Multiple Criteria Decision Analysis in Autonomous Computing: A Study on Independent and Coordinated Self-Management

by

Yağız Onat Yazır
B.Sc., University of Victoria, 2005
M.Sc., University of Victoria, 2007

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Computer Science

Multiple Criteria Decision Analysis in Autonomous Computing: A Study on Independent and Coordinated Self-Management

by

Yağız Onat Yazır
B.Sc., University of Victoria, 2005
M.Sc., University of Victoria, 2007

Supervisory Committee

Dr. Yvonne Coady, Co-Supervisor
(Department of Computer Science)

Dr. Sudhakar Ganti, Co-Supervisor
(Department of Computer Science)

Dr. Jianping Pan, Departmental Member
(Department of Computer Science)

Dr. Adel Guitouni, Outside Member
(Peter B. Gustavson School of Business)

Dr. Stephen W. Neville, Outside Member
(Department of Electrical and Computer Engineering)

Supervisory Committee

Dr. Yvonne Coady, Co-Supervisor
(Department of Computer Science)

Dr. Sudhakar Ganti, Co-Supervisor
(Department of Computer Science)

Dr. Jianping Pan, Departmental Member
(Department of Computer Science)

Dr. Adel Guitouni, Outside Member
(Peter B. Gustavson School of Business)

Dr. Stephen W. Neville, Outside Member
(Department of Electrical and Computer Engineering)

ABSTRACT

In this dissertation, we focus on the problem of self-management in distributed systems. In this context, we propose a new methodology for reactive self-management based on multiple criteria decision analysis (MCDA). The general structure of the proposed methodology is extracted from the commonalities of the former well-established approaches that are applied in other problem domains. The main novelty of this work, however, lies in the usage of MCDA during the reaction processes in the context of the two problems that the proposed methodology is applied to.

In order to provide a detailed analysis and assessment of this new approach, we have used the proposed methodology to design distributed autonomous agents that can provide self-management in two outstanding problems. These two problems also

represent the two distinct ways in which the methodology can be applied to self-management problems. These two cases are: 1) independent self management, and 2) coordinated self-management. In the simulation case study regarding independent self-management, the methodology is used to design and implement a distributed resource consolidation manager for clouds, called IMPROMPTU. In IMPROMPTU, each autonomous agent is attached to a unique physical machine in the cloud, where it manages resource consolidation independently from the rest of the autonomous agents. On the other hand, the simulation case study regarding coordinated self-management focuses on the problem of adaptive routing in mobile ad hoc networks (MANET). The resulting system carries out adaptation through autonomous agents that are attached to each MANET node in a coordinated manner. In this context, each autonomous node agent expresses its opinion in the form of a decision regarding which routing algorithm should be used given the perceived conditions. The opinions are aggregated through coordination in order to produce a final decision that is to be shared by every node in the MANET.

Although MCDA has been previously considered within the context of artificial intelligence—particularly with respect to algorithms and frameworks that represent different requirements for MCDA problems, to the best of our knowledge, this dissertation outlines a work where MCDA is applied for the first time in the domain of these two problems that are represented as simulation case studies.

Contents

Supervisory Committee ii

Abstract iii

Table of Contents v

List of Tables viii

List of Figures ix

Acknowledgements xii

Dedication xiii

1 Introduction 1

1.1 Motivation and Problem Description 2

1.2 Scope 4

1.3 Organization 6

2 A Brief History of Self-Management 7

2.1 Artificial Intelligence, Operations Research and Self-Management . . 8

2.2 Direct Approaches to Self-Management 11

2.3 Summary 14

3 Methodology for Reactive Self-Management Based on MCDA 16

3.1 Observing the Conditions 18

3.2 Detecting Critical Changes 20

3.3 Reacting through MCDA 21

3.3.1 Independent Reactions 22

3.3.2 Coordinated Reactions 23

3.4 Summary 26

4 Multiple Criteria Decision Analysis Methods 27
4.1 Single Synthesizing Criterion Methods 29
4.1.1 Multiattribute Utility Theory 30
4.1.2 UTA Methods 36
4.1.3 Analytic Hierarchy Process 40
4.1.4 Other Single Synthesizing Criterion Methods 44
4.2 Outranking Methods 45
4.2.1 ELECTRE Methods 45
4.2.2 PROMETHEE Methods 54
4.2.3 Other Outranking Methods 60
4.3 Summary 60

5 Simulation Platforms 62
5.1 Platform for Simulation Case Study 1 63
5.1.1 Current State and Possible Improvements 67
5.2 Platform for Simulation Case Study 2 68
5.2.1 Current State and Possible Improvements 72
5.3 Summary 72

6 Simulation Case Study 1: Independent Self-Management 74
6.1 Problem Description 75
6.2 Related Work 77
6.3 IMPROMPTU 80
6.3.1 Reactive Resource Consolidation Management 80
6.3.2 Multiple Criteria Decision Analysis Model 83
6.3.3 Applying PROMETHEE 87
6.4 System Architecture 88
6.5 Empirical Results 93
6.5.1 Simulation Settings 94
6.5.2 Simulation Results 96
6.6 Summary 105

7 Simulation Case Study 2: Coordinated Self-Management 106
7.1 Problem Description 108

7.2 Related Work 110
7.3 Proposed Approach and System Architecture 112
7.3.1 Monitor 115
7.3.2 Local Decider 118
7.3.3 Coordination 120
7.3.4 Cooperation 122
7.4 Simulation Results 123
7.4.1 Testing Environment 123
7.4.2 Test Scenarios 124
7.5 Summary 130

8 Conclusion and Future Work **132**

Bibliography **137**

List of Tables

Table 4.1 Random Index 44

Table 6.1 Statistics of Physical Machine Usage, SLO Violations and Migrations of Different Resource Consolidation Managers on a per Simulation Step Basis 97

List of Figures

2.1 SISO Negative feedback control system. 12
2.2 OODA Loop 13
2.3 IBM Autonomic Manager 14

3.1 The high-level structure of the methodology for reactive self-management using MCDA. 19
3.2 Observing the conditions. 20
3.3 Detecting critical changes. 21
3.4 Reacting through MCDA, the independent case. 23
3.5 Reacting through MCDA, the coordinated case. 25

5.1 General architecture of the platform for simulation changing resource requirements at the IaaS level of a cloud. 64
5.2 Different deployment schemes with respect to centralized and distributed self-management approaches. 66
(a) Layout using a centralized resource consolidation manager. . . . 66
(b) Layout using a distributed resource consolidation manager. . . . 66
5.3 General view of the internal structure of each generic node used in the simulation runs. 69
5.4 General view of the internal structures of the observation and detection modules, and the data flow between them. 70
5.5 General view of the reaction module and the data flow during the coordaintion and coorperation procedures. 71

6.1 The modules of an autonomous node agent. 90
6.2 The states assigned to a virtual machine under the management of an autonomous node agent. 91
6.3 Physical machine usage per simulation step by different resource consolidation managers. 95

6.4 SLO violation per simulation step caused by different resource consolidation managers. 96
6.5 Migrations per simulation step performed by different resource consolidaiton managers. 97
6.6 Overall CPU utilization in the data center at each step using different resource consolidation managers. 98
6.7 Overall Memory utilization in the data center at each step using different resource consolidation managers. 99
6.8 Overall Bandwidth utilization in the data center at each step using different resource consolidation managers. 100
6.9 Total SLO violations and migrations throughout a simulation run as experienced on each physical machine in the data center. 101
(a) IMP-1 101
(b) IMP-2 101
6.10 Total SLO violations and migrations throughout a simulation run as experienced by each virtual machine in the data center. 103
(a) IMP-1 103
(b) IMP-2 103

7.1 Control state diagram of node agents. 113
7.2 The modules of an NA. 115
7.3 Interactivity between the modules during the course of the coordination state. 116
7.4 Interactivity between modules during the course of the cooperation state. 116
7.5 Non-constrained coordinators: The total number of global adaptation processes undertaken in a 5-hour simulation run, and its decomposition into cycles that completed with commit and abort. 125
7.6 Constrained coordinators: The total number of global adaptation processes undertaken in a 5-hour simulation run, and its decomposition into cycles that completed with commit and abort. 126
7.7 The total number backoffs observed in runs with different levels of constraints on coordinators. 127
7.8 The total number timeouts observed in runs with different levels of constraints on coordinators. 128

7.9 The average time-taken for global adaptation processes to complete. 129

ACKNOWLEDGEMENTS

I would like to thank my supervisors Dr. Yvonne Coady and Dr. Sudhakar Ganti for their endless support and mentoring throughout this degree. I would also like to thank Dr. Adel Guitouni for his help with the academic work that led to this dissertation.

Segui il tuo corso, e lascia dir le genti.

DEDICATION

I dedicate this dissertation to my small and wonderful family of women; my mother Gülçin Onat, my grandmother Nimet Onat—may she rest in light—and my younger sisters Meriç, Toprak and Arviş.

Ayın altında kağnılar gidiyordu.
Kağnılar gidiyordu Akşehir üstünden Afyon'a doğru.
Toprak öyle bitip tükenmez,
dağlar öyle uzakta,
sanki gidenler hiçbir zaman
hiçbir menzile erişmiyecekti.
Kağnılar yürüyordu yekpare meşeden tekerlekleriyle.
Ve onlar
ayın altında dönen ilk tekerlekti.
Ayın altında öküzler
başka ve çok küçük bir dünyadan gelmişler gibi
ufacık, kısacıktılar,
ve pırıltılar vardı hasta, kırık boynuzlarında
ve ayakları altından akan
toprak,
toprak
ve topraktı.
Gece aydınlık ve sıcak
ve kağnılarda tahta yataklarında
koyu mavi humbaralar çırılçıplaktı.
Ve kadınlar
birbirlerinden gizliyerek
bakıyorlardı ayın altında
geçmiş kafilelerden kalan öküz ve tekerlek ölülerine.
Ve kadınlar,
bizim kadınlarımız:
korkunç ve mübarek elleri,
ince, küçük çeneleri, kocaman gözleriyle

anamız, avradımız, yârimiz
ve sanki hiç yaşamamış gibi ölen
ve soframızdaki yeri
öküzümüzden sonra gelen
ve dağlara kaçırıp uğrunda hapis yattığımız
ve ekinde, tütünde, odunda ve pazardaki
ve karasabana koşulan
ve ağıllarda
ışıltısında yere saplı bıçakların
oynak, ağır kalçaları ve zilleriyle bizim olan
kadınlar,
bizim kadınlarımız
şimdi ayın altında
kağnıların ve hartuçların peşinde
harman yerine kehribar başaklı sap çeker gibi
aynı yürek ferahlığı,
aynı yorgun alışkanlık içindeydiler.
Ve on beşlik şarapnelin çeliğinde
ince boyunlu çocuklar uyuyordu.
Ve ayın altında kağnılar
yürüyordu Akşehir üstünden Afyon'a doğru.

by Nazım Hikmet Ran

Chapter 1

Introduction

Distributed systems consist of multiple units of computation that communicate over a network to achieve a common goal. In their most ideal form, distributed systems appear to its users as a single coherent system while dividing a problem into many mutually exclusive sub-tasks which are undertaken concurrently by the computational entities in the system [186, 89]. A distributed system is different from a centralized system with respect to the geographical distribution of its computational units. Although a generally accepted definition is yet to be made [87, 186], the two properties of a distributed system that are commonly referred to can be listed as: (1) each computational entity in a distributed system has its own independent memory [12, 60, 87, 128, 148], and (2) the interaction between each unit of computation is carried out through message passing [12, 87, 148].

The idea of undertaking mutually exclusive tasks through concurrent processes with a common goal dates back to operating system research done in the 1960s [12]. One of the earliest examples to geographically distributed systems is ARPANET—the predecessor of Internet [1]. A majority of today's standard protocols for moderating certain types of communication between distributed units of computation—such as Ethernet, TCP/IP and FTP—were integrated into ARPANET by the end of 1970s [1]. At the application level, ARPANET e-mail was introduced in early 1970s and has become the most successful application of ARPANET. ARPANET is also one of the first large-scale distributed applications [153].

In mid-1980s, the technological leap in micro-processors, and the invention of high-speed networks have further fueled the spreading usage of distributed systems [186]. As a result, local area networks (LAN) and wide area networks (WAN), where hundreds-to-thousands of locally or globally distributed computers are connected,

have emerged. Usenet [193] and FidoNet [76] are some of the examples to WANs that were established in mid-1980s, which are still operational.

With the emergence of Internet in 1990s, the usage of distributed systems has incrementally became common. Today, distributed systems are rapidly evolving towards being a necessity rather than being an alternative practice. This is mainly due to the fact that the physical limits of how small a central processing unit (CPU) can be fabricated are almost reached [141, 108]. On the other hand, even if such limitations were somehow alleviated by new paradigms, memory bandwidth remains to be another critical bottleneck regarding how much information can be carried between storage and CPU [108]. These two issues point to the diminishing benefits of building stronger CPUs. As a matter of fact, it is often stated that the amount of computation that can be performed by a single computational unit is not likely to enhance dramatically at least within the next few years [108].

As a result, solutions based on distributed systems are now applied to a wide spectrum of problems in a number of fields. Internet, as a very large-scale distributed system of thousands-to-millions of distributed sub-systems, is perhaps the most obvious example to such applications. A few of the many other good examples are: (1) mobile and stationary ad hoc networks and their usage in areas such as emergency response [142], community networks [85, 144], inter-vehicular communication [113], sensor networks [163, 129], military applications [163], (2) large scale computing for e-commerce, e-services and scientific computing in areas such as cloud computing [7, 8], grid computing [68], overlay networks [154], network testbeds [154, 69], content distribution networks [5, 54, 53], and (3) Internet infrastructure systems such as DNS [139, 140], DNSSEC [13, 14, 15], ConfiDNS [157, 211], etc.

1.1 Motivation and Problem Description

Despite their immediately apparent advantages, distributed systems introduce additional complexities and problems that are not present in centralized systems. Aside from the most obvious ones, such as the necessity for communication, coordination, etc., an important and equally interesting problem lies in the general features of the environments that the distributed systems operate in. A majority of modern distributed systems run in dynamic, open and accessible environments where the conditions change rapidly, unpredictably, and more importantly in a continuous manner [99, 106]. In a general sense, the changes in conditions can be defined as the

fluctuations in the environment that have direct or indirect effects on both the behaviour and the performance of the computing system. The changes that a distributed system may face can be of internal or external nature—or a hybrid of both.

Like every computing system, distributed systems are built with a specific purpose defined around a certain set of pre-determined goals. The changes in conditions usually interrupt with the way that a computing system works. This, in turn, influences the system's accuracy in reaching its goals. The effects of the changes can be of any magnitude ranging from various levels of performance degradations to total system failures. A computing system needs to ensure that its goals are ultimately achieved regardless of the changes in conditions. This requires that a computing system is continuously tuned with respect to the conditions in order to perpetuate on a certain level of performance. This is generally referred to as adaptability, and is often considered a key quality for systems that run in dynamic environments [25].

The process of adaptation can be generally viewed in the form of two sequential activities. The first activity consists of collecting information in order to define the current state of the conditions in the computational environment. Awareness extracted from the collected information is then used in a second activity where the main task is to infer the way in which the system needs to be configured with respect to the perceived conditions. Naturally, these two tasks need to be repeatedly carried out in order to make sure that the system is adapting with the changing conditions continuously.

In a centralized system or a small-scale distributed system it can be feasible to manage adaptation manually. However, even in those cases it is required that the process of adaptation is carried out carefully by one or more skillful and experienced system analysts and system administrators that are fully aware and capable of what needs to be done in order to ensure that the desired level of performance is continuously achieved [106]. For instance, defining the current conditions requires extracting knowledge from the erroneous, different, partial or conflicting perceptions of many physically dispersed computational units. Furthermore, system adaptation process may require that the system parameters are re-configured at multiple locations at once. In modern and large-scale distributed systems where pervasive computing is pushed to its limits with respect to both the scales and complexity of computation, manual adaptation appears to be an unrealistic and infeasible approach—if not impossible—exceeding the abilities of even the savviest system experts [99, 106].

This prompts for a paradigm shift from building more powerful systems to building

more intelligent systems. The goal of this approach is to ultimately design, implement and deploy intelligent distributed systems that can manage their internal behaviour by autonomously adapting with the dynamic conditions in the environments they operate in. Such systems are referred to as self-managing systems [99, 106].

1.2 Scope

In this dissertation, we focus on the problem of self-management in distributed systems. In this context, we propose a new methodology for reactive self-management based on multiple criteria decision analysis (MCDA). The general structure of the proposed methodology is extracted from the commonalities of the former generic approaches that are applied in the same and different problem domains.

The reactive behaviour that is underlined by the proposed methodology focuses on adaptation as a process of reacting to certain changes in the conditions. This is different from the proactive approaches that are generally adopted in a majority of the former applications where self-management process is not a function of perceived changes but rather a continuous procedure that runs regardless [199, 188, 94]. It is important to note here that the reactive behaviour is not a new approach, and has been underlined for the design of reflex agents in the context of multi-agent systems [209, 75]. The main novelty of this work, however, lies in the usage of MCDA during the reaction processes in the context of the two problems that the proposed methodology is applied to. This is fundamentally different from the former approaches to these problems where a vast solution space is searched for an optimal or a near-optimal solution. Instead, the MCDA based reactions focus on selecting the next course of action from a pre-defined and finite set of alternative actions using mathematically explicit multiple criteria aggregation procedures (MCAP). In order to provide a detailed analysis and assessment of this new approach, we have used the proposed methodology to design distributed autonomous agents that can provide self-management in two outstanding problems. Although MCDA has been previously considered within the context of artificial intelligence—particularly with respect to algorithms and frameworks that represent different requirements for MCDA problems [27, 26, 185], to the best of our knowledge, this dissertation outlines a work where MCDA is applied for the first time in the domain of these two problems that are represented as simulation case studies.

These two problems also represent the two distinct ways in which the methodology

can be applied to self-management problems. These two cases are: 1) independent self management, and 2) coordinated self-management. In the simulation case study regarding independent self-management, the methodology is used to design and implement a distributed resource consolidation manager for clouds, called IMPROMPTU. In IMPROMPTU, each autonomous agent is attached to a unique physical machine in the cloud, where it manages resource consolidation independently from the rest of the autonomous agents. On the other hand, the simulation case study regarding coordinated self-management focuses on the problem of adaptive routing in mobile ad hoc networks (MANET). The resulting system carries out adaptation through autonomous agents that are attached to each MANET node in a coordinated manner. In this context, each autonomous node agent expresses its opinion in the form of a decision regarding which routing algorithm should be used given the perceived conditions. The opinions are aggregated through coordination in order to produce a final decision that is to be shared by every node in the MANET.

In summary, the contributions of this work can be listed as follows:

- We introduce a new methodology for reactive self-management in distributed systems which focuses on the idea of reacting to critical changes in the computing environment through MCDA. Each MCDA process attempts to select the most suitable course of action during the configuration of the system.

- We introduce IMPROMPTU, a reactive and distributed resource consolidation manager for clouds, built based on the proposed methodology. In addition, we provide a comprehensive assessment of the introduced system particularly with respect to the impact of local independent configurations—computed through MCDA—on the overall distribution of resources in a data center.

- We introduce a system based on the proposed methodology that provides adaptive routing in MANETs by switching between routing algorithms in real-time with respect to the perceived conditions in the environment. The primary focus of this new system is to coordinate the adaptation process so that the results of local MCDA processes are aggregated to a global configuration that is shared by every node in the network.

The self-managing systems that are designed using the proposed methodology are tested through several simulation runs. The assessment, analysis and comparative studies based on the collected results show that the work outlined in this dissertation

forms a strong alternative to the already existing approaches, and a solid basis for further studies regarding the application of MCDA in the domain of self-management. It is also important to note here that, although the main purpose of the proposed methodology is to provide a general structure for self-managing distributed systems, it can also be used in the case of centralized self-management.

1.3 Organization

The rest of this dissertation is organized as follows. Chapter 2 provides a brief history of self-management in terms of some of the methods that are applied to self-management problems, and the approaches that are directly concerned with the problem of self-management. In Chapter 3, we explain the general structure of the proposed methodology in terms of the activities that need to be undertaken during the course of self-management. Chapter 4 gives an in-depth survey of the mainstream MCDA methods. In Chapter 5, we provide an overview of the simulation platforms that are built or used during the assessment of the proposed methodology in the context of the independent and coordinated self-management. Chapters 6 and 7 cover the application of the proposed methodology to the problems of dynamic resource consolidation management in clouds and adaptive routing in MANETs, respectively; and provides a detailed overview of design, implementation and analyses of collected results. Finally, Chapter 8 summarizes this dissertation and outlines future directions.

Chapter 2

A Brief History of Self-Management

Based on the set of methods that are widely applied within the context of self-management in various application domains, it is safe to state that the idea of a self-managing system is based majorly on the ideas underlined in the two general disciplines of artificial intelligence and operations research [99]. Although a variety of other definitions exist, artificial intelligence can be defined in a general sense as the science and engineering of building intelligent agents and machinery [135, 172, 156]. Whereas, operations research is an interdisciplinary mathematical science that employs techniques from other mathematical disciplines in order to provide optimal or near-optimal solutions to a large variety of decision making problems [206].

The focus of this chapter is to provide a brief history and an introductory explanation of some of the methods used in these two fields that are related to the concept of self-managing systems. It is important to note here that these methods are mostly used in both artificial intelligence and operations research, and are either already applied in the domain of self-managing systems—both in a direct and indirect sense—or have the potential to be used in the future. A number of these methods are further investigated in the related work of Chapters 6 and 7 of this dissertation in terms of their domain specific applications within the context of self-management. Furthermore, we also iterate through a set of fundamental methodologies that can be considered as approaches that are directly concerned with the concept of self-management, and outline a set of common views that can be extracted from them at a glance, which in turn forms the skeleton of a majority of self-managing systems.

In this context, the rest of this chapter is organized as follows. In Section 2.1 we give a general explanation of some of the methods that are widely used in artificial intelligence and operations research, and are deemed to be closely related with the idea of self-managing systems either in whole or in part. In Section 2.2, we outline a set of methodologies and frameworks that are directly concerned with the idea of self-management. Finally, Section 2.3 concludes this chapter with a summary.

2.1 Artificial Intelligence, Operations Research and Self-Management

First introduced in mid-1950s with a focus on building machines that can intelligently undertake tasks as well as human beings, the concept of artificial intelligence quickly evolved towards design and implementation of intelligent computer programs and has been treated in that manner ever since in a majority of the applications. In 1980s, with the emergence and industrial success of expert systems, artificial intelligence once again started to attract a substantial amount of interest. The central focus of artificial intelligence includes interest in general problems such as reasoning, knowledge representation, planning, learning, communication, perception, and manipulation. In a general sense, almost all of these problems overlap with the ones that are central to the field of self-managing systems.

Operations research, on the other hand, is mostly originated from the problems that are encountered in military planning. In the decades following World War II, the techniques used in operations research were rapidly adapted to problems in business, industry and society. Today the application of such methods span a wide range of fields mostly on the mathematical models that can be used to analyze complex systems and has become an area of active academic and industrial research. Its main focus has been the solution of a range of real-life problems focusing on a set of fields such computing and information technologies, decision analysis, environment, energy and natural resources, financial engineering, manufacturing, service sciences, marketing engineering, policy modeling, revenue management, simulation, stochastic models, and transportation.

A number of the methods used in artificial intelligence and operations research and have been explicitly applied to a variety of problems in self-management. Some of these methods can be outlined as as neural networks within the context of ma-

chine learning and control, evolutionary algorithms and genetic programming within the context of machine learning, rule-based systems within the context of knowledge representation, decision making and planning, an extensive set of heuristic methods within the context of various fields, and MCDA within the context of decision making. In addition, both of the disciplines employ a number of mathematical and computational tools and models from fields such as statistics and probability theory, statistical decision analysis, statistical inference, queuing theory, game theory, and mathematical optimization. A majority of these methods have either been applied to the problems in self-management or have the potential to be applied in the near future. In the rest of this section, we are going to provide a brief explanation of some of the methods listed above as they have been applied to the problem of self-management in various contexts. Some of these applications are also going to be investigated in the related work sections of the two self-management problems that are addressed in Chapters 6 and 7.

One of such methods is the concept of a neural network. Neural networks are defined as a sets of interconnected units of computation, often called neurons, that employ mathematical and computational models in order to process information [191, 55]. Neurons in a neural network tend to exhibit complex global behaviour which is determined by the interconnections and parameters of neurons [40, 93]. Accordingly, a neural network is often an adaptive system that evolves in time by changing its structure with respect to the information flowing through the network [11, 93]. In a general sense, a neural network can be considered as a tool for non-linear statistical modeling or decision making, and is used to recognize patterns in data, and determine relationships between input and output by inferring functions from observations [40, 93, 11]. The types of problems that neural networks are applied to generally fall into categories such as function approximation, data processing and classification.

Another method, genetic programming, has its origins in evolutionary algorithms which are first used in evolutionary simulations [18]. In time evolutionary algorithms gained recognition as optimization methods that could be applied to a set of optimization and search problems within various domains, particularly in engineering [97, 114, 115]. Although they were generally applied to simple problems until recently due to their computationally intensive nature, improvements in genetic programming and growth in processing power helped producing new and remarkable results in a variety of fields such as quantum computing, game playing, sorting and searching. Essentially, genetic programming is a machine learning technique based on

biological evolution that is used to optimize a set of computer programs with respect to a measure defined by a program's fitness to perform a given task [114]. In a broad sense, in genetic programming a set of computer programs are evolved generation by generation into new programs that are ideally fitter for the task at hand. Genetic programming is a random process, and its essential randomness can lead to successful, novel and unexpected ways of solving problems [155]. The main steps in a system that follows genetic programming consist of determining the performance of each program through runs, and quantifying their ability to perform the task at hand by comparing the quality of their performances to a given ideal. The result of the second step is a numeric value that is generally called fitness. Based on the fitness values for each program, the programs are deemed fitter are used to breed new programs in order to form the next generation of programs. Two primary genetic operators are employed during the creation of new generations of programs [114, 155]: (1) crossover, and (2) mutation. The crossover operator generates a new program by combining two randomly selected parts from two programs. Whereas, the mutation operator modifies a program by modifying a randomly chosen part of it.

The idea of a rule-based system is another one of the methods that are applied in domains similar to those of neural networks and genetic programming. Rule-based systems are generally employed in problems that involve storing, manipulating and interpreting information and knowledge [121]. Perhaps the most common examples of rule-based systems are domain-specific expert system that essentially employ rules to make choices [88]. In a broad sense, a rule-based system can be created using a list of assertions—that form a working memory, and a list of rules—that determine the action on the list of assertions [121]. In this context, a rule-based system mainly consists of a set of 'if-then' statements. In expert systems, the list of rules represent the general behaviour of an expert, so that, the system acts in a manner similar to that of an expert when exposed to the same conditions [88, 101]. Generally, rule-based systems are applicable to problems where the knowledge can be represented in the form of a relatively short list of rules [121]. The main workflow of a rule-based system can be summarized as follows. Initially, the system starts with a list of rules as representations of knowledge, and a working memory. In the next steps, the system checks all the rule conditions in order to define a subset of rules—called a conflict set—the conditions of which are satisfied based on the working memory. One of the rules in the conflict set is then chosen to be triggered, and the actions specified by that rule are carried out. The rule-base can change as a result of undertaking

these actions [121]. Selecting a rule to be triggered from a conflict set depends on the chosen strategy for conflict resolution. Some of the strategies that are employed during conflict resolution can be listed as 'first applicable', 'random', 'most specific', 'most recently used', 'best rule' [88, 101].

Finally, MCDA provides a set of other methods that is applied in a variety of problems that span a wide range of application domains. Briefly, it is used in the cases where the problem is to select the most suitable alternative given a finite set of candidate alternatives and a finite set of criteria that can be used to evaluate them. The final result is generally reached through an aggregation procedure that helps selecting the most suitable alternative by taking into account the alternatives' evaluations over the criteria. General principles and a set of well-known methods of MCDA are investigated in Chapter 4 of this dissertation in detail.

2.2 Direct Approaches to Self-Management

Although the idea of self-management has been specifically addressed through a generic framework only since 2001 [99], the subject is not new and has been applied in a number of domains. One of the first forms of self-management, which is widely used in various engineering domains, is described in control theory. Control theory deals with the behaviour of dynamic systems, where the primary goal is to tune the output of a system so that it follows a certain reference—the goal of the system—continuously [70, 2]. A controller uses the inputs to the system in order to obtain the reference output of the system. The principles of control theory can be seen in one of the most well-known types of control systems, called the closed-loop—or feedback—control systems [2]. In a closed-loop control system, the output of the system is monitored by a sensor and feeds the collected data back to a controller which tunes the control in order to maintain the reference output. The concept of control loop arises from feeding the observed output back to the controller continuously, where the control attempts to tune the system output, which in turn is observed and fed back to the controller to alter the control. Figure 2.1 depicts a negative feedback loop where the controller uses the difference between the observed and the desired output of the system. This is a simple example to single-input-single-output (SISO) systems. Multiple-input-multiple-output (MIMO) systems are also common [70]. However, control theory focuses, in part, on the mathematical assurance of system properties

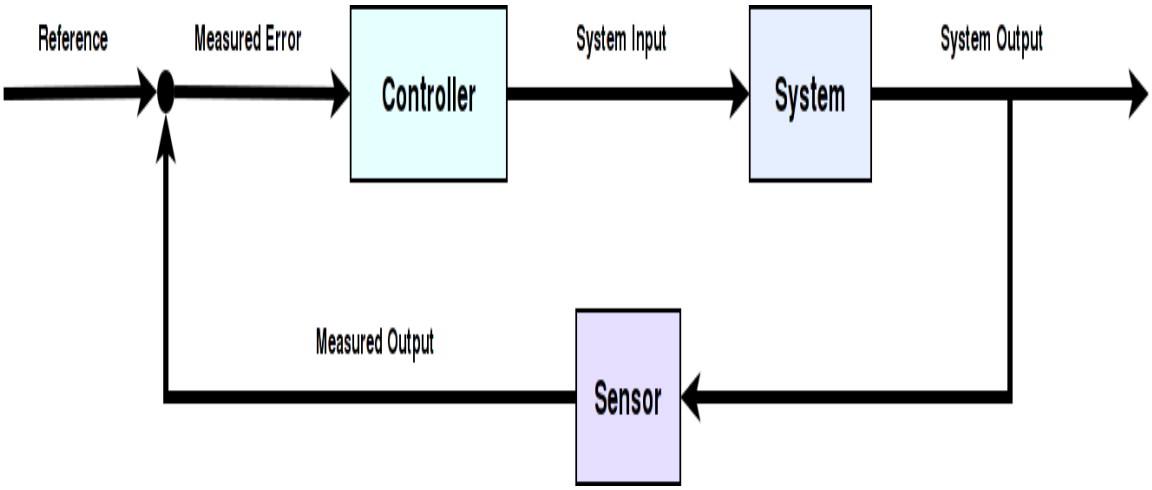


Figure 2.1: SISO Negative feedback control system.

such as stability, response time, etc. Therefore, it generally requires analytical models which limit control theory's applicability as a software control approach.

Another early concept that focuses on self-management is the observe-orient-decide-act (OODA) loop. The concept is originally applied at the strategic level in the military operations [192, 146, 122]. Currently, it is being applied to business operations and learning processes, and is an important concept in both business and military strategy [167, 146]. The concept was developed by military strategist John Boyd [192, 167, 146, 122]. In OODA loop, the decision making process is undertaken in continuous observe-orient-decide-act cycles [146]. The process is carried out in hostile situations where there exists one or more adversaries. The primary focus is to enable an entity—an individual, an organization or a system—to process the OODA loop rapidly by observing the events occurring in the environment, and acting based on the perceived reality faster than an opponent, and therefore gaining the advantage. Figure 2.2 illustrates the general structure of the OODA loop. The diagram shows that all decisions and actions are based on the observation of the conditions in the environment. The observations are basically collections of raw information that are obtained in the observe stage, and used in the orient stage in order to form knowledge and awareness—by also factoring in experience, previous knowledge, etc.—regarding the reality [146]. This knowledge is then used as an input to the decide stage, which consists of determining the next course of action towards adapting to the captured conditions. Finally, the act stage of the OODA loop represents the implementation of the selected action. Although OODA loop is not a concept that is much used in

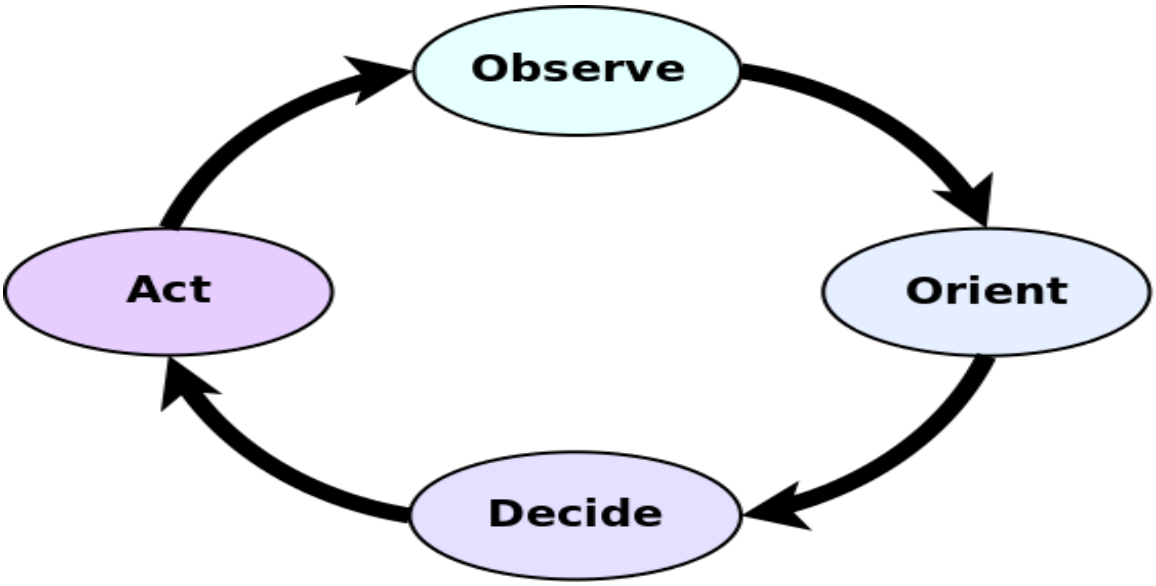


Figure 2.2: OODA Loop

the context of self-managing systems, its resemblance with the closed-loop control systems is notable.

