrison, and A. Kershenbaum, "Mobile agents: Are they a good idea?," in *Mobile Object Systems: Towards the Programmable Internet*, Lecture Notes in Computer Science, vol. 1222, J. Vitek and C. Tschudin, Eds. Springer-Verlag, 1997, pp. 25–47.

47. O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on nonidentical processors," *Journal of the Association for Computing Machinery*, vol. 24, no. 2, pp. 280-289, 1977.

48. D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Transactions on Software Engineering*, vol. 15, no. 11, pp. 1427-1436, 1989.

49. T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 61, pp. 810-837, 2001.

50. D. M. Dilts, N. P. Boyd, and H. H. Whorms, "The evolution of control architectures for automated manufacturing system," *Journal of Manufacturing Systems*, vol. 10, no. 1, pp. 79-93, 1991.

51. F. Heylighen, "Self-organization, emergence and the architecture of complexity," in *Proc. First European Conference on System Science*, 1992, pp. 23-32.

52. J. P. Crutchfield, "The calculi of emergence," *Physica D*, vol. 75, pp. 11-54, 1994.

53. S. M. Manson, "Simplifying complexity: A review of complexity theory," *Geoforum*, vol. 32, no. 3, pp 405-414, 2001.

54. S. Clearwater, *Market-Based Control: A Paradigm for Distributed Resource Allocation*, World Scientific Publishing, 1996.

55. R. P. McAfee and J. Mcmillan, "Auctions and bidding," *Journal of Economic Literature*, vol. 25, pp. 699-738, 1987.

56. T. W. Malone, R. E. Fikes, K. R. Grant, and M. T. Howard, "Enterprise: A market-like task scheduler for distributed computing environments," in *The Ecology of Computation*, B. A. Huberman, Ed. Amsterdam: North-Holland, 1988, pp. 177-205.

57. A. Chavez, A. Moukas, and P. Maes, 1997, "Challenger: A multi-agent system for distributed resource allocation," in *Proc. First International Conference on Autonomous Agents*, 1997, pp. 323-331.

58. D. Ferguson, Y. Yemini, and C. Nikolaou, "Microeconomic algorithms for load balancing in distributed computer systems," in *Proc. 8th International Conference on Distributed Systems*, 1988, pp. 491-499.

59. C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and W. S. Stornetta, "Spawn: A distributed computational economy," *IEEE Transactions on Software Engineering*, vol. 18, no. 2, pp. 103-117, 1992.

60. D. Morley. "Painting trucks at general motors: The effectiveness of a complexity based approach," in *Embracing Complexity: Exploring the Application of Complex Adaptive Systems to Business*, The Ernst and Young Center for Business Innovation, 1996, pp. 53-58.

61. M. A. Gibney, N. R. Jennings, N. J. Vriend, and J.-M. Griffiths, "Market-based call routing in telecommunications networks using adaptive pricing and real bidding," *Lecture Notes in Computer Science*, vol. 1699, pp. 46-61, 1999.

62. M. A. Gibney and N. R. Jennings, "Dynamic resource allocation by market-based routing in telecommunications networks," in *Proc. 2nd International Workshop on Multi-Agent Systems and Telecommunications*, 1998, pp. 102-117.

63. K. H. Low, W. K. Leow, and M. H. Ang, Jr., "Task allocation via self-organizing swarm coalitions in distributed mobile sensor network," in *Proc. 19th National Conference on Artificial Intelligence*, 2004, pp. 28-33.

64. K. Prouskas, A. Patel, J. Pitt, J. Barria, "A Multi-agent system for intelligent network load control using a market-based approach," in *Proc. Fourth International Conference on Multi-Agent Systems*, 2000, pp. 231-.

65. R. E. Bellman, 1957, *Dynamic Programming*, Princeton University Press.

66. E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press, 1999.

67. M. Dorigo, V. Maniezzo, and A. Colorni, "Positive feedback as a search strategy," Dipartimento di Elettronica, Politecnico di Milano, Italy, Tech. Rep. 91-016, 1991.

68. R. Schoonderwoerd, O. Holland, J. Bruten, and L. Rothkrantz, "Ant-based load balancing in telecommunications networks," *Adaptive Behavior*, vol. 5, no. 2, pp. 169-207, 1996.

69. G. Di Caro and M. Dorigo, "An adaptive multi-agent routing algorithm inspired by ants behavior," in *Proc. Fifth Annual Australasian Conference on Parallel and Real-Time Systems*, 1998.

70. V. Cicirello and S. Smith, "Ant colony control for autonomous decentralized shop floor routing", in *Proc. Fifth International Symposium on Autonomous Decentralized Systems*, 2001, pp. 383-390.

71. G. Di Caro, and M. Dorigo, "Two Ant Colony Algorithms For Best-Effort Routing In Datagram Networks," in *Proc. Tenth International Conference on Parallel and Distributed Computing and Systems*, 1998.

72. E. Bonabeau, A. Sobkowski, G. Theraulaz, and J.-L. Deneubourg, "Adaptive task allocation inspired by a model of division of labor in social insects," in *Bio Computation and Emergent Computing*, D. Lundh and B. Olsson, Eds. World Scientific, 1997, pp. 36-45.

73. V. Cicirello and S. Smith, "Wasp nests for self-configurable factories," in *Proc. Fifth International Conference on Autonomous Agents*, 2001.

74. V. Cicirello and S. Smith, "Distributed coordination of resources via wasp-like agents," in *Proc. First NASA GSFC/JPL Workshop on Radical Agent Concepts*, 2002.

75. V. Cicirello and S. Smith, 2001, "Wasp-like agents for distributed factory coordination," Robotics Institute, Carnegie Mellon University, Tech. Rep. CMU-RI-TR-01-39.

76. R. S. Sutton, A. G. Barto, and R. J. Williams, "Reinforcement learning is direct adaptive optimal control," *IEEE Control Systems*, vol. 12, no. 2, 1992.

77. M. Littman and J. Boyan, 1993, "A distributed reinforcement learning scheme for network routing," Computer Science Department, Carnegie Mellon University, Tech. Rep. CS-93-165, 1993.

78. J. Boyan and M. Littman, "Packet routing in dynamically changing networks: a reinforcement learning approach," *Advances in Neural Information Processing Systems*, vol. 6, pp. 671-678, 1994.

79. S. Kumar and R. Miikkulainen, "Dual reinforcement Q-routing: An on-line adaptive routing algorithm," in *Proc. Artificial Neural Networks in Engineering Conference*, 1997.

80. S. Kumar, and R. Miikkulainen, "Confidence-based Q-routing: An on-line adaptive routing algorithm", in *Proc. Artificial Neural Networks in Engineering Conference*, 1998.

81. S. Kumar, "Confidence based dual reinforcement Q-routing: An on-line adaptive network routing algorithm," Department of Computer Sciences, The University of Texas at Austin, Tech. Rep. AI98-267, 1998.

82. A. Schaerf, Y. Shoham, and M. Tennenholtz, "Adaptive load balancing: A study in multiagent learning", *Journal of Artificial Intelligence Research*, vol. 2, pp. 475-500, 1995.

83. D. H. Wolpert, K. Tumer, and J. Frank, **"**Using collective intelligence to route Internet traffic," in *Advances in Neural Information Processing Systems*, vol. 11, pp. 952-958, 1999.

84. K. Tumer and D. Wolpert, "Collective intelligence and Braess' paradox," in *Proc. Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, 2000, pp. 104-109.

85. D. H. Wolpert, S. Kirshner, C. J. Merz, and K. Tumer, "Adaptivity in agent-based routing for data networks," in *Proc. fourth International Conference on Autonomous Agents*, 2000, pp. 396-403.

86. F. W. Molina, "A survey of resource directive decomposition in mathematical programming," *Computing Surveys*, vol. 11, no. 2, pp. 95-104, 1979.

87. G. Danzig and P. Wolfe, "The decomposition algorithm for linear programs," *Econometrica*, vol. 19, pp. 767-778, 1961.

88. J. F. Benders, "Partitioning procedures for solving mixed variables programming problems," *Numerische mathematik*, vol. 4, pp. 238-252, 1962.

89. A. M. Geoffrion, "Generalized Benders decomposition," *Journal of Optimization Theory and Applications*, vol. 10, no. 4, pp. 237-260, 1972.

90. J. D. Thomas and K. Sycara, "Heterogeneity, root-finding, and decentralization," in *Proc. Artificial Societies and Computational Markets Workshop at the Second International Conference on Autonomous Agents*, 1998.

91. W. Vickrey, "Counterspeculation, auction, and competitive sealed tenders," *Journal of Finance*, vol. 16, pp. 8-37, 1961.

92. D. P. Bertsekas, "A new algorithm for the assignment problem," *Mathematical Programming*, vol. 21, no. 1, pp. 152-171, 1981.

93. D. P. Bertsekas, "The auction algorithm for assignment and other network flow problems: A tutorial," *Interface*, vol. 20, no. 4, pp. 133–149, 1990.

94. D. P. Bertsekas, "The auction algorithm for shortest paths," *SIAM journal on Optimization*, vol. 1, pp. 425-447.

95. J. Thomas and K. Sycara, "Heterogeneity, stability, and efficiency in distributed systems," in *Proc. International Conference on Multi-Agent Systems*, 1998, pp. 293-.

96. S. Gadaleta and G. Dangelmayr, "Optimal chaos control through reinforcement learning," *Chaos*, vol. 9, no. 3, pp. 775-788, 1999.

97. S. Gadaleta and G. Dangelmayr, "Control of 1-D and 2-D coupled map lattices through reinforcement learning," in *Proc. Second International Conference on Control of Oscillations and Chaos*, vol. 1, pp. 109-112, 2000.

98. R. Der and M. Herrmann, "Q-learning chaos controller," *IEEE International Conference on Neural Networks*, vol. 4, pp. 2472-2475, 1994.

99. C. Lin and C. Jou, "Controlling chaos by GA-based reinforcement learning neural network," *IEEE Transactions on Neural Networks*, vol. 10, no. 4, pp. 846-859, 1999.

100. P. Milgrom, "Auctions and bidding: A primer," *Journal of Economic Perspectives*, vol. 3, no. 3, pp. 3-22, 1989.

101. N. R. Jennings, K. Sycara, and M. Wooldridge, "A roadmap of agent research and development," *Autonomous Agents and Multi-Agent Systems*, vol. 1, pp. 7-38, 1998.

102. M. Wooldridge and N. R. Jennings, "Intelligent agents: Theory and practice," *The knowledge Engineering Review*, vol. 10, no. 2, pp. 115-152, 1995.

103. D. Good and W. Young, "Mathematical methods for digital systems development," in *Proc. Fourth International Symposium of VDM Europe*, vol. 2, pp. 406-430, 1991.

104. C. J .C .H. Watkins, "Learning from delayed rewards," PhD thesis, King's college, Cambridge, UK, 1989.

105. E. Bonabeau, G. Théraulaz, and J.-L. Deneubourg, "Quantitative study of the fixed threshold model for the regulation of division of labour in insect societies," in *Proc. Royal Society of London B*, vol. 263, pp. 1565-1569, 1996.

106. S. J. Bradtke and M. O. Duff, "Reinforcement learning methods for continuous-time markov decision problems," in *Proc. 7th Annual Conference on Neural Information Processing Systems*, 1994, pp. 393-400.

107. C. Waldspurger, "Lottery and stride scheduling: Flexible proportional share resource management," Ph.D. dissertation, Lab. for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1995.

108. J. B. Rosen and J. C. Orena, "Solution of nonlinear programming problems by partitioning," *Management System*, vol. 10, no. 1, pp. 93-98, 1963.

109. J. B. Rosen, "Primal partition programming for block diagonal matrices," *Numerische Mathematik*, vol. 6, no. 3, pp. 250-260, 1964.

110. D. P. Bertsekas, "Auction algorithms for network flow problems: A tutorial introduction," *Computational Optimization and Applications*, vol. 1, pp. 7-66, 1992.

111. M. P. Wellman, "A market-oriented programming environment and its application to distributed multicommodity flow problems," *Journal of Artificial Intelligence Research*, vol. 1, pp. 1-23, 1993.

112. M. P. Wellman, "A computational market model for distributed configuration design," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 9, no. 2, pp.125-134, 1995.

113. J. Q. Cheng and M. P. Wellman, "The WALRAS algorithm: A convergent distributed implementation of general equilibrium outcomes," *Computational Economics*, vol. 12, no. 1, pp. 1-24, 1998.

114. M. P. Wellman, W. E. Walsh, P. R. Wurman, and J. K. MacKie-Mason, "Auction protocols for decentralized scheduling," *Games and Economic Behavior*, vol. 35, pp. 271-303, 2001.

115. M. Guignard, "Lagrangean Relaxation," *TOP*, vol. 11, no. 2, pp. 151-228, 2003.

116. R. J. Jose and L. H. Ungar, "Auction-driven coordination for plantwide optimization," in Proc. *Foundations of Computer-Aided Process Operation*, 1998.

117. K. Ertogral and S. D. Wu, "Auction-theoretic coordination of production planning in the supply chain," *IIE Transactions*, vol. 32, no. 10, pp. 931-940, 2000.

118. P. R. Wurman, M. P. Wellman, and W. E. Walsh, "The Michigan Internet AuctionBot: A configurable auction server for human and software agents," in *Proc. 2nd International Conference on Autonomous Agents*, 1998, pp. 301-308.

Appendix A

Emergent Properties of Proportional Allocation

The proportional resource allocation policy has several emergent properties that can be found in many self-organizing systems such as social or biological systems. First, it is localized as each node can make decisions independent of others. Second, it requires almost no computation. Third, nevertheless the network can achieve a desirable performance. Fourth, it is adaptive to the stress environments without explicit considerations as will be shown as follows.

Suppose there are some stressors sharing resources with the components. We denote ω_n^s as the amount of shared resource by a stressor in node n. Then, the lower bound performance T_s^{LB} under stress is given by (**A.1**). We denote the completion time under stress as T_s'.

$$T_s^{LB} = \underset{n \in N}{Max} \frac{\omega_n + \omega_n^s}{\omega_n} \sum_{i \in K_n} LI_i \qquad \textbf{(A.1)}$$

Theorem 3. *T_s' equals to T_s^{LB} under proportional allocation.*

Proof. *$RA_i(t)$* becomes:

$$RA_i(t) \geq \frac{w_i(t)}{\omega_{n(i)} + \omega_{n(i)}^s} \quad for\ t \geq 0\ . \qquad \textbf{(A.2)}$$

Then, (**5.9**) results in (**A.3**) under proportional allocation.

$$\frac{T_s^{LB}}{L_i} \geq S_i(t) \quad for\ t \geq 0 \qquad \textbf{(A.3)}$$

Therefore, the network completes at T_s^{LB} under proportional allocation.

Theorem 3 depicts that the proportional allocation policy is optimal independent of the stress environments. Though we do not consider them explicitly, the policy gives lower bound performance adaptively. This characteristic is especially important when the system is vulnerable to unpredictable or unidentifiable stress environments.

The emergent properties hold in the limit of large number of tasks. As the term "large" is obscure we need to give it a concrete definition. We define it with an adequacy criterion, by which one can evaluate if the desirable properties of the proportional allocation hold for a given network. For this purpose we characterize upper bound performance of a network under proportional allocation.

Theorem 4. *Under proportional allocation a network's upper bound T^{UB} of completion time T is given by:*

$$T^{UB} = T^{LB} + \underset{e\in E}{Max}\ \underset{j\in S_e}{Max} \sum_{i\in j} [P_i \sum_{p\in K_{n(i)}} LI_p / LI_i], \qquad \textbf{(A.4)}$$

where E denotes a set of components which have no successor and S_e a set of task paths to component e. A task path to component e is a set of components in a path from a component with no predecessor to component e and does not include component e.

Proof. From (**5.9**) we can induce the lowest upper bound S_i^{UB} of $S_i(t)$ as:

$$S_i^{UB} = P_i \sum_{p\in K_{n(i)}} LI_p / LI_i . \qquad \textbf{(A.5)}$$

So, a component i can complete by T^{LB} and generate tasks at a constant interval of T^{LB}/L_i from $t=S_i^{UB}$ when it receives tasks at a constant interval of T^{LB}/L_i from $t=0$. Now, consider component i's successor s which has only one predecessor. As the successor receives tasks at a constant interval of T^{LB}/L_s from $t=S_i^{UB}$ or more preferably, it can complete by $S_i^{UB}+T^{LB}$. So, a component $e\in E$ (with no successor) can receive tasks at a constant interval of T^{LB}/L_e from maximal task traveling time to the component of:

$$\underset{j\in S_e}{Max} \sum_{i\in j} S_i^{UB} \qquad \textbf{(A.6)}$$

(note that a path j does *not* include component e) or more preferably so that its completion time T_e is bounded as:

$$T_e \le T^{LB} + \underset{j \in S_e}{Max} \sum_{i \in j} S_i^{UB} \quad . \qquad \textbf{(A.7)}$$

And, the upper bound of T is the maximal of the bounds.

Though we formulated the upper bound performance without considering stress environments, one can easily modify it so that the upper bound performance can reflect the stress environments (if each ω_n^s is identifiable or assumable). The adequacy criterion is defined as the ratio between T^{LB} and T^{UB} as in (**A.8**). When the criterion is close to one, a network can achieve the lower bound performance using the proportion allocation policy. Typically, the criterion converges to one as each L_i increases. However, as the criterion approaches zero, the policy become more and more inadequate. The example network in Figure **5.1** is quite adequate because the network's adequacy is 0.99 (300/303).

$$Adequacy = \frac{T^{LB}}{T^{UB}} \qquad \textbf{(A.8)}$$

Appendix B

Topology Determination

In this appendix, we develop an efficient method of quantifying the minimal completion time for distributed component networks. For a given set of resources and components, the performance can vary depending on the way of utilizing distributed heterogeneous resources. *Network topology* assigns components to available machines with a set of constraints. Some components may not be separable to different machines and may be allowed to specific machines. Given a network topology, there can be multiple components in a machine sharing the machine's resources together. These two control facilities determine the performance of a network and the minimal completion time represents achievable QoS by a set of resources and components.

Similar problems can be found in the multiprocessor scheduling literature. There is a set of components with a task flow structure between them and each component without predecessors has one root task. Each component processes exactly one task only after all of its predecessors complete their tasks. A multiprocessor scheduling is composed of an assignment of components to machines (network topology) and a sequence of components for each machine (resource allocation). However, our problem is different especially because a component in our networks can have multiple tasks to process, i.e., a component can process tasks in parallel with its successors or predecessors. The easiest multiprocessor scheduling problem is when components are independent, i.e., there is no task flow between components. However, this problem is known as NP-complete [**47**][**48**]. Considering that the task flow structure of our networks is arbitrary and each component can have multiple tasks to process, our scheduling problem is even harder.

In this context, the method designed here is a *heuristic* which is applicable to the cases where the number of tasks to be processed by each component is large. Though the increase of the number of tasks adds more complexity, it can give us great opportunity to develop an efficient heuristic. Also, our method addresses *resource reservation*. When different applications share resources together, their performance can be guaranteed through the resource reservation. The method quantifies the minimal completion time by incorporating the resource reservations of other applications and also enables to make the resource reservations for the networks under consideration.

B.1 Problem Statement

In this section we detail network topology and resource allocation.

B.1.1 Network Topology

There is a set $K = \{k: k \in K\}$ of available machines and $P_i(k)$ represents CPU time per task of component i at machine k reflecting computation speed difference between machines. Considering that some of the components may not be separable to different machines, we define a set $J = \{j: j \in J\}$ of clusters and denote the components of a cluster j as M_j. Each component is a member of one of the clusters and the components in a cluster should be assigned to the same machine. Each cluster can be assigned to a set of machines and we denote the assignable machine set of cluster j as N_j. We define topology variable set $\boldsymbol{X} = \{x_{jk}: j \in J, k \in K\}$ in which x_{jk} is 1 if cluster j is assigned to machine k and 0 otherwise. The constraints of topology variables are as in (**B.1**).

- **Network topology constraints**

$$\begin{aligned} &\sum_{k \in N_j} x_{jk} = 1 \qquad for\ all\ \ j \in J \\ &\sum_{k \notin N_j} x_{jk} = 0 \qquad for\ all\ \ j \in J \\ &x_{jk} \in \{0,1\} \qquad for\ all\ \ j \in J\ and\ k \in K \end{aligned} \tag{B.1}$$

B.1.2 Resource Allocation

We denote the components assigned to machine k as $S_{I[k]}$ and the clusters assigned to machine k as $S_{J[k]}$. If ω_k^a of total managed weight ω_k is available to assign in machine k (i.e. ω_k-ω_k^a is reserved by other applications), the constraints of resource allocation variables for a given topology are as in (**B.2**).

- **Resource allocation constraints**

$$\sum_{i \in S_{I[k]}} w_i(t) \le \omega_k^a \qquad for\ all\ \ k \in K \tag{B.2}$$

B.1.3 Problem Definition

As the completion time T is a function of network topology (***X***) and resource allocation (***w***), the objective is to quantify the minimal completion time T^* represented in (**B.3**) with the constraints of (**B.1**) and (**B.2**).

$$T^* = \underset{X,w}{\text{Min}}\ T \tag{B.3}$$

B.2 Solution Methodology

As stated earlier, we design a method of quantifying the minimal completion time by limiting to the cases where the number of tasks to be processed by each component is large. In this section, we formulate the problem and provide a heuristic algorithm for solving the problem formulation.

B.2.1 Problem Formulation

For a given topology, the minimal weights required to achieve T^{LB} are constants over time as in (**B.4**) referring to Theorem 1 and the summation of these weights for each machine forms the required amount ω_k^r of resource reservation in the machine as in (**B.5**). Note that ω_k^r is less than or equal to ω_k^a satisfying the resource allocation constraints in (**B.2**).

- **Constant resource allocation**

$$w_i = \omega_{k(i)} \frac{LI_i}{T^{LB}} \quad for\ all\ i \in I\ and\ t \geq 0 \qquad (\mathbf{B.4})$$

- **Resource reservation**

$$\omega_k^r = \frac{\omega_k}{T^{LB}} \sum_{i \in S_{I[k]}} LI_i \quad for\ all\ k \in K \qquad (\mathbf{B.5})$$

As CPU time per task is machine-dependent we rewrite the load index as a function of machine as:

$$LI_i(k) = L_i P_i(k). \qquad (\mathbf{B.6})$$

Considering that the components in a cluster cannot be assigned to separate machines, we define *Cluster Load Index* $CLI_j(k)$ as:

$$CLI_j(k) = \sum_{i \in M_j} LI_i(k). \qquad (\mathbf{B.7})$$

Then, under the constant resource allocation, the completion time for a given topology can be estimated by:

$$\operatorname*{Max}_{k \in K} \frac{\omega_k}{\omega_k^a} \sum_{j \in S_{J[k]}} CLI_j(k). \qquad (\mathbf{B.8})$$

Consequently, the minimal completion time T^* can be formulated as in (**B.9**) by incorporating topology variables and constraints in (**B.1**).

- **Topology problem formulation**

$$
\begin{aligned}
T^* &= \text{Min}\,\text{Max}_{k\in K}\ \frac{\omega_k}{\omega_k^a}\sum_{j\in J} CLI_j(k)\,x_{jk} \\
s.t.\quad & \sum_{k\in N_j} x_{jk} = 1 \qquad for\ all\ \ j\in J \\
& \sum_{k\notin N_j} x_{jk} = 0 \qquad for\ all\ \ j\in J \\
& x_{jk}\in\{0,1\} \qquad for\ all\ \ j\in J\ and\ k\in K
\end{aligned}
\tag{B.9}
$$

B.2.2 Heuristic Solution

The formulation has a simplistic form because it is completely separated from resource allocation variables. As a result, the formulation can be mapped into the easiest multiprocessor scheduling problem, i.e., an assignment of independent clusters to machines. As discussed, this problem is NP-complete and there are diverse heuristic algorithms available in the literature. Eleven heuristics were selected and examined with various problem configurations in [**49**]. They are Opportunistic Load Balancing, Minimum Execution Time, Minimum Completion Time, Min-min, Max-min, Duplex, Genetic Algorithm, Simulated Annealing, Genetic Simulated Annealing, Tabu, and A*. Though Genetic Algorithm always gave the best performance, if algorithm execution time is also considered, it was shown that the simple Min-min heuristic performs well in comparison to others. So, we recommend the Min-min heuristic as an algorithm for solving the problem formulation. By adapting to our context the Min-min heuristic is as follows.

- **Min-min heuristic algorithm**

Step 1: Initialize a set of all unassigned clusters, $U \leftarrow J$, and current machine-level completion times, $mc(k) \leftarrow 0$ for all $k \in K$.

Step 2: Compute the minimal completion time after assignment for each unassigned cluster, $M=\{\min_{k\in N_j}[\frac{\omega_k}{\omega_k^a} CLI_j(k) + mc(k)] : j \in U\}$.

Step 3: Select the minimal from M, $mmc \leftarrow \min M$, and find corresponding cluster and machine, c and m respectively.

Step 4: Assign c to m and update $mc(m)$, $mc(m) \leftarrow mc(m)+mmc$.

Step 5: Remove c from U.

Step 6: If $U=\varnothing$ then go to step 7. Otherwise go to step 2.

Step 7: $T^* \leftarrow \max_{k \in K} mc(k)$.

B.3 Empirical Results

We ran several experiments through discrete-event simulation to validate the designed method.

B.3.1 Network Description

The network is composed of eight components in four clusters as in Table **B.1**. Task flow structure between components is described in Figure **B.1**. There are three available machines $\{K_1, K_2, K_3\}$ with $\omega_k=\omega_k^a=1$ for all k, and each cluster is assignable to any machine.

Table **B.1**: Experimental network parameters

Component	rt_i	$P_i(k)$[a]	Cluster
I_1	0	4	J_1
I_2	0	12	J_2
I_3	0	4	J_3
I_4	0	8	J_3
I_5	200	2	J_4
I_6	200	2	J_4
I_7	200	2	J_4
I_8	200	2	J_4

[a] for all $k \in K$

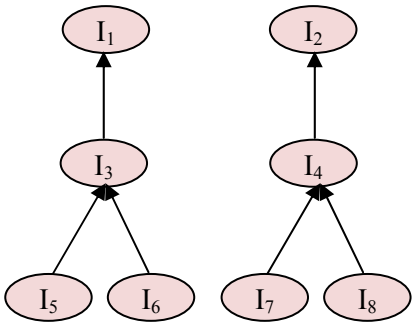
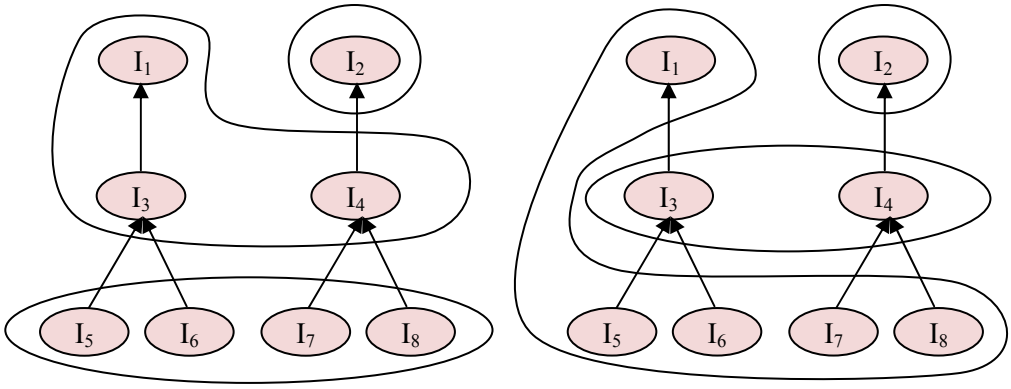


Figure **B.1**: Experimental task flow structure for topology determination problem

B.3.2 Performance Evaluation

The Min-min heuristic algorithm gives T^*=4800 and the resulting topology is as in Figure **B.2(b)**. The heuristic solution is equivalent to the exact solution of (**B.9**) in this experimental network.



(a) Non-optimal *(b) Optimal*

Figure **B.2**: Experimental network topologies

We set up eight different experimental conditions by combining three independent factors as shown in Table **B.2**. We use two different network topologies as in Figure **B.2**, which are non-optimal and optimal topologies. Two resource allocation policies are used: round-robin allocation and constant allocation. In round-robin allocation the components in each machine are assigned

equal weights and in constant allocation according to the components' load indices as in (**B.4**). To implement PS scheduling we use a weighted round-robin scheduling in which CPU time received by each component in a round is equal to its assigned weight. Also, the distribution of $P_i(k)$ can be deterministic or stochastic. While using stochastic distribution we repeat 5 experiments.

Table **B.2**: Experimental design

Condition	Topology	Resource allocation	$P_i(k)$
Con1	Non-optimal	Round-Robin	Deterministic
Con2	Non-optimal	Round-Robin	Exponential
Con3	Non-optimal	Constant	Deterministic
Con4	Non-optimal	Constant	Exponential
Con5	Optimal	Round-Robin	Deterministic
Con6	Optimal	Round-Robin	Exponential
Con7	Optimal	Constant	Deterministic
Con8	Optimal	Constant	Exponential

Numerical results from the experimentation are shown in Table **B.3**. The last two conditions (Con7 and Con8), which use the optimal network topology and constant resource allocation, gives a performance close to T^* and outperforms other conditions significantly. Also, constant allocation for both non-optimal (Con3 and Con4) and optimal (Con7 and Con8) topologies, gives a performance superior to round-robin allocation and close to lower bound performance T^{LB} in both deterministic and stochastic environments. These facts support the optimality of the constant resource allocation and consequently the validity of the method of quantifying the minimal completion time.

Table **B.3**: Experimental results

Condition	T^{LB}	T^*	Actual T	$\%$ [a]
Con1	6400	4800	7215	150.3
Con2	6400	4800	7314	152.4
Con3	6400	4800	6416	133.7
Con4	6400	4800	6404	133.4
Con5	4800	4800	5619	117.1
Con6	4800	4800	5645	117.6
Con7	4800	4800	4820	100.4
Con8	4800	4800	4899	102.1

[a] $\% = 100 * T / T^*$

B.3.3 Resource Reservation

The resource reservations required in the optimal topology are [$\omega_{K1}{}^{r}$=0.667, $\omega_{K2}{}^{r}$=1, $\omega_{K3}{}^{r}$=1] computed from (**B.5**). Our argument is that the network can achieve the optimal performance T^{*} with these reservations even though unreserved resources are allocated to other applications. To validate this, we use eleven different reservations for machine K_1 as shown in Table **B.4**. In each condition, $\omega_{K1}{}^{r}$ is allocated proportional to the load indices of the residing components and unreserved resources are assigned to an application which has infinite work (continuously requiring resources). The numerical results are shown in Table **B.4** and Figures **B.3** to **B.4**. In overall, the completion time decreases as $\omega_{K1}{}^{r}$ increases. However, when $\omega_{K1}{}^{r}$ is greater than 0.667, there is no significant advantage in deterministic environment. In contrast, the threshold in stochastic environment is somewhere between 0.667 and 0.7. Considering that the other applications may not require resources continuously, such a slight difference ($\leq$ 0.033) does not seem to be significant.