In the field of computer systems, the idea of self-management has also been central in a number of domains particularly in the contexts of high throughput computing (HTC) [187, 125, 71, 124, 123], matchmaking and classified advertisements [162, 161], resource management [20, 19, 127, 159], checkpointing [21, 160], data intensive computing [47, 110, 111], master-worker computing [95, 96], scheduling [178], load balancing [126], storage systems [10, 9, 203], and various others. However, a generic framework for self-management was—to the best of our knowledge—first proposed by IBM in 2001 [99]. As in control theory and the concept of OODA loop, IBM's general design of an autonomic manager is also based on a feedback loop. Figure 2.3 depicts the high-level structure of the tasks that need to be carried out by an autonomous element in order to provide self-management in a system. According to this model, the autonomic manager continuously monitors the environment that the system is running in. The information collected throughout the monitoring process is stored in a knowledge base, which is used in analysis, planning and execution. Analysis is much like the orient stage in OODA loop concept, that is, the collected raw information is transformed into knowledge and awareness. Furthermore, planning is the stage where the next course of action—or set of actions—is selected, whereas, the execution stage represents the implementation of the selected strategy [106]. Since it

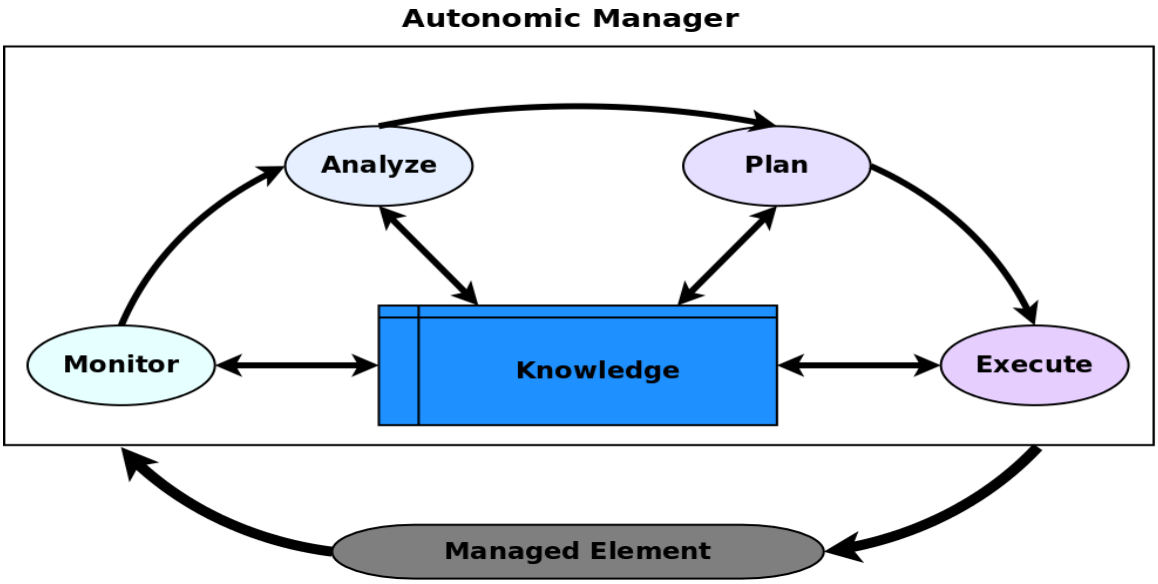


Figure 2.3: IBM Autonomic Manager

is proposed, this general framework has been commonly accepted in the domain of autonomous computing, and is used in a variety of research in the context of different applications.

Considering the fact that each of these frameworks—illustrated in Figures 2.1, 2.2, and 2.3—are from different fields, the apparent resemblance between them is remarkable. This resemblance can be used to extract a common process flow for self-management in any system. Each of these frameworks point to a general process of (1) observing the environment, (2) extracting knowledge from the collected observations, and (3) acting—tuning the system—with respect to the captured knowledge. These three activities draw the general behaviour of self-managing systems, independent from the dimensions—such as configuration, security, optimization, etc.—that need to be managed.

2.3 Summary

In this chapter, we provided a brief history of self-management. In this context, we first listed some of the methods that had a history of application in self-management problems. Some of these applications are further investigated in the related work sections of Chapters 6 and 7 of this work. It is important to note here that the listed methods are not limited to self-management problems, but also used in a variety of

domains related to the fields of artificial intelligence and operations research. Furthermore, we provided an overview of a set of approaches that are directly concerned with the problem of self-management—such as control theory, the concept of OODA loop, and IBM autonomous computing, and have addressed some of the similarities that may help in forming the skeleton of newer approaches to the problem of self-management. In the next chapter, we propose a new self-management methodology based on MCDA, and provide details regarding its general working principles and how they are extracted from the similarities in former approaches.

Chapter 3

Methodology for Reactive Self-Management Based on MCDA

In this chapter, we provide an overview of the proposed methodology for reactive self-management using MCDA. The proposed methodology outlines the general activities that need to be undertaken by the autonomous agents in a distributed system in order to facilitate the adaptation of the overall system behaviour with respect to the perceived conditions. In that sense, the primary concern of the methodology is to provide a general procedure for design and implementation of autonomous agents in systems where the adaptation process is controlled by multiple entities. However, it is important to underline that the methodology is not limited to the cases of distributed self-management. As a matter of fact, the proposed methodology can be used to design centralized autonomous control by employing the same structure with minimum modifications to certain activities.

The general structure of the proposed methodology is built on two tenets. First, the self-management is deemed as a reactive process that is undergone only when certain conditions are captured. That is, overall system adaptation is event-driven where the events are defined as specific critical, anomalous or problematic situations. Autonomous agents trigger reactions when such conditions are locally captured. Second, the triggered reactions consist of using MCDA in order to select the next course of action from a pre-defined set of alternative actions. Once again, the decisions made during the course of a reaction is produced on a strictly local basis. The local decisions produced by the autonomous agents are either treated as final decisions or used

in a system-wide aggregation process depending on the context of self-management problem at hand.

The proposed methodology adopts a reactive approach in order to avoid the potential difficulties that may arise as a result of proactive control. In the proactive approaches, the process of self-management is not a function of changes in the environment. That is, self-management is a continuous process where management cycles are undergone regardless of the changes in the conditions. However, such an approach may often impose too much burden on the system since self-management is a process that requires extra resources. In order to avoid this issue, most proactive self-management approaches aim at turning the continuous process of self-management into a series of discrete cycles. In a majority of the cases, this is done by triggering management cycles only at the end of fixed time intervals [199, 188, 94]. Although this approach removes the extra burden on the system, it introduces a new question as to how the length of management intervals are selected. The main problem with assigning a value for management intervals is that in a majority of modern distributed systems it is very hard to pre-determine a suitable value due to the level of unpredictability and uncertainties with respect to how the conditions may evolve. In addition, in some critical applications, calibration of such a value over time may not be a feasible option. Inaccurate determination of management intervals may cause multiple issues. For instance, if the intervals are too short, the system may be performing self-management unnecessarily frequent which results in wasting valuable system resources [215]. Whereas, if the intervals are too long, the responsiveness of the self-managing system is reduced [215].

A number of previous approaches to self-management in distributed systems have adopted reactive approaches to avoid such problems [215, 214, 212, 83, 82, 61]. In reactive approaches, it is not necessary to define control intervals. Instead, the self-management cycles are triggered by certain conditions. However, this approach also introduces other types of additional tasks. For instance, in the case of reactive self-management, the changes in conditions that trigger management cycles and the indicators that help capturing such changes need to be well-defined before the system is operational. This requires expert knowledge and statistical information regarding the system behaviour to be used by the autonomous agents.

Moreover, a system that is implemented using the proposed methodology carries out the reactions using MCDA. In a majority of the autonomous systems, the general view regarding self-management is based on optimization methods where either a

set of feasible solutions or an optimal is searched for in a relatively large solution space to be defined as the next course of action [199, 188, 94]. Generally, such an approach results in long management cycles bringing in the issue of outdated solutions regardless of how good the output is. In the MCDA based approach, the problem of self-management is viewed as a problem of choice instead. That is, the main idea is to pre-determine a finite—preferably a limited—set of alternative actions, and selecting the next course of action from that set following certain principles. This approach results in inherently fast management cycles, which, in turn, produces more up-to-date solutions [215]. Moreover, it is also possible to re-model most of the optimization problems as MCDA problems and apply the methodology in that context. Such a view will be assessed in detail in Chapter 6.

The proposed methodology structures self-management through four distinct activities to be undertaken by an autonomous agent. These four activities are defined to be: 1) *eliciting system objectives and performance indicators*, 2) *observing the conditions*, 3) *detecting critical changes*, and 4) *reacting through MCDA*. The elicitation activity needs to be carried out before the system is operational. During this step, the main focus is first to define the ideal system behaviour in terms of the goals to be achieved by the system, and second, to define the indicators that help assessing how well the system goals are being achieved. After this activity is undertaken, the system is ready to start operation, in which the self-management is undertaken through the activities of observing the conditions, detecting critical changes, and reacting through MCDA in an ongoing cycle. Figure 3.1 illustrates the high-level structure of the proposed methodology in terms of the pre-operation and in-operation activities.

In the rest of this chapter, we elaborate more on the details of the in-operation activities with respect to their inputs, their expected progress, and their outputs. Accordingly, Sections 3.1, 3.2, and 3.3 overviews each in-operation activity that needs to be undertaken by the autonomous agents, respectively.

3.1 Observing the Conditions

An autonomous agent's main purpose during this activity is to collect information regarding the current state of the system that it is controlling and the surrounding environment in order to have an up-to-date understanding of the general conditions. The information that is being collected must reflect on the system's current ability to fulfill its objectives. In that sense, observations are done strategically by closely

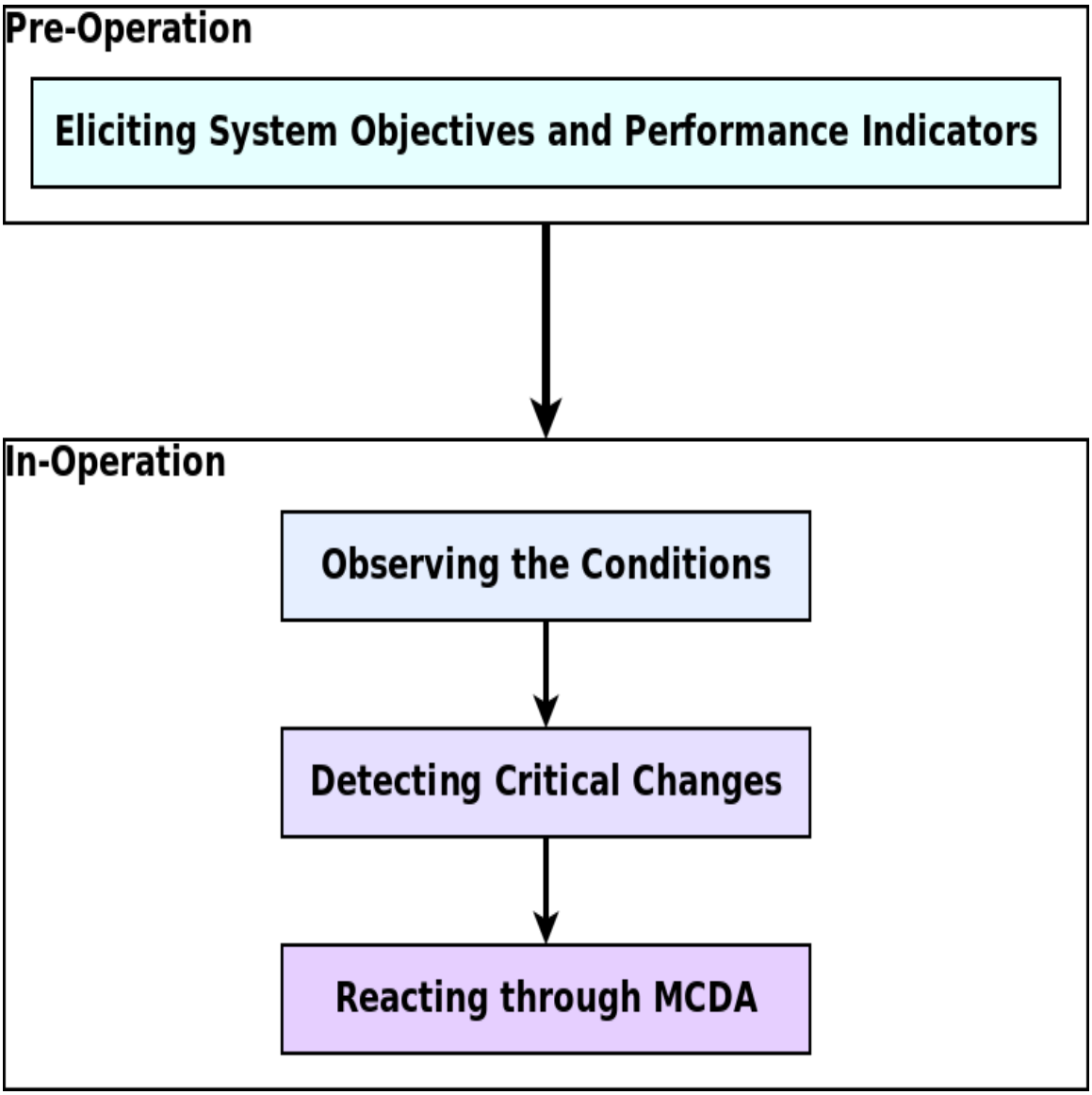


Figure 3.1: The high-level structure of the methodology for reactive self-management using MCDA.

watching a limited set of indicators that are defined before the system is operational, which are deemed to provide insight on how well the system objectives are being achieved.

In this context, the indicators can be of internal or external nature. The internal indicators are used to capture information regarding the internal dynamics of the system. Accordingly, these are used to determine the conditions on a strictly local basis. Whereas, external indicators are used to reach an understanding regarding

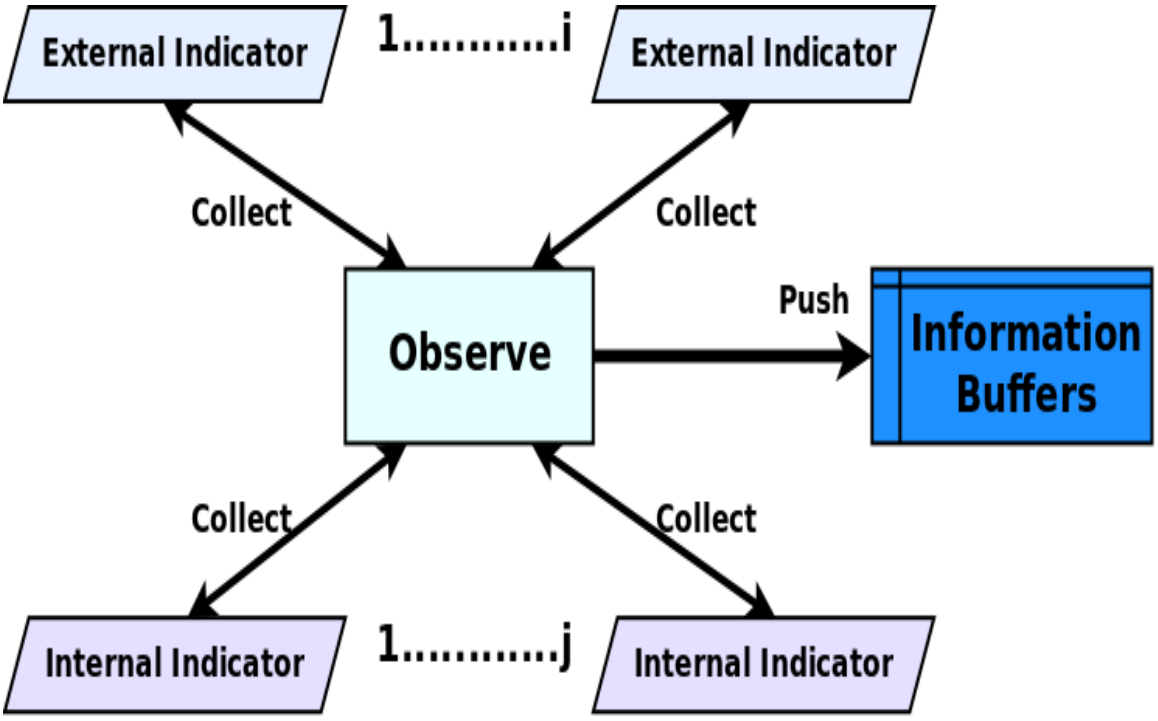


Figure 3.2: Observing the conditions.

the environment that surrounds the system, and are of partial or global value. The observation activity may need to gather information using one or both types of such indicators. In that sense, information collection may be performed by polling certain system variables and other internal sources of information, or querying sensors, certain peripherals and other external sources of information. The frequency and the manner in which the information collection is carried out is mostly application dependent.

The information that are collected during the observation activity need to be stored by the autonomous agents for further use during other activities. In this particular activity, the output of the process is used as input by the detection activity. Figure 3.2 illustrates a high-level view of the observation activity in terms of the sources of input and output buffers.

3.2 Detecting Critical Changes

An autonomous agent's main purpose in this activity is to generate a perception of change using the raw information collected during the observation activity. The perceptions are formed in terms of capturing whether the observed conditions require

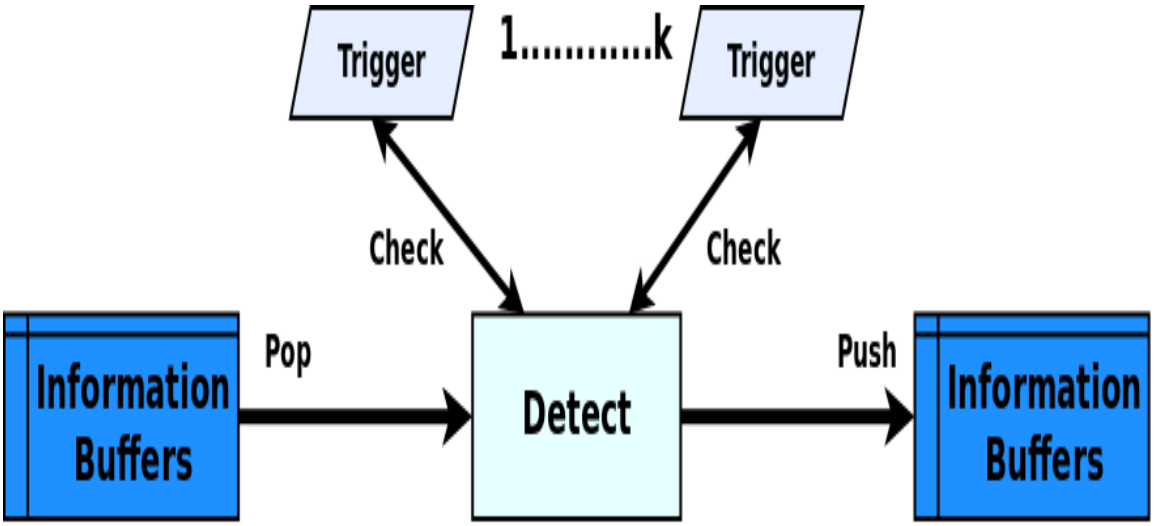


Figure 3.3: Detecting critical changes.

a reaction from the system. This is done by feeding collected observations into predefined triggers which, in turn, are used to determine if the systems performance is desirable given the current conditions. Note that the triggers that are used during the detection activity can require one or more indicators as input, where the set of indicators that are fed to the triggers can be both disjoint or intersecting. If any critical changes in conditions that require a reaction are captured during this activity, the autonomous node agent initiates a reaction by storing necessary information to be used during the course of the reaction activity. Figure 3.3 illustrates a high-level view of the detection activity.

3.3 Reacting through MCDA

This activity consists of choosing the best course of action from a set of alternative actions that is determined during the elicitation activity. The process of selection is carried out based on the critical changes captured during the detection activity. Accordingly, the reaction activity uses the necessary criteria in order to evaluate each alternative actions, and as a result elect the most suitable course of action—often a compromise alternative—depending on the perceived changes in conditions. The implementation of the selected alternative is carried out simply by modifying a set of certain system parameters. Accordingly, the course of reaction is mainly defined

by the types of system parameters to be changed. In a broad sense, the types of variables are defined by the methodology as local variables and global variables.

The local variables can be identified as variables which can be modified by an autonomous agent without the consent of the rest of the system. That is, if a variable to be configured does not need to share a common value throughout the distributed system, then it is deemed to be local, which in turn implies that it can be configured independently. Some examples to such parameters can be listed as the transmission power in a wireless network, IP table entries on network nodes, etc [61, 212]. Note that, tempering with the values of such a parameter may have global effects on system behaviour, but, configurations can still be done on a strictly local basis without causing any vital failures.

On the contrary, global variables are identified as variables that must have a value that is common to every computational unit throughout the system at any given time. Then, configuration of such variables requires coordination among autonomous agents where the opinions of each autonomous agent is taken into account during the configuration procedure. Some examples to such parameters in the context of computer networks can be listed as routing algorithms, media access methods, low-level error detection methods etc [61, 212].

In this context, the general process that is undertaken during the reaction activity differs based on these two types of system variables. If the the variable to be configured is local to each autonomous agent then this activity can be generally called an *independent reaction*. Whereas, if the system variable is of global nature, the term *coordinated reaction* is more suitable. The following two sections—Sections 3.3.1 and 3.3.2—gives details on these two different procedures, respectively.

3.3.1 Independent Reactions

An independent reaction is carried out by an autonomous agent in a completely isolated manner. During this procedure the autonomous agent selects the best course of action using MCDA, and implements the output of the MCDA process directly without articulating its perception with the rest of the autonomous agents in the system. The process is carried out as follows. Once the reaction process is triggered, the information that are outputted by the detection activity are used for the evaluations of alternative actions over each criterion. These evaluations are used during the process of MCDA in order select a dominating action—or, to be more realistic,

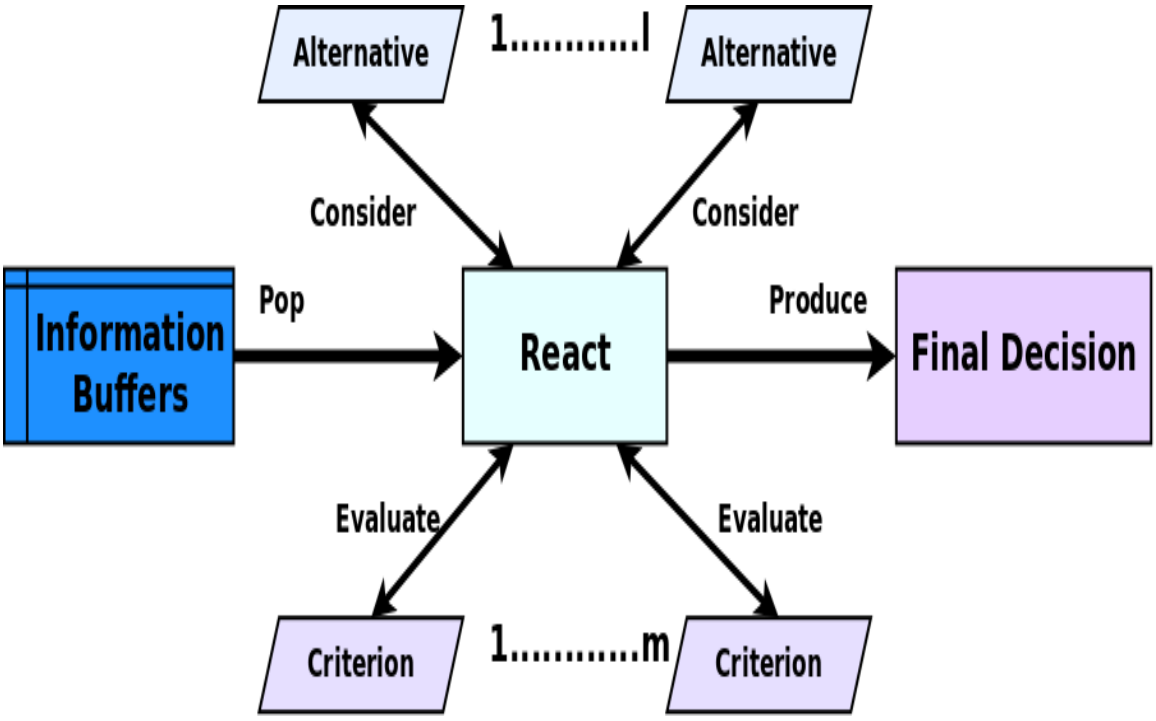


Figure 3.4: Reacting through MCDA, the independent case.

the best compromise action—as the next course of action to be implemented. The implementation of the final decision is carried out through re-configuring a predefined set of system parameters to the decided values. A high-level view of the independent reaction activity is illustrated in Figure 3.4 in terms of the criteria and the alternative actions used during the MCDA process and the final decision.

3.3.2 Coordinated Reactions

In the case of the coordinated reactions, an autonomous agent first computes a local decision—or an opinion—using MCDA, following the exact same procedure as in the case of independent reactions. However, differently from the independent reactions, the local decision that is produced by an autonomous node agent is not deemed as a final decision, and it now needs to be articulated throughout the system as a representation of an individual opinion. The same process is carried out by every other autonomous agent in the distributed system. The aim is to form an accurate and complete knowledge based on the individual opinions collected from physically dispersed autonomous agents. Note that this is fundamentally different from approaches that articulate raw information to form a complete view [83, 82, 61].

Accordingly, such an approach also requires additional measures indicating the quality of each opinion being communicated during the reaction process. In essence, this implies that the opinions that are generated by different autonomous agents have varying levels of effect on the final decision to be generated as a result of the coordination process. As a result, the coordination process need to support such a view taking into account different quality measures during the aggregation of multiple opinions into a final network-wide decision. The measure to be used should reflect on the level of trust assigned to the judgement of different autonomous agents, which in turn is a function of the accuracy of their view of the reality. For instance, an autonomous agent that acts as a hub in a certain application is likely to have a better—or at least a more complete—view of the reality in comparison to autonomous agents that are leafs. This needs to be represented in the aggregation process by assigning the hub node a higher impact factor on the final decision. Note that the impact values that are assigned to the autonomous agents are not necessarily constant throughout the lifetime of the system. For instance, assuming that such values remain constant in a self-management application in mobile ad hoc networks may be erroneous, since each node—and thus the autonomous agent that is attached to it—may have changing views of the reality over time simply due to changing measures such as mobility.

Another issue that needs to be taken into account during the coordination is rather technical, and stems from the difficulties of forming a complete and unified perception in a distributed system. It is plausible to think that it is necessary to have an identical collection of opinions—with an identical collection of impact factors—at each autonomous agent after the articulation process. That is, in order to configure a global parameter in a synchronized manner, each autonomous agent need to undertake the exact same aggregation process over the exact same set of opinions. However, there are technical limitations as to how the articulation process can be carried out. One important problem that needs to walked around is the LFP impossibility [78, 82] since one or more autonomous agents can fail to articulate their opinions during the management cycles. Thus, in order to overcome this issue, the proposed methodology adopts a multiple phase commit approach—with weak fault detection, such as the usage of time-bounds, etc.—to be employed during the coordinated reactions. Naturally, the process can no longer be purely distributed due to the usage of a coordinator during the multiple phase commit procedure. In this context, the coordination is lead by a single autonomous agent that takes on the responsibility to provide a final decision on behalf of the entire system using the individual opinions that it receives from

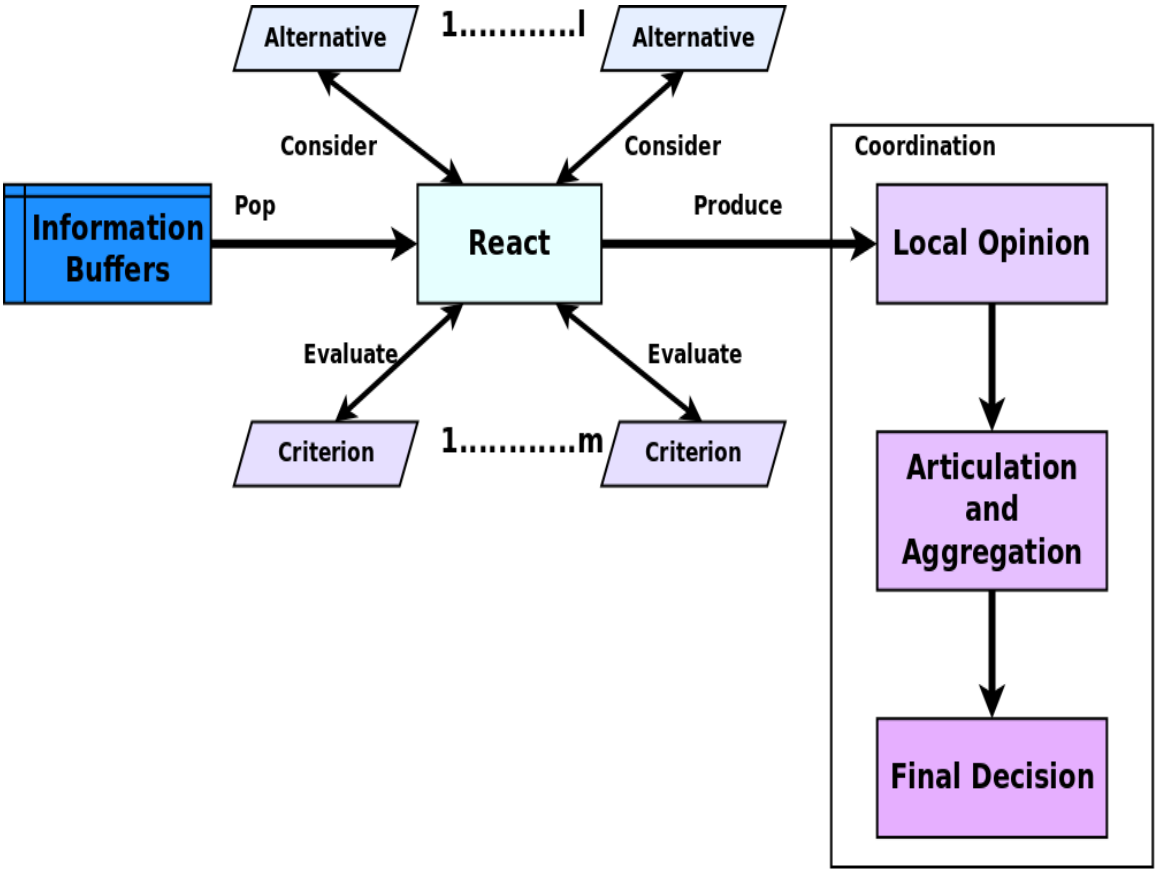


Figure 3.5: Reacting through MCDA, the coordinated case.

other autonomous agents. It is important to note here that, unlike an elected leader, the coordinator emerges as the first one to capture a critical change. However, this also means that the system needs go through a synchronization procedure in order to eliminate the issue of multiple emerging coordinators. Once that issue is resolved, a single coordinator carries out the process of aggregation. In a sense, the coordinator is merely a ballot box—or simply a moderator—with the ability to express its opinion during the aggregation process, and has no extra significance or impact on the final decision. Once the coordinator has the opinions from each autonomous agent, it generates a final network-wide decision factoring in the importance of each autonomous agent. The final decision is then imposed on every autonomous agent and the new value of the global parameter is set identically at every point of control. Figure 3.5 illustrates a high-level view of the coordinated reaction activity.

3.4 Summary

In this chapter we have introduced a general methodology for reactive self-management based on MCDA. In this context, we have viewed the general procedures that is necessary to build a self-managing distributed system. The methodology views the adaptation process from a reactive point of view where the self-management cycles are reactions to encountering certain critical changes in conditions. Once a reaction is triggered by the conditions, the methodology uses MCDA to select the next course of action from a set of alternative actions. In this context, system objectives, performance indicators, critical changes, set of alternative actions and the multiple criteria decision model must be carefully determined before the system is operational. Once the modeling is complete, autonomous agents that are designed using the methodology manage system adaptation through a continuous cycle of observe-detect-react activities. The manner in which the reaction is carried out is defined by the type of parameters to be configured in order to perform self-management. Accordingly, we have covered these different types in the context of independent reactions and coordinated reactions. These two approaches will be applied to two outstanding problems in Chapters 6 and 7. However, before we go into the details of how the methodology can be applied to these problems, we will first review the main stream MCDA methods and the simulation platforms built and employed for the assessment of the methodology in Chapters 5 and 6, respectively.

Chapter 4

Multiple Criteria Decision Analysis Methods

In a majority of real-world multiple criteria decision making problems, it is very difficult to come across an alternative that performs better that all of the other alternative actions over each and every one of the evaluation criteria. For instance, consider a multiple criteria decision making problem with A as the set of alternative actions and C as the set of criteria to be used to evaluate the performances of the alternative actions in A. For $a_1, a_2 \in A$, alternative a_1 is said to dominate an alternative a_2 if, $c(a_1) \geq c(a_2)$, $\forall c \in C$, and $\exists c_l \in C$ such that $c_l(a_1) > c_l(a_2)$. That is, in order for an alternative action to dominate another, it has to be at least as good as the other alternative action with respect to its performance on every single criteria, and there needs to be at least one criteria on which it performs strictly better than the other alternative action. Accordingly, an alternative action is said to be *Pareto Optimal*, if it is outperformed by no other alternative action over any evaluation criteria. In most of the situations, an alternative action that dominates all of the rest is often absent. That is, the performances of alternative actions over different criteria can reveal conflicting evaluations where an alternative action may be deemed as the most suitable one whereas it performs rather poor on another criterion. Moreover, the evaluations over different criteria may span a set of heterogeneous and non-commensurable measurement scales. In such situations, the main problem is to form a comprehensive judgment as to which alternative action should be considered as the better-performing one by taking into account the performances of each alternative

action over every evaluation criteria. The problem of forming such a comprehensive judgment is generally referred to as an *aggregation problem* [169].

Aggregation problems are the main focus of various operational approaches in the MCDA literature. These approaches are generally considered to fall under one of the following two categories [169]: 1) Methods that are based on mathematically explicit multiple criteria aggregation procedures (MCAP), and (2) Methods that make use of interactivity with the decision maker when the mathematical procedure remains implicit.

Methods that are based on MCAPs aim at giving a clear answer to the aggregation problem for any pair of alternative actions by considering a number of *inter-criteria parameters* and a *logic of aggregation* [169]. Inter-criteria parameters help in defining the specific roles and importances that each criterion can be assigned with respect to the others. Weights, scaling constants, veto thresholds, aspiration levels and rejection levels are some of these inter-criteria parameters [30, 168]. On the other hand, a logic of aggregation considers both the specific types of dependencies between the evaluation criteria and the conditions where compensation between performances is accepted or refused. The inter-criteria parameters need to be assigned numerical values following the logic of aggregation of the considered MCAP, for the assigned values have no meaning outside of this logic [30, 168]. A majority of the methods based on MCAPs are categorized under two operational approaches: 1) Multiattribute Utility Theory Methods and 2) Outranking Methods [197]. These two categories are also referred to as Methods Based on a Synthesizing Criterion and Methods Based on a Synthesizing Preference Relational System [169], or alternatively as Single Synthesizing Criterion Approach without Incomparability and Outranking Synthesizing Approach [92]. It is also important to note here that not all methods based on MCAPs necessarily fall under these two categories [42, 169]. Some of the other approaches include methods based on evolutionary algorithms and simulated annealing [49], rough sets [90, 183], artificial intelligence [151], and fuzzy subsets [137]. Note that the MCAP based approaches differ from the approaches based on multi-objective optimization in a sense that they do not aim for reaching a set of feasible solutions as a result of the process they undertake. Rather, they focus on outputting either a single alternative action as the most suitable solution, a ranking of the alternative actions or a sorting of the alternative actions in to categories, depending on the choice problematic.

In the methods that make use of interactivity, a formal procedure for asking questions to the decision maker is modeled [92]. The procedure leads to an ad hoc sequence

of judgments and progression by trial and error [169, 92]. Interactive methods are also applied to a number of multiple criteria decision making problems [86, 152, 184, 195].

In this chapter, we focus on the methods that are based on MCAPs since they are inherently more applicable to the problems addressed in this dissertation due to their mathematically explicit nature. In the rest of this chapter, we investigate the two operational approaches to methods based on MCAPs separately. In this context, we provide an overview of the Multiattribute Utility Theory Methods and the Outranking Methods in Sections 4.1 and 4.2, respectively. It is important to note here that this overview covers a representative set of well-known and commonly used MCDA methods rather than providing an exhaustive survey of every existing method that has been proposed and applied in the domain. Finally, Section 4.3 concludes this chapter with a brief summary.

4.1 Single Synthesizing Criterion Methods

The approach based on the Single Synthesizing Criterion is often considered as the most traditional one [169]. Methods that are based on this approach lead to a complete pre-order of the alternative actions where there is no room for incomparability between alternative actions. The complete pre-order is reached through formal rules that map each alternative action to a position on an appropriate scale. In general, the formal rules consist of mathematical formulae that explicitly define a unique criterion which synthesizes all of the criteria that are relevant to the decision problem at hand—hence the alternative names for this approach. Furthermore, imperfect knowledge can also be taken into account in Single Synthesizing Criterion methods through probabilistic or fuzzy models [169, 64].

In the rest of this section, we continue our investigation further by focusing on some of the most well-known MCDA methods based on this approach. Section 4.1.1 briefly explains the Multiattribute Utility Theory (MAUT) as the foundations of this approach [64]. In Sections 4.1.2 and 4.1.3 we describe the UTA Methods and the Analytic Hierarchy Process (AHP) respectively [182, 173]. Finally, Section 4.1.4 briefly introduces some of the other methods based on this approach.

4.1.1 Multiattribute Utility Theory

There are several multiattribute models based on alternative sets of axioms, which are covered by the MAUT. For this very reason, the term multiattribute preference theory will be used instead of MAUT as it is a more general form that covers each of these models [64]. In order to facilitate a better understanding of the subject, we begin this overview with an investigation of the single attribute preference theory by exploring the basic representations of a decision maker's preferences, reflections of which will also be seen in multiattribute preference theory.

Preference theory studies the general aspects of individual choice behaviour by attempting to provide ways in which an individual's preferences over a set of alternative actions can both be identified and quantified in addition to providing outlines to construct appropriate preference representation functions for decision making [64]. In this context, preference theory is based on rigorous axioms that are essential for establishing this point of view.

Preference theory investigates the preference behaviour of a decision maker under two categories: 1) Preference under certainty, and 2) Preference under risk. Accordingly, preference representation functions under certainty and preference representation functions under risk are generally referred to as *value functions* and *utility functions*, respectively [105].

Preference theory studies an individuals preference behaviour through a binary preference relation $\succ$ on a set of alternative actions A. Let b and c be two alternative actions in A. If b is preferred to c, then $b \succ c$, where $\succ$ represents a strict preference. On the other hand, if b and c are indifferent, then $b \sim c$, where $\sim$ represents the absence of strict preference—that is, $b \not\succ c$ and $c \not\succ b$. In addition, a weak preference relation, $\succeq$, can also be defined as the union of strict preference, $\succ$, and indifference, $\sim$—that is, both $b \succ c$ and $b \sim c$.

Preference theory is based on some basic axioms of individual preference behaviour. One of these axioms is *asymmetry* which states that a decision maker's preferences should be expressed without contradiction, that is, a decision maker does not strictly prefer b to c and prefer c to b simultaneously. This can also be considered as an axiom of preference consistency [64]. Another basic axiom of preference theory is *negative transitivity* which states that if a decision maker makes a judgment that b is preferred to c, then it should be possible to place any other alternative action

$d \in A$ somewhere on the ordinal scale. That is, if $b \succ c$, then it follows that either $b \succ d$ or $d \succ c$, or both.

A preference relation that is asymmetric and negatively transitive is called a weak order. The weak order is a basic assumption in many preference studies, and implies a certain set of desirable properties for a preference ordering [64]. Some of these properties can be summarized as [32, 80, 117]: 1) Strict preference, $\succ$, is transitive, 2) Indifference, $\sim$, is transitive, reflexive and symmetric, 3) Exactly one of $b \succ c$, $c \succ b$ and $b \sim c$ holds for each pair of alternative actions b and c, and 4) Weak preference, $\succeq$, is transitive and complete. Hence, a decision maker whose preference behaviour can be represented by a weak order can rank all alternative actions in a unique order.

If strict preference, $\succ$, is a weak order and A is finite or denumerable, then preferences under certainty can be represented numerically through a real-valued function, $\mathring{v}$, on A such that $b \succ c$ if and only if $\mathring{v}(b) > \mathring{v}(c)$, for all $b, c \in A$. Since $\mathring{v}$ is a preference representation under certainty, it is referred to as a value function [64, 105]. The value function, $\mathring{v}$, is order preserving since the values of $\mathring{v}(b), \mathring{v}(c), \ldots$ ordered by $>$ are consistent with $b, c, \ldots$ ordered by $\succ$. Accordingly, monotonic transformations of $\mathring{v}$ are also order preserving. Note that, the value function, $\mathring{v}$, implies no meaning other than the ordering of the alternatives in the set of alternative actions. Therefore, $\mathring{v}$ is an ordinal value function.

Instead of merely ordering the alternative actions, the preference behaviour of a decision maker can further be granulated through notions such as *strength of preference between pairs of alternative actions* in order to reduce ambiguity. The term *measurable value function* is used to represent such value functions which can be employed to order the difference in the strength of preferences between pairs of alternative actions. In order to elaborate more on this, let us consider a set of alternative actions $b, c, d, e, b', c', d', e' \in A$, and define A^* and $\succeq^*$ as a non-empty subset of A^2 and a binary relation on A^*, respectively. Note that $bc \succeq^* de$ mean that the strength of preference for b over c is greater than or equal to the strength of preference for d over e. For reasons such as organization, simplicity and consistency, in the rest of this discussion, we are going to focus on an axiom system that is based on a positive difference structure—that is negative differences in strength of preferences are not allowed [116].

The set of axioms necessary for such representations include several technical assumptions in addition to the weak order assumption with no significant implica-

tions on preference behaviour [64]. However, a key axiom is the following one. If $bc, cd, b'c', c'd \in A^*$, $bc \succeq^* b'c'$ and $cd \succeq^* c'd'$, then $bd \succeq^* b'd'$. That is, if the difference in the strength of preference of b over c exceeds the difference in the strength of preference of b' over c', and the difference in the strength of preference of c over d exceeds the difference in the strength of preference of c' over d', then the difference in the strength of preference of b over d must exceed the difference in the strength of preference of b' over d'. The set of axioms imply that there is a real-valued function v on A such that, for all $b, c, d, e \in A$, if b is preferred to c and d is preferred to e, then $bc \succeq^* de$ if and only if $v(b) - v(c) \geq v(d) - v(e)$. Furthermore, v is unique up to a positive transformation, that is, v' also satisfies the same property for real numbers $x > 0$ and y such that $v'(b) = xv(b) + y$ for all $x \in A$. Finally, v is a cardinal function that provides an interval scale of measurement [64]. The binary relation $\succeq$ on A is defined from the binary relation $\succeq^*$ on A^* by requiring $bc \succeq^* de$ if and only if $b \succeq c$ for all $b, c, d \in A$. Then it is clear that $b \succeq d$ if and only if $v(b) \geq v(d)$. Thus, v is a measurable value function on A. A measurable value function can be thought of as a unique preference function under certainty that reveals the marginal value of additional units of the underlying commodity [64, 105].

In the case of preference representations under risk, the risk is generally defined in terms of lotteries or gambles, outcomes of which depend on the occurrences from a set of mutually exclusive and exhaustive events [198, 64]. In this context, a lottery or a gamble can be exemplified as the flip of a coin with different gains assigned to each outcome. The most significant contribution in this field is the formalization of the expected utility theory [198] which is commonly represented in terms of three basic axioms as follows [80]. Let P be a convex set of probability distributions, random variables, lotteries or gambles, $\{p, q, r, \dots\}$, on a non-empty set A of alternative actions. Then for random variables $p, q, r \in P$ and all λ, with $0 < \lambda < 1$: 1) Strict preference, $\succ$, is a weak order. 2) If $p \succ q$ then $(\lambda p + (1 - \lambda)r) \succ (\lambda q + (1 - \lambda)r)$ for all r in P, and 3) If $p \succ q \succ r$ then there exist some $0 < \alpha < 1$ and $0 < \beta < 1$ such that $\alpha p + (1 - \alpha)r \succ q \succ \beta p + (1 - \beta)r$. These axioms are referred to as the axioms of *ordering*, *independence*, and *continuity*, respectively [64]. In the expected utility theory, these three basic axioms hold if and only if there exists a real-valued utility function u such that for all $p, q \in P$, $p \succ q$ if and only if $\sum_{x \in A} p(b)u(b) \geq \sum_{x \in A} q(b)u(b)$. Furthermore, u is unique up to a positive linear transformation.

This model is also used in characterizing risk attitude, where the decision maker is deemed to be risk averse, risk neutral or risk seeking if the decision maker's utility

function is concave, linear or convex, respectively. The theory is also extended to a subjective expected utility representation by allowing for subjective probabilities to be determined for the outcomes of a utility function [177]. Although, the expected utility theory has played a major role in the prescriptive analysis of decision problems, its assumptions are often challenged by empirical studies [104].

This concludes our brief overview of the single attribute preference representations under certainty and risk. From this point on, we investigate multiattribute preference representations by focusing on notions such as ordinal and measurable multiattribute value functions and cardinal multiattribute utility functions.

Ordinal Multiattribute Value Functions

The primary emphasis of the work on MAUT is on the decomposition of a preference function into simple polynomials. In order to investigate this further, let us suppose that the alternative actions are now represented by a vector of attributes. To be more specific, let $A = \prod_{i=1}^{n} A_i$ be the set of alternative actions, where A_i represents the set of alternative actions for the i^{th} attribute—or criterion. Naturally, multiattribute cases focus on the determination of $(b_1, b_2, \ldots, b_n) \succeq (c_1, c_2, \ldots, c_n)$ if and only if $\mathring{v}(b_1, b_2, \ldots, b_n) \geq \mathring{v}(c_1, c_2, \ldots, c_n)$. In essence, the only property that is required is that the decision maker's preferences are a weak order on the vectors of attribute performances over each alternative action. As for the decomposition of a multiattribute preference function, additional conditions need to be defined.

Perhaps the most general approach to multiattribute decision problems is the usage of additive representations [64]. In order to further investigate additive representations, let us assume that there is a most preferred alternative action b_i^* and a least preferred alternative action b_i^0 on each attribute $i = 0, 1, \ldots, n$. A real value, $\mathring{v}$, is assigned to each alternative action, $(b_1, b_2, \ldots, b_n)$, using $\mathring{v}(b_1, b_2, \ldots, b_n) = \sum_{i=1}^{n} \mathring{v}_i(b_i)$ where each $\mathring{v}_i$ is a single attribute value function. It is also possible to use scaling through the least preferred alternative action, $\mathring{v}_i(b_i^0) = 0$, and the most preferred alternative action, $\mathring{v}_i(b_i^*) = 1$, which results in $\mathring{v}(b_1, b_2, \ldots, b_n) = \sum_{i=1}^{n} \mathring{\lambda}_i \mathring{v}_i(b_i)$ where $\sum_{i=1}^{n} \mathring{\lambda}_i = 1$.

In the cases where the decision maker's purpose is to rank-order the alternative actions, the key condition for building an additive representation of a multiattribute preference function is *mutual preference independence*. Let us assume that A_I is a subset of the set of all attributes, where $I \subset \{1, 2, \ldots, n\}$ is a subset of the at-

tribute indices; and let $\bar{A}_I$ represent the complement of A_I. Then the conditions for mutual preference independence are: 1) A_I is preference independent of $\bar{A}_I$ if $(b_I, \bar{b}_I) \succeq (c_I, \bar{b}_I)$ for any $b_I, c_I \in A_I$ and $\bar{b}_I \in \bar{A}_I$ implies $(b_I, \bar{c}_I) \succeq (c_I, \bar{c}_I)$ for all $\bar{c}_I \in \bar{A}_I$, and 2) The attributes $A_1, A_2, \ldots, A_n$ are mutually preference independent if for every subset $I \subset \{1, 2, \ldots, n\}$ the set A_I of these attributes is preference independent of $\bar{A}_I$. That is, mutual preference independence requires that all pairs of attributes A_i and A_j are preference independent, and preference independence holds if the trade-off between each pair are independent of the rest of the attributes. Briefly, mutual preference independence implies that the indifference curves for any pair of attributes are unaffected by the fixed levels of the remaining attributes [64]. Along with some additional conditions and technical assumptions, mutual preference independence implies the existence of an additive multiattribute value function for $n \geq 3$ attributes, which is unique up to a positive linear transformation.

Cardinal Multiattribute Utility Functions

In the cases that involve risk, a multiattribute utility function, $u(b_1, b_2, \ldots, b_n)$, can be decomposed into additive, multiplicative or other well-structured forms given that $A = \prod_{i=1}^{n} A_i$ is a von Neumann-Morgenstern utility model and a certain set of independence conditions are met by the decision maker's preferences. Some of these independence conditions are: 1) *utility independence*, and 2) *additive independence*.

An attribute, A_i, is utility independent of its complementary attributes, $\bar{A}_i$, if preferences over lotteries with different levels of A_i do not depend on the fixed levels of the rest of the attributes [105]. Accordingly, attributes $A_1, A_2, \ldots, A_n$ are mutually utility independent if all proper subsets of these attributes are utility independent of their complements [64]. Furthermore, if attributes are mutually preference independent, then they are also mutually utility independent if pairs of attributes are utility independent of their complements [105].

If the attributes are mutually utility independent, $u(b_1, b_2, \ldots, b_n)$ can have the multiplicative form, $1 + ku(b_1, b_2, \ldots, b_n) = \prod_{i=1}^{n} [1 + kk_i u_i(b_i)]$, where u_i are single attribute functions over A_i scaled from 0 to 1, $0 \leq k_i \leq 1$ are positive scaling constants, and k is an additional scaling constant. Note that in the cases where k is set to 0, the multiplicative form reduces to the additive form $\sum_{i=1}^{n} k_i u_i(b_i)$ where $\sum_{i=1}^{n} k_i = 1$.

In order to decompose a von Neumann-Morgenstern utility function into an additive form additive independence is a required property where the key condition is

the marginality condition [79]. Marginality condition states that the preferences for any lotteries $p, q \in P$ must depend only on the marginal probabilities of the attribute values, and not on their joint probability distributions. Also, there are other independence conditions that are defined for non-additive decomposition of multiattribute utility functions [74].

Measurable Multiattribute Value Functions

Some extra notions are needed to investigate the strength of preferences on multiattribute alternatives. In order to do that, let us consider the preference relation $\succeq$ on A, which is a weak order. Also, let $A^* = \{bc : b, c \in A\}$ be a nonempty subset of A^2, and $\succeq^*$ denote a preference relation, which is a weak order on A^*. As it is used in the case of single attribute measurable value functions, $bc \succeq^* de$ states that the preference difference between b and c is greater than the preference difference between d and e. There is an apparent relationship between $\succeq$ on A and $\succeq^*$ on A^*. For a better understanding of this relationship, let us assume that the attributes $A_1, A_2, \ldots, A_n$ are mutually preference independent. The orders gathered through $\succeq$ and $\succeq^*$ are difference consistent if, for all $b, c \in A_i$, $(b_i, \bar{b}_i) \succeq (c_i, \bar{b}_i)$ if and only if $(b_i, \bar{b}_i)(c_i^o, \bar{b}_i) \succeq^* (c_i, \bar{b}_i)(c_i^o, \bar{b}_i)$ for some $c_i^o \in A_i$ and some $\bar{b}_i \in A_i$, and for any $i \in \{1, 2, \ldots, n\}$, and if $b \sim c$ then $bd \sim^* cy$ or $db \sim^* dc$ or both for any $d \in A$ [64]. That is, if a multiattributed alternative is preferred to another differing only on its performance over attribute A_i, then the preference difference between that alternative and some common reference alternative $(c_i^o, \bar{b}_i)$ will be larger than the difference between the alternative that is not preferred and the common reference alternative.

A condition that can be used to construct multiplicative and other non-additive forms of measurable value functions is referred to as *weak difference independence*. This condition is similar to the utility independence condition in multiattribute utility theory [64, 63]. Briefly, A_I is weak difference independent of $\bar{A}_I$ if the ordering of preference differences depends only on the performances over the attributes A_I and not on the fixed values of $\bar{A}_I$. That is, given any $b_I, c_I, d_I, e_I \in A_I$ and some $\bar{b}_I \in \bar{A}_I$, if $(b_I, \bar{b}_I)(c_I, \bar{b}_I) \succeq^* (d_I, \bar{b}_I)(e_I, \bar{b}_I)$ then $(b_I, \bar{c}_I)(c_I, \bar{c}_I) \succeq^* (d_I, \bar{c}_I)(e_I, \bar{c}_I)$ for any $\bar{c}_I \in \bar{A}_I$. Accordingly, the attributes are *mutually weak difference independent* if all of their proper subsets are weak difference independent of their complements. Furthermore, if the attributes are mutually preference independent, then they are also mutually weak difference independent given that any pair of attributes is weak

difference independent of its complement [63]. Weak difference independence leads to the decomposition of a measurable value function that is identical to the one implied by utility independence for utility functions [64]. More specifically, a measurable multiattribute value function $v(b_1, b_2, \ldots, b_n)$ on A can have the multiplicative form, $1 + \lambda v(b) = \prod_{i=1}^{n}[1 + \lambda\lambda_i v_i(b_i)]$ if and only if $A_1, A_2, \ldots, A_n$ are mutually weak difference independent, where v_i is a single attribute measurable value function over A_i scaled from 0 to 1, λ_i are positive scaling constants, and λ is an additional scaling constant. If $\lambda = 0$, then the multiplicative form reduces to the additive form, $v(b) = \sum_{i=1}^{n} \lambda_i v_i(b_i)$, where $\sum_{i=1}^{n} \lambda_i = 1$.

In order to construct an additive measurable multiattribute value function, a condition called *difference independence* is required. In this context, A_i is difference independent of $\bar{A}_i$, if $(b_i, \bar{b}_i)(c_i, \bar{b}_i) \sim^* (b_i, \bar{c}_i)(c_i, \bar{c}_i)$ for any $\bar{c}_i \in \bar{A}_i$—for all $b_i, c_i \in A_i$ such that $(b_i, \bar{b}_i) \succeq (c_i, \bar{b}_i)$ for some $\bar{b}_i \in \bar{A}_i$. That is, the preference difference between two alternative actions differing only on one attribute does not depend on the equal performances over other attributes. The attributes are mutually difference independent if all proper subsets of these attributes are difference independent of their complements. Furthermore, the attributes that are mutually preference independent are also mutually difference independent, if A_I is difference independent of $\bar{A}_I$ [63]. In the case where $n \geq 3$, mutual difference independence—in addition to some other conditions—ensure that $bc, de \in A^*$, then $bc \succeq^* de$ if and only if $\sum_{i=1}^{n} \lambda_i v_i(b_i) - \sum_{i=1}^{n} \lambda_i v_i(c_i) \geq \sum_{i=1}^{n} \lambda_i v_i(d_i) - \sum_{i=1}^{n} \lambda_i v_i(e_i)$ and $c \succeq d$ if and only if $\sum_{i=1}^{n} \lambda_i v_i(c_i) \geq \sum_{i=1}^{n} \lambda_i v_i(d_i)$, where v_i is a single attribute measurable value function over A_i scaled from 0 to 1, and $\sum_{i=1}^{n} \lambda_i = 1$. Furthermore, each v_i is unique up to a positive linear transformation.

4.1.2 UTA Methods

The UTA (UTilitès Additives) is a method based on MAUT that aims at constructing one or more additive multiattribute value functions from a given ranking on a set of alternative actions, A [102]. The method uses linear programming techniques to assess whether the rankings on A produced by the constructed functions are consistent with the given ranking [102, 182]. The aggregation model in UTA is in the form of an additive value or utility function, $u(g) = \sum_{i=1}^{n} w_i u_i(g_i)$ subject to normalization constraints $\sum_{i=1}^{n} w_i = 1$ and $u_i(g_{i^*}) = 0, u_i(g_i^*) = 1, \forall i = 1, 2, \ldots, n$, where $u_i, i =$

$1, 2, \ldots, n$ are non decreasing real valued functions, named marginal value or utility functions, which are normalized between 0 and 1, and w_i is the weight of u_i.

The UTA method also constructs an unweighted form of additive value function, $u(g) = \sum_{i=1}^{n} u_i(g_i)$, subject to normalization constraints $\sum_{i=1}^{n} u_i(g_i^*) = 1$ and $u_i(g_{i^*}) = 0, \forall i = 1, 2, \ldots, n$. Naturally, this model exists only under the condition of mutual preference independence—as explained in Section 4.1.1. Based on this additive model, each alternative action, $a \in A$, can be evaluated using

$$u'[g(a)] = \sum_{i=1}^{n} u_i[g_i(a)] + \sigma(a), \forall a \in A \tag{4.1}$$

where $\sigma(a)$ is the error relative to $u'[g(a)]$ [182].

Furthermore, linear interpolation can be used to estimate the corresponding marginal value functions in a piecewise linear form [102]. In this approach, the interval $[g_{i^*}, g_i^*]$ is cut into $(\alpha_i - 1)$ equal intervals for each criterion, and the end points are obtained through $g_i^j = g_{i^*} + \frac{j-1}{\alpha_i - 1}(g_i^* - g_{i^*})$, $\forall j = 1, 2, \ldots, \alpha_i$, where the marginal value of an alternative action is approximated by linear interpolation. Thus, for $g_i(a) \in [g_i^j - g_i^{j+1}]$,

$$u_i[g_i(a)] = u_i(g_i^j) + \frac{g_i(a) g_i^j}{g_i^{j+1} - g_i^j}[u_i(g_i^{j+1}) - u_i(g_i^j)] \tag{4.2}$$

The set of alternative actions, $A = \{a_1, a_2, \ldots, a_m\}$ is ranked in a weak order, where a_1 is the best action—hence, the head of the ranking—and a_m is the worst action—hence, the tail of the ranking [182]. Thus, if $\Delta(a_k, a_{k+1}) = u'[g(a_k)] - u'[g(a_{k+1})]$, then either $\Delta(a_k, a_{k+1}) \geq \delta$ if and only if $a_k \succ a_{k+1}$ or $\Delta(a_k, a_{k+1}) = 0$ if and only if $a_k \sim a_{k+1}$, where δ is a small positive number used to discriminate two successive equivalence classes of the weak order. The marginal values, $u_i(g_i)$, must satisfy $u_i(g_i^{j+1}) - u_i(g_i^j) \geq s_i$, $\forall j = 1, 2, \ldots, \alpha_i - 1, i = 1, 2, \ldots, n$, where $s_i \geq 0$ are indifference thresholds defined on each criterion g_i. Note that, it is not absolutely necessary to use thresholds in the UTA model, however, they are often useful in avoiding cases such as $u_i(g_i^{j+1}) = u_i(g_i^j)$ when $g_i^{j+1} \succ g_i^j$.

Finally, the marginal value functions are estimated through the following linear

program [102, 182]:

$[min]F = \sum_{a \in A_R} \sigma(a)$

subject to

$$\begin{array}{l} \Delta(a_k, a_{k+1}) \geq \delta, \text{ if } a_k \succ a_{k+1}, \forall k \\ \Delta(a_k, a_{k+1}) = 0, \text{ if } a_k \sim a_{k+1}, \forall k \\ u_i(g_i^{j+1}) - u_i(g_i^j) \geq 0, \forall i, j \\ \sum_{i=1}^{n} u_i(g_i^*) = 1 \\ u_i(g_{i^*}) = 0, u_i(g_i^j) \geq 0, \sigma(a) \geq 0, \forall a \in A, \forall i, j. \end{array} \tag{4.3}$$

where $\sigma(a)$ indicates the amount of total deviation.

The stability analysis of the results provided by the linear program 4.3 is considered to be a post-optimality analysis problem. If the optimum $F^* = 0$, the polyhedron of solutions for $u_i(g_i)$ is not empty and many value functions lead to a perfect representation of the given weak order [102]. The post-optimal solution space is defined by the polyhedron:

$$\begin{array}{l} F \leq F^* + k(F^*) \\ \text{all the constraints of the linear program 4.3} \end{array} \tag{4.4}$$

where $k(F^*)$ is a positive threshold which is a small portion of F^* [182].

The polyhedron given by 4.4 can be explored by several methods such as branch and bound methods [58], labyrinth techniques in graph theory [45], etc. In the original UTA Method the following linear program is used to explore the polyhedron [102]:

$$\begin{array}{ll} [min]u_i(g_i^*) \text{ and } [max]u_i(g_i^*) & \\ & \text{in } \forall i = 1, 2, \ldots, n. \\ & \text{Polyhedron 4.4} \end{array} \tag{4.5}$$

UTASTAR Method

The UTASTAR Method is an enhanced version of the UTA Method. The UTA Method is improved through introducing a double positive error function [181]. The UTA Method uses a single error, $\sigma(a)$, for each alternative action, $a \in A$, to be minimized. This error function is often not enough to minimize the error around the monotonous curve. A double positive error function is employed in order to eliminate

this issue, and thus each alternative action can now be evaluated using

$$u'[g(a)] = \sum_{i=1}^{n} u_i[g_i(a)] - \sigma^+(a) + \sigma^-(a), \forall a \in A, \tag{4.6}$$

where σ^+ and σ^- are the overestimation and the underestimation errors respectively.

The UTASTAR Method improves the UTA Method further by modifying the monotonicity conditions on the criteria through transformation of the variables:

$$z_{ij} = u_i(g_i^{j+1}) - u_i(g_i^j) \geq 0, \forall i = 1, 2, \ldots, n \text{ and } j = 1, 2, \ldots, \alpha_i - 1 \tag{4.7}$$

where, for $s_i = 0$, the monotonicity conditions can be replaced by the non-negative constraints for the variables z_{ij} [182].

Accordingly, the UTASTAR Method is carried on in four steps as follows. In step 1, the global value of reference actions, $u[g(a_k)], k = 1, 2, \ldots, m$, are expressed first in terms of marginal values, $u_i(g_i)$, and second in terms of the variables z_{ij} as given in Formula 4.7, through the following expressions:

$$\begin{array}{ll} u_i(g_i^1 = 0) & \forall i = 1, 2, \ldots, n \\ u_i(g_i^j) = \sum_{t=1}^{j-1} z_{it} & \forall i = 1, 2 \ldots, n \text{ and } j = 2, 3, \ldots, \alpha_i - 1 \end{array} \tag{4.8}$$

In step 2, two error functions, σ^+ and σ^-, on A are introduced by using the following expressions for each pair of consecutive actions in the given ranking:

$$\Delta(a_k, a_{k+1}) = u[g(a_k)] - \sigma^+(a_k) + \sigma^-(a_k) - u[g(a_{k+1})] + \sigma^+(a_{k+1}) - \sigma^-(a_{k+1}). \tag{4.9}$$

In step 3, the following linear program is solved:

$[min]z = \sum_{k=1}^{m} [\sigma^+(a_k) + \sigma^-(a_k)]$

subject to

$$\begin{array}{l} \Delta(a_k, a_{k+1}) \geq \delta, \text{ if } a_k \succ a_{k+1}, \forall k \\ \Delta(a_k, a_{k+1}) = 0, \text{ if } a_k \sim a_{k+1}, \forall k \\ \sum_{i=1}^{n} \sum_{j=1}^{\alpha_i - 1} z_{ij} = 1 \\ z_{ij} \geq 0, \sigma^+(a_k) \geq 0, \sigma^-(a_k) \geq 0, \forall i, j, k \end{array} \tag{4.10}$$

with σ being a small positive number. Finally, in step 4, existence of multiple or

near optimal solutions of the linear program is tested. In the case of non-uniqueness, step 4 involves finding the mean additive value function of the near optimal solutions which maximize the objective functions:

$$u_i(g_i^*) = \sum_{j=1}^{\alpha_i - 1} z_{ij}, \forall i = 1, 2, \ldots, n \tag{4.11}$$

on the polyhedron of the constraints of the linear program, bounded by the new constraint:

$$\sum_{k=1}^{m} [\sigma^+(\alpha_k) + \sigma^-(\alpha_k)] \leq y^* + \varepsilon, \tag{4.12}$$

where y^* is the optimal value of the linear program in Step 3 and ε is a very small positive number.

Variants of the UTA Method

There is a number of other variants and extensions of the UTA method incorporating different forms of global preference or optimality criteria. An extension to the UTA Method is based on the usage of alternative optimality criteria where subjective preferences obtained by pairwise judgements are used [102, 59]. Another extension the Meta-UTA Techniques which aim at improving the value function with respect to near optimality analysis or to its exploitation for decision support [59]. Also, Stochastic UTA Method is built for multiple criteria decision analysis under uncertainty, in which the aggregation model to infer from a reference ranking is an additive utility function [180, 179]. Last but not least, UTA-type Sorting Methods are built as an extension to the UTA method for the cases of discriminant analysis model where the main aim is to infer u from assignment examples in the context of problem statement [102].

4.1.3 Analytic Hierarchy Process

Analytic Hierarchy Process (AHP) theory is mainly concerned with the prioritization of alternative actions, which focuses on extracting priorities from the relative judgements of a decision maker, and expressing them numerically on an absolute scale [173, 175]. The AHP deals with both tangible and intangible criteria—with an explicit focus on the latter—by taking into account the judgements of domain ex-

perts and collected statistics needed to make a decision. The judgements are generally made in a qualitative fashion, and are expressed numerically [173]. Briefly, AHP uses reciprocal pairwise comparisons rather than merely assigning scores to each alternative action. Priorities of measurable quantities and non-measurable qualities are combined using a ratio scale, in order to facilitate addition and multiplication of alternative actions under certain independence conditions among the elements of a decision making problem [174]. There is also a link between the ratio judgements used in AHP and measurable value functions [64]. In this context, AHP is mainly based on four axioms [173]: (1) reciprocal judgments, (2) homogeneous elements, (3) hierarchical or feedback dependent structures, and (4) rank order expectations. It combines multidimensional scales of measurement into single scale of priorities [174], where decisions are expressed by a single number for the best alternative action or an order that gives a proportionate ranking of the alternative actions.

In order to investigate the mathematical structure of AHP, let us consider a case where a decision maker is to express judgements on a set of alternative actions, $A = \{a_1, a_2, \ldots, a_n\}$. Suppose that the decision maker starts by attaching a positive real number, v_i, to each alternative action, a_i, where v_i expresses the importance of alternative a_i. The main assumption here is that once each alternative action is assigned a numerical value denoting its importance, the relative importance of two alternative actions, a_i and a_j, can be expressed as a ratio of importance. That is, the decision maker can conclude that a_i is more important than a_j by a factor of v_i/v_j. If $(v_i/v_j) < 1$, then a_j is more important than a_i by a factor of v_j/v_i. Naturally, $(v_i/v_i) = 1$ for any alternative action a_i. Using these relative measurements of importance, the decision maker can now form an $n \times n$ matrix $M = (m_{ij})$, where $m_{ij} = v_i/v_j$, which has the following properties [175, 17]:

1. $m_{ii} = 1$, $m_{ij} > 0$, and $m_{ji} = m_{ij}^{-1}$ for all i, j.

2. $m_{ij}m_{jk} = m_{ik}$ for all i, j, k.

3. The matrix M has rank 1, with each column proportional to the vector $C = (v_1, v_2, \ldots, v_n)^T$ and each row proportional to the vector $R = (v_1^{-1}, v_2^{-1}, \ldots, v_n^{-1})$.

4. 0 is an eigenvalue of M with multiplicity $n - 1$, and the trace of M is n; it follows from this that there is a remaining eigenvalue which is simple and equal to n;

5. C is a column eigenvector and R is a row eigenvector of M corresponding to the eigenvalue n. Thus, in this special case, our relative importance measurements of the options a_i appears in the form of an eigenvector corresponding to the largest positive eigenvalue of a matrix with positive entries.

In the case where the entries of the matrix, M, is perturbed, its eigenvectors and eigenvalues will also be perturbed. If this perturbation is small, then there is an eigenvalue close to n whose column eigenvector can be regarded as a good approximation to the relative importance judgements of the alternative actions. Thus, the ranking problem can be viewed as follows [17]. For the n alternative actions there is an ideal, yet unknown, ranking—in the form of a vector—of importances. In order to find this ranking, each pair of alternative actions, (a_i, a_j), is assigned a positive real number, m_{ij}, which measures the relative importance of a_i and a_j. The only condition on the assignment of m_{ij} is the first property. A matrix M of entries m_{ij} that satisfies this first property is called a reciprocal matrix. If the entries, m_{ij}, also satisfy the second property, the matrix M is consistent. Then, the k^{th} column is equal to a_{jk} times the j^{th} column, so the rank of M is 1, and M satisfies the fourth property [17, 173]. Furthermore, if $(c_1, c_2, \ldots, c_n)^T$ is any column eigenvector and $(r_1, r_2, \ldots, r_n)$ any row eigenvector with eigenvalue n, then $r_i/r_j = c_i/c_j = m_{ij}$. That is, the eigenvectors can be used to weight the alternative actions that is consistent with the pairwise judgements [17, 173].

Note that, AHP uses a fundamental scale of absolute numbers to assign the decision maker's judgements to m_{ij}, where each judgment represents the dominance of an element in the column on the left over an element in the row on top. Judgments reflect on two views on the elements in comparison: 1) representing which of the two elements is more important with respect to a given criterion—or a higher level criterion, and 2) representing the strength of preference using the fundamental scale, where the strength of preference is represented using numbers from 1 to 9. The fundamental scale represents the strength of preference from indifference to extreme preference, incrementally. The 1-9 scale is derived from stimulus response ratios [173, 175, 174].

Naturally, a multiple criteria decision making problem involves taking into account several criteria during the ranking of alternative actions or selecting the most suitable alternative action. The different criteria are taken into account by first attaching weights to each criteria, $c_1, c_2, \ldots, c_s$ using the same procedure for determining the relative importance of alternative actions. Let the column vector $Z = (z_1, z_2, \ldots, z_s)$ hold these entries, each of which are normalized so that they add up to 1. Now, the

question is to evaluate the alternative actions with respect to each criteria separately. For the j^{th} criterion, suppose the weights are the entries of the column vector $Y = (y_{1j}, y_{2j}, \ldots, y_{nj})^T$—entries of which are also normalized. An overall determination of the importance of alternative actions is reached by taking a weighted average—using Z—of the weights of the criteria. Then, an overall importance of the alternative actions, $(w_1, w_2, \ldots, w_n)^T$, can be found with

$$w_i = y_{i1}z_1 + y_{i2}z_2 + \cdots + y_{in}z_n \quad (1 \leq i \leq n). \tag{4.13}$$

where $w_1 + w_2 + \cdots + w_n = 1$. This procedure can be generalized to more complex decision processes where there is a hierarchy of multiple levels of criteria. The alternative actions are at the bottom of the hierarchy, the top consists of the most general criteria and in between are the sub-criteria that are considered to be overridden by the more general criteria. For each pair of adjacent levels, an $r \times s$ matrix is formed, where the s columns represent the weights of the r lower level criteria with respect to the s higher level criteria. Accordingly, if $Y_1, Y_2, \ldots, Y_m$ are the matrices, with normalized columns, for each pair of adjacent levels from lowest to the highest respectively, and Z is the weighting of the highest level criteria. The overall importance of the alternative actions is given by a matrix product $Y_1Y_2 \ldots Y_mZ$ [17].

In a number of cases, the pairwise comparisons that form the matrix M can lead to inconsistent judgements. As an example, let us consider the case where there are three pairs of alternative actions (a_1, a_2), (a_2, a_3), (a_1, a_3) and pairwise comparisons 3, 5 and 1/2 respectively. Although this is an extreme case in which transitivity is violated, some inconsistency is bound to emerge in most of the cases. Moreover, AHP allows such inconsistencies up to an extent [173], and provides the mathematical ways to handle such situations. In the cases of inconsistency, an inconsistent matrix M is subject to Perron's Theorem [176], which states that if M is a matrix with strictly positive entries, then M has a simple positive eigenvalue, λ_{max} which is not exceeded in absolute value by any of its complex eigenvalues; every row eigenvector or column eigenvector corresponding to λ_{max} is a constant multiple of an eigenvector with strictly positive entries.