Table **B.4**: The effects of resource reservation

$\omega_{K1}{}^{r}$	Actual T	
	Deterministic $P_i(k)$	Exponential $P_i(k)$
0.1	32284	34502
0.2	16132	16605
0.3	10746	11069
0.4	8057	8237
0.5	6439	6635
0.6	5369	5503
0.667	4829	5135
0.7	4827	4941
0.8	4824	4965
0.9	4822	4984
1.0	4820	4946

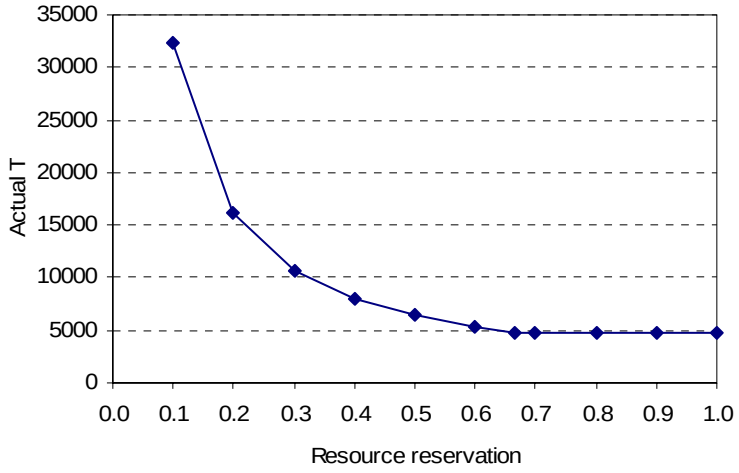


Figure **B.3**: The effects of resource reservation in deterministic environment.

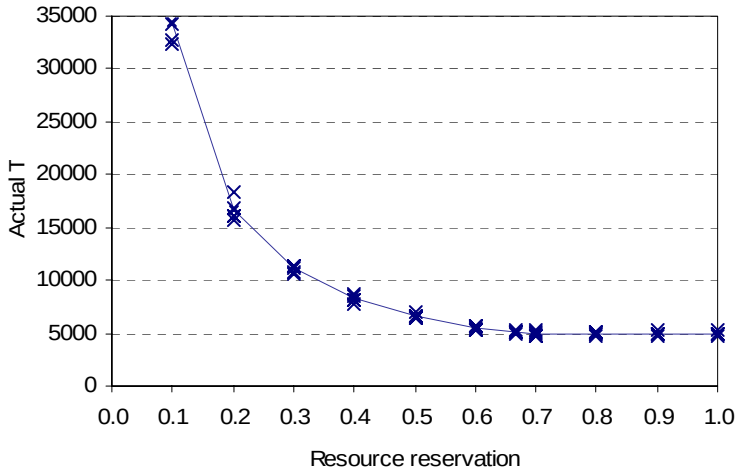


Figure **B.4**: The effects of resource reservation in stochastic environment.

Appendix C

System Behavior in No Stress Environments

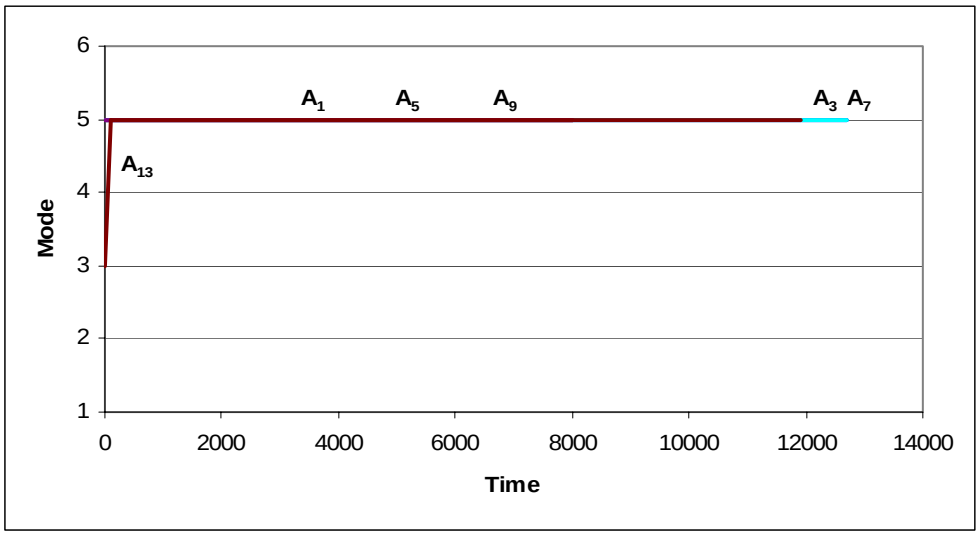


Figure **C.1**: Behavior of v_i^* under RCYY in deterministic environment with no stress (Con1-1)

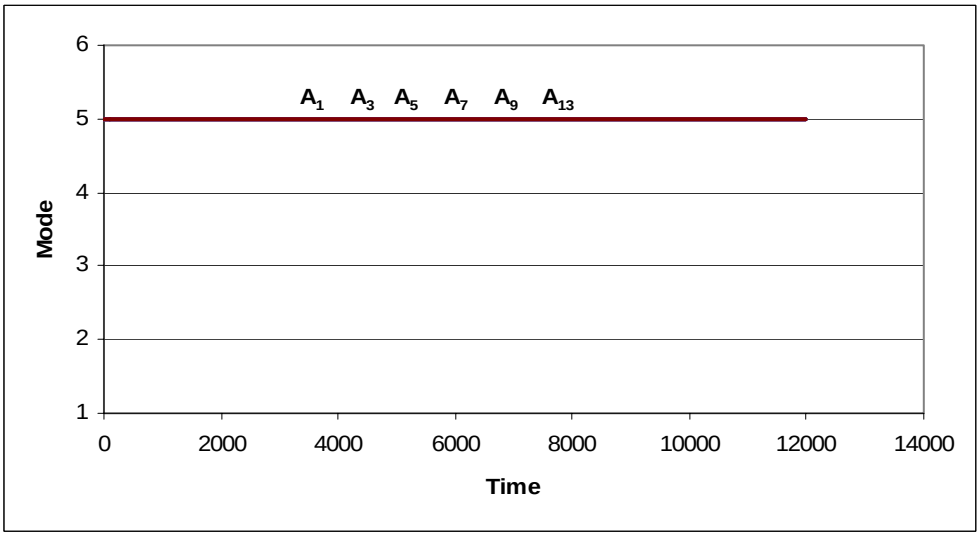


Figure **C.2**: Behavior of v_i^* under PCNN in deterministic environment with no stress (Con1-1)

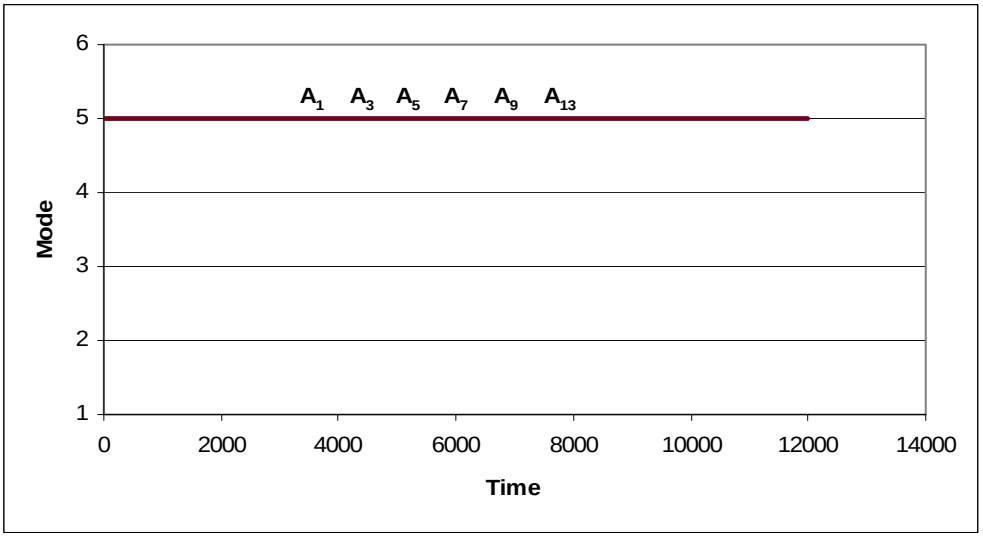


Figure **C.3**: Behavior of v_i^* under PCYN in deterministic environment with no stress (Con1-1)

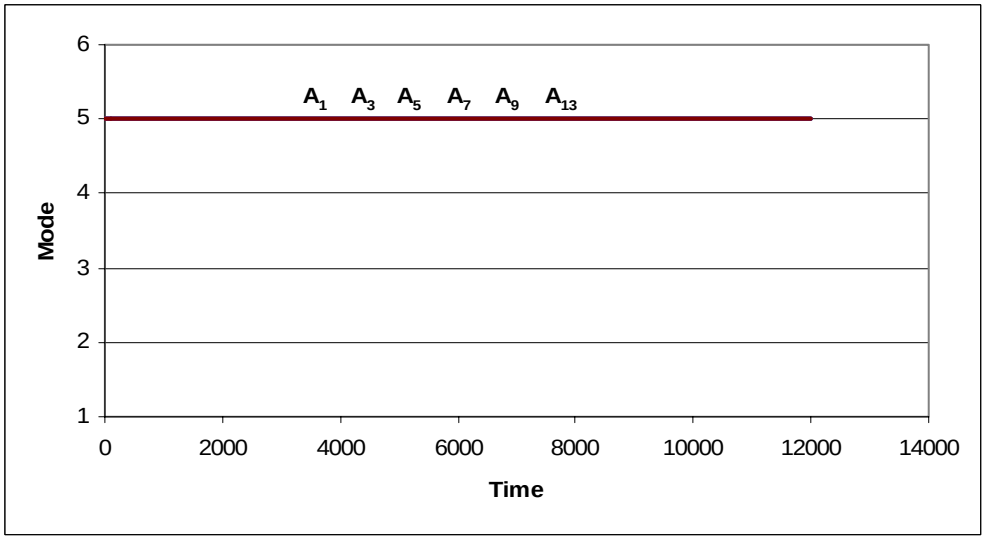


Figure **C.4**: Behavior of v_i^* under PCYY in deterministic environment with no stress (Con1-1)

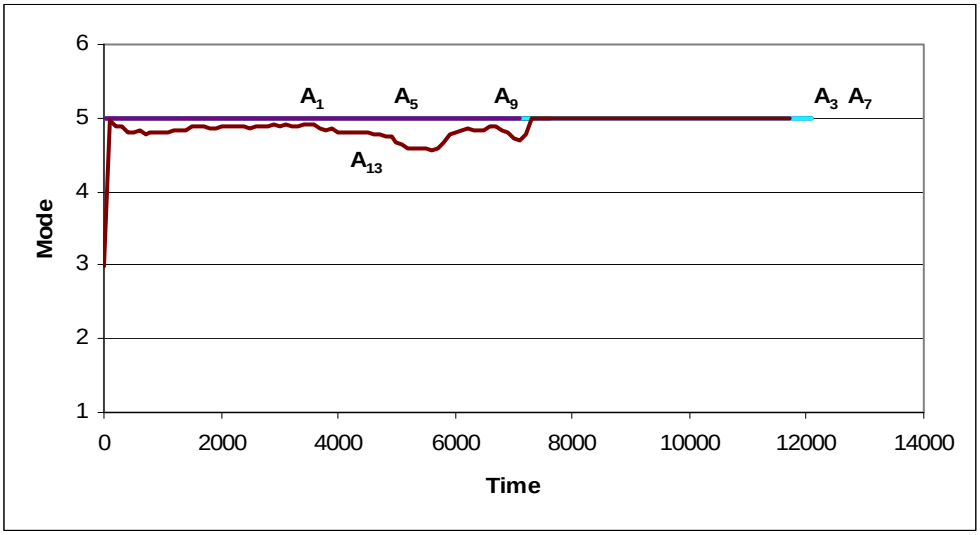


Figure **C.5**: Behavior of v_i^* under RCYY in stochastic environment with no stress (Con1-2)

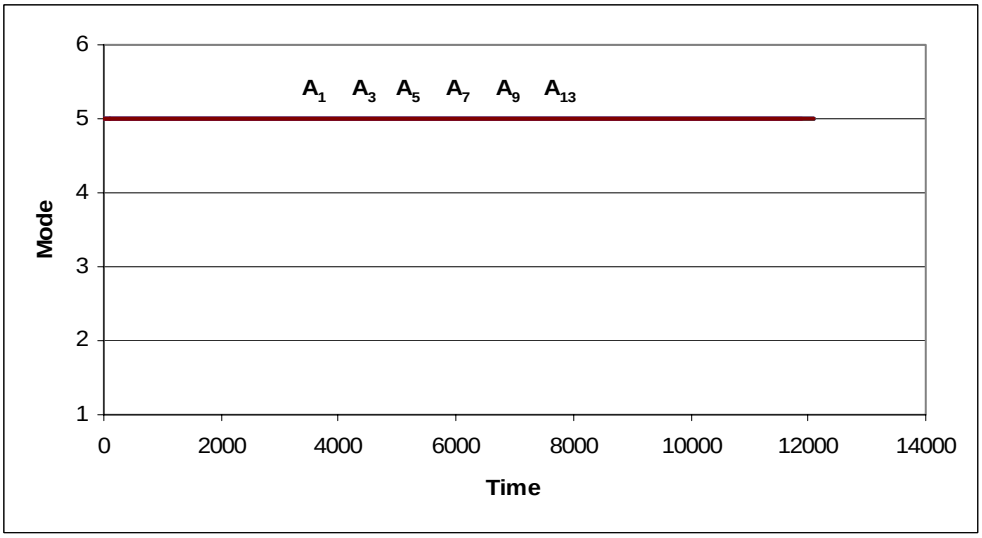


Figure **C.6**: Behavior of v_i^* under PCNN in stochastic environment with no stress (Con1-2)

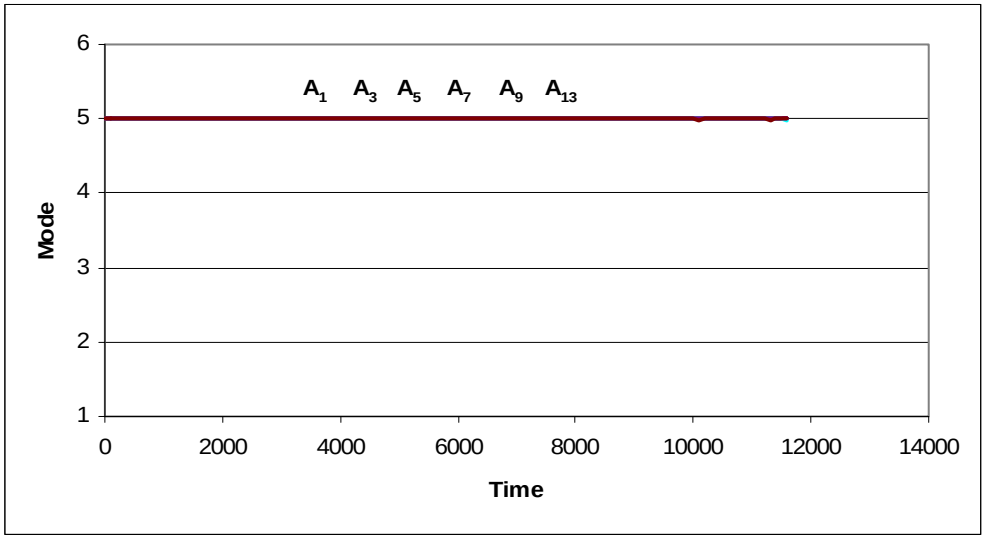


Figure **C.7**: Behavior of v_i^* under PCYN in stochastic environment with no stress (Con1-2)