As an example, let us review a case where M is a 3×3 matrix. Then, the Perron eigenvalue $1 + y + y^{-1}$ is always at least 3, with equality when $y = 1$. Also, the largest eigenvalue of M exceeds 3 if and only if the matrix is not consistent, and equals 3 otherwise [17]. In a more general case, suppose that $M = (m_{ij})$ is an $n \times n$ matrix

n:	1	2	3	4	5	6	7	8	9	10
μ:	0.00	0.00	0.58	0.90	1.12	1.24	1.32	1.41	1.45	1.49

Table 4.1: Random Index

with positive entries where $m_{ij} = 1/m_{ji}$ holds. Then, if λ_{max} is M's eigenvalue of maximum absolute value, $\lambda_{max} \geq n$, and M is consistent if and only if $\lambda_{max} = n$. Thus, the difference $\lambda_{max} - n$ can be used as means to measure the consistency of a given matrix M [173, 17]. Since the sum of all the eigenvalues of M is n—the trace of M, $\lambda_{max} - n$ is the negative sum of the remaining eigenvalues of M. The average of these eigenvalues is $-\mu$,

$$\mu = \frac{\lambda - n}{n - 1}. \tag{4.14}$$

where μ is referred to as the consistency index of M. In practical cases, μ is deemed to be satisfactory if it is no more than about 10% of the mean consistency index for a sample of 500 randomly generated matrices where $a_{ij} = a_{ji}^{-1}$ holds. Each matrix is formed using entries drawn from the set $\{1/9, 1/8, \ldots, 1/2, 1, 2, \ldots, 9\}$. Table 4.1 lists the order of such matrices and the random mean consistency indexes. If μ is too large, then the process is likely to be defective and the judgements made should be reviewed [173].

4.1.4 Other Single Synthesizing Criterion Methods

There is a number of other methods that are based on the Single Synthesizing Criterion approach. Some of the well-known ones among these methods can be summarized as follows. Technique for Order Preference by Similarity to Ideal Solution (TOPSIS) focuses on finding an alternative action that has the nearest and farthest profile to the ideal solution and the negative ideal solution, respectively [98, 92]. Simple Multi-Attribute Rating Technique (SMART) is a simplified method mainly based on the multiattribute utility theory [66, 92]. It uses weighted linear averages, and provides a very close approximation to utility functions. SMART is enhanced in later works, some of which can be exemplified as SMARTS and SMARTER [67]. EVAMIX is another method that calculates a dominance index for ordinal evaluation of the alternative actions and a dominance index for the cardinal evaluation of the alternative actions. The two indexes are combined to reach a measure of dominance between each pair of alternative actions [92]. Also, there are other approaches based on fuzzy

subset theory. For instance, fuzzy weighted sum technique uses α-level sets to derive fuzzy utilities based on a simple additive weighted model [16]. Finally, Measuring Attractiveness by a Categorical Based Evaluation Technique (MACBETH) is a method that requires only qualitative judgements about differences of value to quantify the relative attractiveness of alternative actions [65].

4.2 Outranking Methods

The operational approach based on outranking has its roots in social choice theory [92]. The methods based on this approach deal with a decision problem by successively comparing each of the alternative actions with each other. That is, unlike the Single Synthesizing Criterion methods, outranking methods do not consider alternative actions in isolation. To be more specific, the aggregation problem is not addressed in terms of defining a complete pre-order an a set of alternative actions. Instead, the approach based on outranking uses pairwise comparisons of alternative actions in order to design a synthesizing preference relational system, during which the performances of each alternative action on each criteria is compared to the performances of other alternative actions on the same criteria [169].

However, there are certain difficulties when this type of approach is considered. First of all, the pairwise comparisons can cause some intransitive preference behaviour to appear. Furthermore, incomparability may be the best conclusion when certain pairs of alternative actions are compared. As a result, outranking methods are not to be immediately used for making a decision, and require completion of an additional step called the exploitation procedure [169].

This operational approach has given rise to several methods. In this section, we focus on some of the most well-known ones among them. Accordingly, Section 4.2.1 focuses on the earliest of these methods, called ELECTRE [77]. In Section 4.2.2, we explain another of these methods, namely PROMETHEE [36]. Finally, Section 4.2.3 provides a brief list of other methods that under under the operational approach of outranking methods.

4.2.1 ELECTRE Methods

ELECTRE methods model preference behaviour of a decision maker using a binary outranking relation, S, that stands for the notion "at least as good as" [77, 168]. When

the two alternative actions, $a, b \in A$, are considered along with the binary outranking relation, S, four specific situations are bound to occur: 1) aSb and not bSa, or in a more concise form, aPb, which denotes the case where a is strictly preferred to b, 2) bSa and not aSb, or bPa, which denotes the case where b is strictly preferred to a, 3) aSb and bSa, or in a more concise form, aIb, which denotes the case where a is indifferent to b, and 4) not aSb and not bSa, or in a more concise form, aRb, which denotes the case where a is incomparable to b. In ELECTRE methods, one or more outranking relations—crisp, fuzzy or embedded—are constructed. The outranking relations used to model the preference behaviour of a decision maker introduce a new preference relation that is not present in the MAUT methods, called incomparability, R [77].

The outranking relations are constructed based on the two major conditions of *concordance* and *non-discordance* [170, 171]. The concordance condition states that an outranking relation, aSb, is valid only if a sufficient majority of criteria is in favour of this assertion. Whereas, the non-discordance condition states that an outranking relation, aSb, is valid only if none of the criteria in the minority opposes too strongly to this assertion. Accordingly, any assertion, aSb, that satisfies these two conditions is deemed valid. Also, note that an outranking relation is not necessarily transitive, which is due to the Condorcet effect and incomparabilities. The intransitivities make it necessary to go through a secondary procedure once the outranking relations are constructed, called the exploitation procedure, in order to extract results that fit a given problematic—choice problematic, ranking problematic, or sorting problematic [77].

In ELECTRE methods, the relative importance attached to each criteria is defined by two distinct sets of parameters [170, 171]: 1) the importance of coefficients, and 2) the veto thresholds. The importance coefficients are the intrinsic weights of each criterion. That is, for a given criterion, its weight w_j, reflects is voting power when it contributes to the majority which is in favour of an outranking. The weights do not depend on the ranges or the encoding scales, and cannot be interpreted as substitution rates as in MAUT based MCAPs [169]. On the other hand, the veto thresholds represent the power assigned to a given criterion to be against the assertion, "a outranks b", when the difference between the evaluation of a and evaluation of b on criterion $g(\cdot)$—that is the difference between $g(a)$ and $g(b)$—is not greater than the threshold assigned to that criterion [170, 171]. The thresholds can be variable or constant along a scale [169]. Moreover, ELECTRE methods use discrimination—or

indifference and preference—thresholds in order to take into account the imperfect evaluation of actions. Discrimination thresholds lead to a pseudo-criterion model, and are used for modeling situations in which the difference between evaluations associated with two alternative actions on given criterion may: 1) justify the preference in favour of one of the two actions—preference threshold, p_j, 2) be compatible with indifference between the two actions—indifference thresholds, q_j, or 3) be interpreted as hesitation in concluding for a preference or an indifference between the two alternative actions. Like the veto thresholds, the discrimination thresholds can be variable or constant along a scale. There are various techniques to assign values to such thresholds. Some of these techniques are derived directly from the definition of a threshold, while others use an additional concept called a dispersion threshold. Dispersion thresholds allow the decision analyst to take into account notions such as probable, optimistic, and pessimistic evaluations [169].

From this point on, we investigate the different ELECTRE methods based on the problematics they address. In this context, we first focus on the choice problematic and provide an outline of ELECTRE I, ELECTRE Iv, and ELECTRE Is methods. Secondly, we focus on the methods, such as ELECTRE II, ELECTRE III, and ELECTRE IV, that address the ranking problematic where a set of alternative actions are ranked from the best to the worst with a possibility of equalities. Finally, we conclude with an overview of ELECTRE TRI within the context of the sorting problematic where each alternative action is assigned to a set of pre-determined categories. The reaon behind this organization is to facilitate a better understanding of why and how the notions such as veto thresholds and pseudo-criteria are introduced in ELECTRE methods.

ELECTRE I

ELECTRE I is a very simple method and is applied only in the cases where all the criteria are coded in numerical scales with identical ranges [77]. In such cases, "a outranks b", aSb, can be asserted when the two conditions of concordance and non-discordance hold.

In this context, the assertion aSb must be sufficiently supported by the strength of a concordant colation which is the sum of the weights of the criteria that form the coalition. The strength of concordant coalition is defined by the concordance index, $c(aSb) = \sum_{\{j:g_j(a)\geq g_j(b)\}} w_j$, where J is the set of criteria indices, $\sum_{j\in J} w_j = 1$,

and $\{j : g_j(a) \geq g_j(b)\}$ is the set of indices for all the criteria belonging to the concordant coalition with the outranking relation aSb [169]. The concordance index must be at least equal to a pre-determined concordance level or threshold s, $c(aSb) \geq s$ [170, 171]. Furthermore, there should be no discordance against the assertion "a is at least as good as b". In order to measure that, the discordance index, $d(aSb) = max_{\{j:g_j(a)<g_j(b)\}}\{g_j(b) - g_j(a)\}$, is defined, which measures the power of the discordant coalition. If $d(aSb)$ exceeds a certain level or threshold v, then the assertion is deemed to be not valid. In the case where $d(aSb) \leq v$, the discordant coalition does not represent a considerable power over the assertion at hand. Note that, both the concordance and discordance indices need to be evaluated for every pair of alternative actions, $(a, b) \in A^2$, where a and b are unique alternative actions.

This procedure leads to preference-indifference framework with a chance for incomparability to occur. However, such a framework does not necessarily indicate which alternative action is the best choice [170, 171, 77]. Therefore, the process continues with a secondary procedure which consists of exploiting the outranking relation produced by the first procedure. The main purpose of the exploitation procedure is to identify a small subset of A where a compromise alternative action can be found. In order to identify such a subset, let us consider building a graph, $G = (V, U)$, using the binary realtions on the set of alternative actions, A, where V is the set of vertices and U the set of arcs. Accordingly, each alternative action $a \in A$ is associated with a vertex $i \in V$, and for each pair of alternative actions, $(a, b) \in A^2$, there exists an arc (i, l) either if aPb or aIb. Alternative action a outranks alternative action b if and only if the arc (i, l) exists. If there is no arc between vertices i and l, then a and b are incomparable. In the case where there are two reversal arcs, then a and b are indifferent. Based on such a graph, the graph kernel concept can be used to identify such a small subset $\bar{A}$ [77]. If a graph contains no direct cycles, then there exists a unique kernel, otherwise, the graph contains either no kernels or several kernels. If the graph G contains direct cycles, a preprocessing step is necessary where maximal direct cyles are reduced to singleton elements. Thus, forming a partition $\bar{A}$ on A. Each class on $\bar{A} = \{\bar{A}_1, \bar{A}_2, \dots\}$ consists of a set of equivalent actions.

$$\bar{A}_p \succ \bar{A}_q \Leftrightarrow \exists a \in \bar{A}_p \text{ and } \exists b \in \bar{A}_q \text{ such that } aSb \text{ for } \bar{A}p \neq \bar{A}_q \tag{4.15}$$

Note that, a new preference relation, $\succ$, is defined on $\bar{A}$ [77].

ELECTRE Iv

ELECTRE Iv extends ELECTRE I with the veto threshold which is a very useful adddition that overcomes the difficulties related to homogeneity of scales [134]. Regardless of the types of scales, ELECTRE Iv is capable of selecting the best compromise alternative action, or identifying a subset of alternative actions that can be further analyzed [77].

IN ELECTRE Iv, the veto threshold ,v_j, can be assigned to certain criteria, $g_j \in F$. Although the concepts of a veto threshold and the discordance level—as in ELECTRE I—are in a sense related, they are essentially different since the discordance level is related to the scale of a criterion g_j in absolute terms for an alternative action $a \in A$, whereas a veto threshold is related to the preference difference between the evaluation of two alternative actions, $g_j(a)$ and $g_j(b)$.

In terms of general structure, the only difference between ELECTRE I and ELECTRE Iv is the requirement for a *no veto* condition to hold, instead of a non-discordance condition. The no veto condition is expressed as, $g_j(a) + v_j(g_j(a)) \geq g_j(b)$, $\forall j \in J$. The assertion "a outranks b" is validated only if it is not vetoed by any criterion in the minority of criteria that are against this assertion.

ELECTRE IS

The main contribution of ELECTRE IS is the use of pseudo-criteria instead of true-criteria. ELECTRE IS is an extenion to ELECTRE Iv that aims at taking into account two objectives [77]: 1) the use of possible no nil indifference and preference thresholds for certain criteria belonging to F, and 2) a reinforcement of the veto effect when the importance of the concordant coalition decreases. Accordingly, both the concordance condition and the no veto condition are changed in ELECTRE IS.

The concordance condition is extended by defining two indices sets J^S and J^Q that concerns the coalition of criteria in which aSb and the coalition of criteria in which bQa, respectively. These two indices sets are defined as

$$\begin{aligned} J^S &= \{j \in J : g_j(a) + q_j(g_j(a)) \geq g_j(b)\} \\ J^Q &= \{j \in J : g_j(a) + q_j(g_j(a)) < g_j(a) \leq g_j(b) + p_j(g_j(b))\} \end{aligned} \tag{4.16}$$

based on which the concordance condition is now

$$c(aSb) = \sum_{j \in J^S} w_j + \sum_{j \in J^Q} \varphi_j w_j \geq s \tag{4.17}$$

where

$$\varphi_j = \frac{g_j(a) + p_j(g_j(a)) - g_j(b)}{p_j(g_j(a)) - q_j(g_j(a))} \tag{4.18}$$

Note that the coefficient φ_j decreases linearly from 1 to 0, when g_j describes the range $[g_j(a) + q_j(g_j(a)), g_j(a) + p_j(g_j(a))]$ [77].

Furthermore, ELECTRE IS defines the no veto condition as follows

$$g_j(a) + v_j(g_j(a)) \geq g_j(b) + q_j(g_j(b))\eta_j \tag{4.19}$$

where

$$\eta_j = \frac{1 - c(aSb) - w_j}{1 - s - w_j} \tag{4.20}$$

In ELECTRE I all the actions which form a cycle in graph G are considered different, which is often criticized [77]. ELECTRE IS is designed to mitigate this inconvenience [77]. Accordingly, in the exploitation procedure, alternative actions that form a cycle are not considered different. Instead, the degree of robustness of the asertion "a outranks b" is taken into account, which is a reinforcement of the veto effect that makes it possible to build true equivalence classes so that an acyclic graph can be built. As stated earlier, there is always a single kernel in such cases [170, 171, 77].

ELECTRE II

ELECTRE II is the first of the ELECTRE methods that was based on constructing sequences of embedded outranking relations. Since ELECTRE II is a true-criteria based method, its construction procedure is very close to that of ELECTRE Iv. Accordingly, the no veto condition in ELECTRE II is identical to the no veto condition in ELECTRE Iv. However, the concordance condition is modified in order to express the notion of embedded outranking relations. In ELECTRE II, there are two embedded outranking relations: 1) a strong outranking relation, and 2) a weak outranking relation. These two outranking relations are constructed based on the definition of two concordance levels, s^1 and s^2. Thus, the modified concordance condition on the assertion "a

outranks b" is defined as $c(aSb) \geq s^r$ and $c(aSb) \geq c(bSa)$, for $r = 1, 2$ [170, 171, 77]. After the outranking relations are constructed, ELECTRE II uses a four-step exploitation procedure which is carried out as follows [170, 77].

The first step consists of partitioning the alternative set A. Let us consider the relation S^1 over A. As in ELECTRE I, this relation may define one or more cycles on A. Then, a partition $\bar{A}$ on A is obtained by grouping each alternative action that belong to each maximal cycle into a single class. If a class of $\bar{A}$ is not a singleton, the alternative actions in that class are considered to be indifferent. Identical to the notion, $\succ$, used in ELECTRE I, the comparison between the elements of $\bar{A}$ is done using the preference relation $\succ^1$. The second step builds a complete pre-order Z_1 on $\bar{A}$ as follows. Once $\bar{A}$ is obtained, a subset B^1 is found in the classes of $\bar{A}$ following the rule "no other is preferred to them" using the relation $\succ^1$, and consequently removed from $\bar{A}$. The procedure iterates the same way until all the subsets, $B^1, B^2, \ldots$, on $\bar{A}$ are found and removed. Thus, on the basis of S^1, a rough version of the complete pre-order, Z_1, is defined where the head is all classes of B^1 followed by all classes of B^2, and so forth. The complete pre-order is further refined on the basis of S^2. This refinement is obtained by using S^2 in order to define over subset B^p a complete pre-order that takes place between B^{p-1} and B^{p+1}. The third step involves determining a complete pre-order Z_2 on $\bar{A}$. The third step is similar to the second step with only the following two modifications: 1) Obtain the partitions $B^{1'}, B'^2, \ldots$ following the rule "they are not preferred to any other" instead of "no other is preferred to them", and 2) Obtain the rough version of the complete pre-order Z_2 by queuing all classes of $B^{1'}$ followed by all classes of $B^{2'}$, and so forth. Finally, the fourth step consists of defining the partial pre-order Z which is an intersection of Z_1 and Z_2, $Z = Z_1 \cap Z_2$, and is defined as $aZb \Leftrightarrow aZ_1band aZ_2b$.

ELECTRE III

ELECTRE III is an improvement to ELECTRE II, which addresses the consideration of inaccurate, imprecise, uncertain or ill-determined information. Accordingly, the main contribution of ELECTRE III is the usage of pseudo-criteria instead of true-criteria in the context of ranking problematic [170, 77]. ELECTRE III defines the outranking relation as a fuzzy relation, construction of which requires the definition of a credibility index. The credibility index, $\rho(aSb)$, characterizes the credibility of the assertion "a outranks b", aSb. The credibility index $\rho(aSb)$ is defined using

both the concordance index, $c(aSb)$,—which defined the same way as it is defined in ELECTRE IS—and the discordance index, $d_j(aSb)$, for each criterion $g_j \in F$ [77].

The discordance index has the maximum possible value when criterion g_j vetos the assertion at hand, and has the minimum value when the criterion g_j is not discordant with that assertion. The value of the discordance index for the intermediate zone between the maximum and minimum values are assigned proportionally to the difference $g_j(b) - g_j(a)$. Then the discordance index can be represented as

$$d_j(aSb) = \begin{cases} 1 & \text{if } g_j(b) > g_j(a) + v_j(g_j(a)) \\ 0 & \text{if } g_j(b) \leq g_j(a) + p_j(g_j(a)) \\ \frac{g_j(b) - g_j(a) - p_j(g_j(a))}{v_j(g_j(a)) - p_j(g_j(a))} & \text{if otherwise} \end{cases} \tag{4.21}$$

The credibility index is defined based on the definitions of the concordance index and the discordance index as

$$\rho(aSb) = c(aSb) \prod_{\{j \in J : d_j(aSb) > c(aSb)\}} \frac{1 - d_j(aSb)}{1 - c(aSb)} \tag{4.22}$$

where $\rho(aSb) = 0$, if $d_j(aSb) = 1$ since $c(aSb) < 1$.

This definition of the credibility index is based on three main reasons [77]: 1) If there is no discordant criterion, the credibility of the outranking relation is equal to the comprehensive concordance index, 2) If a discordant criterion vetos the assertion at hand, that assertion is not credible at all, and 3) In the cases where the comprehensive concordance index is strictly lower than the discordance index on the discordant criterion, the credibility index becomes lower than the comprehensive concordance index, because of the opposition effect on this criterion. Thus, the credibility index corresponds to the concordance index weakened by possible veto effect. The explitation procedure consists of deriving two complete pre-oreders Z_1 and Z_2 from the constructed fuzzy relations. The process is very similar to the exploitation procedure of ELECTRE II, where the partial pre-order Z is defined as the intersection of Z_1 and Z_2. A complete pre-order is finally suggested taking into account the partial pre-orders and some additional considerations.

ELECTRE IV

ELECTRE IV is a also procedure based on the construction of a set of embedded outranking relations, which extends ELECTRE III. In ELECTRE IV, there are five different outranking relations, $S^1, \ldots, S^5$. The S^{r+1} relation ($r = 1, 2, 3, 4$) accepts an outranking in less credible circumstances than the relation S^r [170]. That is, ρ_r is assigned as the credibility value $\rho(aSb)$ for the assertion aSb. The values assigned to the credibility index value must follow $\rho_r > \rho_{r+1}$, where moving from credibility value ρ_r to ρ_{r+1} must be perceived as a considerable loss [77]. ELECTRE IV exploitation procedure is identical to that of ELECTRE III.

ELECTRE TRI

ELECTRE TRI is designed to address the sortgin problematic, that is, assignment of alternative actions into certain categories. In ELECTRE TRI, the categories are ordered between a two categories that are deemed to be the worst, (C_1) and the best, (C_k) [77]. Each category is chracterized by a lower profile and an upper profile. Let $C = \{C_1, \ldots, C_h, \ldots, C_k\}$ denote a set of such categories. The assignment of an alternative action a to a certain category C_h is done by comparing a with the lower profile of C_{h+1} and upper profile of C_h, where b_h is the upper profile of category C_h and the lower profile of category C_{h+1}, for all $h = 1, 2, \ldots, k$. The comparisons depend on the credibility of assertions aSb_h and b_hSa for a given b_h, where the credibility index is defined in a way identical to that of ELECTRE III.

Once the credibility index is determined, an $\alpha - cutting$ level of the fuzzy relation is introduced in order to obtain a crisp outranking relation [77]. This level is defined as the smallest value of the credibility index that is compatible with the assertion aSb_h. Now, let $\succ$, I and R denote preference, indifference and incomparability relations respectively. The alternative action a and the profile b_h can be realted as: 1) aIb_h if and only if aSb_h and b_hSa, 2) $a \succ b_h$ if and only if aSb_h and not b_hSa, 3) $b_h \succ a$ if and only if not aSb_h and b_hSa, and 4) aRb_h if and only if not aSb_h and not b_hSa.

The exploitation procedure aims at proposing an assignment for the alternative actions, where each assignment is based on the conjuntive—pessimistic—logic or the disjunctive—optimistic—logic. In the conjunctive logic, an alternative action is assigned to a category when its evaluation on each criterion is at least as good as the lower limit which has been defined on the criterion to be in this category. The action is hence assigned the highest category fulfilling this condition [77]. In the disjunc-

tive logic, an action is assigned to a category if there exist a criterion on which it has an evaluation at least as good as the lower limit defined on the criterion to be in this category. The action is hence assigned to the highest category fulfilling this condition [77]. If there is no incompatibility in the comparison of an action a to the limits of categories, a is assigned to the same category by the optimistic and the pessimistic procedures. If a is assigned to different categories by the optimistic and pessimistic procedures, a is incomparable to all intermediate limits within the highest and lowest assignment categories. ELECTRE TRI generalizes the conjunctive logic and the disjunctive logic as: 1) The condition "on each criterion" is replaced by "on a sufficient majority of criteria and in the absence of veto" in the conjunctive rule, and 2) The condition "on at least one criterion" is replaced by "on a sufficient minority of criteria and in the absence of veto" in the disjunctive rule.

4.2.2 PROMETHEE Methods

The PROMETHEE methods are form another family of procedures that treat multiple criteria decision analysis problems based on the outranking approach. The information that PROMETHEE needs in order to assist a decision maker in a decision problem are rather clear and simple, and consists of [33, 36]: 1) Information between criteria, and 2) Information within each criterion.

Information between criteria can be identified as the representation of relative importance of each criterion's influence during the process of decision making. The relative importance of criteria are represented by a set, $w_j, j = \{1, 2, \ldots, k\}$, of weights each of which is a non-negative number independent from the scales of evaluation that are associated with the criteria [34, 35]. Naturally, criteria with higher weights have a stronger influnce on the decision to be made. These weights are normalized so that $\sum_{j=1}^{k} w_j = 1$.

Information with each criterion is gathered in terms of pairwise comparisons of the alternative actions based on each criterion in isolation. On each criterion, the deviation between the evaluations of two alternative actions is considered where small deviations represent either weaker preferences or no preference at all depending on the perception of a decision maker. Accordingly, the strength of preference grows proportionally with the growth in deviations. In PROMETHEE, the strength of preference is represented using real numbers between 0 and 1, gathered through a preference function, $P_j(a, b) = F_j[d_j(a_b)]$, $\forall a, b \in A$, assigned to a particular criterion

where $d_j(a,b) = g_j(a) - g_j(b)$ and $0 \leq P_j(a,b) \leq 1$ [39, 34, 35]. In the case where the decision maker is looking for maximum evaluations—that is, higher values for the evaluation of an alternative action over a criterion is deemed more preferable, the preference function, P_j, gives the preference of alternative action a over alternative action b for observed deviations between the evaluations of these two alternative actions on criterion $g_j(\cdot)$. Negative deviations imply that the preference is equal to 0, $P_j(a,b) > 0 \Rightarrow P_j(b,a) = 0$. In the case where the decision maker is looking for minimum evaluations—that is, lower values for the evaluation of an alternative action over a criterion is deemed more preferable, the preference function P_j is reversed so that $P_j(a,b) = F_j[-d_j(a,b)]$ [36].

In PROMETHEE, the pair $\{g_j(\cdot), P_j(a,b)\}$ attached to criterion $g_j(\cdot)$ is called the generalized criterion, which should be identified for each criterion. In order to facilitate ease of identification of the generalized criteria, PROMETHEE proposes six types of particular preference functions [36]:

Type 1: Usual Criterion

$$P(d) = \begin{cases} 0 & \text{,if } d \leq 0 \\ 1 & \text{,if } d > 0 \end{cases}$$

Type 2: U-Shape Criterion

$$P(d) = \begin{cases} 0 & \text{,if } d \leq q \\ 1 & \text{,if } d > q \end{cases}$$

Type 3: V-Shape Criterion

$$P(d) = \begin{cases} 0 & \text{,if } d \leq 0 \\ \frac{d}{p} & \text{,if } 0 \leq d \leq p \\ 1 & \text{,if } d > p \end{cases}$$

Type 4: Level Criterion

$$P(d) = \begin{cases} 0 & \text{,if } d \leq q \\ \frac{1}{2} & \text{,if } q < d \leq p \\ 1 & \text{,if } d > p \end{cases}$$

Type 5: V-Shape with Indifference Criterion

$$P(d) = \begin{cases} 0 & \text{,if } d \leq q \\ \frac{d-q}{p-q} & \text{,if } q < d \leq p \\ 1 & \text{,if } d > p \end{cases}$$

Type 6: Gaussian Criterion

$$P(d) = \begin{cases} 0 & \text{,if } d \leq 0 \\ 1 - e^{-\frac{d^2}{2s^2}} & \text{,if } d > 0 \end{cases}$$

where each type identifies 0, 1 or 2 parameters, the significance of which can be listed as: 1) q is a threshold of indifference, 2) p is a threshold of strict preference, and 3) s is an intermediate value between q and p. The indifference threshold q represents the largest deviation that is considered negligible. The preference threshold p represents the smallest deviations that is considered sufficient for full preference. In the Gaussian Criterion, s defines the inflection point of the preference function between the indifference threshold q and the strict preference threshold p. In the context of these six types of preference functions, the identification of a generalized criterion is limited to the selection of these thresholds. It is important to note here that altghough these six types of preference functions have been sufficient in the application of PROMETHEE to many real-world decision problems, other generalized criteria can also be used.

Once the information between criteria, w_j, and the information within each criterion, $\{g_j(\cdot), P_j(a,b)\}$, is obtained—in addtion to the evaluation of each alternative action on each criterion, $\{g_j(\cdot)\}$— the PROMETHEE procedure can be applied. As in ELECTRE methods, the PROMETHEE procedure is also based on pairwise comparisons. The pairwise comparisons are then used in notions that are key to the PROMETHEE procedure, such as *aggregated preference indices* and *outranking flows* [33].

The definition of aggregated preference indices is as follows. Let us consider the two alternative actions $a, b \in A$, and let

$$\pi(a,b) = \sum_{j=1}^{k} P_j(a,b) w_j$$
$$\pi(b,a) = \sum_{j=1}^{k} P_j(b,a) w_j$$

where the aggregated preference indices $\pi(a,b)$ and $\pi(b,a)$ idicate with which degree a is preferred to b and with which degree b is preferred to a over all criteria, respectively [39, 34]. In a number of cases, there are criteria on which a is preferred to b, whereas there are other criteria on which b is preferred to a. Consequently, $\pi(a,b)$ and $\pi(b,a)$ are usually positive. The following properties hold for all $(a,b) \in A^2$: 1) $\pi(a,a) = 0$, 2) $0 \leq \pi(a,b) \leq 1$, 3) $0 \leq \pi(b,a) \leq 1$, and 4) $0 \leq \pi(a,b) + \pi(b,a) \leq 1$. Accordingly, $\pi(a,b) \sim 0$ implies a weak global preference of a over b and $\pi(a,b) \sim 1$ implies a strong global preference of a over b. Once the aggregated preference indices are computed for each pair of alternative actions in A, a complete valued outranking graph with two arcs between each pair of alternatives is obtained.

Based on the consideration that each alternative action is facing $n-1$ other alternative actions in A, let us now define the two outranking flows as [33, 39, 34]:

Positive Outranking Flow:

$$\phi^+ = \frac{1}{n-1} \sum_{x \in A} \pi(a,x)$$

Negative Outranking Flow:

$$\phi^- = \frac{1}{n-1} \sum_{x \in A} \pi(x,a)$$

where the positive outranking flow expresses how an alternative action a is outranking the rest of the alternative actions, whereas, the negative outranking flow expresses how an alternative action a is outranked by the other alternative actions. Thus, the positive outranking flow represents the outranking character or the outranking power of an alternative action, whereas, the negative outranking flow represents the outranked character or outranking weakness of an alternative action. Accordingly, a higher value of $\phi^+(a)$ and a lower value of $\phi^-(a)$ increases the suitability of an alternative a as the final choice [36].

PROMETHEE I: Partial Ranking

PROMETHEE I, partial ranking (P^I, I^I, R^I), is obtained from positive and negative outranking flows. Since positive and negative outranking flows do not usually induce the same rankings, PROMETHEE I takes their intersection where

aP^Ib **if and only if**

$$\begin{aligned}&\phi^+(a) > \phi^+(b) \text{ and } \phi^-(a) < \phi^-(b), \text{ or}\\ &\phi^+(a) = \phi^+(b) \text{ and } \phi^-(a) < \phi^-(b), \text{ or}\\ &\phi^+(a) > \phi^+(b) \text{ and } \phi^-(a) = \phi^-(b)\end{aligned}$$

aI^Ib **if and only if**

$$\phi^+(a) = \phi^+(b) \text{ and } \phi^-(a) = \phi^-(b)$$

aR^Ib **if and only if**

$$\begin{aligned}&\phi^+(a) > \phi^+(b) \text{ and } \phi^-(a) > \phi^-(b), \text{ or}\\ &\phi^+(a) < \phi^+(b) \text{ and } \phi^-(a) < \phi^-(b)\end{aligned}$$

where P^I, I^I, R^I stand for preference, indifference and incomparability, respectively [36]. In the case of aP^Ib, a higher power of a is associated to a lower weakness of a with respect to b, that is both outranking flows is consistent. In the case where aI^Ib, both positive and negative flows are equal. In the case of aR^Ib, a higher power of one alternative is associated to a lower weakness of the other. This occurs when a is strong on a set of criteria on which b is weak, whereas, b is strong on some other set of criteria on which a is weak. In such situations, the outranking flows are deemed to be inconsistent, and thus, the two alternatives are considered to be incomparable. The PROMETHEE I is prudent in such cases, and it will not decide which alternative action is the best.

PROMETHEE II: Complete Ranking

In the cases where a decision maker requests a complete ranking on the set of alternative actions. PROMETHEE II deals with such situations, and consists of the (P^{II}, I^{II}) complete ranking [33]. In PROMETHEE II, a new notion called the *net outranking flow* is considered in order to provide a complete ranking. The net outranking flow is defined as

$$\phi(a) = \phi^+(a) - \phi^-(a).$$

Esentially, the net outranking flow is the balance between the positive and the negative outranking flows. Accordingly, an alternative action is deemed as a more suitable choice when it has a high net outranking flow, so that: 1) $aP^{II}b$ if and

only if $\phi(a) > \phi(b)$, and 2) $aI^{II}b$ if and only if $\phi(a) = \phi(b)$. In PROMETHEE II, all the alternatives are comparable. However, the resulting complete rank is more disputable due losing information by taking the difference between the positive and the negative outranking flows. In PROMETHEE II, the following two properties hold: 1) $-1 \leq \phi(a) \leq 1$, and 2) $\sum_{x \in A} \phi(a) = 0$. In the cases where $\phi(a) > 0$, a's ooutranking character is stronger, whereas, in the cases where $\phi(a) < 0$, a's outranked character is stronger [35, 34, 39].

Extensions of PROMETHEE

There are multiple extensions to the initial PROMETHEE I and II methods. GAIA Visual Interactive Module is one of such extensions where the GAIA Plane is formed using the single criterion net flows for each alternative action, which are called the profiles of alternative actions. These profiles are computed as follows. Based on the definitions of the positive outranking flow, negative outranking flow and aggregated preference indices:

$$\begin{aligned}
&\phi(a) = \phi^+(a) - \phi^-(a) = \tfrac{1}{n-1} \textstyle\sum_{j=1}^{k} \sum_{x \in A} [p_j(a,x) - P_j(x,a)] w_j \\
&\phi(a) = \textstyle\sum_{j=1}^{k} \phi_j(a) w_j \\
&\phi_j(a) = \tfrac{1}{n-1} \textstyle\sum_{x \in A} [P_j(a,x) - P_j(x,a)]
\end{aligned}$$

where $\phi_j(a)$ is the single criterion net flow obtained when only one criterion $g_j(\cdot)$ is considered with 100% weight [36]. The single criterion net flow represents how an alternative action a is outranking, $(\phi_j(a) > 0)$, or outranked, $(\phi_j(a) < 0)$, by all the other alternative actions on criterion $g_j(\cdot)$. Accordingly, the profile of an alternative is the set of all the single criterion net flows: $\phi_j(a), j = 1, 2, \ldots, k$. The profiles of the alternatives represent their quality on the different criteria. Also, note that the global net outranking flow of an alternative alternative is the scalar product between the vector of weights and its profile vector. The GAIA plane is built using this particular property.

Moreover, in certain decision making applications, a subset of alternatives must be identified given a set of constraints. PROMETHEE V extends PROMETHEE I and II—where only one alternative action is selected—to such cases [38]. Also, PROMETHEE VI module provides the decision makers with additional information on their personal views on a particular mutliple criteria decision analysis problem [36]. Finally, PROMCALC and DECISION LAB are multiple criteria decision support

software based on PROMETHEE methods which help a decision maker in the process of modeling their views and making decisions [37].

4.2.3 Other Outranking Methods

There is a number of other methods that are based on the outranking approach. Some of the well-known ones among these methods can be summarized as follows. MELCHIOR [119] is an example of such methods that extends ELECTRE IV [92]. ORESTE is a method that uses ordinal evaluations of alternative actions and ranking of the criteria in terms of importance [92]. REGIME builds a pairwise comparison matrix where $+1$, 0 and -1 are used to represent dominance, indifference and negative dominance among alternative actions, respectively [92, 132]. Moreover, Novel Approach to Imprecise Assesment and Decision Environments (NAIADE) is a method that uses distance semantics operators to assess the pairwise comparisons of alternative actions. Similar to PROMETHEE, NAIADE also uses positive and negative outranking flows [92]. QUALIFLEX, ARGUS, EVAMIX, TACTIC, PACMAN, IDRA, PRAGMA and MAPPAC are some of the other methods that are based on the outranking approach [132].

4.3 Summary

In this chapter, we have reviewed some of the well-known MCDA methods that are based on MCAPs. In this context, we covered the two general approaches: Single Synthesizing Criterion methods and Outranking methods. In addition to the explanation of the formal background of some of the fundamental methods, we have also provided a comprehensive list of other methods that can be applied to the problem of providing adaptability in distributed systems through MCDA. However, the provided overview still spans only a representative subset of a vast number of available MCDA methods based on MCAPs. In that sense, some of the elementary methods such as Weighted Sum, Lexicographic Method, Conjunctive and Disjunctive Methods, and Maximin Method are not included in the scope of this chapter. Although the diversity in MCDA methods can be seen as an advantage in a general sense, it is also viewed as a weakness due to reasonable questions regarding how a suitable method can be selected given a specific multiple criteria decision making problem [31, 92]. Accordingly, some studies focused on providing formal frameworks for selecting MCDA methods

given the nature of a decision making problem [92]. The next chapter focuses on the simulation of certain cases in order to test this approach.

Chapter 5

Simulation Platforms

In this chapter, we overview the two simulation platforms that are used to assess the two systems that we have built using the proposed methodology to the problems of independent self-management and coordinated self-management. The main reason behind using simulation platforms in these two simulation case studies is to eliminate possible issues with respect to high expenses of setting physical test beds—which can be substantial in both simulation case studies. In addition, using simulation platforms is a well-established way to avoid the apparent time-consuming activities such as hardware configuration, calibration and maintenance. However, it is also important to note here that simulation platforms and employed scenarios are simply models of reality and are not always very accurate in representing the actual situation. As a result, the conclusions drawn from simulation runs most often require further analysis and validation due to possible inaccuracies that may stem from erroneous input models.

The first simulation platform is built to run test scenarios and facilitate comparative studies on the usage of different methods within the context of dynamic resource consolidation management in clouds. Accordingly, we have built the platform from ground up for simulation of changing resource requirements in a cloud based on a three-layered architecture. The platform mainly focuses on the infrastructure as a service (IaaS) layer of clouds, and abstracts away from the various higher-levels of software by representing them in terms of their resource requirements in the form of virtual machines. Similarly, the resources available in a cloud are represented by physical machines. The platform allocates its top layer for resource consolidation management where virtually any resource consolidation manager can be deployed and run. The resource consolidation managers work in terms of changing the locations of

the virtual machines in a cloud in order to control overall resource utilization. Using this platform, an autonomous system based on the principles outlined by the proposed methodology is built as a solution to the problem of independent self-management in a distributed system.

The second simulation platform is used for the analysis of a self-management system that is built based on the proposed methodology for providing adaptive routing in MANETs through switching between routing algorithms in real-time. For reasons due to the level of fidelity, OMNET++ [145] is used as the main platform with certain extensions.The platform uses the general structure provided by INETMANET [100] framework built for OMNET++, and extends it through implementation of specific modules to facilitate decision making in MANET nodes and coordination within the network. The extensions do not interfere with the general functionality of the platform and only provide additional functionality to be used by the autonomous agents. The platform is used to test the applicability of the proposed methodology within the context of coordinated self-management in distributed systems.

In the rest of this chapter, we cover the two simulation platforms that were implemented or used for the purpose of testing the methodology's applicability in the two specific problem domains. Accordingly, Section 5.1 covers the simulation platform that we have built for the independent-self management case, and Section 5.2 covers the extensions that are made to INETMANET framework within OMNET++ platform to be used for coordinated self-management case.

5.1 Platform for Simulation Case Study 1

This simulation platform is built for running scenarios regarding the problem of dynamic resource consolidation management in clouds. The simulation platform is built using the Python programming language based on an initial implementation in C programming language. The new platform has certain additional features extending the simulation basics that were outlined in our previous studies [215]. The choice of changing the programming language was mainly due to reasons such as code organization and maintenance. The platform is designed based on a three-layered architecture: (1) resource layer, (2) simulation engine, (3) consolidation management. A high-level view of the general platform architecture is illustrated in Figure 5.1.

Resource layer forms the lowest layer of the simulation platform. It consists of physical machines and virtual machines, and is the most abstract representation of

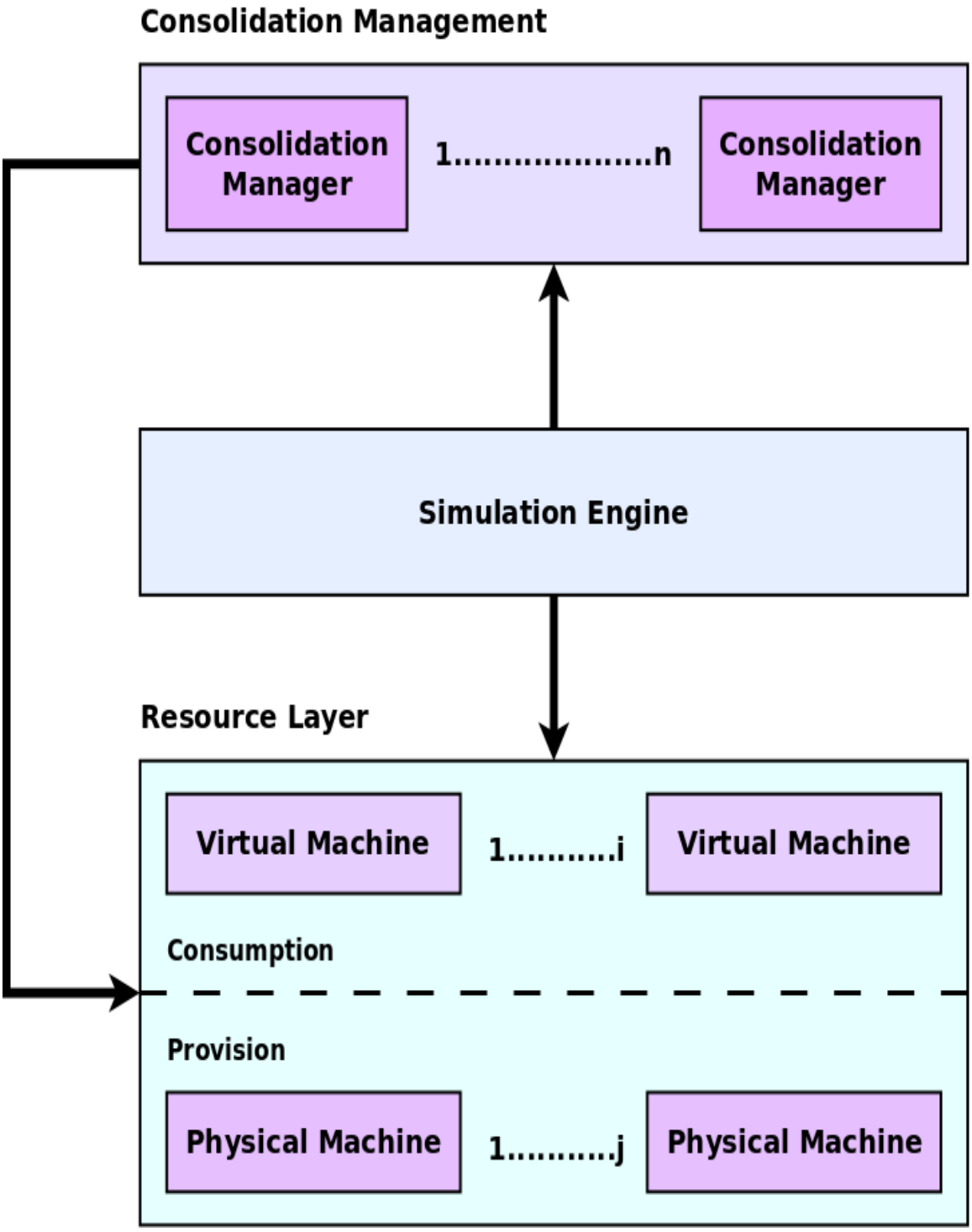
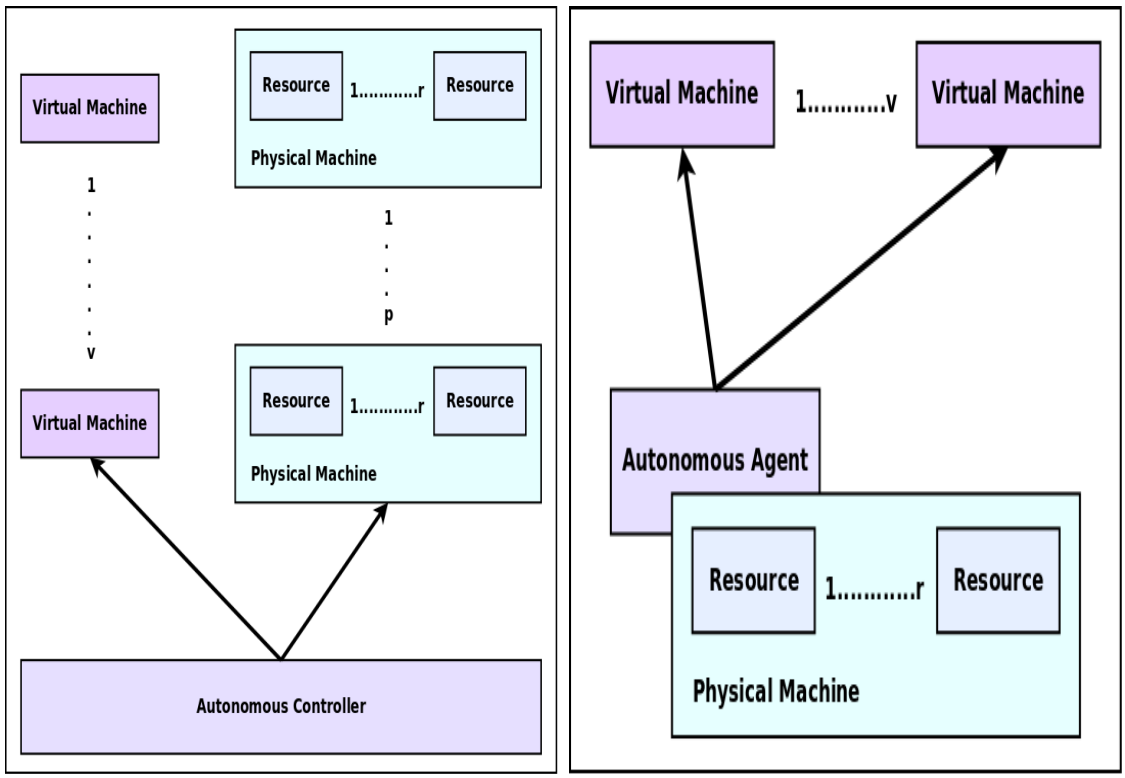


Figure 5.1: General architecture of the platform for simulation changing resource requirements at the IaaS level of a cloud.

a cloud as to how the resources are made available by a provider and consumed by clients. The resource layer holds the general information regarding the definition of both the physical machines and the virtual machines. Each physical machine is considered as a basic unit that provides a portion of the resources available in the data center. Accordingly, they have certain limits on the resources that they can provide on each resource dimension. As long as the total resource consumption on a physical machine is kept within the limits on each available resource dimension, it can host any number of virtual machines without encountering violations of service level objectives (SLO). On the other hand, each virtual machine represent a component or the whole of a software environment to be served by the cloud. Virtual machines receive the amount of resources they require by being hosted on a physical machine in the data center. Virtual machines are mainly defined by how their consumptions change on different resource dimensions. Each virtual machine's consumption is defined based on standard statistical distributions, e.g., Gaussian, Exponential, Pareto, etc. The resource requirements change independently on different resource dimensions following a set of statistical distributions assigned to a virtual machine during the initialization of the simulation scenario. Once a simulation run starts, each virtual machine change their resource requirements based on these predefined statistical distributions when ordered by the simulation engine.

The changes proceed differently based on the type of virtual machines. Currently there are two types of virtual machines in the simulation platform. The first type represent a behaviour much like the batch processes, changing resource requirements on each resource dimension as continuous statistical noise throughout the simulation run. Whereas, the second type of virtual machines follow the behaviour of on-line processes where resource requirements are continuous statistical noise only during the on-load periods. During the off-load periods a virtual machine does not require any resources except for the ones that are permanently necessary, such as memory. In the second type of virtual machines, the length of on-load and off-load periods are also generated using certain statistical distributions, which are specifically chosen to generate a more realistic pattern of change [81].

Consolidation management is at the top of the three-layered architecture. This layer is the main location where the resource consolidation managers are deployed. Based on the type of resource consolidation manager to be used at a simulation run, this layer assumes the control of how the virtual machines are mapped and assigned to the physical machines in the data center. The assignments depend on the knowledge



(a) Layout using a centralized resource consolidation manager.

(b) Layout using a distributed resource consolidation manager.

Figure 5.2: Different deployment schemes with respect to centralized and distributed self-management approaches.

of resources required by each virtual machine, and the resources available on each physical machine. Using this knowledge, the resource consolidation layer computes a mapping from the virtual machines to physical machines using the currently deployed resource consolidation manager. Based on this mapping it performs resource assignments by migrating virtual machines to their newly assigned locations. A general view of the two types of consolidation managers that can be deployed in the consolidation management layer are given in Figure 5.2. Accordingly, the consolidation management layer can be used to deploy two types of consolidation managers, namely the centralized and distributed managers. Figure 5.2a illustrates the appearance of centralized consolidation manager in the overall scheme of the simulation platform. Note that the autonomous controller here represents the main unit of consolidation; and it has a full view of the cloud including the states of both the virtual and physical machines. In this case, the consolidation management is controlled from a single uni-

fied point in the cloud. Whereas, Figure 5.2b illustrates the layout for the distributed consolidation managers. In this case, every autonomous agent is responsible for a unique location in the data center, which is also the general design used in Chapter 6 of this work. As a result, the simulation platform is not rigidly designed based on a distributed view, but also supports any arbitrary implementation including other possible approaches such as semi-distributed consolidation management, etc.

Simulation engine lies in the middle of the resource layer and the consolidation management, and acts as the coordinator of simulation runs. One of its main responsibilities is to initialize simulation runs. In order to do that, simulation engine loads a user defined scenario description file that specifies simulation parameters such as the number of virtual machines to be managed in the data center, the resource limits on each physical machine, the types of virtual machines, the types of resource dimensions available in the cloud, the statistical distributions to be used as descriptions of requirement changes of virtual machines, the type of resource consolidation manager to be employed, etc. It is also responsible for managing the flow of simulation runs. It manages the simulation in a step-wise manner. Each step consist of two turns. During the first turn, it conducts the changes in the cloud by signaling the resource layer. In the second turn, it activates the resource consolidation management and assists it by providing the information regarding the current state of the data center. The result of each step is a new mapping between virtual machines and the physical machines in the data center. In a general sense, it acts as an intermediary that carries the information and action commands between the two layers throughout the process of resource consolidation. The simulation runs are carried out in turn-based steps in order to measure the reactions of each resource consolidation manager under the exact same conditions at the exact same points in time in each simulation run.

5.1.1 Current State and Possible Improvements

It is important to note here that this simulation platform is not designed and implemented to test cloud service levels above IaaS. That is, the platform does not address the simulation of platform as a service (PaaS) and software as a service (SaaS) layers of a cloud. There are other simulation platforms that focus on these layer in an extensive manner, such as CloudSim [52]. Note that such platforms also provide facilities to support simulations at the IaaS layer. However, since such platforms are rather heavy-weight general-purpose platforms that provide a lot of functionality, we

have built a light-weight special-purpose platform solely for the IaaS layer of a cloud. The current implementation of the platform is modular enough to facilitate further improvements in the form of certain modifications and extensions. Some of those improvements can be outlined as follows.

Currently, the input models regarding the resource requirement changes that are used by the platform are extracted from previous studies regarding general patterns in the Internet. However, this information may need to be renewed based on actual information as released by the data center providers to have a more complete and up-to-date input model, which can produce results that are more realistic in terms of the assessment of different resource consolidation managers. The simulation platform currently views the events in the data center merely as the resource requirement changes of the virtual machines. This behaviour can be further granulated by modifying the environment so that it supports the processing of events in the form of incoming service requests to virtual machines. However, such an extension also requires that the amount of load that will be inflicted on the physical machines by each request should be well-determined per each virtual machine or the software environments that are running inside them. In addition, it is necessary to identify or extract the input models for received requests for each virtual machine in the data center over-time. The platform is implemented with such a view in mind and will be extended to have that event view in the near future. Finally, another future goal is to provide a complete set of implementation for resource consolidation managers that are or are to be proposed for comparative analysis. Implementation may also require certain considerations to be taken into account regarding the design and implementation of the platform that may not have been considered previously. Once the consolidation management layer is extended by deploying other well-known methods, the platform can be used to provide a better view of the current state in the field based on extensive comparative studies.

5.2 Platform for Simulation Case Study 2

This simulation environment for the case regarding coordinated self-management uses the OMNET++ discrete event simulation platform. The simulation scenarios are implemented on OMNET++ with certain extensions to the general framework. The extensions include implementation and deployment of a number of convenience modules at the different layers of the INATMANET framework. The convenience modules

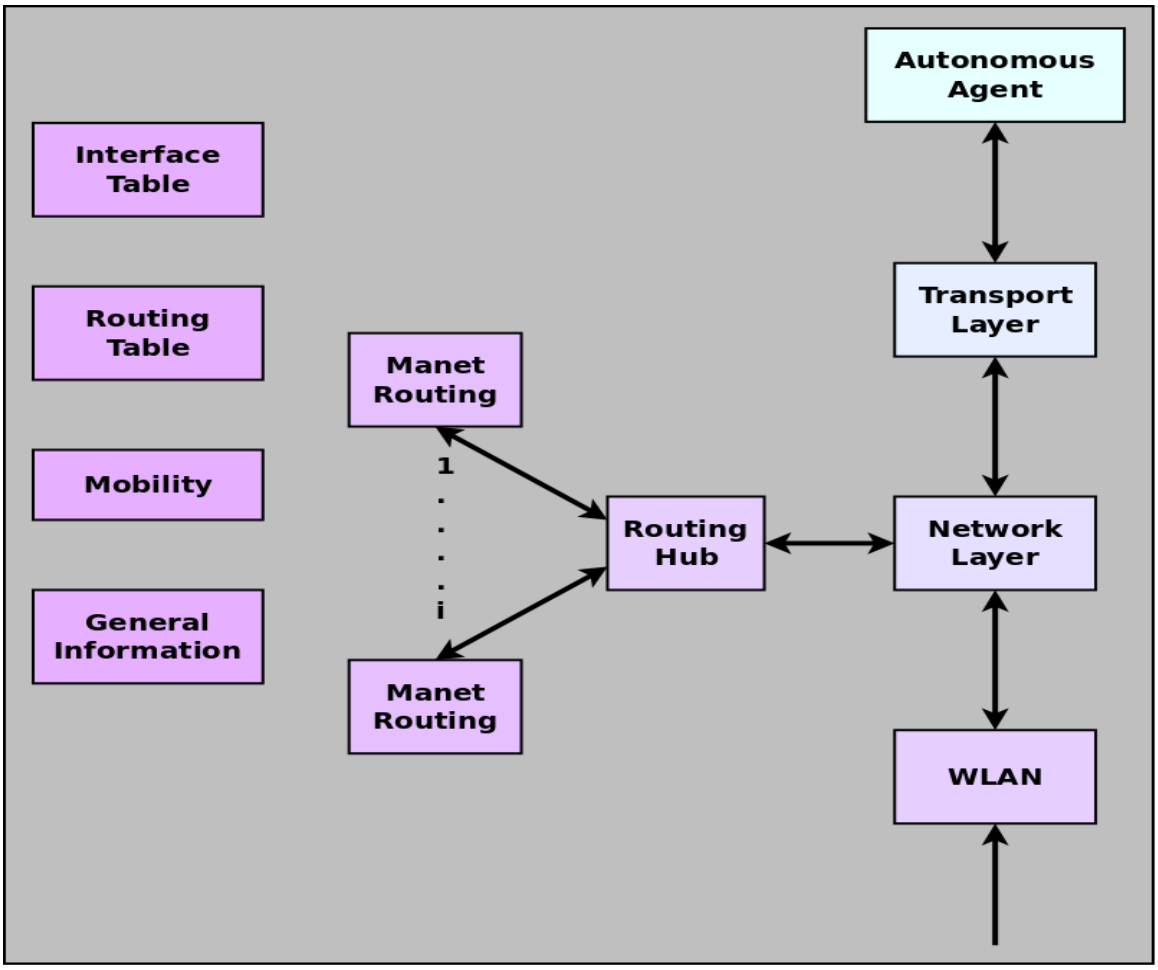


Figure 5.3: General view of the internal structure of each generic node used in the simulation runs.

do not interfere with the existing functionality of the underlying platform; they are merely a set of extensions that access the data structures (IP tables, ARP, etc.) that hold the general information needed by the autonomous agents.

The generic MANET nodes' capabilities are further extended in order to have the capability of carrying out MCDA and coordination procedures. The extensions include additional compound modules to provide the functionality during the observation, detection and reaction activities. In this context, each generic node is attached an autonomous agent that provides this functionality, and certain additional data structures are added to hold the general information regarding the states and perceptions of the MANET nodes. Figure 5.3 illustrates the extended MANET nodes. Note that in the extension, each node is capable of using multiple routing algorithms;

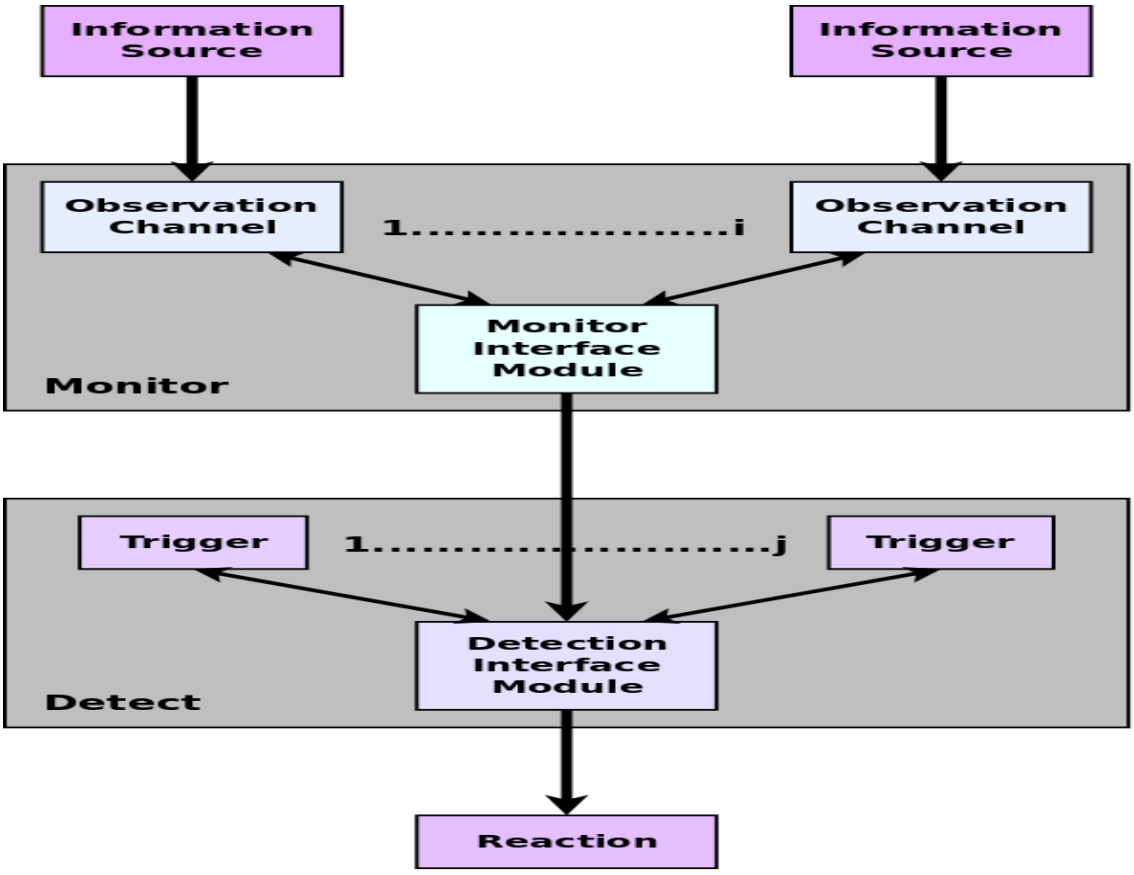


Figure 5.4: General view of the internal structures of the observation and detection modules, and the data flow between them.

and the state information is held in data structures called *GeneralInformation* to be used by the autonomous agent.

Each autonomous agent contains modules that support the observe-detect-react cycles as outlined by the proposed methodology. In this context the monitor module is designed to collect information by querying lower-level modules or polling certain variables on a regular basis. The monitor module carries out the responsibilities as outlined for the observe activity. Collected information are sent to the detection module through signals where it is processed in order to capture trigger conditions. The information flow and the interaction between the monitor and detect modules are illustrated in Figure 5.4. Note that, these modules are implemented specifically to support any method of observation, or trigger detection since they represent merely an intermediate layer between the lower-levels of the system, and the reaction activity of the autonomous agent.

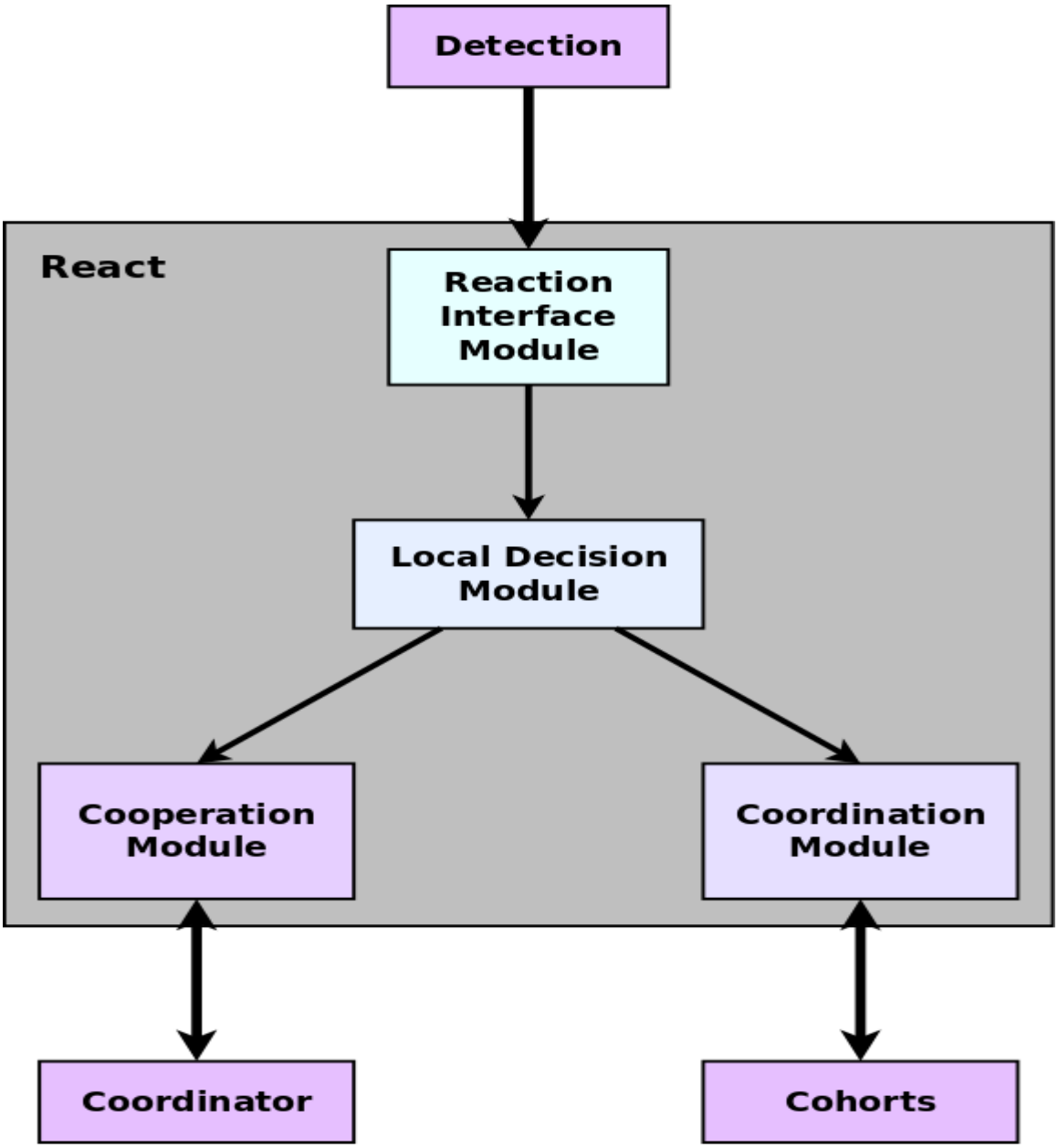


Figure 5.5: General view of the reaction module and the data flow during the coordaintion and coorperation procedures.

Similarly, the reaction module includes a set of modules that facilitate the decision making capability, cooperation with a coordinator when requested, and taking the responsibility of coordination. Once again, these modules are built so that any decision making method or cooperation/coordination algorithm can be implemented as simple or compound modules into them. A general view of the internal structure

of the reaction module and the data flow between it and the other modules, or the rest of the network is illustrated in Figure 5.5. Furthermore, in order to facilitate switching between routing algorithms, we have implemented a routing hub module that can be connected to multiple MANET routing modules simultaneously and can transmit the packets in the right directions based on the current routing algorithm in use.

5.2.1 Current State and Possible Improvements

Currently, the implemented extension modules help us demonstrate an approach that is capable of arriving at a group decision without the usage of a centralized authority, which can be used as a basis for addressing the problem of adaptive routing in MANETs. However, our future goals include certtain modifications to the general design and implementations of the extension modules so that the same structure can be utilized in a more generic manner in order to address other self-management problems within the context of MANETs. This requires certain modules that interact with the OMNET++ simulation engine to be slightly changed towards generic purposes. In particular, the general purpose compound and simple modules that are used for information collection or acting as intermediary points in the system—such as the routing hub—need to be further tailored to support any self-management problem. In addition, once the necessary changes are implemented towards a more generic extension for decision support, new scenarios including self-management problems in other domains within the context of MANETs can be implemented and assessed.

5.3 Summary

In this chapter, we have outlined the general structure of the simulation platforms that are built and used during the assessment of the application of the proposed methodology. In this context, two platforms were overviewed. The first platform is implemented in order to apply the proposed methodology to the problem of dynamic resource consolidation management in clouds as a representation of the independent self-management. The second platform is an extended version to OMNET++ to support the activities that the proposed methodology underlines for coordinated self-management. The platform is used during the simulation case study of adaptive routing in mobile ad hoc networks. In the next two chapter, we will provide an in-

depth analysis of how the methodology is applied to the problems at hand using the simulation platforms that are outlined in this chapter.

Chapter 6

Simulation Case Study 1: Independent Self-Management

Cloud computing is an emerging trend in current computing. The main tenet of this new paradigm is its business-centric view, in which computational resources are deemed as measurable and billable utilities. Clouds aim to provide services much like the way that other common utilities, such as electricity and water, are provided to the public. In this context, clouds deliver computational resources as services—on a pay-per-use basis—that can be simultaneously accessed by multiple clients without a detailed knowledge of the underlying infrastructure. Although cloud computing shares a common vision, architecture and technologies with former paradigms, it is different particularly with respect to security, programming model, business model, compute model, data model, applications and abstractions [84].

In a general sense, clients view a cloud as an abstract pool of computational resources, where they can deploy and run their software environments. The amount of resources to be assigned to each software environment are defined through SLOs. Typically, a software environment in a cloud consists of one or more virtual machines with resource requirements that reflect on its individual SLO. In this context, virtualization plays a critical role in cloud computing. Particularly at the IaaS layer, it highlights interesting problems and opportunities. One of these is the problem of autonomous resource consolidation management [94]. The main goal of solving this problem is to provide an autonomous IaaS layer that distributes computational resources among software environments dynamically as the workloads change. Currently, static resource consolidation is generally deemed acceptable. However, it is

likely that the structure of SLOs will evolve towards higher-level definitions, which implies further abstraction of how the low-level computational resources are assigned to software environments [199, 215]. In such settings, static resource consolidation can cause problems with respect to lack of responsiveness. Moreover, manual administration to compensate the shortcomings of static resource consolidation may not be feasible. Therefore, an autonomous IaaS layer that is capable of managing resource consolidation dynamically will be necessary in the future.

In this chapter, we present IMPROMPTU, a distributed resource consolidation manager for clouds. The main contribution of this work is two-fold. First, IMPROMPTU fully distributes the responsibility of resource consolidation management among autonomous node agents that are tightly coupled with the physical machines in the cloud. Second, autonomous node agents manage resource consolidation using MCDA through PROMETHEE method [36]. To the best of our knowledge, neither a distributed approach nor MCDA has been applied to the problem at hand. Moreover, IMPROMPTU improves on our previous studies [215, 214] substantially by introducing extensions that further granulate the MCDA model. Simulation results show that IMPROMPTU is a solid alternative to the previous work in the field.

The rest of this chapter is organized as follows. In Section 6.1, we provide a detailed definition of the problem domain. Section 6.2 outlines a survey of previously proposed solutions. In Section 6.3, we decsribe the MCDA model used by IMPROMPTU in detail. Section 6.4 presents IMPROMPTU's distributed system design. In Section 6.5, we provide an in-depth performance analysis of the proposed system. Finally, Section 6.6 summarizes this work, raises further questions, and underlines future directions within this specific problem domain.

6.1 Problem Description

In cloud computing, virtualization [158] is a major ingredient that provides the necessary abstraction of computational resources, and encapsulation of software environments. It ensures a certain level of security and isolation which is essential to almost any cloud [84, 120]. In addition, since a virtual machine is loosely coupled with the physical machine it runs on, not only can it be started, suspended, stopped, resumed, etc. on any physical machine, but also it can be migrated to others at run-time. The ability to migrate virtual machines, in particular live migration [50], facilitates flexible resource consolidation management in clouds, and provides easier

means to dynamically configure the distribution of computational resources among virtual machines with minimal interruption to their services. At the IaaS layer both clients and providers can benefit from this enhanced level of manageability. Clients can increase or decrease the resources assigned to their software environments at any point in time, whereas, providers can relocate software environments within the cloud for reasons such as meeting upgrade/downgrade requests from clients, or scheduled maintenance operations. The advantages facilitated by virtualization are leveraged by certain IaaS providers in the market such as Amazon EC2 [7] and Amazon S3 [8].

Manageability of a cloud at the IaaS layer can be enhanced further by automating resource consolidation management. However, in the current state of cloud computing, resource consolidation is still done statically. That is, once a required level of computational resources is assigned to a software environment, it remains at the location where it was initially deployed unless the resources are reallocated by cloud administrators manually. For the time being, static resource consolidation is acceptable, since today's SLO are generally defined through rigid low-level metrics such as memory, bandwidth, etc [199]. Therefore, unless a customer dynamically alters the resources assigned to a software environment—given that it is permitted by the provider—in a way that the current location of the software environment can no longer provide the required resources, there is no need for relocation.

However, in the future, it is likely that the SLOs will evolve towards higher-level definitions that will further abstract the low-level computational resources [199, 215]. For instance, response time is a particularly good example to some of the soft metrics that can be used in high-level SLOs. The necessary resources to meet a certain response time for an on-line application may not remain constant throughout its lifetime in the cloud. That is, the amount of computational resources needed to deliver a certain response time for a software environment will change with respect to the changes in its workload. In such settings, static resource consolidation may cause problems due to under-provisioning or over-provisioning. Under-provisioning results in more frequent violations of SLOs, whereas, over-provisioning results in wasted computational resources.

Moreover, in large-scale settings like clouds, manual administration to compensate the shortcomings of static resource consolidation is not a feasible option. From the clients' point of view, an IaaS layer must continuously ensure that their software environments are assigned enough resources regardless of changing workloads so that SLO violations are minimized. Whereas, the providers expect that the resource utilization

in their data centers is maximized. Then, an autonomous IaaS level is responsible for continuously finding solutions in the face of two conflicting goals in order to ensure that the resources are allocated in a way that both parties are satisfied as much as possible. Therefore, an autonomous IaaS level that is capable of continuously performing dynamic resource consolidations in order to reach the goals defined by cloud customers and cloud providers is a necessity in the future of cloud computing.

This problem has been investigated in a number of previous studies [199, 94, 194, 3, 62]. Besides the efforts to solve a seemingly one-piece problem, the general approach provides an insight as to how the problem can be decomposed into two consecutive, yet, distinct sub-problems. First, software environments need to be profiled in terms of defining a mapping from their workloads to the amount computational resources that guarantees meeting individual SLOs. Second, the computational resources need to be distributed among software environments based on the provided mappings in an autonomous manner. We believe that such a decomposition can facilitate a more in-depth understanding, and aid in a more focused analysis of two disjoint components of the general problem. In this work, we focus only on the automation of resource consolidation management by assuming that the mappings from workloads to computational resources are already provided.