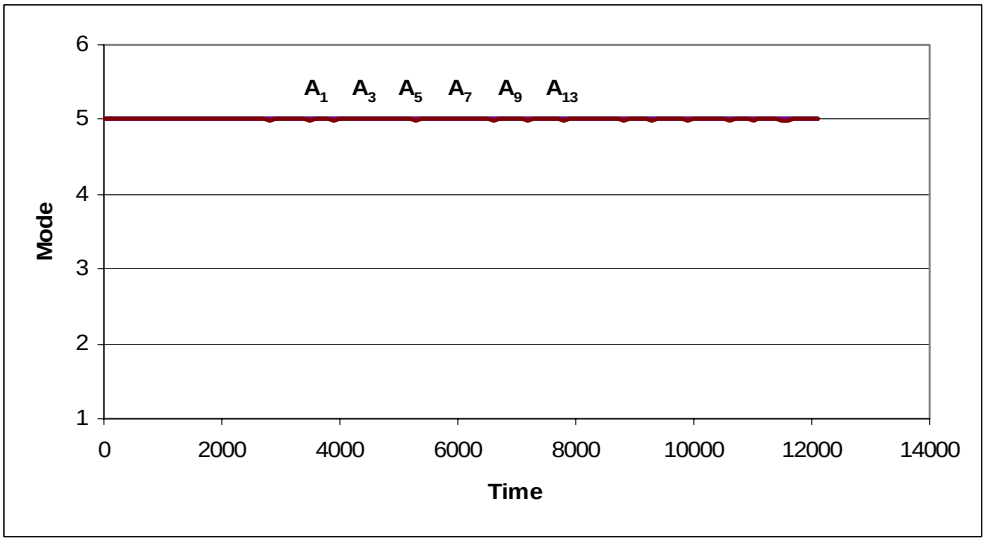


Figure **C.8**: Behavior of v_i^* under PCYY in stochastic environment with no stress (Con1-2)

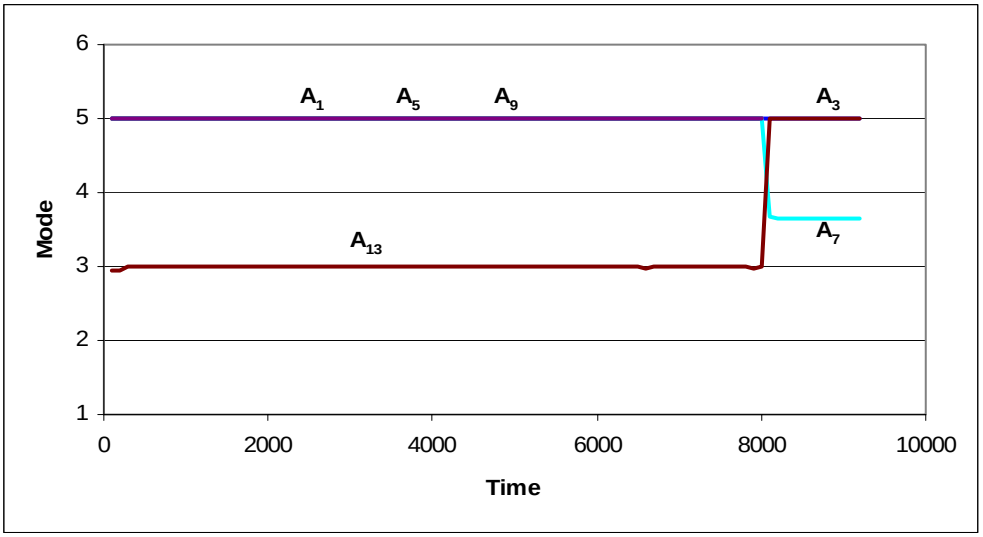


Figure **C.9**: Behavior of v_i^* under RCYY in deterministic environment with no stress (Con2-1)

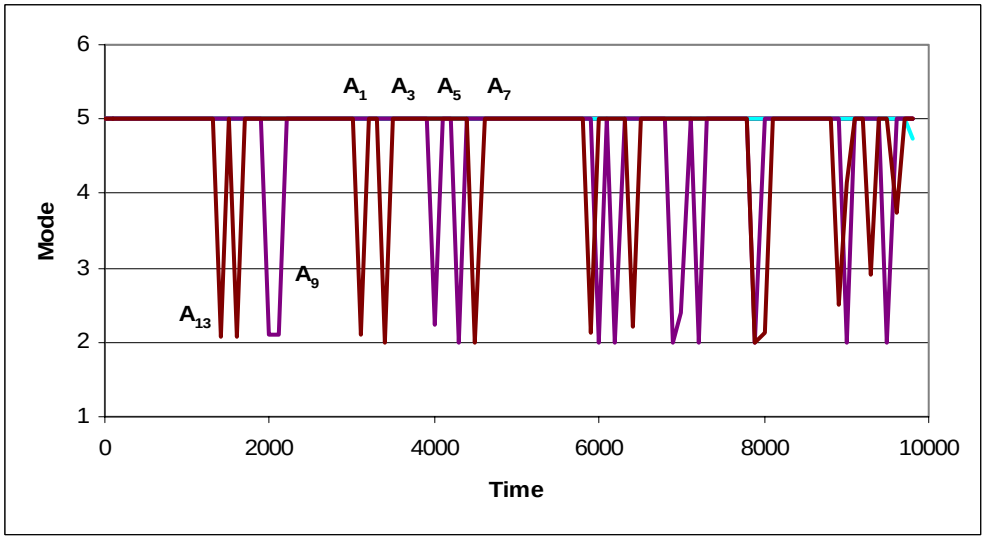


Figure **C.10**: Behavior of v_i^* under PCNN in deterministic environment with no stress (Con2-1)

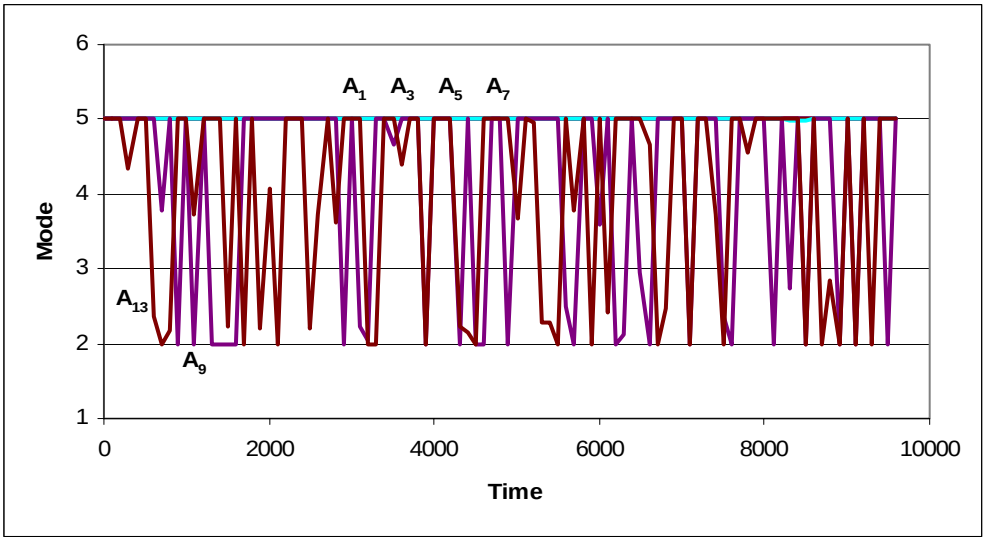


Figure **C.11**: Behavior of v_i^* under PCYN in deterministic environment with no stress (Con2-1)

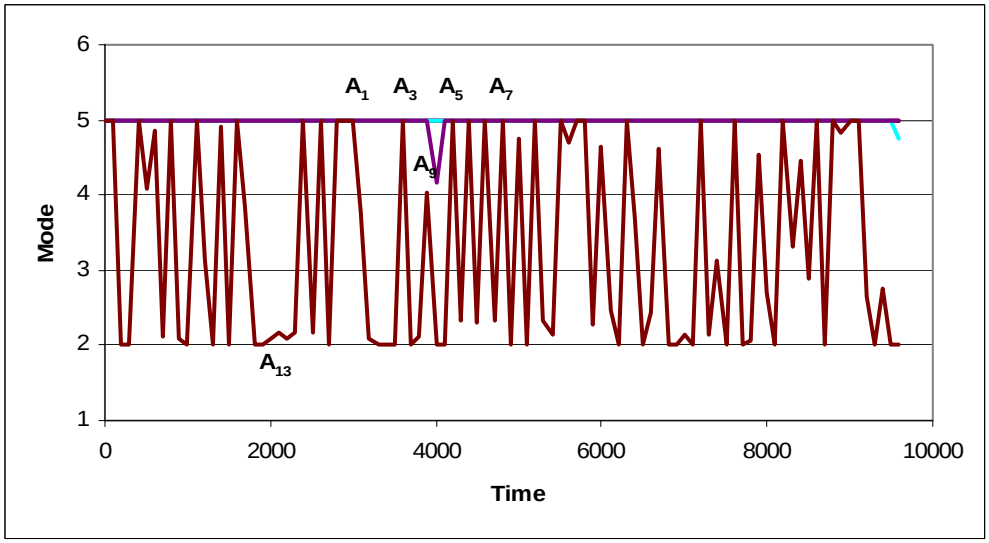


Figure **C.12**: Behavior of v_i^* under PCYY in deterministic environment with no stress (Con2-1)

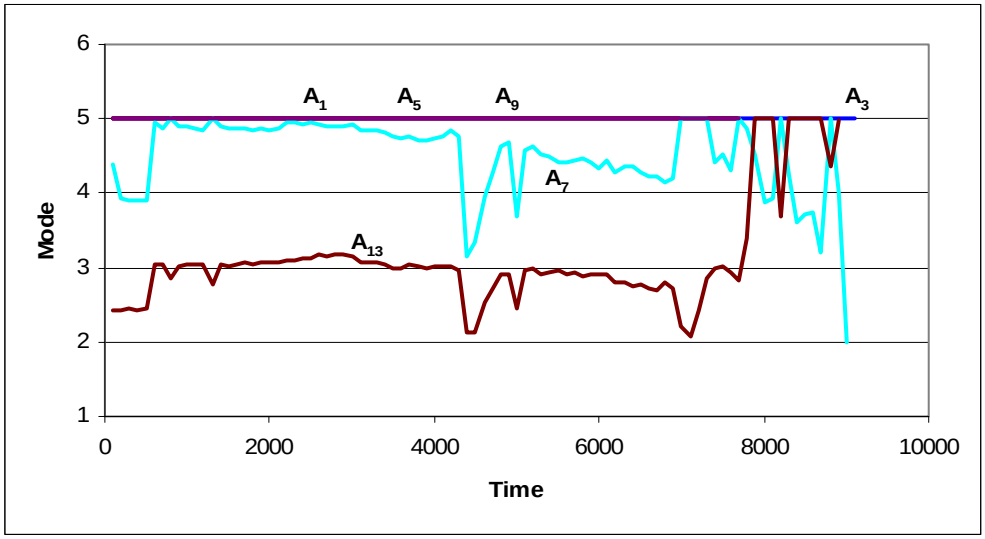


Figure **C.13**: Behavior of v_i^* under RCYY in stochastic environment with no stress (Con2-2)

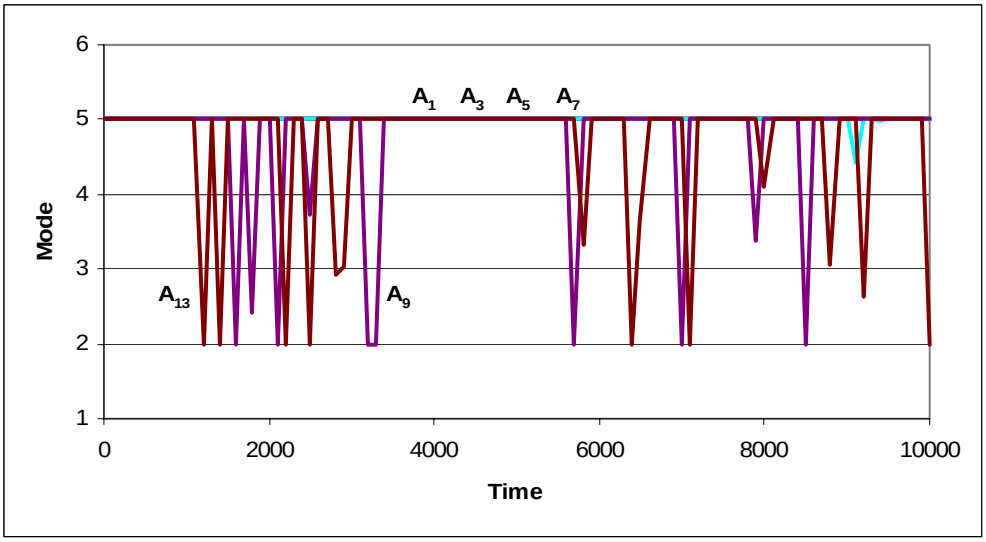


Figure **C.14**: Behavior of v_i^* under PCNN in stochastic environment with no stress (Con2-2)

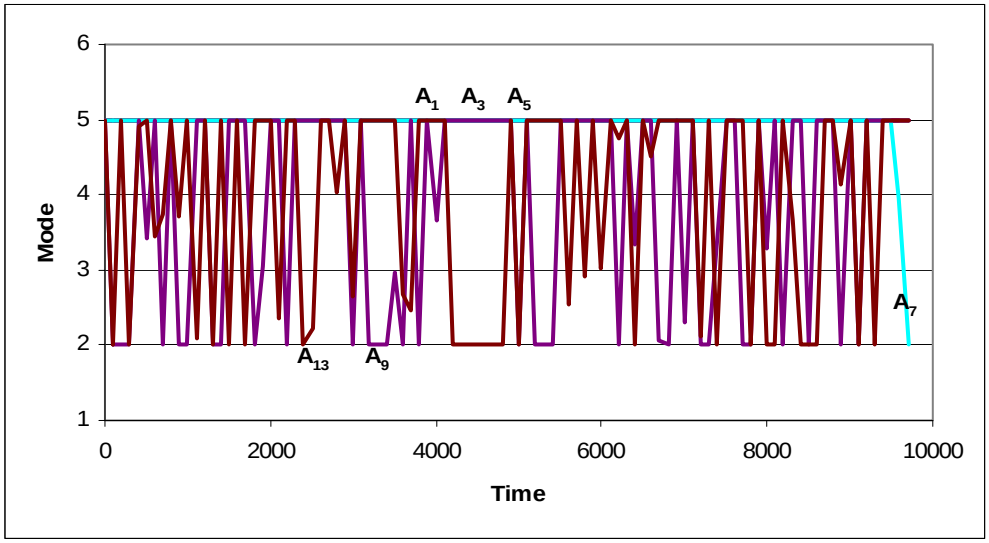


Figure **C.15**: Behavior of v_i^* under PCYN in stochastic environment with no stress (Con2-2)

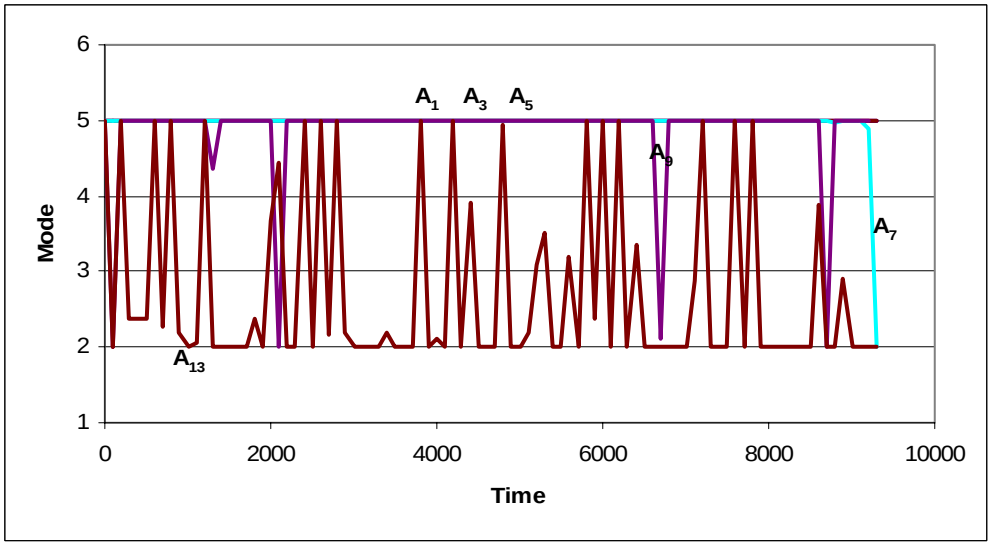


Figure **C.16**: Behavior of v_i^* under PCYY in stochastic environment with no stress (Con2-2)

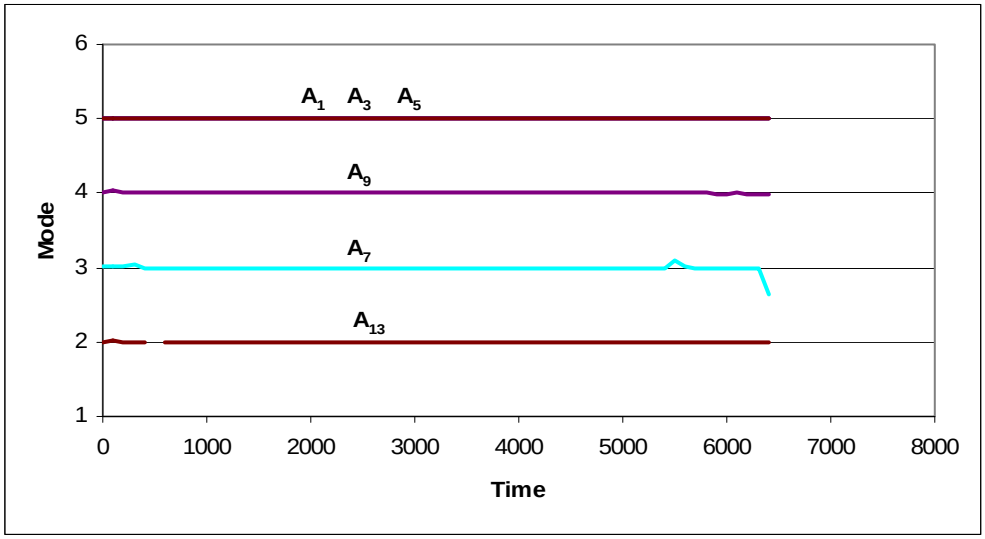


Figure **C.17**: Behavior of v_i^* under RCYY in deterministic environment with no stress (Con3-1)

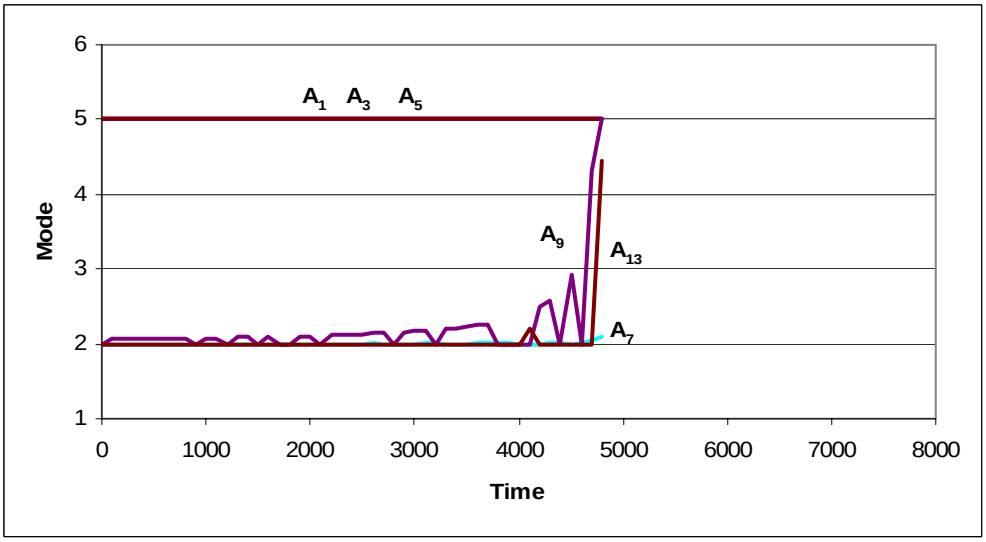


Figure **C.18**: Behavior of v_i^* under PCNN in deterministic environment with no stress (Con3-1)

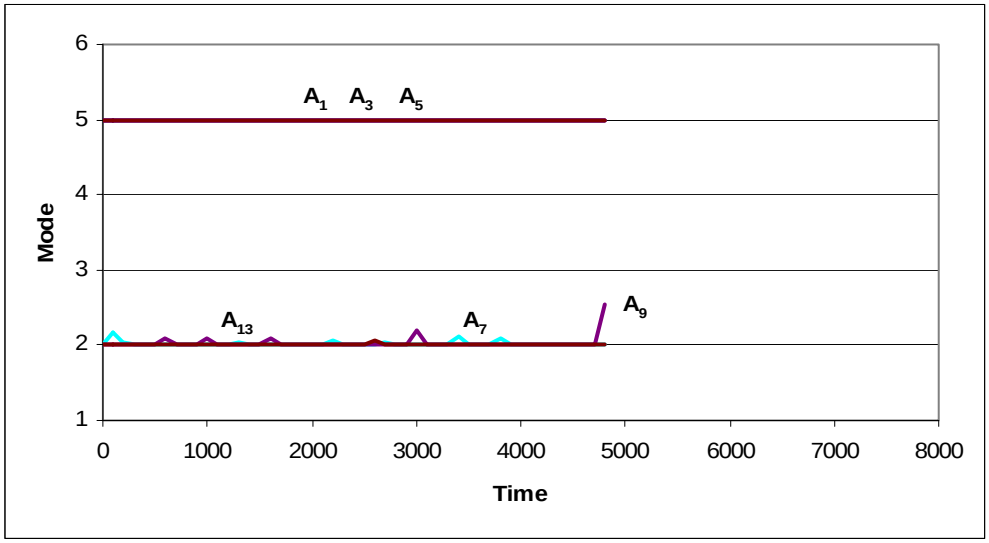


Figure **C.19**: Behavior of v_i^* under PCYN in deterministic environment with no stress (Con3-1)

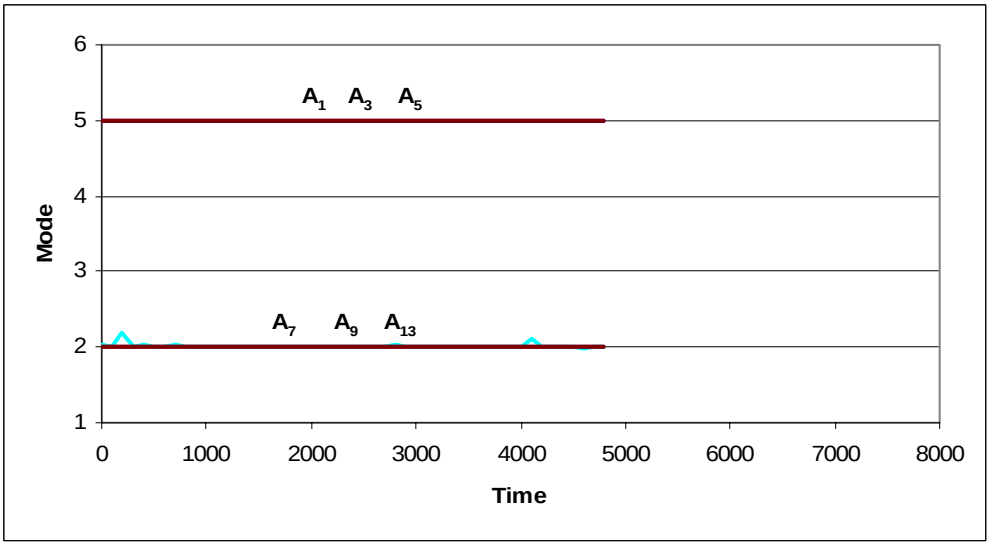


Figure **C.20**: Behavior of v_i^* under PCYY in deterministic environment with no stress (Con3-1)

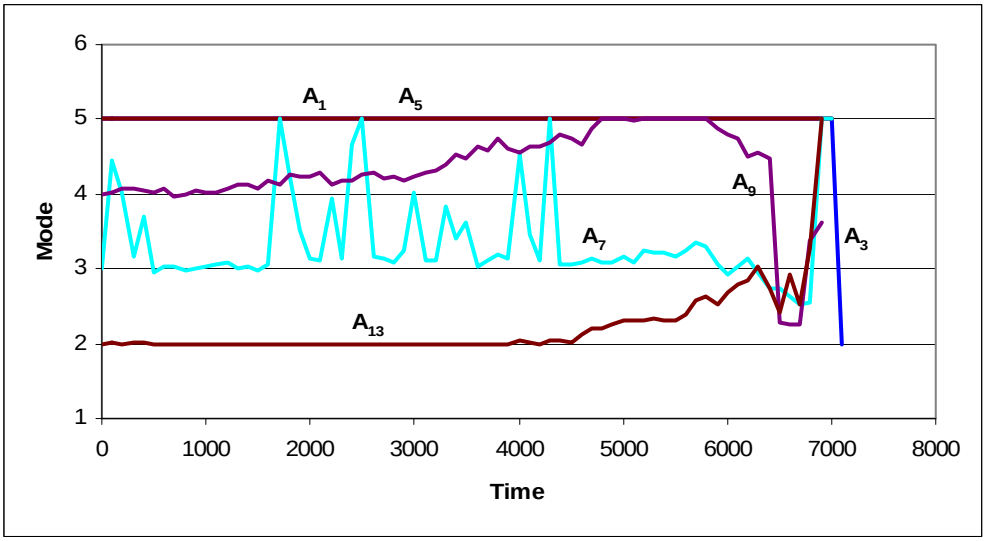


Figure **C.21**: Behavior of v_i^* under RCYY in stochastic environment with no stress (Con3-2)

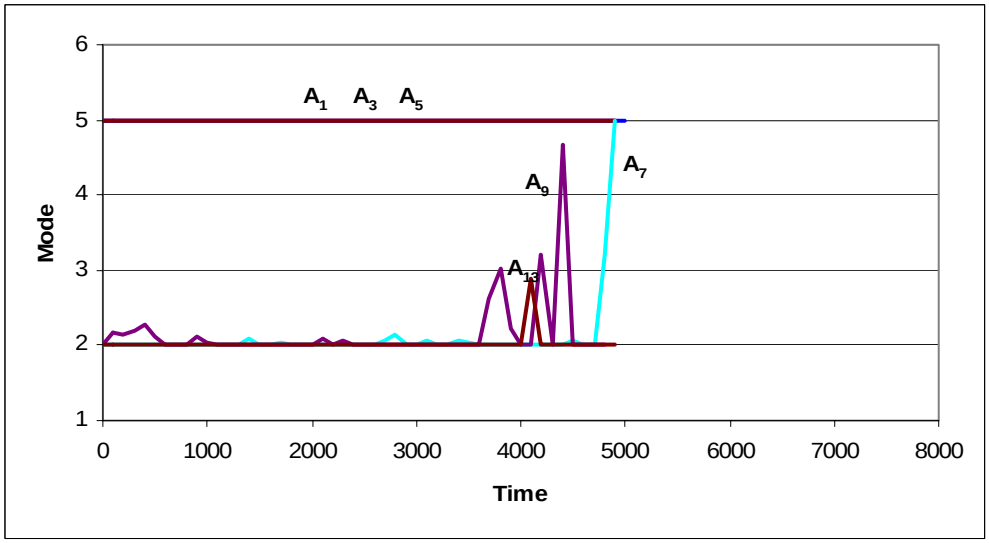


Figure **C.22**: Behavior of v_i^* under PCNN in stochastic environment with no stress (Con3-2)

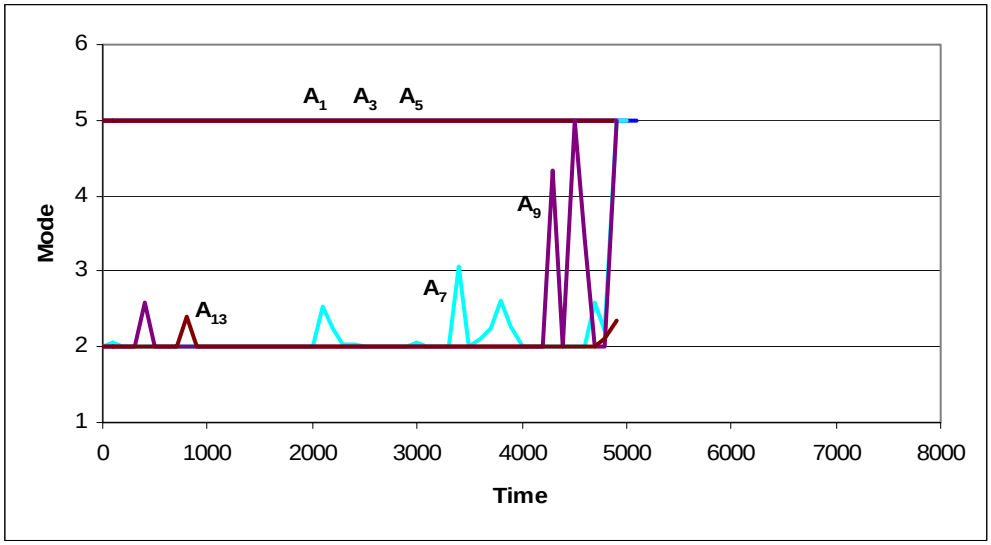


Figure **C.23**: Behavior of v_i^* under PCYN in stochastic environment with no stress (Con3-2)

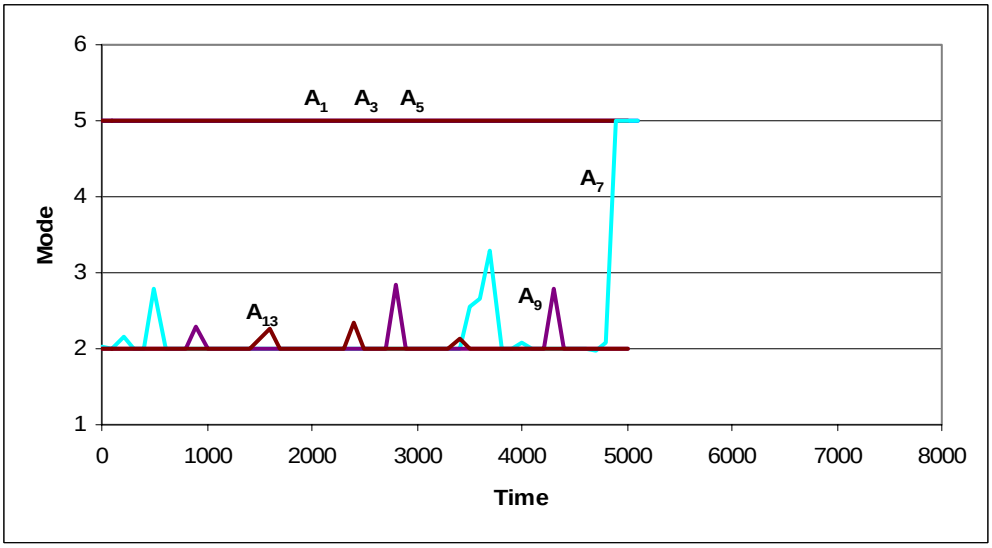


Figure **C.24**: Behavior of v_i^* under PCYY in stochastic environment with no stress (Con3-2)