6.2 Related Work

Since first proposed in the seminal work of Walsh et al. [199], a general two-tier architecture that uses utility functions has been commonly adopted in the context of autonomous resource consolidation management. The architecture consists of local decision modules and a centralized global arbiter. Each local decision module is attached to a certain software environment where it is responsible for generating the software environment's preferences over a set of different levels of computational resources. Preferences are determined in the form of utility values that are produced using a certain utility function. The utility values are communicated to the global arbiter as representations of mappings from workloads to computational resources for each software environment. Using these values, the global arbiter attempts to find a solution that distributes computational resources among the software environments in a way that maximizes the total utility in the data center. In a general sense, this approach maps the problem to the Knapsack Problem, or a certain variant of it, the Vector Bin Packing Problem, both of which are known to be NP-Hard [199, 94].

Studies outlined in Bennani et al. [22], Chess et al. [48], Tesauro [190], Tesauro et al. [188, 189], and Das et al. [56] adopted this approach for non-virtualized data centers. They have also focused on performance modeling of software environments by leveraging forecasting methods based on different analytical models. While Bennani and Menasce [22] mainly focused on a queuing theoretic approach to the performance modeling problem, Tesauro [190] and Tesauro et al. [188, 189] considered a pure decompositional reinforcement learning approach along with its hybridization with the queuing theoretic approach. Furthermore, Chess et al. and Das et al. [48, 56] used this same architecture and utility model to build a commercialized computing system, called Unity.

More recent research has focused on virtualized data centers. Starting with the work of Almeida et al. [6], data center utilization has been explicitly considered as a major criterion in resource consolidation management. In addition, this work outlined data center utilization as a major factor in both short-term and long-term resource planning. Other research outlined in Khanna et al. [107], Bobroff et al. [23], Wood et al. [208], Wang et al. [201], Kochut [109], Hermenier et al. [94], and Van and Tran [194] have also adopted a centralized architecture. In some studies the cost of migrations was taken into account [208, 107, 201, 94, 194] in order to increase the feasibility of the new resource allocations. Hermenier et al. [94] focused on reducing the number of migrations through constraint solving. The main goal of this approach is to find a set of possible solutions, and selecting the most suitable one among them with respect to maximizing global utility and minimizing the number of migrations. The same method is also adopted in the follow-up work of Van and Tran [194].

In the work of Agrawal et al. [3], an approach based on a grouping genetic algorithm, which takes into account further factors such as placement conflicts, was proposed. Placement conflicts are used mainly to determine which virtual machines can or cannot be hosted on the same physical machines. In a more recent work, Dutreilh et al. [62] provide a comprehensive study on the difficulties of solving the problem by considering queuing theory and control theory approaches. Similar models using different methods have also been proposed for single-server settings where multiple software environments are hosted on a single physical machine. Some of those approaches are outlined in the work of Chandra et al. [46], Mahabhashyam [130], and Menasce et al. [136].

These proposed approaches can be very efficient in clouds that operate on small-scale data centers. However, we believe that they can potentially suffer from scalabil-

ity and feasibility issues in large-scale settings, which are more realistic case studies given the size of today's commercial clouds. The main source of the problems due to scalability is the centralized control over resource consolidation management. Centralized management of resource consolidation—coupled with the usage of complex algorithms that take into account a number of criteria—brings along a heavy computational burden of searching a massive solution space in order to compute good solutions. This may often take a long time—generally seconds to minutes—in realistic settings, and in turn, the computed solution might be stale due to the speed at which the conditionns of the cloud might change.

The computational burden can be alleviated through the usage of simpler algorithms. In this case, however, certain feasibility issues arise. Mainly, solutions reached by those simpler algorithms enforce too many migrations; meaning that the solutions can be found in a shorter time, and yet, may be physically impossible to carry out due to limitations on certain resources such as network bandwidth [215]. In a more general sense, feasibility issues may also arise in complex solutions proposed in the previous studies due to high rates of SLO violations. This is mainly due to the uni-objective view that is generally adopted. In most of the proposed solutions, the main objective is to prevent under-utilization. Based on this view, most of the previous solutions aim for packing tightly the virtual machines on the physical machines in a cloud. This view is destined to produce an increased number of SLO violations since the physical machines will not have the necessary margin to tolerate sudden increases in the resource requirements of the virtual machines. This may in turn cause an increase in the number of migrations per unit time since the resource consolidation manager is likely to look for new solutions more frequently due to the need that arises from a high number of encountered SLO violations.

IMPROMPTU adapts a distributed architecture that leverages MCDA in order to solve the problem of autonomous resource consolidation management. The proposed solution takes into account the views of both clients and providers, and aims to manage resource consolidation in a scalable and feasible way. IMPROMPTU distributes the responsibility of resource consolidation management accross multiple units of computation in order to eliminate the scalability issues that can be seen in centralized approaches. The distribution of responsibility makes IMPROMPTU inherently scalable by partitioning the solution space to scales that are strictly local to the physical machines, which results in fast and up-to-date solutions regardless of the size of the cloud. In addition, resource consolidation is performed in a reactive

manner, which in turn—coupled with the distribution of responsibility—produces a significantly lower number of SLO violations and feasible solutions that require a very low number of migrations. In the next section, we provide a detailed overview of this new approach in terms of its MCDA model, and how the new solutions are computed using PROMETHEE method.

6.3 IMPROMPTU

IMPROMPTU distributes the responsibility of resource consolidation management among autonomous node agents. Each autonomous node agent is tightly coupled with a unique physical machine in the cloud, where it assumes full control over local resource consolidation management. The primary responsibilities of an autonomous node agent are (1) observing the current conditions of the physical machine that it is attached to, (2) capturing undesirable situations as they occur, and (3) ensuring that the desirable condition on a physical machine is restored immediately after an undesirable condition is captured. In this context, autonomous node agents work in a completely reactive manner, carrying out solutions only when prompted by the problematic situations. Upon detecting such situations, the autonomous node agents independently decide on their next course of action based on an MCDA model.

The rest of this section describes the general working principles of IMPROMPTU as follows. In Section 6.3.1, we outline the reactive behaviour of the proposed system in terms of formally defining how the reactions are triggered by each autonomous node agent. Section 6.3.2 explains the general MCDA model that is followed by an autonomous node agent during the reactions. Finally in Section 6.3.3, we provide a brief explanation of PROMETHEE as the MCDA method that is employed for concluding the reaction processes following the MCDA model of IMPROMPTU.

6.3.1 Reactive Resource Consolidation Management

IMPROMPTU views a cloud in terms of two main entities. Virtual machines represent specific components or the whole of a software environment, whereas physical machines represent bins that can accommodate virtual machines by providing them with computational resources. Although resources can be provided by physical entities that do not host virtual machines, physical machines can be considered as the most general and abstract unit of resources in a cloud. Therefore, the proposed solu-

tion assumes that the physical machines in a cloud are the direct representations of the available computational resources. Then, the resources in a cloud can be represented in terms of the physical machines as $P = \{p_i \in P \mid i = 1, 2, ..., I\}$, where each p_i represent a unique physical machine, and P is the set of all physical machines in the cloud. Similarly, we can define the set of all virtual machines being hosted on a physical machine p_i at a given point in time as $V^i = \{v^i_j \in V^i \mid j = 1, 2, ..., J\}$, where each v^i_j represent a unique virtual machine. Then, the set of all virtual machines in the cloud at a certain instance can be defined as $V = \bigcup_{i=1}^{I} V^i$, given that,

$$V^i \neq V^{i'} \text{ and } V^i \cap V^{i'} = \emptyset, \ \forall i, i' \text{ where } i \neq i'. \tag{6.1}$$

which underlines that each unique virtual machine can be hosted by only one physical machine at a specific instance.

Each autonomous node agent needs a set of parameters in order to define the conditions on the physical machine that it is coupled with. The most obvious of these parameters are the measured usages of available resource types—referred to as *resource dimensions* henceforth—such as compute, storage, communication, etc. Then, let us define the set of resource usage metrics as $G_R = \{g_n(\cdot) \in G_R \mid n = 1, 2, ..., N\}$, where each $g_n(\cdot)$ denotes the measured usage on a specific resource dimension n. From an autonomous node agents point of view, total resources in use on a physical machine are defined in terms of the computational resources assigned to the virtual machines that are currently being hosted. Let $g_n(v^i_j)$ denote the amount of resources assigned to a virtual machine v^i_j on resource dimension n, which represents a certain portion of the resources available on dimension n of physical machine p_i. Then, the total proportion of resources being used at a specific instance on resource dimension n of physical machine p_i can be defined as $g_n(p_i) = \sum_{j=1}^{J} g_n(v^i_j)$, where $0 \leq g_n(v^i_j) \leq 1$. It is important to note here that IMPROMPTU assumes that hypervisors are assigned predetermined amounts of resources on each resource dimension n statically, which is considered separate from the total amount of resources available on each dimension. Thus, the hypervisor's own resource usage is neglected during the calculation of $g_n(p_i)$.

The values of each $g_n(\cdot)$ are used by the autonomous node agents to define the current state of physical machines. This is performed by autonomous node agents through checking whether on their respective physical machines resource usage on each dimension are within desirable limits. The desirable limits on each resource

dimension is defined by predetermined lower and higher resource thresholds L_n and H_n, where $0 < L_n \leq H_n \leq 1$. In this context, L_n and H_n are used for defining under-utilization and over-utilization limits on resource dimension n respectively. In particular, H_n is associated with the perception of SLO violations. We assume that no virtual machine in the data center can violate the higher threshold on any resource dimension of a physical machine on its own, then, $g_n(v_j^i) < H_n, \forall i, j, n$.

Using the lower and higher thresholds, autonomous node agents continuously update their perception of the current state of their physical machines with respect to all available resource dimensions. The current state S_i of a physical machine p_i, is defined by an autonomous node agent as,

$$S_i = \begin{cases} 1 & \text{if } \exists n,\ g_n(p_i) > H_n, \\ 2 & \text{if } (\exists n,\ g_n(p_i) \neq 0) \ \wedge \ (\forall n,\ g_n(p_i) < L_n), \\ 3 & \text{if } \forall n,\ g_n(p_i) = 0, \\ 4 & \text{if otherwise.} \end{cases} \tag{6.2}$$

which defines the desirable states on physical machine p_i as $S_i = 3$ and $S_i = 4$. One of the two desirable states, $S_i = 3$ is encountered when a physical machine is not hosting any virtual machines, which implies that there is no resources assigned to the clients' software environments. In this case a physical machine is assumed to be in a state similar to hibernation, which in turn causes its autonomous node agent to stand-by until resources are in use again. In the second desirable state, $S_i = 4$, the resource usages on each resource dimension of a physical machine is between the corresponding lower and higher threshold values.

On the other hand, the undesirable states are defined as $S_i = 1$ and $S_i = 2$. One of the undesirable states, $S_i = 1$, represent the cases where there is the danger of an increasing number of SLO violations on the physical machines due to over-utilization on at least one of the resource dimensions. The second undesirable state $S_i = 2$, represents the conditions where there is a problem of under-utilization on all resource dimensions.

Autonomous node agents react to changes in conditions in specific manners based on the captured value of S_i. The type of reaction to be carried out on a physical

machine is defined as,

$$R_i = \begin{cases} \text{Tune} & \text{if } S_i = 1, \\ \text{Evacuate} & \text{if } S_i = 2, \\ \text{No Reaction} & \text{if } S_i = 3 \ \vee \ S_i = 4. \end{cases} \tag{6.3}$$

where *Tune* and *Evacuate* are reactions that require different procedures. Naturally, *No Reaction* represents the behaviour of an autonomous node agent under desirable conditions.

6.3.2 Multiple Criteria Decision Analysis Model

Tune is a reaction where an autonomous node agent attempts to bring the total usage on each resource dimension back below the higher threshold, so that, SLO violations are avoided as much as possible. In order to do this, it needs to migrate one or more virtual machines that it holds to other locations in the cloud. Virtual machines to be migrated need to be chosen in way that resolves the undesirable condition as quickly as possible. On the other hand, the new destinations for each virtual machine to be migrated need to have enough resources to host them without encountering any immediate problems. The reaction is carried out in an iterative manner where each iteration consists of selecting a single pair of virtual machine and physical machine. This is done in order to avoid searching for combinations of pairs in a potentially very large combinatorial search space. Then each iteration can be defined in terms of two sequential decisions: (1) choosing a virtual machine to migrate, and (2) choosing a physical machine that the selected virtual machine will be migrated to.

In the *Evacuate* reaction an autonomous node agent attempts to migrate all of the virtual machines that are currently being hosted on its physical machine to other physical machines in the cloud order to avoid wasting resources. This reaction is also carried out in an iterative manner. However, since all of the virtual machines are to be migrated, the autonomous node agent needs to choose only a physical machine for each of the virtual machines. The two distinct types of decision involved in the *Tune* and *Evacuate* are henceforth referred to as *decision type 1* and *decision type 2* respectively. When combined, these decision processes are required to output the most suitable course of action under the given conditions. Accordingly, each decision

type defines a sets of alternatives to select from and a set of criteria used for evaluating each alternative's suitability.

In *decision type 1*, the goal is to find the most suitable virtual machine to migrate from the set of virtual machines that are currently being hosted on a physical machine. Let A_1 denote the set of alternatives in decision type 1. Then, we can define the set of alternatives on a problematic physical machine p_i as $A_1 = V^i$. The most obvious way of evaluating each virtual machine's suitability as a candidate for migration is through their usages on different resource dimensions. Then the set of criteria for decision type 1 can be defined as the resource usage measurements of virtual machines, G_R. Evaluation should favour alternatives with higher values over the elements in G_R, since decision type 1 is only performed when there is a higher threshold violation. That is, from a local point of view, the virtual machines with higher usages on the resource dimensions are deemed more suitable for migration, since removal of such virtual machines is more likely to resolve a higher threshold violation.

In *decision type 2*, the autonomous node agent aims at find the most suitable physical machine in the cloud for a given virtual machine. Let A_2 denote the set of alternatives in decision type 2. Then, A_2 is the set of all physical machines in the data center that can accommodate the given virtual machine. An autonomous node agent attempts to generate the set of alternatives as,

$$A_2 = \{p_{i'} \in A_2 | (g_n(p_{i'}) + g_n(v_j^i) < H_n, \forall n) \ \wedge \ (S_{i'} = 4), \forall i^{'}\} \tag{6.4}$$

In this step, an autonomous node agent tries to find physical machines that have usages within L_n and H_n on all of the resource dimensions, and can accommodate the given virtual machine without immediately violating any of the higher resource thresholds. If $A_2 \neq \emptyset$, the autonomous node agent proceeds with the decision making process by evaluating each alternative in A_2.

In the case where $A_2 = \emptyset$, it attempts to redefine the set of alternatives as,

$$A_2 = \{p_{i'} \in A_2 | (g_n(p_i^{'}) + g_n(v_j^i) < H_n, \forall n) \ \wedge \ (S_{i'} = 2), \forall i^{'}\} \tag{6.5}$$

In this second attempt, the goal is to form the set of alternatives from the physical machines that have resources that are currently being used by one or more virtual machines on at least one of the resource dimensions, which are still residual due to the fact that the lower thresholds on all of the resource dimensions are violated. Similar

to the first step, if $A_2 \neq \emptyset$, autonomous node agent proceeds with the decision making process. However, if $A_2 = \emptyset$, the decision making process is by-passed through picking the first idle physical machine in the cloud with $S_i = 3$ and migrates the virtual machine to that location. If there is no idle machines left, the autonomous node agent does not continue with the decision making process, which also implies that SLO violations will be encountered.

In the cases where $A_2 \neq \emptyset$, the decision making process proceeds through the evaluation of each alternative in A_2. The set of criteria for decision type 2 can be defined as the total resources used on each physical machine in the alternative set, G_R. The evaluation should favour the physical machines that have smaller values over the elements of G_R. That is, the physical machines with more available resources are deemed more suitable. The reason behind this is to elect a new physical machine, so that, its resource usages on each dimension is below the higher threshold as much as possible.

For both decision types, let us define the relative importance weigth for each criterion. Let w_n be the relative weight of criterion $g_n(\cdot)$ of G_R. All weights are normalized such that $\sum_{n=1}^{N} w_n = 1$. These relative weights can be set statically by the cloud providers to reflect on their preference value systems. However, these relative importance weights might be reset dynamically based on the situation. IMPROMPTU introduces dynamic weight assignment and includes additional decision criteria.

Dynamic weights are used to reflect the importance of each resource dimension with respect to the strength of violations of higher thresholds on the resource dimension. That is, stronger violations on certain resource dimensions increase the corresponding criterion's weight during the evaluation process. The extended model uses dynamic weights only during decision type 1, since weight modulation is only used for responding to problematic situations. Modulation of weights can be done through defining a separate violation parameter y_n for each resource dimension as,

$$y_n = \begin{cases} \frac{g_n(p_i) - H_N}{1 - H_n} & \text{if } g_n(p_i) > H_n, \\ 0 & \text{if otherwise.} \end{cases} \tag{6.6}$$

The normalized relative importance weight with respect to the higher threshold vio-

lation on each resource dimension n is obtained as follows,

$$w_n \;=\; \frac{y_n}{\sum_{n=1}^{N} y_n} \tag{6.7}$$

where $\sum_{n=1}^{N} w_n = 1$. In certain cases, where there is a higher threshold violation on only one of the resource dimensions, the usage of dynamic weights change the decision type 1 to a single criterion decision process where virtual machines in A_1 are evaluated based on only one criterion. This is due to the fact that a criterion with weight of 0 has no influence on the decision.

In addition to the dynamic weights, the extended model uses an additional set of criteria, called the influence criteria, during the evaluation of alternatives in both decision types. Let us define the influence criteria as $G_I = \{g_m(\cdot) \in G_I \mid m = 1, 2, ..., M\}$. In decision type 1, these criteria are used to evaluate the influence of each virtual machine in A_1 on the threshold violations captured on the physical machines that they were hosted on. In order to reflect on the history of virtual machines, it is measured as the average influence of each virtual machine over time. The influence measures of a virtual machine needs to be captured for each resource dimension separately, since its general behaviour in changing requirements may not be the same on every resource dimension. Then, there is a bijective relation between G_R and G_I. Based on this, the influence of a virtual machine v_j^i on the threshold violations on a certain resource dimension n can be defined as,

$$g_m(v_j^i) = \frac{\sum_{t=0}^{T}(e_n^t \;\Delta t)}{T} \tag{6.8}$$

T is the total time that v_j^i has been serviced in the cloud, and e_n^t represents the share of impact that a virtual machine's change in resource usage—as encountered during the period $\Delta t = t - t^{'}$—had on the violation of the lower or higher thresholds on resource dimension n. Then, e_n^t can be obtained as,

$$e_n^t = \begin{cases} \frac{g_n^t(v_j^i) - g_n^{t^{'}}(v_j^i)}{g_n^t(v_j^i) - H_n} & \text{if } S_i = 1 \;\wedge\; g_n^t(v_j^i) - g_n^{t^{'}}(v_j^i) > 0, \\ \frac{g_n^{t^{'}}(v_j^i) - g_n^t(v_j^i)}{L_n - g_n^t(v_j^i)} & \text{if } S_i = 2 \;\wedge\; g_n^t(v_j^i) - g_n^{t^{'}}(v_j^i) < 0, \\ 0 & \text{if otherwise.} \end{cases} \tag{6.9}$$

depending on the type of violation that the change in the resource usages of v_j^i have

influenced between time $t^{'}$ and time t. In decision type 2, the physical machines in A_2 are also evaluated using the influence criteria. The influence measure of a physical machine for resource dimensions n is gathered from the influence measures of the virtual machines that it is currently hosting as $g_m(p_i) = \sum_{j=1}^{J} g_m(v_j^i)$.

When the influence criteria are considered, the evaluation process in decision type 1 should favour the virtual machines with higher influence measures. Whereas, the evaluation process in decision type 2 should favour the physical machines with lower influence measures. The reason behind this is to blend virtual machines with higher influence measures with the ones with lower influence measures in order to reduce further threshold violations, which in turn reduces the occurance of undesirable conditions.

6.3.3 Applying PROMETHEE

IMPROMPTU decision models are based on PROMETHEE [131], which is an outranking method. PROMETHEE has been selected for this application for to its practicality, simplicity and computationally light-weight nature. The PROMETHEE method is based on pairwise comparisons of alternatives based on the difference between evaluations along each criterion. The relative preference between alternatives is computed based on the evaluation difference. The realtive preferences are interpreted as fuzzy numbers between 0 and 1. In PROMETHEE, there is a generalized criterion associated with each evaluation criterion represented by the pair $(g_k(\cdot), P_k(a,b))$, where $P_k(a,b)$ is a preference function associated with a criterion $g_k(\cdot)$. PROMETHEE offers six predefined preference functions. We selected three of those preference functions based on the characteristics of the resource usage metrics explained. As a result, the criteria that represent load dependent resource dimensions, such as CPU and network bandwidth, are associated with a Gaussian preference function in order to reflect on the non-linear relation between utilization and stability of a service channel. Whereas, V-Shaped preference function is associated with resource dimensions, such as memory, that do not exhibit those characteristics. Each influence criterion is associated with Usual preference function, where the strength of preference is represented by a Boolean value.

Using the generalized criteria, autonomous node agents calculate aggregated preference indices to express with what degree an alternative is preferred to another. Let us denote the set of alternatives in both decision types in a general form as A—instead

of A_1 and A_2, and the set of all criteria as G, where for decision type 1 $G = G_R$ and for decision type to $G = G_R \cup G_E$. Then, for each $a, b \in A^2$, the aggregate preference indices are defined as:

$$\pi(a,b) \quad = \quad \sum_{k=1}^{|G|} P_k(a,b) w_k \tag{6.10}$$

Using these indices, an autonomous node agent ranks the alternatives based on their strength and weaknesses. For each alternative a a positive outranking flow—representing its strength—and a negative outranking flow—representing its weakness—are computed as follows:

$$\begin{aligned} \phi^+(a) &= \tfrac{1}{|A|-1} \textstyle\sum_{x \in A} \pi(a,x) \\ \phi^-(a) &= \tfrac{1}{|A|-1} \textstyle\sum_{x \in A} \pi(x,a) \end{aligned} \tag{6.11}$$

PROEMTHEE I uses these flows to generate a partial preorder of all the alternatives, where PROEMTHEE II introduces the net outranking flow to generate a total preorder of the alternatives. IMPROMPTU uses PROMETHEE II where the net outranking flow is computed as follows:

$$\phi(a) = \phi^+(a) - \phi^-(a) \tag{6.12}$$

The alternatives are ranked based on their net outranking flows. As a result, it returns the alternative with the highest net outranking flow, which is then used as the most suitable course of action.

6.4 System Architecture

IMPROMPTU is designed as a distributed network of autonomous node agents. Each autonomous node agent manages resource consolidation locally on the physical machine that it is coupled with. Resource consolidation management is carried out by certain modules that encapsulate distinct activities. The system model is described in abstract functional and operational terms based on the Abstract State Machine (ASM) paradigm [28] and the CoreASM open source tool environment [72] for modeling dynamic properties of distributed systems. ASMs are known for their versatility in modeling of complex distributed systems with an orientation towards practical applications [29]. The description of the model in ASM reveals the underlying design

concepts and provides a concise yet precise blueprint for reasoning about the key system properties.

The primary modules of each autonomous node agent are (1) Monitoring, (2) Selection, (3) Placement, and (4) Migration Management. The following ASM program abstractly captures the behavior of an autonomous node agent in terms of its modules.

Autonomous Node Agent Program

Autonomous Node Agent ≡
  Monitoring
  Selection
  Placement
  MigrationManagement

Note that according to ASM semantics, the composition of the four activities in this program denotes a parallel execution of these activities. The general view of the modules in an autonomous node agent, and its communication with the rest of the network is depicted in Figure 6.1.

In IMPROMPTU, a virtual machine goes through different states throughout its life-time in the cloud under resource consolidation management. The states include *steady*, *toBeMigrated*, *scheduledForMigration*, and *beingMigrated*. These states and transitions are depicted in Figure 6.2. State *steady* denotes the cases where a virtual machine is being hosted on a physical machine without being involved in any explicit actions. In *toBeMigrated* state, a virtual machine is chosen for migration and waits for a suitable physical machine to be found by the autonomous node agent. State *scheduledForMigration* is the state in which a virtual machine is assigned a new host, and is waiting to be migrated. Finally, state *beingMigrated* denotes the state where a virtual machine is being transmitted to its new location. Each of these states are associated with the activities carried on by the four modules.

Monitoring module is responsible for observing the local status, and triggering reactions as described in Section 6.3.1. It triggers the reactions in terms of communicating the necessary type of reaction to the Selection module with the appropriate signal. Based on the required reaction, it signals the Selection module with either a Tune or an Evacuate signal. Once the triggering is performed, Monitoring module proceeds with its responsibility of observing the local status.The general behaviour of the

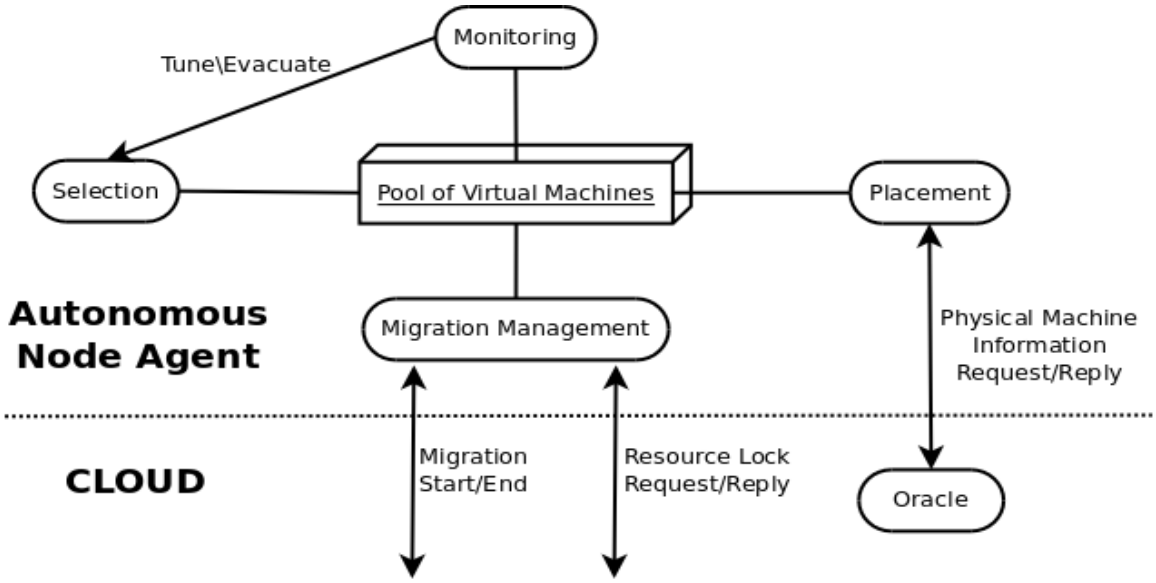


Figure 6.1: The modules of an autonomous node agent.

Monitoring Module is outlined in the following ASM program, where *isOverUtilized* and *isUnderUtilized* are definitions of a physical machines current state as gathered from Equation 6.2. Accordingly, the necessary reaction is triggered by sending the necessary signals to the Selection module, which are defined in Equation 6.3.

Monitoring Module

Monitoring ≡
  if *isOverUtilized* **then**
    signal Selection **with** *TuneSignal*
  if *isUnderUtilized* **then**
    signal Selection **with** *EvacuateSignal*

Upon receiving a signal, the Selection module initiates one of the two reactions based on the type of signal. If a Tune signal is received, the Selection module proceeds with the decision type 1 and ranks the virtual machines from the most suitable one to be migrated to the least suitable one following one of the MCDA models—which is a predetermined before operation—defined in Section 6.3.2 using PROMTHEE. The states of virtual machines with the highest rankings are changed from *steady* to *toBeMigrated* one at a time until the undesirable conditions are potentially resolved. Note here that the undesirable conditions are not deemed resolved until the selected

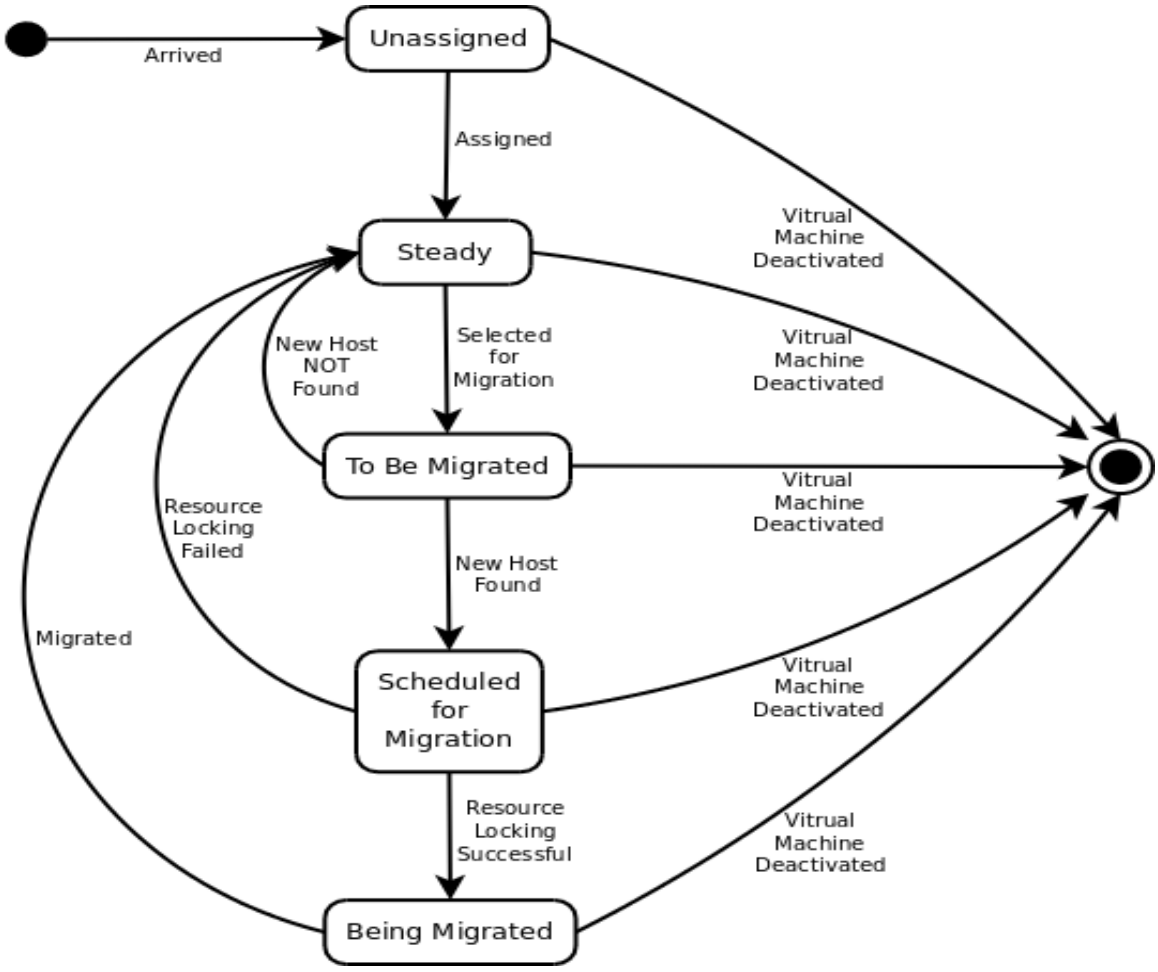


Figure 6.2: The states assigned to a virtual machine under the management of an autonomous node agent.

virtual machines are actually migrated away. On the other hand, if an Evacuate signal is received, every virtual machine's state is updated as *toBeMigrated*. The following ASM program abstractly captures this behaviour.

Selection Module

Selection ≡
  onsignal s **of** *TuneSignal* **do**
    let *ranked* = *rankWithPROMETHEE(virtualMachines)* **in**
      updateUntilResolved(ranked)
  onsignal s **of** *EvacuateSignal* **do**
    updateAll

Placement module's main responsibility is to select new locations in the cloud for the virtual machines that are in the *toBeMigrated* state. If a virtual machine is marked as *toBeMigrated*, Placement module proceeds with decision type 2 to find a suitable location for the given virtual machine. During the process, it needs to have a knowledge of what physical machines are available in the data center. This knowledge is acquired through querying an outside entity called an Oracle. In the proposed design, we assume that the Oracle holds a complete knowledge about the usages on all resource dimensions of the physical machines in the cloud. In the context of cloud computing, there is already a few monitoring mechanisms [133]. If the Placement module successfully finds a new location, then the status of the given virtual machine is updated from *toBeMigrated* to *scheduledForMigration*. If a new host was failed to be found the status of the virtual machine is updated backed to *steady*. This behaviour is outlined in the following ASM program.

Placement

Placement ≡
  foreach *vm* **with** *vm.isState(ToBeMigrated)*
    let *pm* = *selectWithPROMETHEE(getAvailablePMs())* **in**
      if *pm* **then**
        vm.updateStatus(scheduledForMigration, pm)
      else
        vm.updateStatus(steady, none)

Migration Management module's primary responsibility is to coordinate the general migration process. It deals with both inbound and outbound migrations by making sure that the virtual machine in migration is to receive the required resources. In this sense, it is the module that communicates with the other autonomous node agents. During the outbound migrations, it looks for local virtual machines with in state *scheduledForMigration*. Upon finding such virtual machines, it sends a resource lock request to the physical machine that was selected by the placement module. Upon receiving an acceptance signal, it immediately starts migrating the virtual machine to its new location. If a rejection signal is received, it updates the state of the virtual machine back to *steady*. During inbound migrations, it replies to lock requests with either an accept or a reject signal. An accept signal is sent only when the resources

requested are still locally available, whereas a reject signal is sent otherwise. A virtual machine's state is updated to *beingMigrated* right after resource locks are acquired, and remains so until migration is completed. Upon completion, the state is changed back to *steady*. The behaviour of Migration Management module is abstractly captured in the following ASM program.

Migration Management Module

Migration Management ≡
  foreach *vm* **with** *vm.isState(scheduledForMigration)*
    let *pm* = *vm.getDestination()* **in**
      signalLockRequest(pm)
  onsignal s **of** *LockRequest* **do**
    let *vm* = *lockRequest.vmInfo* **in**
      if *isOverUtilizedWith(vm)* **then**
        replyWithReject(s.source)
      else
        replyWithAccept(s.source)
  onsignal s **of** *LockReply* **do**
    let *vm* = *lockRequest.vmInfo* **in**
      if *s.isAccept()* **then**
        migrate(vm)
      else
        vm.updateStatus(steady, none)

6.5 Empirical Results

In this section, we discuss the performance of IMPROMPTU by applying it to the problem domain, observing its behaviour, and comparing its performance to other possible approaches. The tests are run on a platform that is designed and implemented to simulate the time-variant nature of clouds as realistically as possible.

In order to clearly represent the assessment and validation process, we first describe how the simulation environment works and the general settings in which the different resource consolidation managers are tested in Section 6.5.1. Finally in Section 6.5.2, we provide a comprehensive performance analysis of IMPROMPTU with

respect to how it compares to other resource consolidation managers and how it addresses the problem domain.