Appendix D

System Behavior in Stress Environments

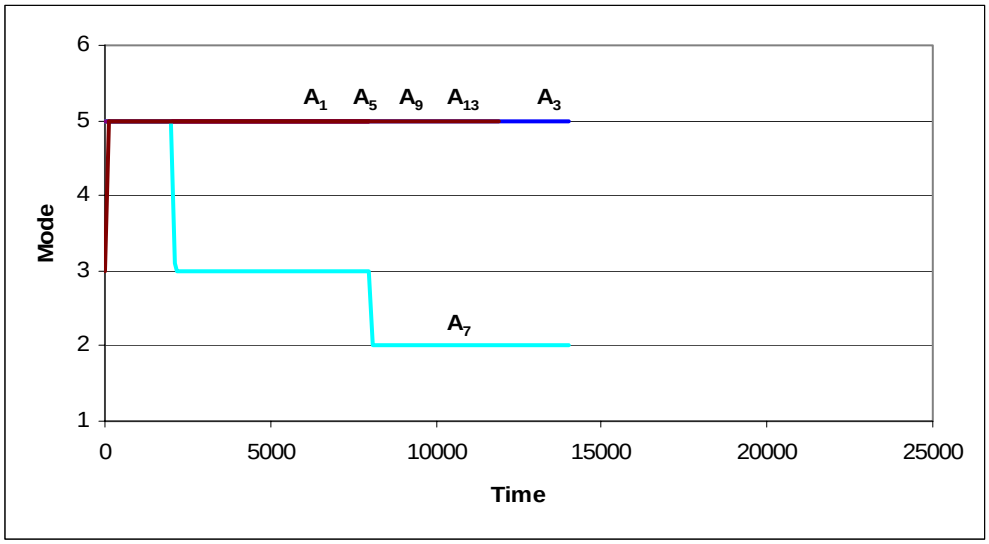


Figure **D.1**: Behavior of v_i^* under RCYY in deterministic environment with no stress (Con4-1)

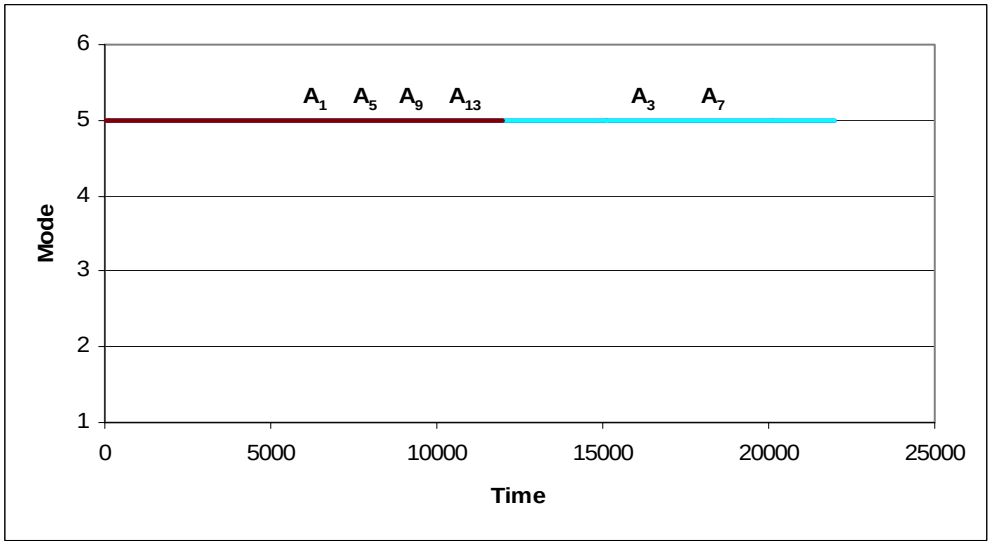


Figure **D.2**: Behavior of v_i^* under PCNN in deterministic environment with no stress (Con4-1)

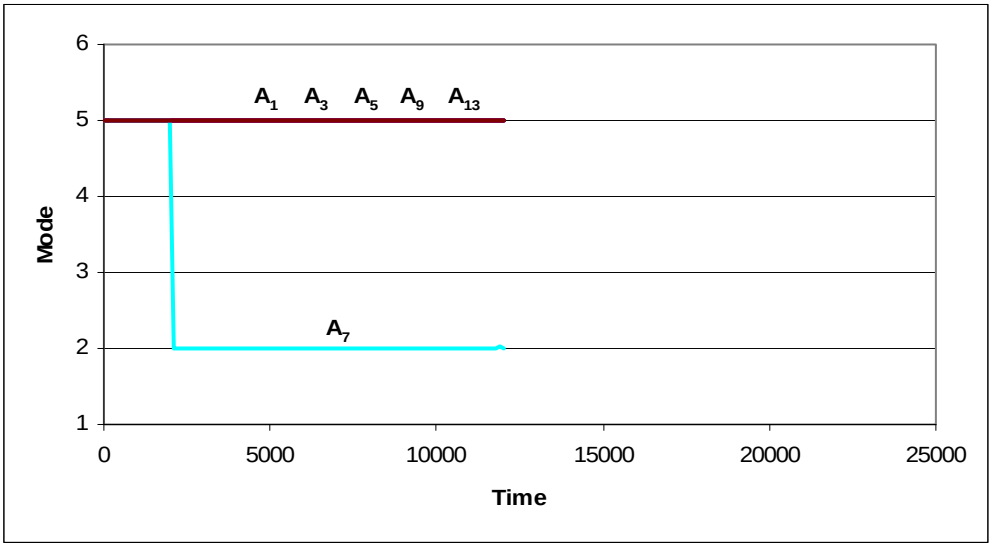


Figure **D.3**: Behavior of v_i^* under PCYN in deterministic environment with no stress (Con4-1)

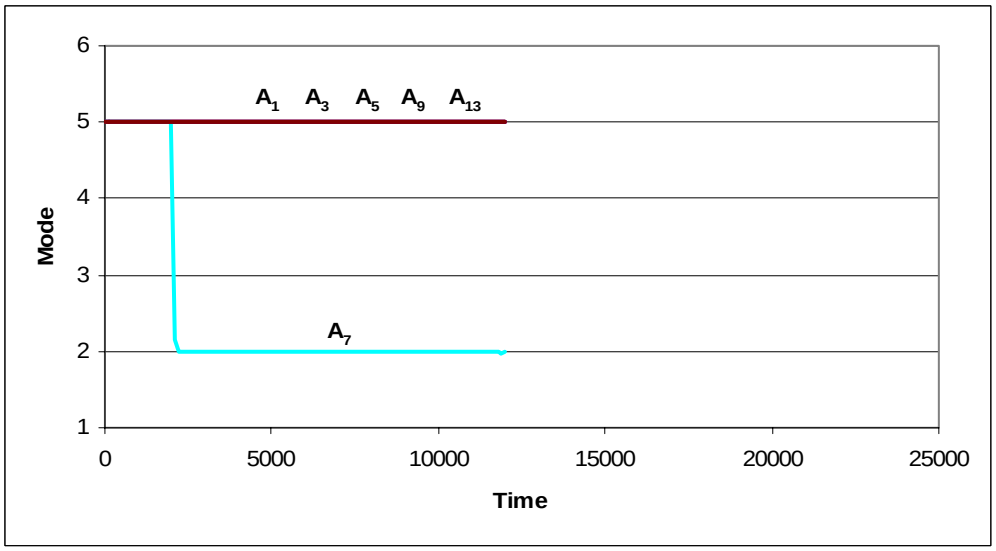


Figure **D.4**: Behavior of v_i^* under PCYY in deterministic environment with no stress (Con4-1)

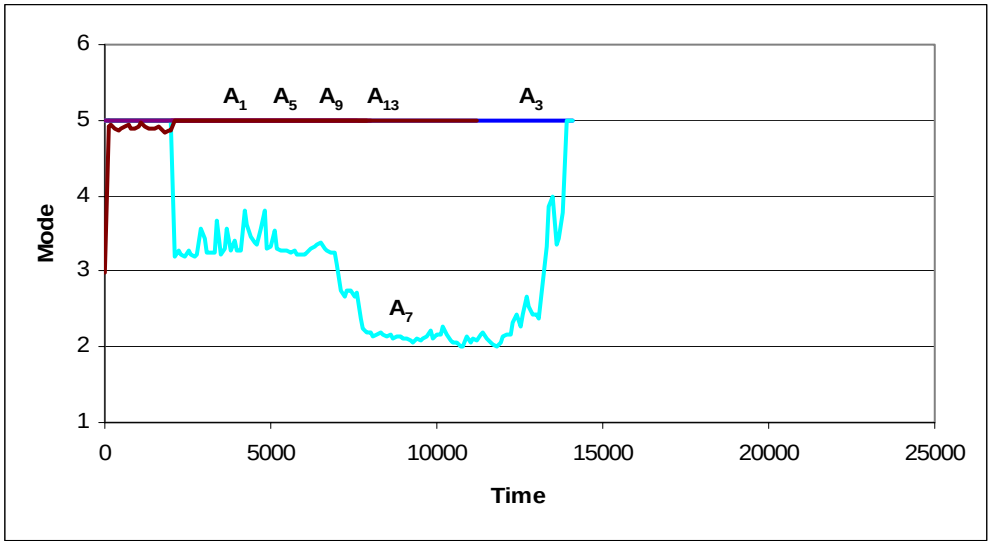


Figure **D.5**: Behavior of v_i^* under RCYY in stochastic environment with no stress (Con4-2)

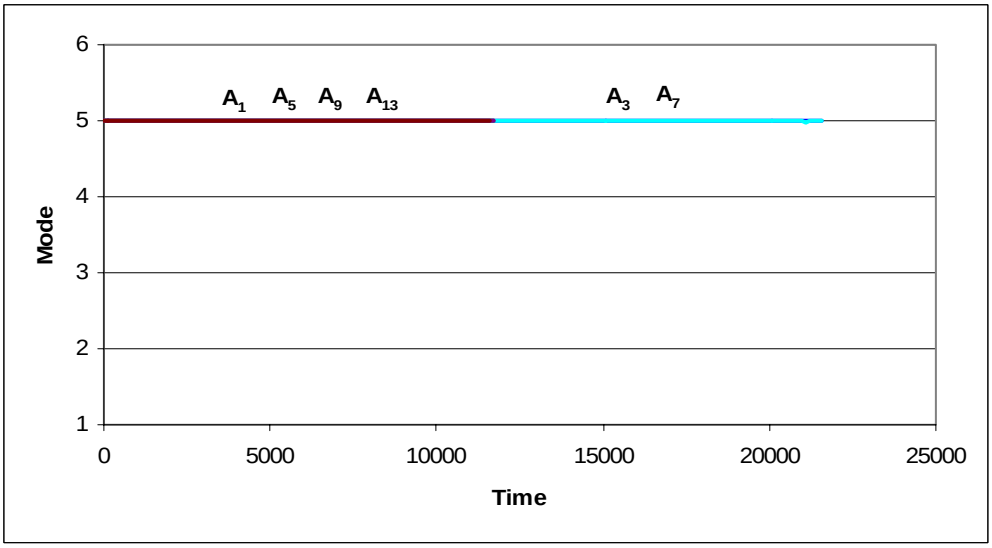


Figure **D.6**: Behavior of v_i^* under PCNN in stochastic environment with no stress (Con4-2)

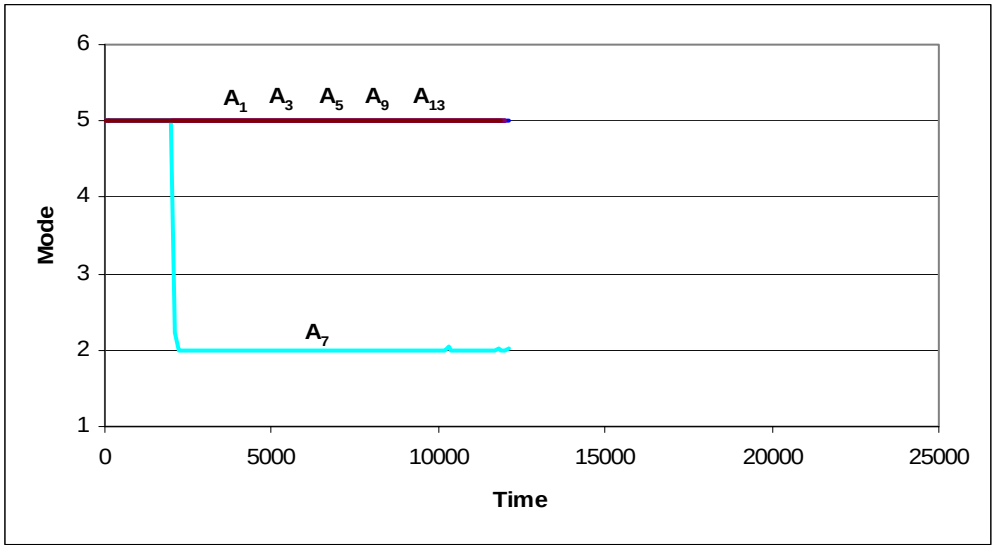


Figure **D.7**: Behavior of v_i^* under PCYN in stochastic environment with no stress (Con4-2)

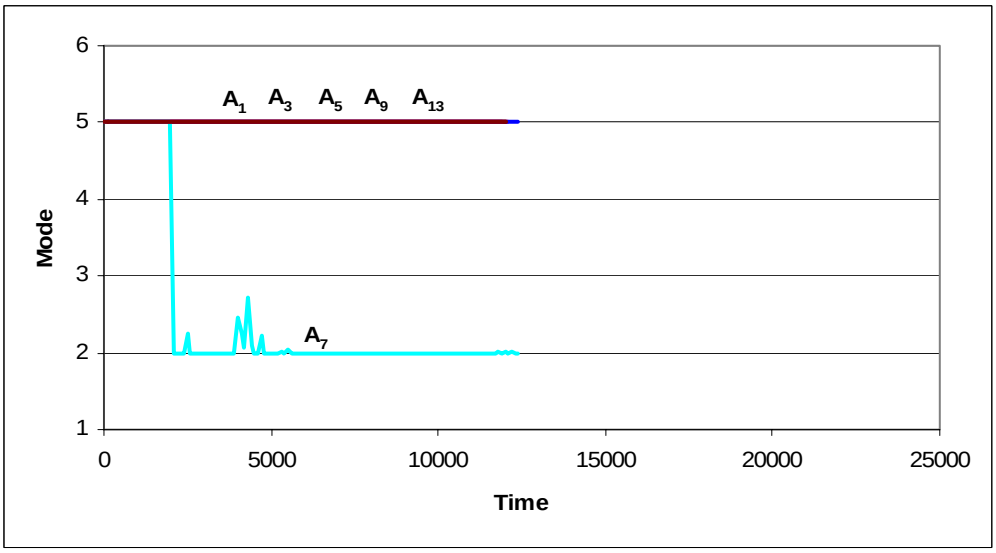


Figure **D.8**: Behavior of v_i^* under PCYY in stochastic environment with no stress (Con4-2)