6.5.1 Simulation Settings

Using the simulation platform explained in Section 5.1, each different consolidation managers are given control in isolated yet identical runs. Each scenario is run for 2000 steps in a data center of 5000 virtual machines and a virtually unlimited number of physical machines. Each simulation run keeps the set of virtual machines constant without adding new ones or removing existing ones throughout the run. The number of physical machines that is necessary to host the set of virtual machines is solely defined by the resource consolidation manager in use. The resource dimensions that are available in the data center are defined as CPU, memory and network bandwidth. The limit for SLO violations are defined separately for each resource dimension. Each physical machine to be used in the data center are set to have a single processing unit and one network interface. In addition, each simulation run assumes that the live migrations are carried out on an isolated network—much like a control line—that does not affect the regular communications of the virtual machines. The resource limits are defined as 60% for CPU and network bandwidth based on knee utilizations and service channel counts [138] due to their load dependent nature. Since memory does not necessarily exhibit the same characteristics it is assigned a limit of 90% such that 10% is reserved to prevent starvation. SLO violation measurements are done based on the values set for each limit. Accordingly, SLO violations occur when a physical machine's thresholds are exceeded. We have implemented five different resource consolidation managers, the behaviour of which is the major focus of the simulation analyses: Centralized Static First Fit (CSFF), Centralized Dynamic First Fit (CDFF), Distributed Dynamic Random (DDR), IMPROMPTU Model 1 (IMP-1), and IMPROMPTU Model 2 (IMP-2). In CSFF, the resources are consolidated only once at the beginning of a simulation run using a simple packing algorithm called First Fit. CDFF is a dynamic resource consolidation manager that is derived from CSFF. It uses the same packing algorithm, yet, it manages resource consolidation dynamically at each step throughout the simulation run. Both CSFF and CDFF are centralized methods where the resource consolidation is carried out by a global arbiter. DDR is a distributed resource consolidation manager that follows the reactive model described in Section 6.3.1. However, in DDR, the decisions are made by randomly

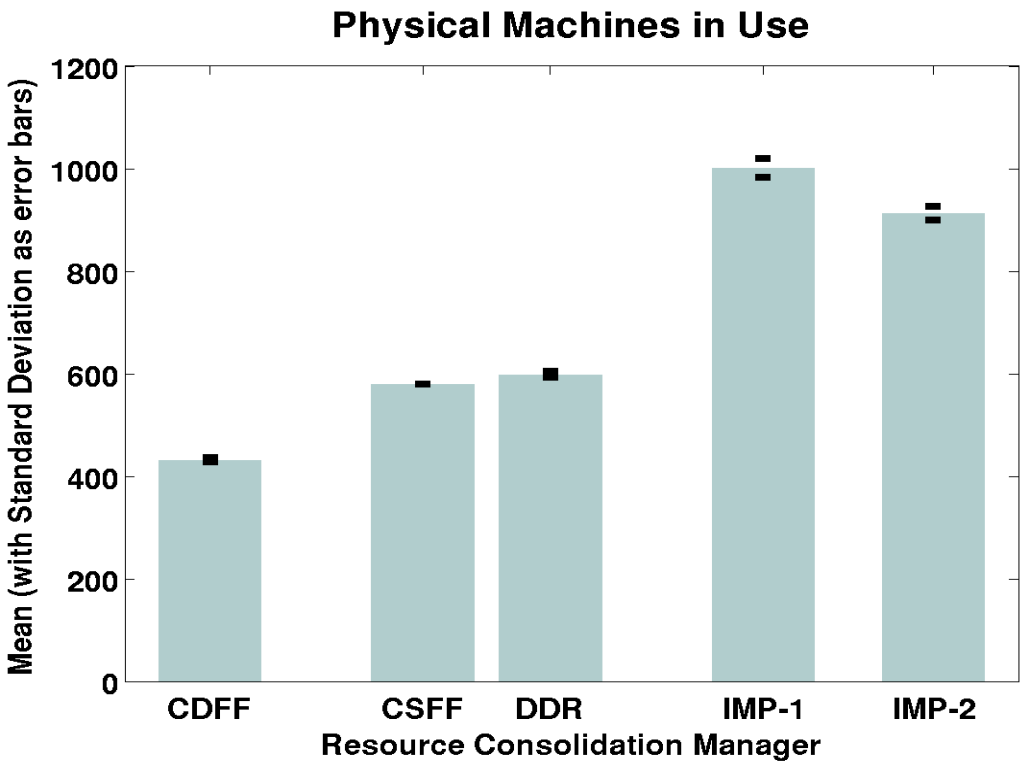


Figure 6.3: Physical machine usage per simulation step by different resource consolidation managers.

selecting from the set of alternatives instead of applying any formal strategy. IMP-1 and IMP-2 represent the two models—as described in Section 6.3.2—based on MCDA through reactive and distributed control. IMP-1 is the implementation for the basic model, whereas, IMP-2 is the latest implementation of IMPROMPTU which uses the extended model with dynamic weights and influence criteria.

Finally, each simulation run is conducted in a homogeneous data center where the total amounts of resources and their corresponding limits on each physical machine are identical. It is important to note here that, although these are the default settings to be considered for the validation and assessment process throughout Section 6.5.2, the simulation platform can easily support any properly defined scenario where the above mentioned simulation parameters can be changed individually or in bundles for any set of entities in the platform.

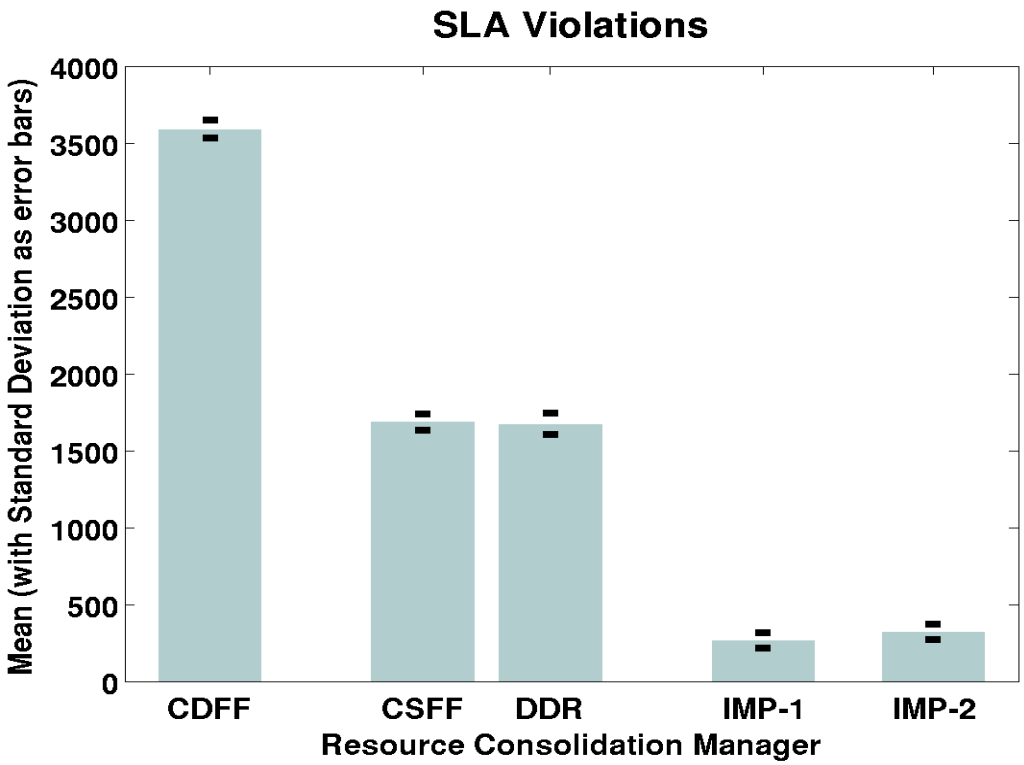


Figure 6.4: SLO violation per simulation step caused by different resource consolidation managers.

6.5.2 Simulation Results

The assessment and validation process is carried out based on a set of primary performance metrics. A widely used metric is the total number of physical machines that are used. This metric defines the efficiency of a resource consolidation manager purely from a provider's point of view. If a resource consolidation manager uses fewer physical machines than other managers under the same conditions, it can be considered as the more efficient one with respect to criteria such as hardware deployment, maintenance, power consumption, etc. In the literature, physical machine usage tends to be used as the only metric to define the efficiency of resource consolidation management.

However, in clouds client-centric metrics also become important, the most obvious being the total number of SLO violations encountered. Naturally, resource consolidation managers that limit total SLO violations can be viewed as more efficient. Furthermore, as opposed to the total number of physical machines, this metric represents the perspectives of both the clients and the providers.

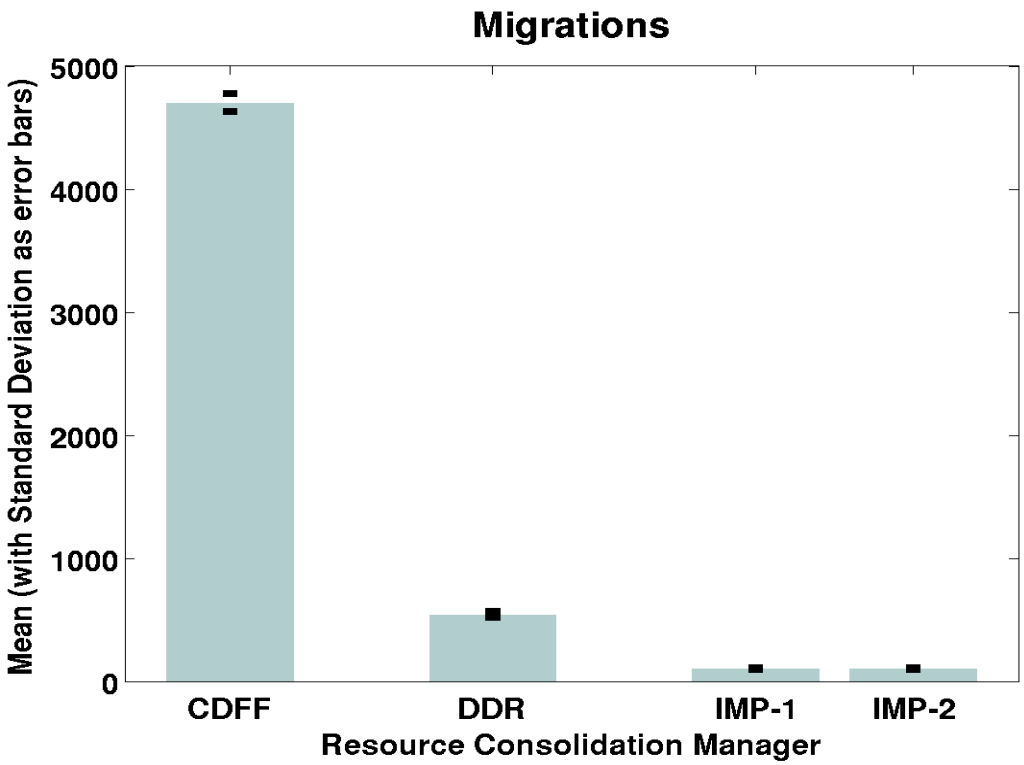


Figure 6.5: Migrations per simulation step performed by different resource consolidaiton managers.

Table 6.1: Statistics of Physical Machine Usage, SLO Violations and Migrations of Different Resource Consolidation Managers on a per Simulation Step Basis

	# of Physical Machines		# of SLO Violations		# of Migrations	
	Mean	STD	Mean	STD	Mean	STD
CDFF	431.6	8.5	3588.7	119.7	4700.6	146.9
CSFF	579	0	1683.9	106.5	0	0
DDR	598.1	10.8	1672.1	137.1	540.1	49.9
IMP-1	1001	35.6	264.2	95.4	99.8	15.3
IMP-2	912.4	26.5	319.3	101.9	101.7	15.4

However, metrics that represent the provider's and clients' perspectives are not necessarily enough in defining the efficiency of a resource consolidation manager. In comparing resource consolidation managers, it is also important to assess the burden placed on the data center. One metric that can be used in this assessment is the total number of migrations per management cycle, which is a measure of how many virtual

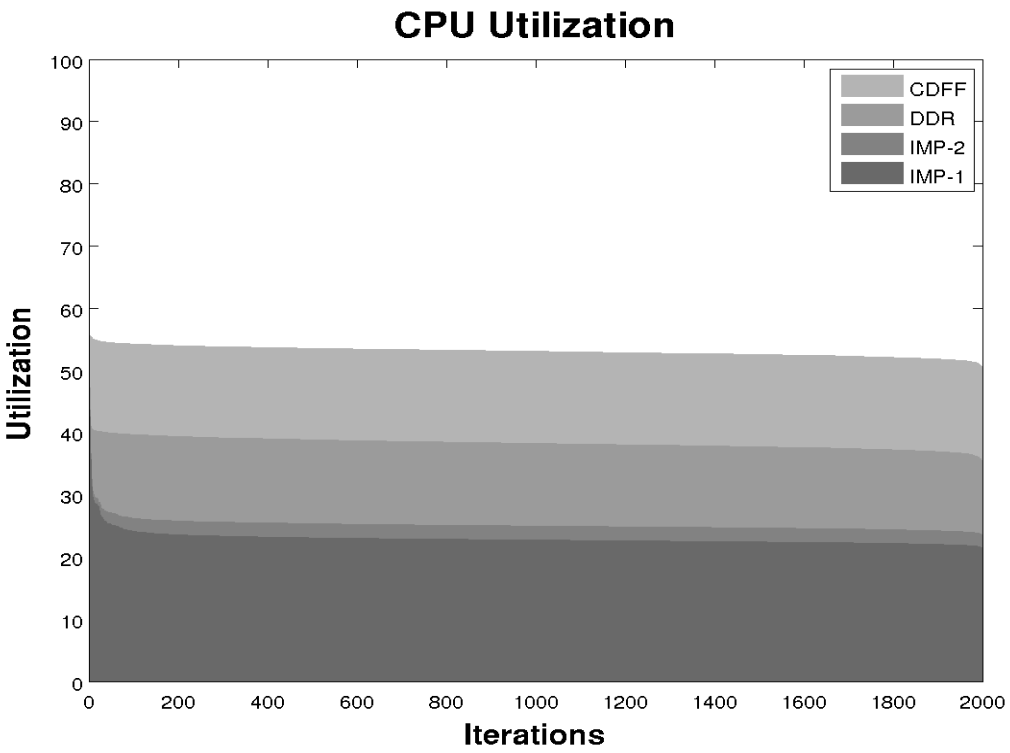


Figure 6.6: Overall CPU utilization in the data center at each step using different resource consolidation managers.

machines need to be migrated to new locations in order to carry out the solutions computed by a resource consolidation manager. In the context of efficiency, it can be used as a measure of both feasibility and scalability. It is a measure of feasibility since there are certain physical limitations as to how many virtual machines can be migrated to new locations in the data center simultaneously. It is a measure of scalability since those limitations become severe as the data center is scaled up with more virtual machines and physical machines. As a result, resource consolidation managers that require a smaller number of migrations are deemed more efficient.

Figures 6.3, 6.4 and 6.5 show the results of simulation runs employing different resource consolidation managers with respect to the three primary metrics as described above—Table 6.1 shows the values of the data points in each of these figures. Each figure provides a mean value per simulation step and a standard deviation around the depicted means for the different metrics under the control of different resource consolidation managers. When the total number of physical machines is considered CDFF is the most efficient method. However, the results regarding the total number

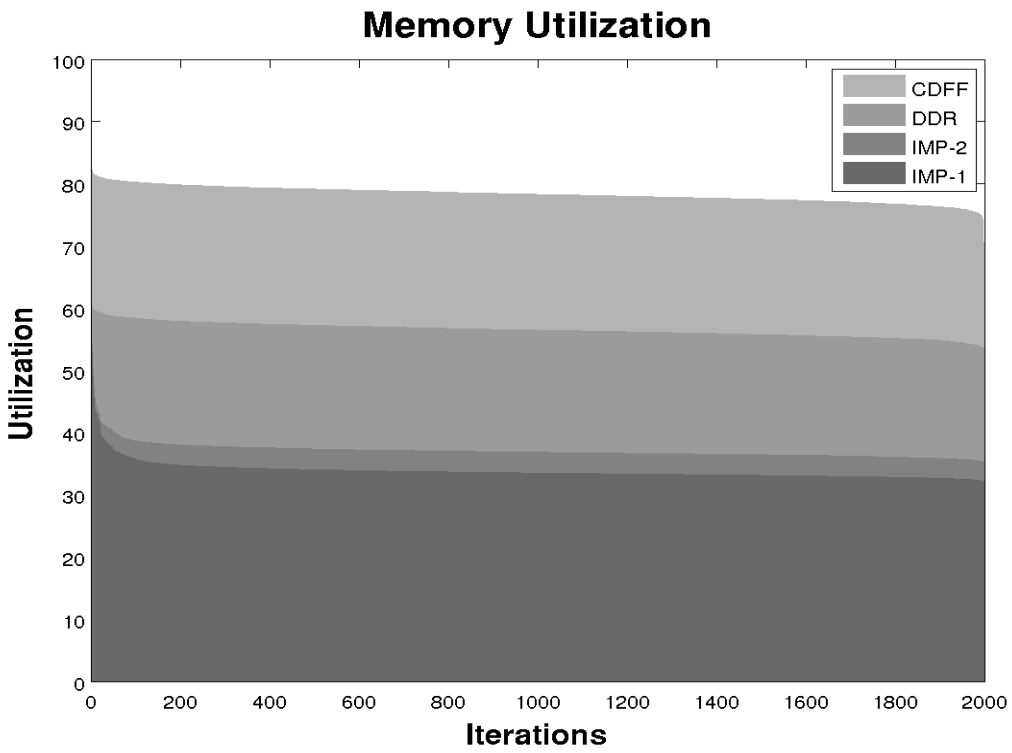


Figure 6.7: Overall Memory utilization in the data center at each step using different resource consolidation managers.

of SLO violations show that the apparent benefits of CDFF diminish since the SLOs of a large portion of the virtual machines are violated. CSFF, which does not perform any dynamic management, and is more efficient than CDFF when the total number of SLO violations are taken into account. This result is due to CDFF's main objective of trying to pack the virtual machines as tightly as possible based on the First Fit algorithm; a general problem for any resource consolidation manager that aims for high resource utilizations. The results illustrated in Figures 6.3 and 6.4 show that there is a direct relation between high resource utilization and high numbers of SLO violations.

On the other hand the distributed resource consolidation managers seem to produce results that are more efficient in taking both metrics into account. For instance, when DDR is given the control, it causes less than half as much SLO violations as CDFF while using slightly more than 150 physical machines on average. Figures 6.3 and 6.4 show approximate results for CSFF and DDR, however it is important to note that the actual data samples are not statistically identical. On the other hand, the

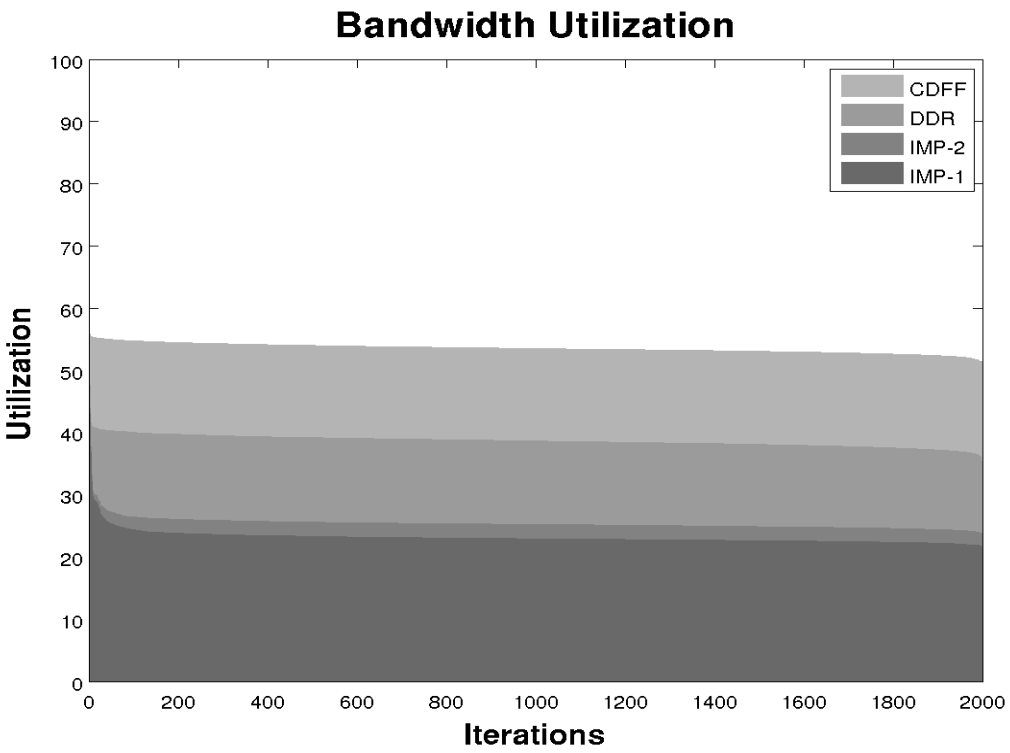
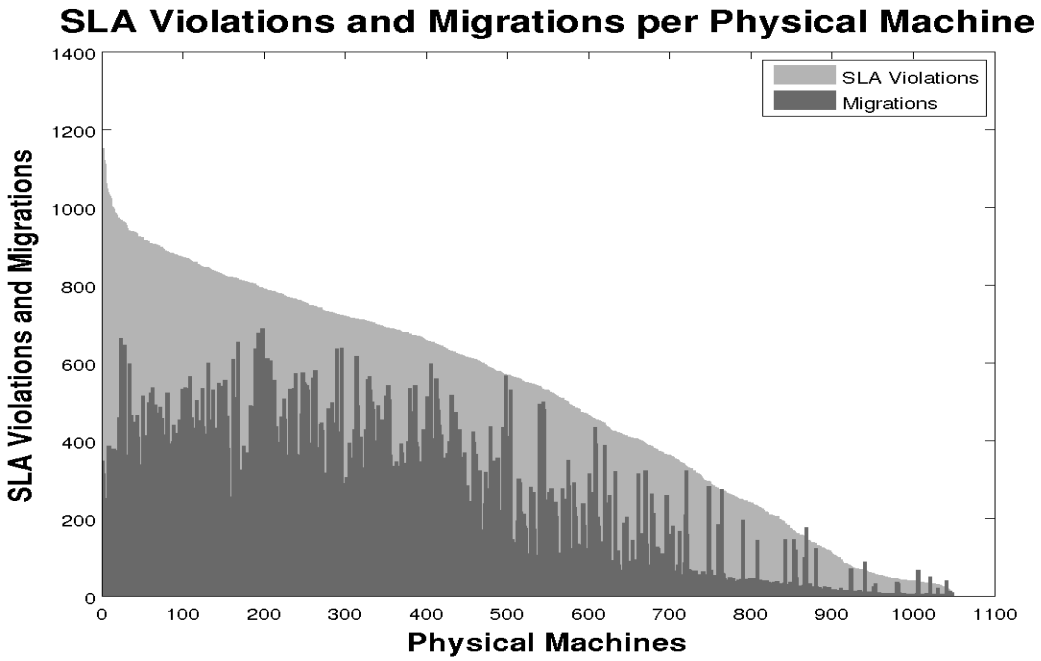


Figure 6.8: Overall Bandwidth utilization in the data center at each step using different resource consolidation managers.

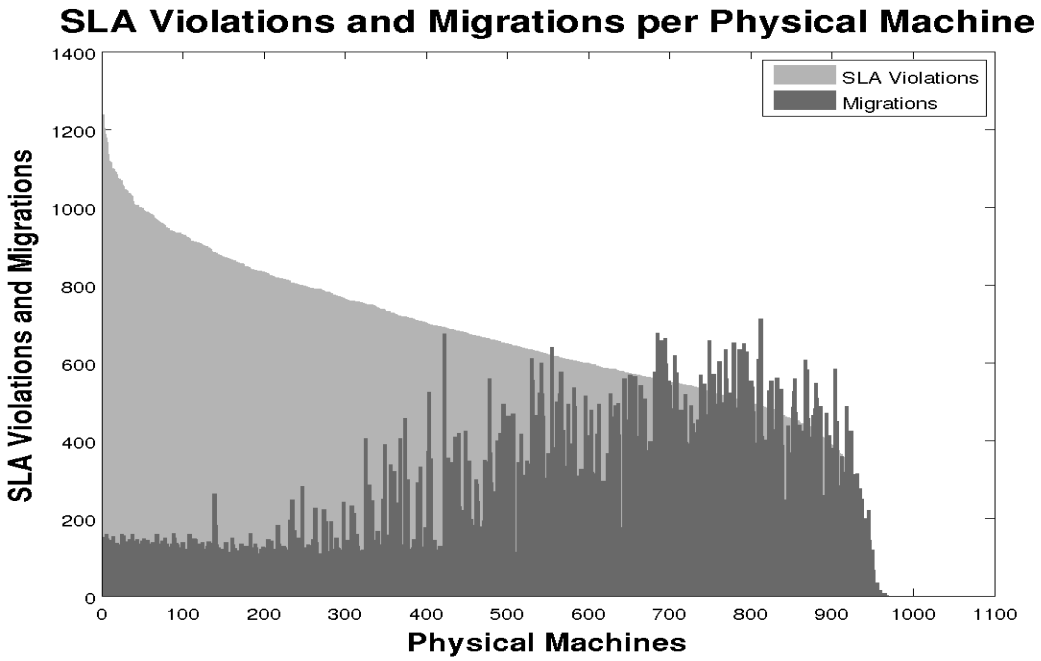
two IMPROMPTU models reduce the per-step average of SLO violations to roughly a sixteenth of the total number of virtual machines in the data center while using more physical machines in the process.

Considering the metric regarding the total number of migrations extends the assessment process further. In Figure 6.5, it can be clearly seen that CDFF's aim at tight packing causes a vast majority of the virtual machines in the data center to be migrated every time resource consolidation is carried out. Note that CSFF is not included in the figure since it does not do any migrations due to its one-time resource consolidation approach. On the other hand, DDR needs substantially less migrations to manage the resources in the data center. Moreover, IMP-1 and IMP-2 further improve on DDR by reducing the number of migrations to the marginal level of roughly a fiftieth of all the virtual machines.

When the information in Figures 6.3, 6.4 and 6.5 is taken into account, it also sheds light on the way that the centralized resource consolidation managers are designed to be used. In its most general form as represented in the previous studies [199, 48, 56],



(a) IMP-1



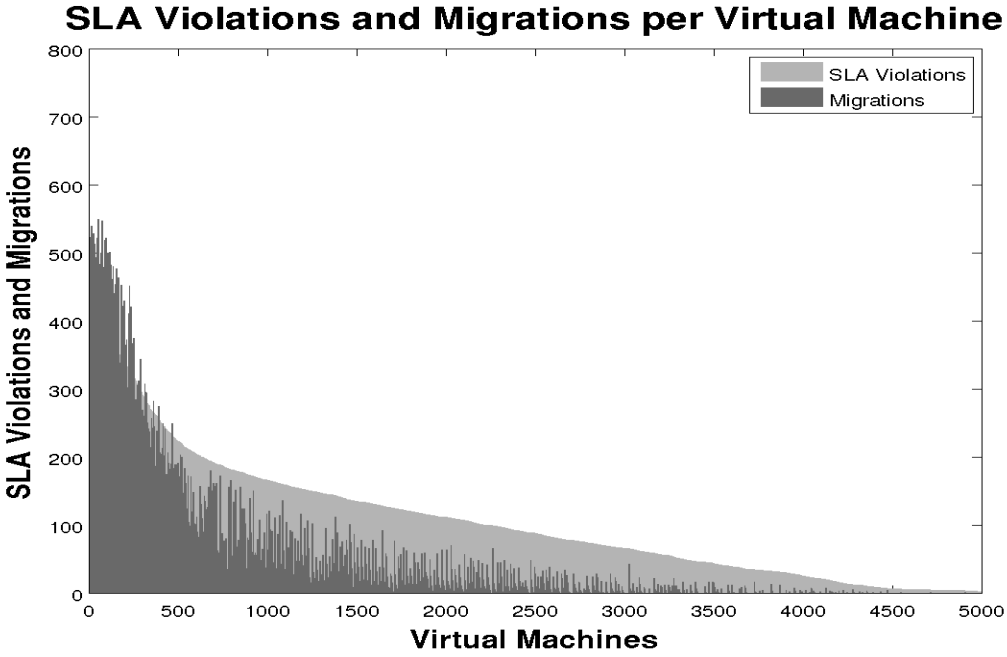
(b) IMP-2

Figure 6.9: Total SLO violations and migrations throughout a simulation run as experienced on each physical machine in the data center.

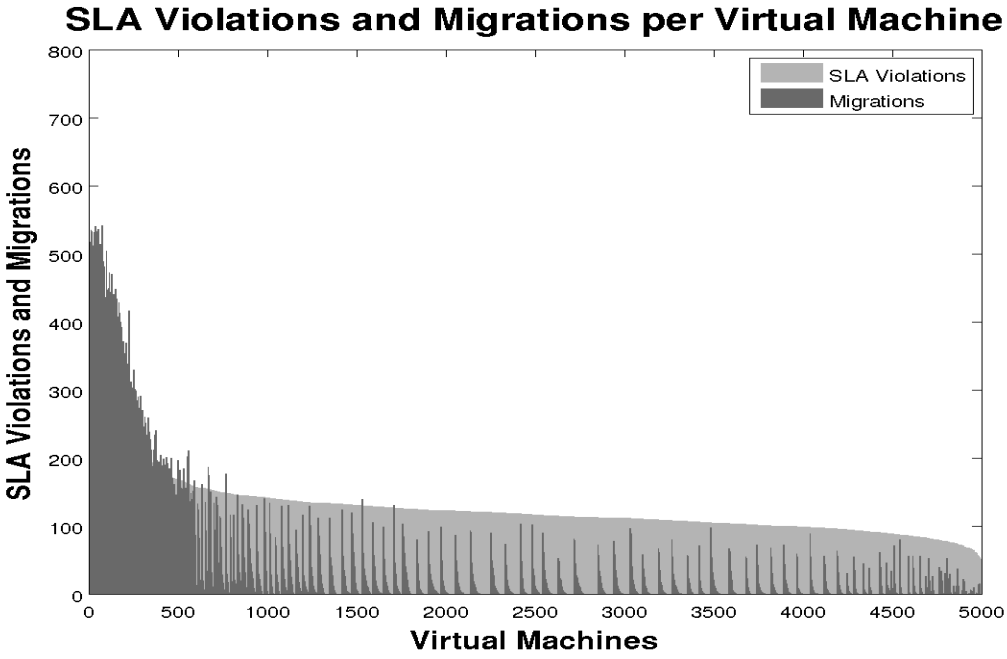
the centralized resource consolidation management is carried out periodically at the end of fixed time intervals. These periods are assigned relatively high values, often in the order of minutes. Since the centralized approaches generally perform resource consolidation management taking into account all of the physical machines and virtual machines in the data center, the solutions they compute require large numbers of migrations that cannot be undertaken frequently. It is obvious that waiting for longer periods of time between the resource consolidation processes reduces the extra burden on the data center. However, in turn, it also reduces responsiveness. The high number of migrations is addressed as a problem in certain previous solutions that adapt centralized approaches, such as Entropy [94], where reducing the number of migrations is defined as one of the primary goals. Yet the fundamental problem remains where SLO violations increase as packing densities increase.

Distributed approaches such as DDR, IMP-1 and IMP-2 are inherently resilient to such problems. They are highly responsive since the need for resource consolidation is dealt with immediately through localized solutions that are naturally fast—in an order of seconds—due to the limited size of the solution space. In addition, their reactive behaviour—which ensures that resource consolidation is carried out only when absolutely necessary—also limits the number of migrations per resource consolidation process. Figures 6.6, 6.7 and 6.8 depict the overall utilization on each resource dimension in the data center. These figures provide further information underlining why the sharp difference in the number of migrations and SLO violations exist between CDFF and the distributed approaches such as DDR, IMP-1 and IMP-2. The main conclusion is that in the distributed approaches, the resources on all dimensions are delivered to the virtual machines in a much looser manner, whereas, CDFF produces near-threshold usage levels throughout the data center that are far more likely to cause SLO violations.

Distributed approaches can be investigated further by considering DDR as the most basic approach that can be used as a benchmark for IMP-1 and IMP-2 due to its simplistic and rigid nature. IMP-1 enhances this simple behaviour through the usage of MCDA in the decision making process. The results observed in Figure 6.4 and 6.5 show that IMP-1 is more efficient than DDR with respect to the number of SLO violations and the number of migrations. The improvement is achieved with the cost of using roughly a third more physical machines than DDR, which may raise questions as to whether it is MCDA or the usage of extra physical machines that improves the efficiency with respect to SLO violation and migration counts. However, the results



(a) IMP-1



(b) IMP-2

Figure 6.10: Total SLO violations and migrations throughout a simulation run as experienced by each virtual machine in the data center.

of using IMP-2 show that the improvement on SLO violations and migrations are not only due to using more physical machines. As a matter of fact, the results in Figure 6.3 show that IMP-2 reduces the number of physical machines used by IMP-1 by a tenth while holding the SLO violation and migration counts roughly the same as in IMP-1 through the usage of dynamic weights and further granulation of MCDA process by adding the influence criteria.

The reasons behind this improvement can be seen in Figures 6.9 and 6.10 where both the total number of SLO violations and the total number of migrations throughout a simulation run are investigated on a per physical machine and per virtual machine basis respectively. It is important to note here that the high values of SLO violations and migrations depicted in Figure 6.9b is simply due to IMP-2's higher utilization of resources. Figure 6.9a shows that most of the migrations are made by the autonomous node agents of the physical machines that have the highest total count of SLO violations, which are naturally more utilized than the rest.

Contrarily, Figure 6.9b shows that IMP-2 produces the majority of migrations on the under-utilized physical machines in order to reduce the number of physical machines in use. That is, while IMP-1 performs resource consolidation mostly as a reaction to over-utilization, IMP-2 performs resource consolidation mostly as a reaction to under-utilization. This shows that the multiple criteria decision analysis model used in IMP-2 is more accurate in terms of the decisions that are made. For this very reason, IMP-2 manages the same number of virtual machines with less physical machines than IMP-1, while encountering approximately the same amount of SLO violations and performing approximately the same number of migrations. That is, data center is managed more efficiently without increasing the effort or the burden of migrating virtual machines. Moreover, Figure 6.10 reveals more information regarding how the decision models in IMP-1 and IMP-2 differ. The main difference between the two models can be seen in how the migrations are distributed among the virtual machines. The general tendency seen in Figure 6.10a show that the migration count of virtual machines decrease with their total count of SLO violations under IMP-1's management, whereas the spikes in Figure 6.10b indicate that the influence criteria used by IMP-2 remove this tendency. This means that in IMP-2, every virtual machine's count of migration change due to its influence on the encountered SLO violations.

6.6 Summary

In this chapter, we have introduced IMPROMPTU, distributed resource consolidation manager for clouds dsigned based on the proposed methodology. The proposed system has been investigated in detail in terms of its reactive behaviour, MCDA model, distributed architecture and performance assessment through simulated runs. Simulation results show that a reactive and distributed approach to autonomous resource consolidation management can be a solid alternative to the existing centralized architectures. Furthermore, it is shown that resource consolidation management can be substantially improved through an MCDA model and proper extensions to it. The proposed system achieves scalability through a distribution of responsibility that reduces the complexity of computing new resource distributions by localizing the management process. In addition, it produces feasible solutions with respect to the observed marginal number of migrations that are performed in order to apply new distributions of computational resources. Moreover, IMPROMPTU triggers less SLO violations by smoothly distributing the overall resource utilization over more physical machines in a cloud. As a reactive and distributed resource consolidation manager, IMPROMPTU proves to be a strong candidate in the domain.

In the next stages of this work, our goal is to further extend the MCDA model by including new criteria to reflect on the overhead of migrations. We are currently working on modeling the overhead of migrations as a new criteria in terms of virtual machine size and virtual machine activity. It is important to note here that although PROMETHEE is the default MCDA method used in the current implementation of IMPROMPTU, other MCDA methods will be applied in the near future. Accordingly, we plan to produce a comprehensive study on the effects of using different MCDA methods on the quality of decisions that are made during the management cycles. Finally, the observations represented in the work of Grier [91]—particularly that power consumption increases quadractically with CPU utilization—underlines the critical issue of energy conservation within resource management and consolidation. Hence, future opportunities exist to extend IMPROMPTU to enable a more carbon footprint friendly IaaS layer. Our future plans also involve investigating this possibility further.