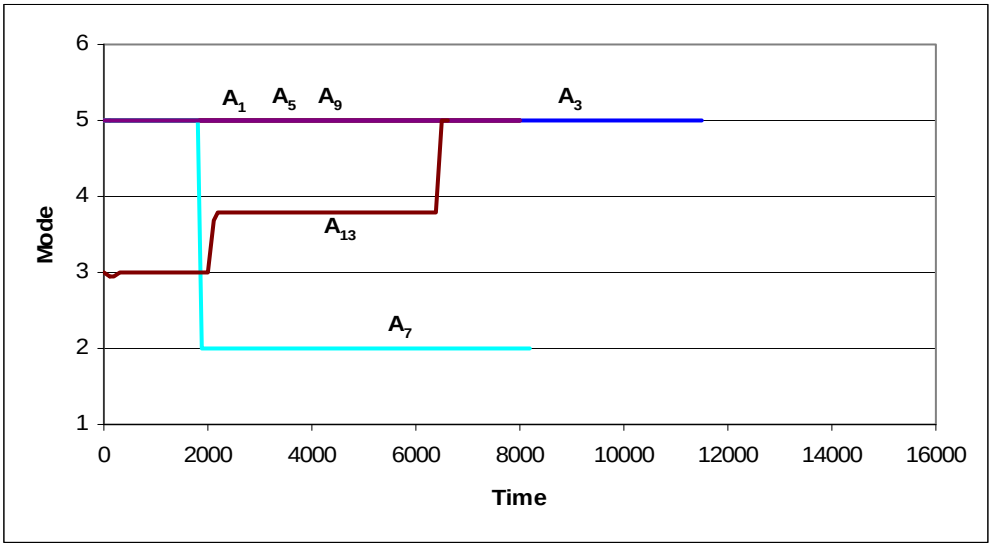


Figure **D.9**: Behavior of v_i^* under RCYY in deterministic environment with no stress (Con5-1)

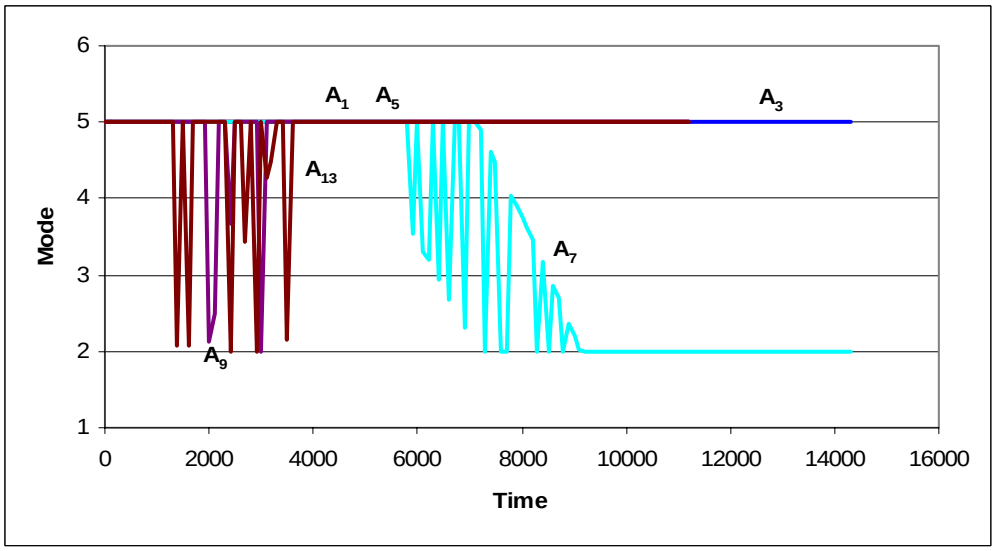


Figure **D.10**: Behavior of v_i^* under PCNN in deterministic environment with no stress (Con5-1)

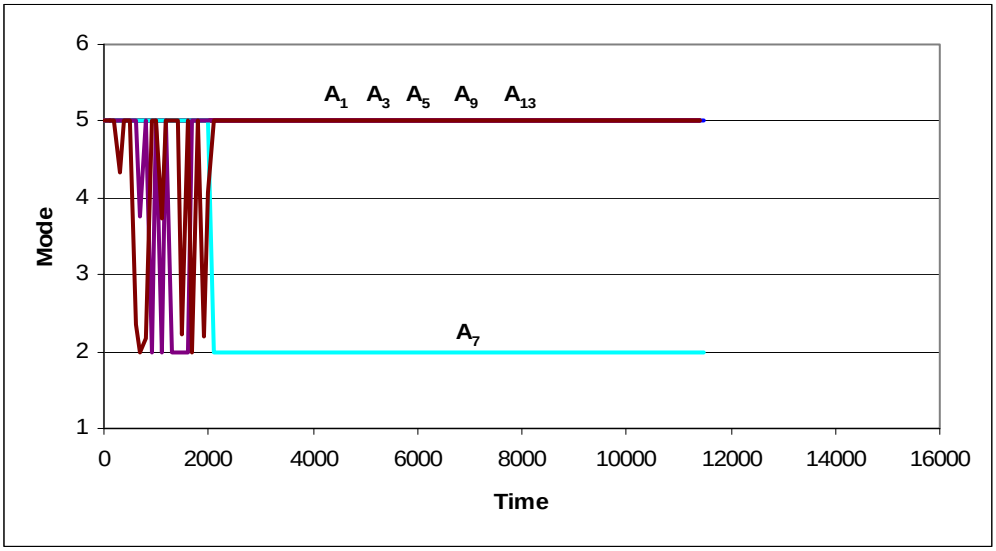


Figure **D.11**: Behavior of v_i^* under PCYN in deterministic environment with no stress (Con5-1)

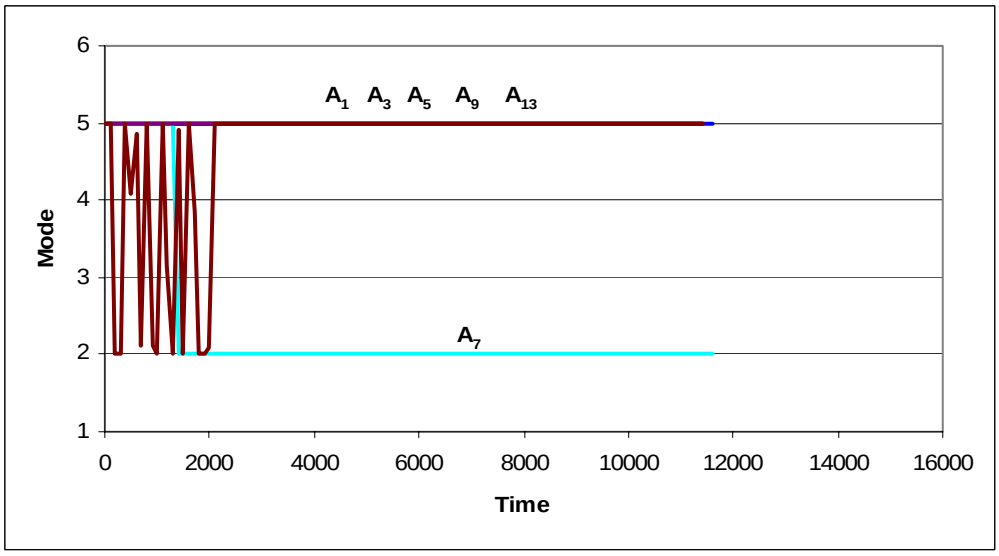


Figure **D.12**: Behavior of v_i^* under PCYY in deterministic environment with no stress (Con5-1)

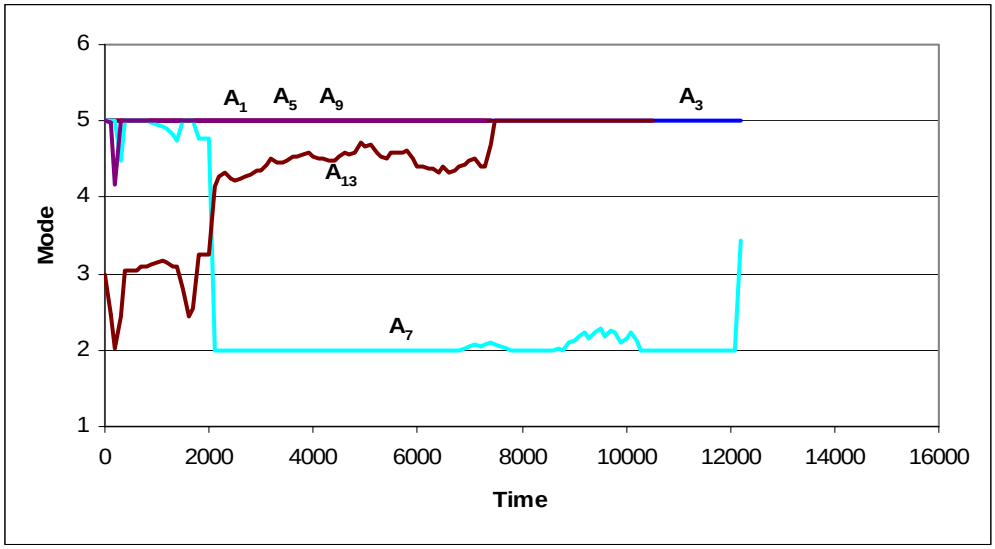


Figure **D.13**: Behavior of v_i^* under RCYY in stochastic environment with no stress (Con5-2)

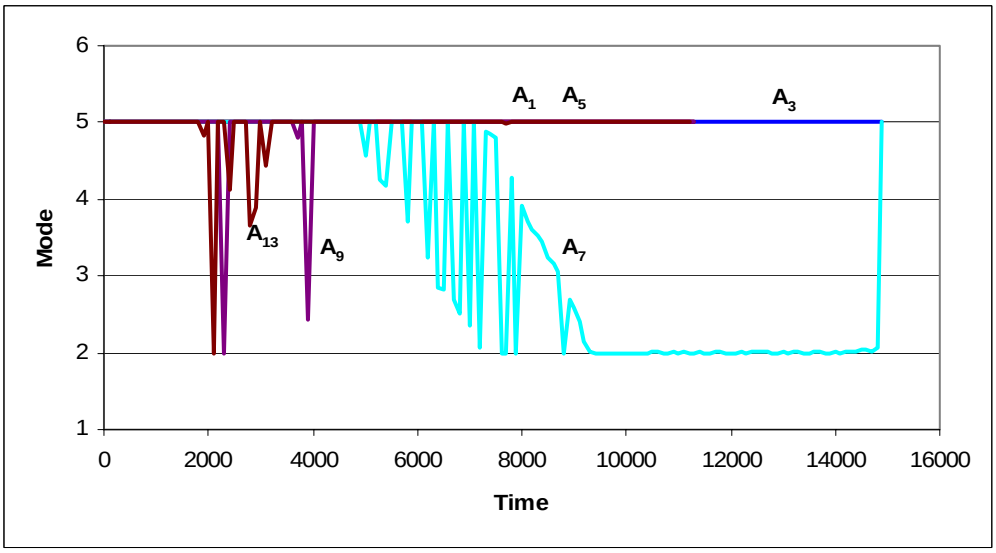


Figure **D.14**: Behavior of v_i^* under PCNN in stochastic environment with no stress (Con5-2)

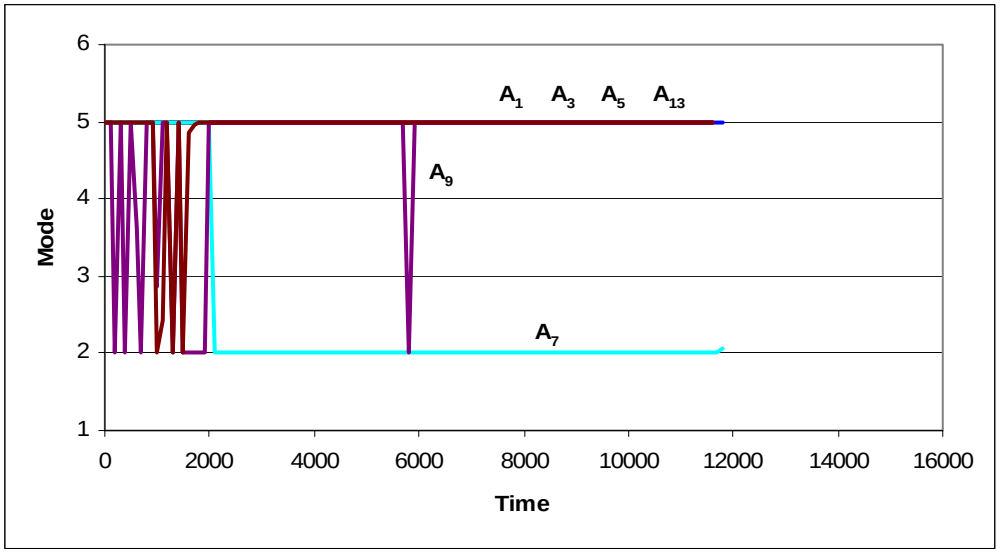


Figure **D.15**: Behavior of v_i^* under PCYN in stochastic environment with no stress (Con5-2)

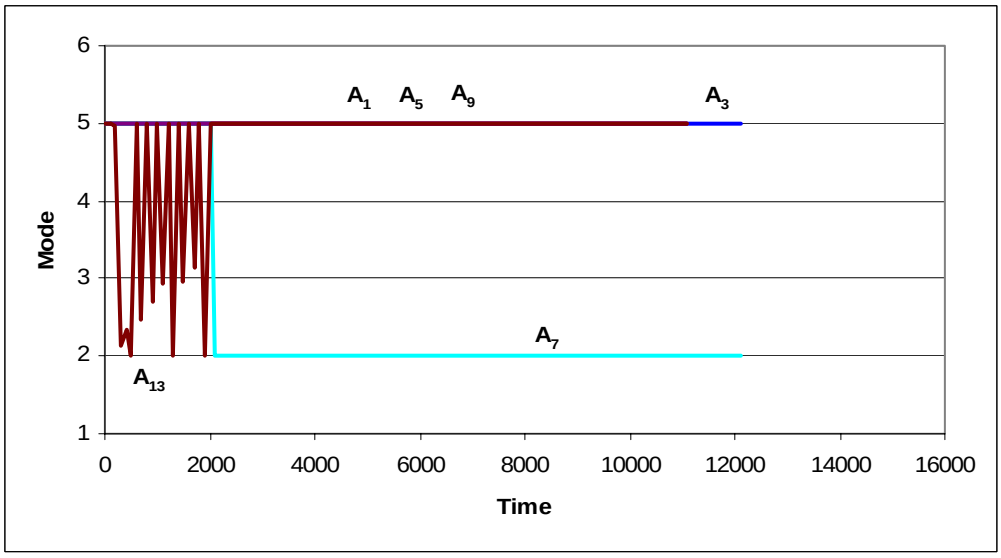


Figure **D.16**: Behavior of v_i^* under PCYY in stochastic environment with no stress (Con5-2)

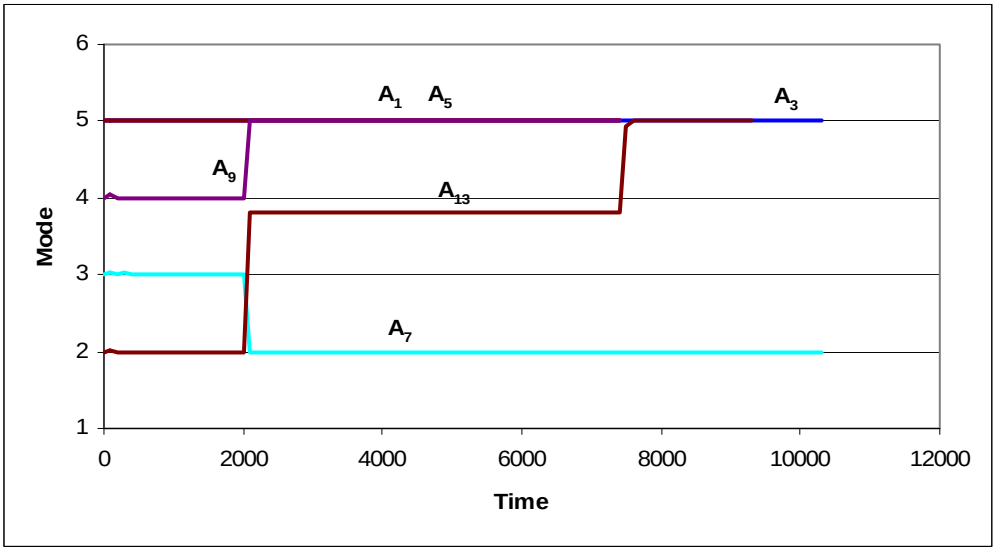


Figure **D.17**: Behavior of v_i^* under RCYY in deterministic environment with no stress (Con6-1)

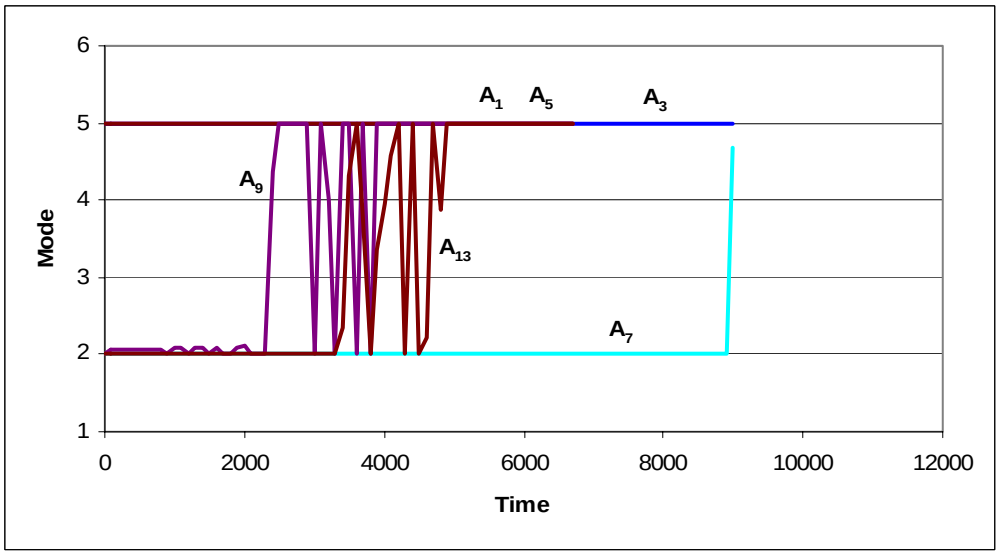


Figure **D.18**: Behavior of v_i^* under PCNN in deterministic environment with no stress (Con6-1)

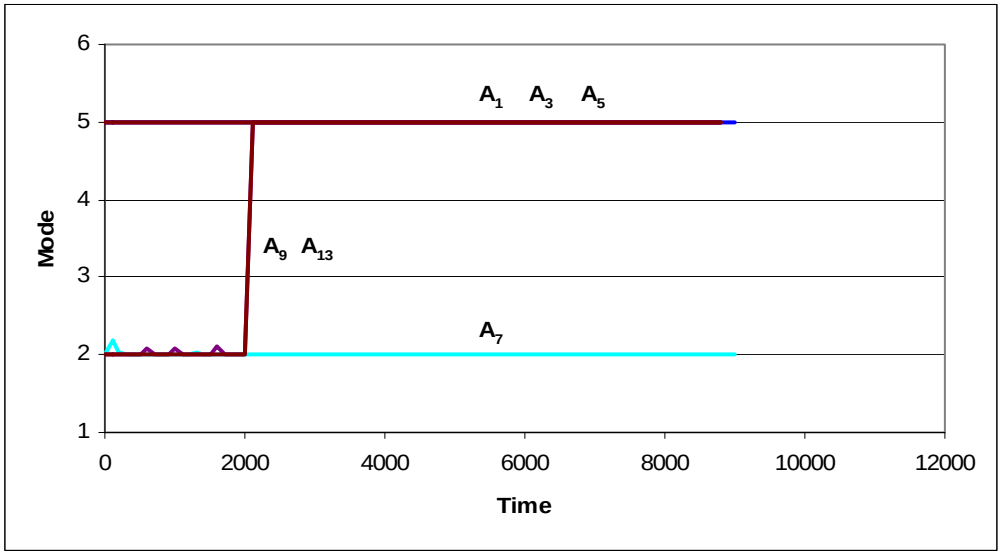


Figure **D.19**: Behavior of v_i^* under PCYN in deterministic environment with no stress (Con6-1)

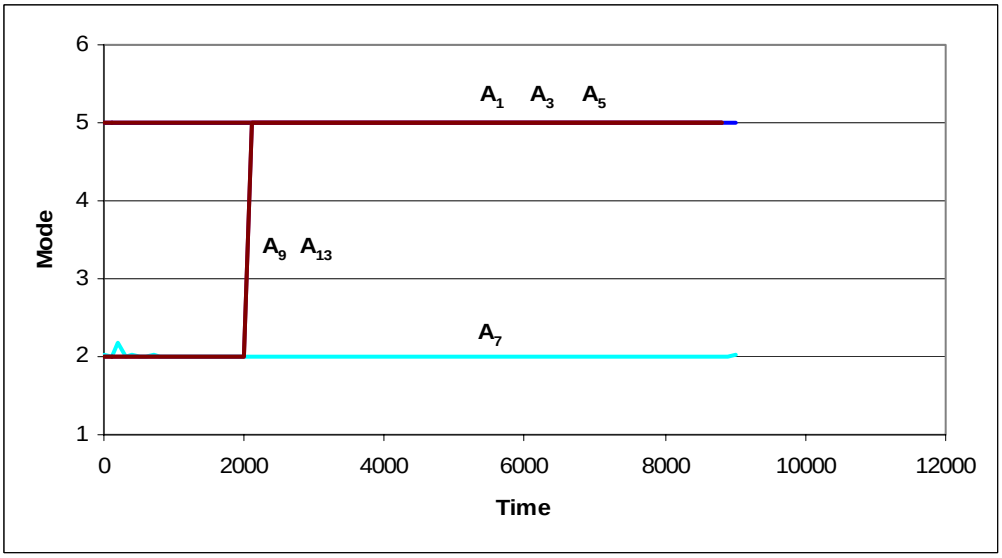


Figure **D.20**: Behavior of v_i^* under PCYY in deterministic environment with no stress (Con6-1)

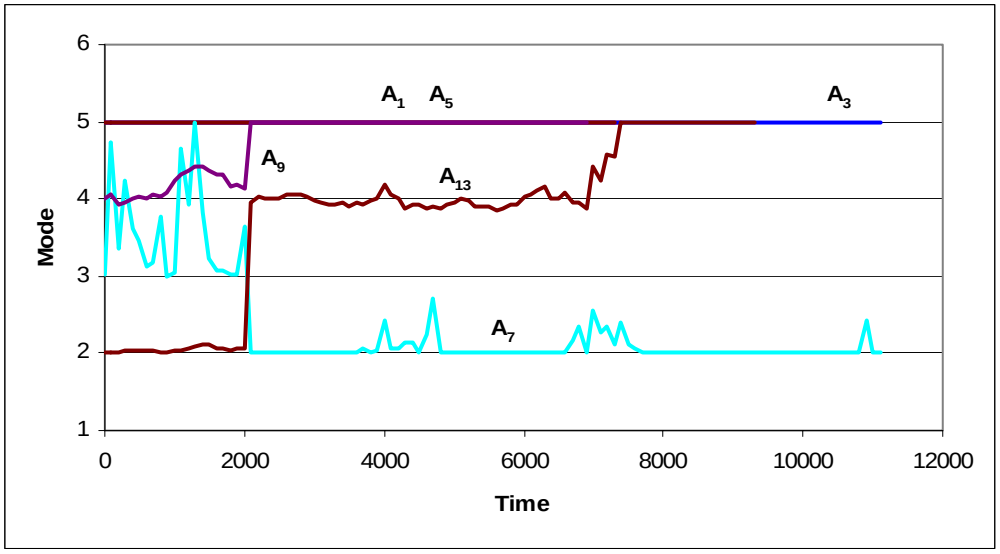


Figure **D.21**: Behavior of v_i^* under RCYY in stochastic environment with no stress (Con6-2)

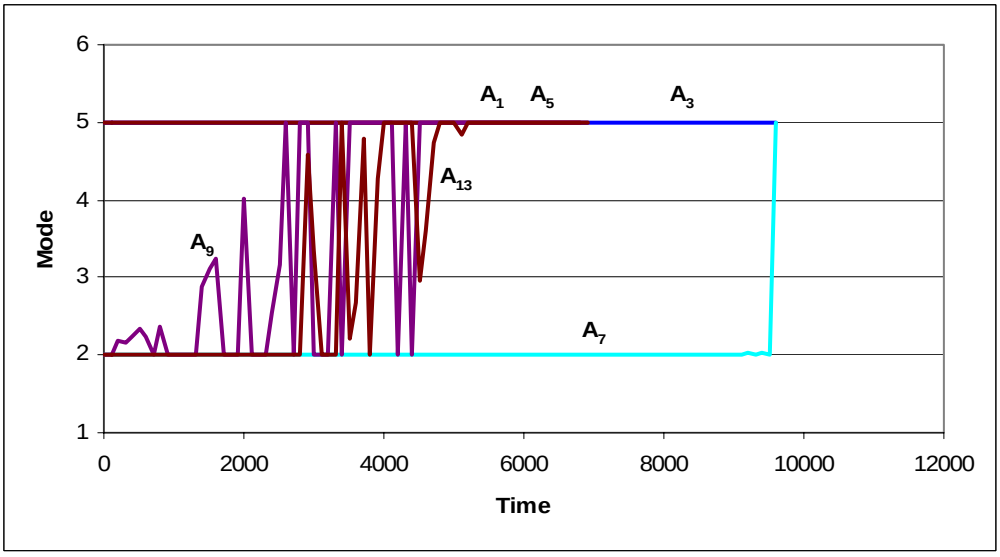


Figure **D.22**: Behavior of v_i^* under PCNN in stochastic environment with no stress (Con6-2)

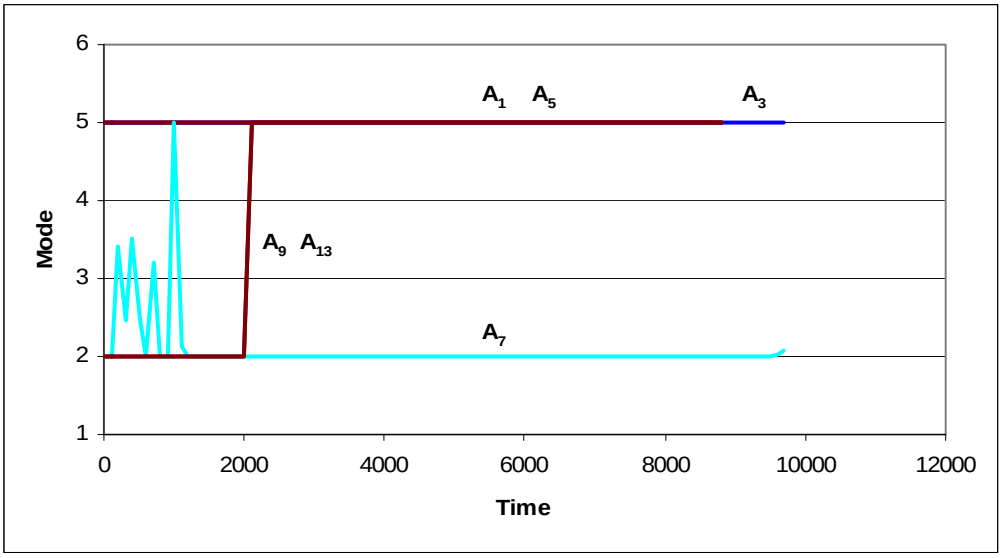


Figure **D.23**: Behavior of v_i^* under PCYN in stochastic environment with no stress (Con6-2)

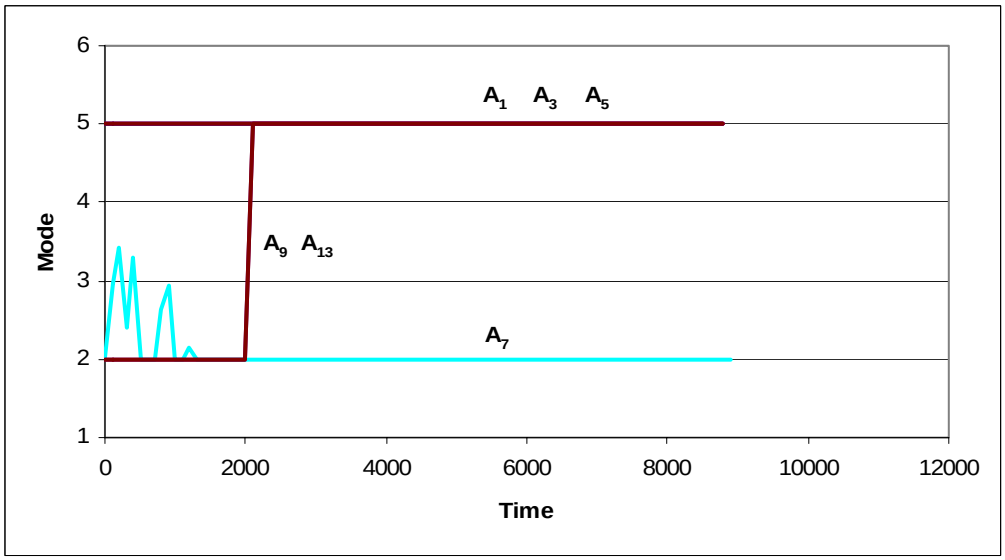


Figure **D.24**: Behavior of v_i^* under PCYY in stochastic environment with no stress (Con6-2)

VITA

Seokcheon Lee

Seokcheon Lee was born in Korea in 1969. He received his B.S. and M.S. degrees in Industrial Engineering from the Seoul National University in 1991 and 1993 respectively. During his M.S. study his major research interest was in the area of *human factors*. After he finished his M.S. study he joined Daewoo Motor Co., Inchon, Korea. For seven years in the Technical Center of the company, he mostly worked in vehicle safety area. In 2000 fall, he enrolled in the Ph.D. program in Industrial Engineering at the Pennsylvania State University. During his Ph.D. study he was employed as a research assistant in the Department of Industrial Engineering, participating in a research project "Chaos, Situation Extraction, and Control: A novel Integrated Approach to Robust and Scalable Cognitive Agent Design" sponsored by DARPA UltraLog program, under the supervision of Dr. Soundar Kumara. His current primary research interest is in the area of *distributed control* in the context of (geographically or organizationally) distributed systems such as supply chain/manufacturing systems, computer/communication networks, and ubiquitous systems.