Chapter 7

Simulation Case Study 2: Coordinated Self-Management

Routing in MANETs has proven to be a challenging problem. As a result, a substantial number of routing algorithms were proposed over the past two decades. However, studies on comparative assessment of the existing MANET routing algorithms have revealed that although certain algorithms perform better than the others under certain conditions, there is no universal method that dominates all of the rest under every condition [196, 150, 41, 57]. This finding has motivated a stream of research that views routing in MANETs as a self-adaptation problem [25]. The main idea is to design an autonomous network where the nodes can collectively switch from one routing algorithm to another in real-time as the network conditions change.

In this chapter, we propose a new solution to the problem of adaptive routing in small scale MANETs based on the principles outlined by the proposed methodology within the context of coordinated self-management. Adaptively switching routing protocols based on the MANET's state is one method of overcoming per-protocol limitations. This solution introduces the additional problem that all of the nodes with the MANET must jointly decide which routing protocol to switch without the benefit of assigning this task to a centralized ruling authority. Accordingly, a mechanism is required that allows for collaborative decision making in a purely distributed environment which is known to be an open problem [78, 82, 83, 61]. This chapter focuses on this problem and shows that MCDA can be used as the basis for such a distributed decision systems. The proposed solution focuses on leveraging existing routing algorithms. This approach is based on switching between routing algorithms in real-time

with the purpose of achieving the best routing performance under changing network conditions. All the nodes of the network use the same routing algorithms that are considered as the set of alternatives in the previous research, which are OLSR [51], DSR [103], and AODV [149]. Also note that the criteria and the conditions to compare the different routing algorithms are based on former research [83], and are not in the scope of this work. That is, this study focuses on how the adaptation is carried out in the proposed system rather than the validation of why it is carried out; the reasons for which are clearly outlined in the former studies.

In the proposed solution, each node participates during the selection of a new routing algorithm through observing the conditions, making local decisions, and either leading the switching task as a coordinator or cooperating as a cohort during the course of transition. The contribution of this work is three-fold: (1) Nodes use a MCDA method called PROMETHEE [131] to make local decisions, (2) The final decision is evaluated as an aggregation of all the local decisions in the MANET through weighted voting, and (3) The network-wide decision process is carried out through a hybrid method that aims to benefit from the advantages of both centralized and purely distributed methods, and limit their disadvantages. Although the primary focus of this chapter is adaptive routing in small MANETs, the application of the proposed solution can be extended to medium and large scale settings by incorporating strategies that divide the network into smaller fragments.

The analyses of the simulation results underline that the proposed approach leads to stable configurations with minimal interruptions. Furthermore, the completion of global adaptation cycles are performed an order of magnitude faster in comparison to the results outlined in the previous work. These results indicate that the proposed solution is a promising alternative to the existing solutions, and forms a strong basis for further research on larger scale MANETs.

The rest of this chapter is organized as follows. In Section 7.1, we provide a more detailed description of the problem of switching between routing algorithms in real-time. Section 7.2 outlines the former research in the domain that adopts a similar approach to the one outlined in this work, and underline the important differences. Section 7.3 provides the detailed overall design of our approach. In Section 7.4, we provide simulation results that characterize the proposed approach based on the data extracted from simulation runs. Finally, Section 7.5 summarizes our conclusions and outlines our future directions.

7.1 Problem Description

In the self-adaptive approach to the problem of routing in MANETs, the routing algorithm used by the nodes in the network is considered as a parameter that can be configured/re-configured in real-time. Each configuration proceeds in two steps: (1) undertaking a decision making process to choose the most suitable routing algorithm from a set of predefined alternatives, and (2) network-wide adaptation of the chosen alternative.

The first step is carried out in terms of continuously observing the environment, reaching a network-wide situation awareness of the conditions, and deciding which routing algorithm is the most suitable one for the observed state of the network. The decision making process is based on a set of criteria that can be used to evaluate the suitability of routing algorithms given the observed conditions. Such criteria are defined in the literature as network size, mobility and traffic [150, 41, 57, 83, 82]. Once the decision is made, the second step consists of setting the value of the routing algorithm parameter to the new decision.

There are two general ways of undertaking these tasks in a MANET [61]. The first one is a centralized solution where the responsibility of control is assigned to a leader that is elected by the nodes as the most knowledgeable node in the network. In this approach, the elected leader decides on behalf of the entire network based on its local view of the network conditions, and communicates its decision to the rest of the network in form of a configuration command. The second way focuses on a distributed approach where each node in the network generates its own local view of the conditions as either a local decision or raw observations, and shares it with the rest of the network. After the articulation, it is assumed that each node in the MANET has full knowledge of the others' views. In the final stage of the adaptation, nodes use these complete views to elect a routing algorithm as the final configuration, which is the result of a consensus.

However, both of these approaches have a number of shortcomings that may cause serious problems in practical applications. In highly dynamic settings like MANETs, the centralized approach may suffer from the necessity to continuously elect new leaders due to the unpredictably mobile nature of the network. Briefly, this is due to the fact that a node elected to be the leader at one point in time is not guaranteed to be the most suitable—or even sufficiently suitable—for leading the configuration at a later instant. In addition, in the absence of alternatives, a node may be elected

as a leader even if its knowledge is not sufficient enough to control network-wide configurations—since a leader must be elected nonetheless. The distributed approach also has some known weaknesses. First of all, this approach requires that the nodes have an identical situation awareness of the network once the articulation of information is complete. Unfortunately, it is known that reaching such a network-wide knowledge that will drive consensus is impossible even if only one node in the network fails to communicate or fails to function correctly [78]. If any failures occur, the nodes in the network will configure to different routing algorithms, which is not tolerable within the context of configuring shared parameters. Furthermore, even if it is assumed that the nodes and the communication channels are flawless at all times, the message complexity is much higher than the centralized approach since the view of each node needs to be articulated to the rest of the network.

Regardless of the approach, one of the most important constraints is the time-taken to configure the routing algorithm network-wide. In highly dynamic settings like MANETs, the configuration needs to be made within a reasonable amount of time to be able to adapt with the changes as they occur. The configurations that take long to perform may result in stale reactions that have no positive—or even negative—effects on the routing performance. Moreover, in certain cases, trying to make a network-wide decision may not be reasonable or even possible due to physical limitations. For instance, if a MANET grows into a size that network-wide articulation of individual observations is not feasible due to time and resources needed to accomplish the task, or if a group of inter-linked nodes have views that are common within the group but in conflict with the rest of the network. In such cases, it may be necessary to divide the network into smaller sub-networks (or *pockets*), allowing them to use a routing algorithm different from the rest of the network. However, this requires that a set of nodes in each pocket run multiple routing algorithms to keep the group connected with the rest of the network.

In this work we focus on small scale MANETs without considering pocketing. However, as detailed in the following sections, the proposed solution can be used within the pockets of larger MANETs. Through this approach the network can dynamically form pockets that can communicate with each other. In a sense, in this work, we deal with a fundamental fraction of the problem of routing which can be used as a ground basis in later research regarding larger MANETs.

7.2 Related Work

The idea of self-adaptation in MANETs through configuring of certain parameters is not new. Formerly, this approach was adopted in the context of both system layers—e.g. dynamic addressing [24], dynamic configuration parameters within a routing algorithm [202, 166], etc.—and application layers [205, 207]. However, to the best of our knowledge, there are only a few studies that focused on adaptive routing in MANETs from a perspective similar to ours [82, 83, 61].

Forde et al. [82] present a reactive auto-configuration protocol based on the concept of self-stabilization by local checking and local correction. This protocol aims to ensure a consistent configuration of the routing algorithm parameter that persists across the connected groups of nodes. In particular, the cases where new nodes join the MANET or existing nodes roam between different groups are tested. This work forms the basis of the later solutions that involved real-time configuration of the routing algorithm parameter in the cases where conditions change, particularly with respect to nodes propagating their views and reaching a consensus within the network.

In a more recent work, Forde et al. [83] present a solution for global consensus in MANETs based on the theory of diffusion of innovations, a social science theory. This solution is applied to the problem of adaptive routing using the formerly proposed reactive auto-configuration protocol [82]. The innovation process is carried out in the three stages of persuasion, decision, and implementation. During persuasion stage a node attempts to make a judgment about its preferred routing algorithm based on its local perception of the network. There are four actor types that are involved in the persuasion stage: (1) Early adopters (EA), (2) Early majority (EM), (3) Late majority (LM), and (4) Laggards (LD). Each node observes its environment with respect to three indicators—such as mobility, size, traffic as underlined in Section 7.1. If the observations of a node has high correlation with the observations of its neighbours, the node is considered as an EA and its judgment is pursued in the network. Persuasion proceeds as follows. A node is an EM if a majority of its neighbours are EAs, or is an LM if a majority of its neighbours are EMs and EAs. Finally, a node is considered an LD if it has no such majorities in its neighbourhood. The decision stage is not entered unless a judgment stems from an EA or persists a certain time period. Once the decision stage is entered it is immediately followed by the implementation state which is merely the act of switching from the current routing algorithm to the newly

selected one. Briefly, EAs have a domino-like effect on the network, where the EMs and LMs follow when they have the required majorities.

In a later work, Doyle et al. [61] represent a Markov Random Field (MRF) Maximum a Posteriori (MAP) approach to the problem based on the same network layer auto-configuration protocol. The decision making process is carried out using the proposed MRF-MAP framework, where the nodes attempt to achieve consensus within their one-hop neighbourhoods. This technique is used similar to its application in the image processing field, which is based on tying connected of pixels to a selected colour based on the correlation of data. The same approach is applied in the context of adaptive routing in MANETs in order to make nodes within a neighbourhood use the same routing algorithm based on their correlated observations. MRF is used to capture this dependency within neighbourhoods. MRF is a field in which the conditional probability of a site being in some state depends only on a subset of the sites nearby and not all the other sites available. It is through these local dependencies that longer term dependencies can be modeled. In other words, the MRF as a prior can be used to drive consensus in the entire network through diffusing the consensus of small cliques.

All of these approaches aim for reaching a certain level of resiliency to failures by limiting the neighbourhood of a node to a one-hop distance. However, this approach merged with the proposed methods of Diffusion of Innovations and MRF-MAP may result in fast and uncontrolled pocketing in the network. In essence, this means that an indefinite number of pockets may emerge as a result of conflicting views even though it is not absolutely necessary to fragment the network. This may pose problems in settings like MANETs where the views of each node may be significantly different even from the nodes at close proximity such as a one-hop distance. In larger MANETs, uncontrolled pocketing may result in difficulty to manage a various number of routing algorithms on the border nodes. Finally, in both of the proposed methods, nodes are treated as equal entities during the decision making process. However, due to the physical dispersion of nodes, it is apparent that some nodes in a MANET may have more comprehensive and better views of the conditions in the environment (e.g. seeing more neighbours). Accordingly, assigning equal weights to each node during the decision making may result in loss of information and inaccurate decisions to be made.

7.3 Proposed Approach and System Architecture

In this section, we present a hybrid method that aims to benefit from the advantages of centralized and distributed methods, while limiting their disadvantages. Briefly, the control of coordination is handed to a single node as in the centralized approaches, yet, this single node is not the necessarily the same node at each coordination cycle. A coordinator node is promoted based on which node detects the change first. Theoretically, any node in the network can be a coordinator. However, we believe that limiting the responsibility of coordination to the nodes with situation awareness above a certain level can enhance the stability of the decisions. The coordinator takes into account the opinions of all the nodes in the network during coordination. This, in a sense, utilizes a leader at each coordination without explicitly electing it, and produces a final decision based on the different opinions distributed over the network through collecting them. The coordinator does not possess any ability to overpower the rest of the network during the course of a global adaptation process. Instead, it can be considered as a moderator that collects, collates and communicates the final decision of the network.

The system is designed as a distributed network of autonomous node agents (NA), where each NA is tightly coupled with a unique node in the MANET. The system model is described in abstract operational terms based on the Abstract State Machine (ASM) paradigm [28] for modeling dynamic properties of distributed systems. This description reveals the underlying design concepts and provides a concise and precise blueprint for reasoning about the system behaviour. The asynchronous computation model of a distributed ASM describes concurrent and reactive behavior as observable in distributed computations performed by autonomously operating computational agents [73].

Each NA has three primary responsibilities: (1) observing the network conditions, (2) coordinating global adaptation when there is a need to change the routing algorithm, and (3) cooperating when the global adaptation is being coordinated by another node in the MANET. These responsibilities are undertaken by NAs in three distinct control states of *observation*, *coordination*, and *cooperation*. An outline of transitions between these states is illustrated in Figure 7.1.

Each NA starts off at the control state of observation. In the observation state, an NA continuously collects network measurements in order to form an up-to-date situation awareness of the network conditions. During observation, if the network

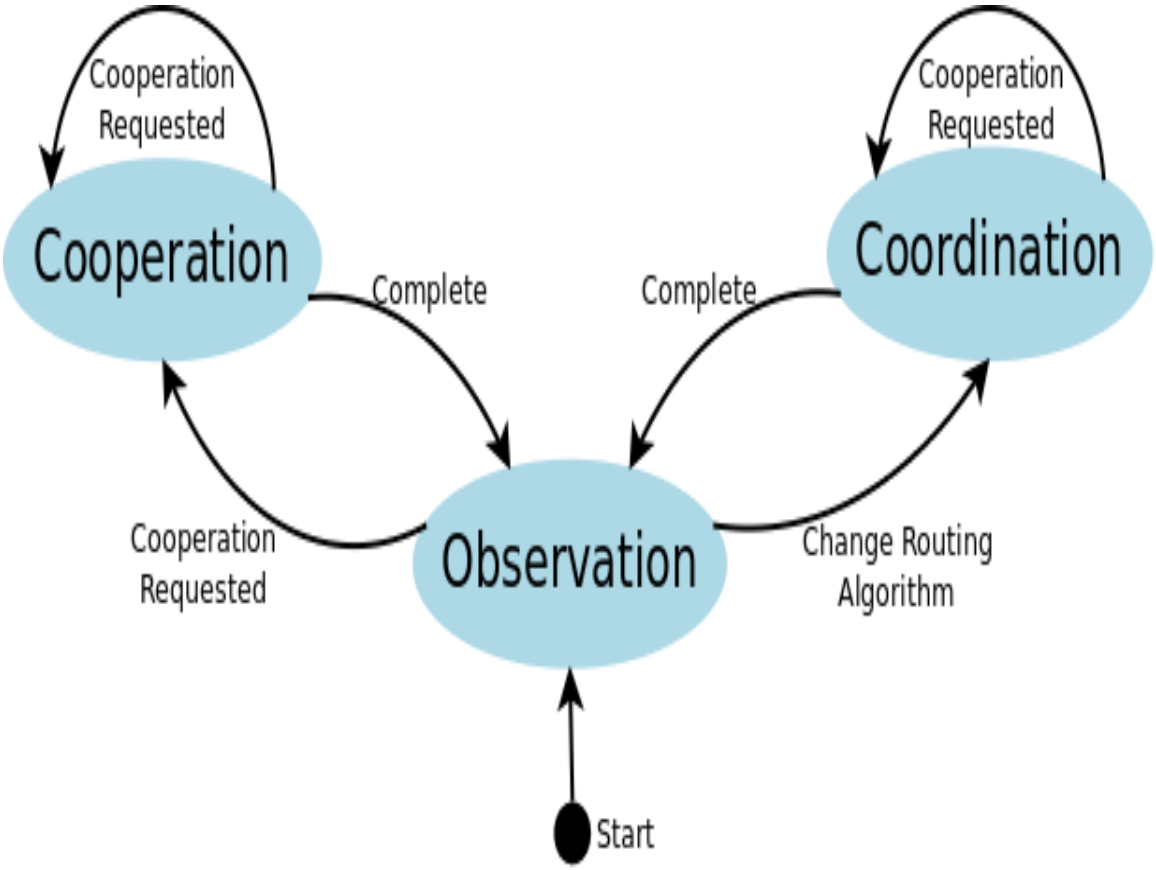


Figure 7.1: Control state diagram of node agents.

conditions reveal a need to switch to another routing algorithm, the NA moves to the coordination state. In the coordination state, the NA leads an election process where it transmits cooperation requests to the rest of the network in order to collect the opinions of the other nodes in the MANET. Once the opinions are collected, the NA aggregates them to decide which routing algorithm is more suitable under the current conditions. If the result of the election process contradicts the status quo (the current routing algorithm), then the NA carries out the rest of the coordination to complete the adaptation. If any failure occurs during the coordination—such as missing opinions, or an indifferent new decision, the coordination completes by aborting the global adaptation process. Upon completion, the NA moves back to the observation state. During the coordination state, receiving a cooperation request from another coordinator results in backing off for a random period of time and retrying coordination.

If a cooperation request is received from a coordinator—another node in the

MANET—during the observation state, the NA moves to the cooperation state. In the cooperation state, the NA's local opinion is sent back to the coordinator as a reply. If a global adaptation command is received from the coordinator after the reply is sent, the cooperation state completes and the NA moves back to the observation state. If the global adaptation command is not received within a time limit, the cooperation state completes with an abort, and the NA moves back to the observation state. During the cooperation state, cooperation requests received from other coordinators are ignored.

In order to carry out the execution in each state, each NA is equipped with four modules: (1) monitor, (2) local decider, (3) coordinator, and (4) cooperator. Every NA is modeled as a Distributed ASM (DASM) agent that continuously runs its main program through these modules. The following ASM program abstractly captures the behavior of an NA. Note that according to ASM semantics, the composition of the four activities in this program denotes a parallel execution of these activities.

Node Agent Program

NodeAgentProgram ≡
  Monitor
  LocalDecider
  Coordinator
  Cooperator

Each module runs in parallel with the ability to handle messages from the modules that its connected to or from the other nodes in the MANET. The connections of modules to each other and to the outside network, and the types of messages passed between modules are illustrated in Figure 7.2.

Modules carry out certain activities during the course of execution in different states. Monitor and local decider modules take part during the states of observation and cooperation. In the observation state, they cooperate in order to produce an opinion based on the observed network conditions periodically. Their primary role in this state is to facilitate the transition to the coordination state when necessary. Once the NA is in the coordination state, they pause the periodical task of producing opinions until the NA transitions back to the observation state. In the cooperation state, these two modules work on an on-demand basis where the opinions are produced

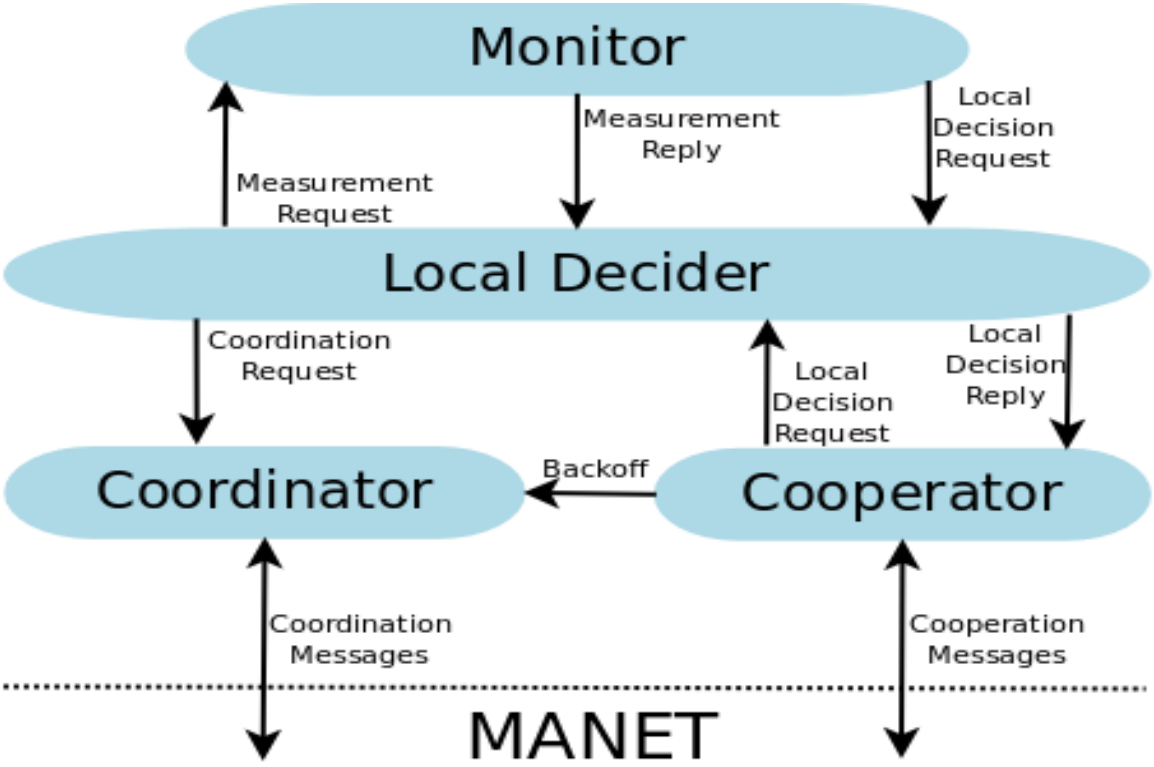


Figure 7.2: The modules of an NA.

only when requested by the cooperator module. Coordinator and cooperator modules form the core of the coordination and cooperation states respectively, where they carry out the task of leading or taking part in the global adaptation process. An outline of the interactivity between the modules during the coordination and cooperation states is illustrated in Figures 7.3 and 7.4 respectively.

In the rest of this section, we focus on the activities carried out by the four modules in more detail.

7.3.1 Monitor

The monitor module is responsible for collecting measurements in order to define the current conditions in the network. The measurements collected by the monitor module are used during both cooperation and coordination states by the associated modules.

As shown in Figure 7.2, the monitor module communicates only with the local decider module. The ways in which the communication is carried out between these two modules depend on the current state of the NA. The monitor module works differ-

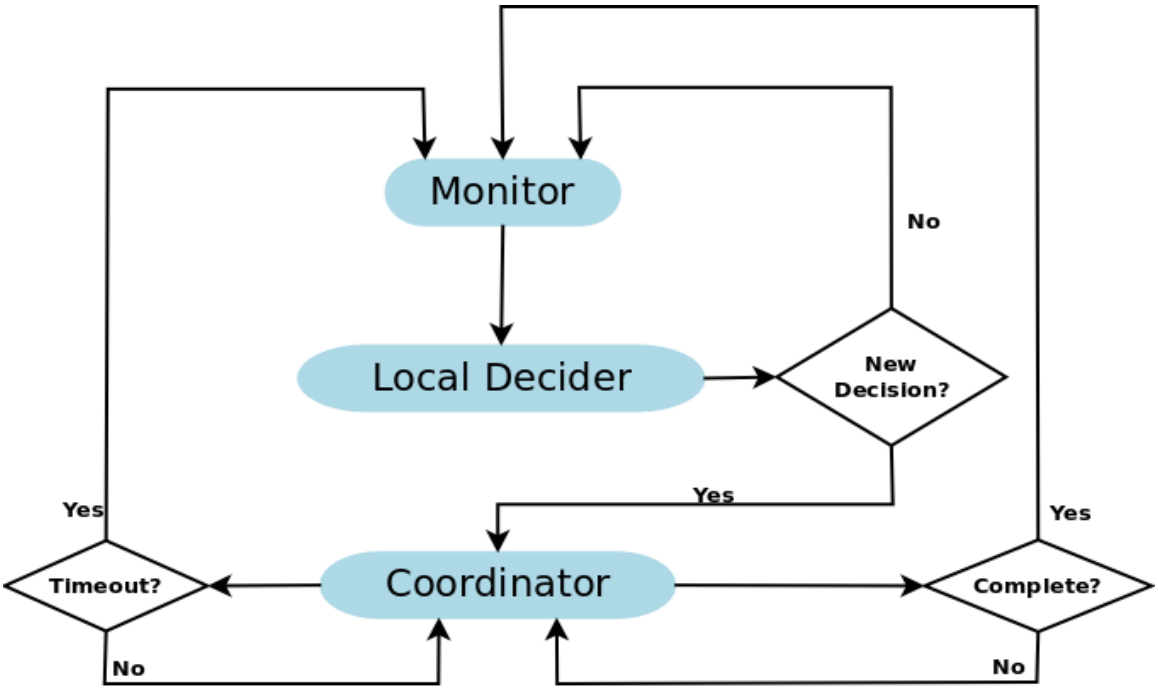


Figure 7.3: Interactivity between the modules during the course of the coordination state.

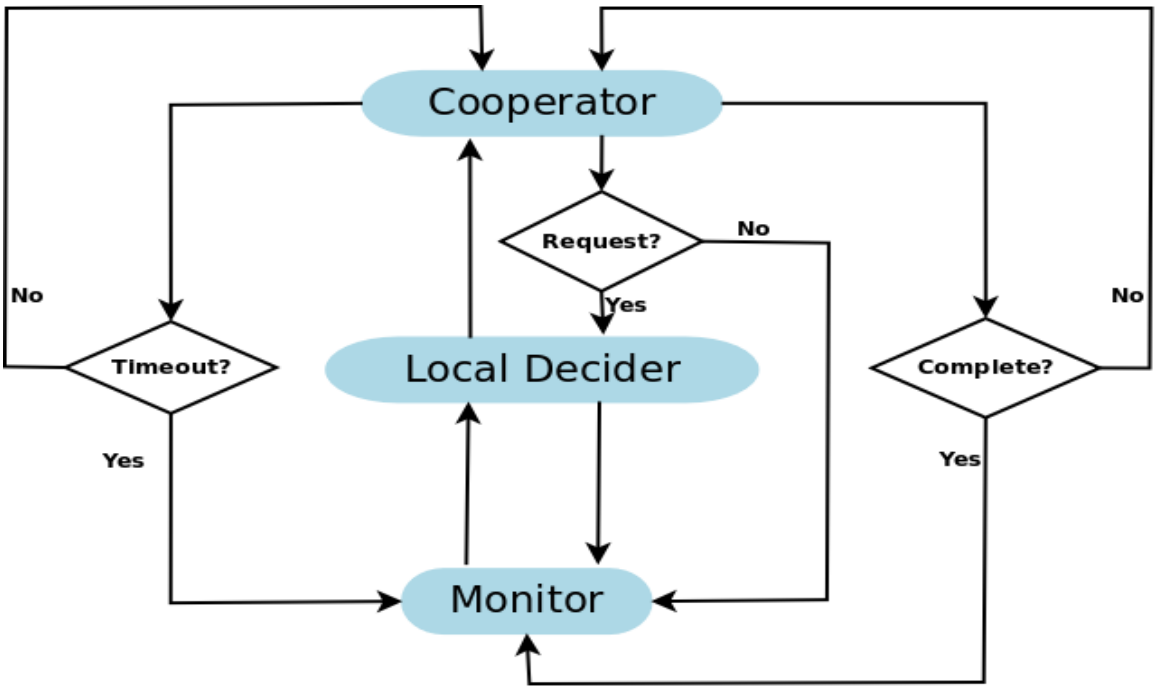


Figure 7.4: Interactivity between modules during the course of the cooperation state.

ently in observation and cooperation states. In the observation state, measurements are collected at the end of fixed time intervals. These measurements are then sent to the local decider module encapsulated in routine local decision requests. Whereas, in the cooperation state, the monitor module works on an on-demand basis where it collects and sends measurements in order to reply to requests from the local decider module.

The behaviour of the monitor module in different states is abstractly captured in the following ASM program as:

Monitor Module

Monitor ≡
  if *isState*(*Observation*) **then**
    if *observationPeriodReached*() **then**
      let *measurements* = *collectMeasurements*() **in**
        SignalLocalDecider(*measurements*)
  if *isState*(*Cooperation*) **then**
    choose *message* **in** *receivedLocalMessages*()
                with *isMeasurementRequest*(*message*) **do**
      let *measurements* = *collectMeasurements*() **in**
        ReplyToLocalDecider(*message*, *measurement*)

The measurements are collected with respect to certain indicators. The three indicators that are often referred to as key metrics to define the conditions in a MANET are: network size, mobility, and traffic load. Due to the distributed nature of MANETs, however, NAs are not guaranteed to have a complete overall view of the environment. Instead, they have partial views limited to their range of sight. Thus, from an NA's point of view, collecting measurements consist of taking local snapshots of the environment, and capturing changes as they occur. In order to do this, *collectMeasurements()* extracts information from different layers of the network stack (network layer, link layer, etc.) without generating any extra network load through querying other nodes in the network.

Network size, mobility and network load measurements are collected as follows. The network size is simply gathered through finding out the number of neighbours that the node is currently aware of. On the other hand, mobility is measured by

capturing the changes in the number of neighbours that an NA perceives throughout time [118]. This is a virtual measurement rather than an actual one which depends on the physical movements of nodes. From an NA's point of view a neighbouring node that moves does not impose any changes to mobility as long as the link persists. Therefore, mobility can be expressed in terms of appearing and disappearing links. Let us define the number of unique links perceived by an NA at time t and time $t' = t + \Delta t$ as sets L_t and $L_{t'}$ respectively. Then, mobility measure M at time t' can be defined as:

$$M_{t'} = \frac{|L_{t'} - L_t| + |L_t - L_{t'}|}{\Delta t} \tag{7.1}$$

where $|L_{t'} - L_t| + |L_t - L_{t'}|$ denotes the total number of unique links. Finally, traffic load is measured by observing the number of packets—both control and application packets—received by a node between observations. Thus, traffic load depends on the current routing algorithm, since the number of control packets is different with respect to different routing algorithms.

7.3.2 Local Decider

The main functionality of the local decider module is to produce opinions upon receiving local decision requests based on the measurements provided by the monitor module.

When a request is received during the observation state, the local decider produces an opinion and compares it to the status quo. If they are different, it signals the coordinator module with the opinion. When it receives a request during the cooperation state, it replies to the cooperator module with the new opinion regardless of it being different than the status quo. The behaviour of the local decider module in different states is abstractly captured in the following ASM program:

Local Decider Module

Local Decider ≡
  if *isState*(*Observation*) **then**
    choose *message* **in** *receivedLocalMessages*()
          with *isRoutineLocalDecisionRequest*(*message*) **do**
      let *newRoutingAlgorithm* = *selectRoutingAlgorithm*() **in**
        if ¬*isStatusQuo(newRoutingAlgorithm)* **then**

Local Decider Module

SignalCoordinator(*newRoutingAlgorithm*)

if *isState*(*Cooperation*) **then**
  choose *message* **in** *receivedLocalMessages*()
    with *isLocalDecisionRequest*(*message*) **do**
  let *newRoutingAlgorithm* = *selectRoutingAlgorithm*() **in**
    SignalCooperator(*newRoutingAlgorithm*)

The opinions are generated using MCDA. During this process the three indicators are used as the criteria to evaluate the quality of each routing algorithm in the alternative set. The criteria g_1, g_2 and g_3 are defined as mobility, traffic load and network size respectively. Each criteria is assigned a weight w_i such that $\sum_{i=1}^{3} w_i = 1$. These weights are used to emphasize more or less on different criteria.

MCDA is performed by *selectRoutingAlgorithm*() function using the PROMETHEE method [131] to rank each routing algorithm in the set of alternatives, A, with respect to the criteria. The PROMETHEE method is based on pairwise comparisons of the alternatives. That is, it considers the difference between the evaluation of two alternatives over each criterion. The preference of one alternative over another is modulated as a function of this difference. The preferences are quantified as real numbers between 0 and 1 through certain proposed preference functions. Each criterion is associated with a generalized criterion which is represented by the pair $(g_i, P_i(a, b))$, where $P_i(a, b)$ denotes the preference of alternative a over b for criterion g_i. In this study, we use the *Usual Criterion* as a common generalized criterion for all of the criteria.

Using the generalized criteria, NAs calculate aggregated preference indices to express with what degree an alternative is preferred to another. For all $a, b \in A$, the aggregate preference indices are defined as:

$$\pi(a, b) \quad = \quad \sum_{i=1}^{3} P_i(a, b) w_i \tag{7.2}$$

Using these indices, an NA calculates how much each alternative routing algorithm outranks the other alternatives and how much it is outranked by the others. For this,

positive and negative outranking flows are calculated as:

$$\begin{aligned} \phi^{+}(a) &= \frac{1}{n-1}\sum_{x\in A}\pi(a,x) \\ \phi^{-}(a) &= \frac{1}{n-1}\sum_{x\in A}\pi(x,a) \end{aligned} \tag{7.3}$$

Finally, the net outranking flow is used to create a complete ranking of the alternatives:

$$\phi(a) = \phi^{+}(a) - \phi^{-}(a) \tag{7.4}$$

The alternatives are ranked based on their net outranking flows, the alternative with highest outranking flow being the best alternative. As a result, *selectRoutingAlgoritm*() returns the routing algorithm with the highest net outranking flow, which is then used by the coordinator or cooperator modules.

7.3.3 Coordination

Upon receiving a coordination request from the local decider module, the coordinator module initiates the global adaptation process. Unless interrupted by the cooperator module, the global adaptation process is carried out by the coordinator module through: (1) collecting the opinions of the nodes in the network by broadcasting cooperation requests, (2) electing a routing algorithm from the received opinions, and (3) communicating the result of the election to the rest of the network through a global decision command. If the coordinator module receives a signal from the cooperator module during the global adaptation process, it backs off for a random period of time before re-trying to coordinate. Any message received during the backoff period, makes the coordinator module abort the global adaptation. As a result of this, the NA moves to either the observation state or cooperation state depending on the type of the received signal. This behaviour is captured by the following ASM program:

Coordinator Module

Coordinator ≡
  if *isState*(*Observation*) **then**
    choose *message* **in** *receivedLocalMessages*()
          with *isCoordiantionRequest*(*message*) **do**
      SendCooperationRequests

Coordinator Module

```
if isState(Coordination) then
  CollectCooperationReplies
  if allRepliesCollected() then
    let globalDecision = electRoutingAlgorithm(replies) in
      ProceedWithCommit(globalDecision)
  else
    if isTimeout() then
      AbortCoordination()
  choose message in receivedLocalMessages()
          with isBackoffMessage(message) do
    Backoff(pickBackoffTime())
```

The SendCooperationRequests rule broadcasts a cooperation request to the MANET. Each of the received replies include an opinion and a weight field. Weight field carries an indicator for the number of votes that are assigned to the source of the reply.

In the *electRoutingAlgorithm*() function, the opinions with their associated weights are used in a weighted voting scheme to find the routing algorithm that is agreed upon. In the current design of our system, the weights are a representation of the number of nodes that a node perceives at the time that the cooperation request was received. Through weights, the coordinator assigns different number of votes to each cooperator during the selection process, where the nodes with more neighbours are assumed to have a more reliable knowledge of the network—thus a more important opinion. Let us define the weighted voting system as $\{q : w_1, w_2, ..., w_N\}$ where q, the quota, is the minimum number of votes for an alternative to be eligible as a winner, N is the total number of nodes that participate in the voting, and w_i is the number of votes assigned to a participant i. Then, the result of *electRoutingAlgorithm*() is a routing algorithm that has the most votes, and has at least as many votes as the quota.

Finally, ProceedWithCommit communicates the result of the election to the rest of the nodes in the MANET in order to complete the global adaptation process. In the current design, the coordinator module uses a commit protocol similar to Tree 2PC protocol [204] during the global adaptation process. Each phase during the coordination is bound by a timeout that prevents blocking. When a timeout occurs, the coordinator aborts the global adaptation, and the NA moves to the observation

state. The choice of Tree 2PC is due to the low message complexity and implementation simplicity. However, the system can be modified to use other multiple-phase protocols.

7.3.4 Cooperation

The main responsibility of the cooperator module is to respond to cooperation requests from the coordinators in the MANET. If a request is received during the observation state, the NA moves to the cooperation state. In the cooperation state, the cooperator module first interacts with the local decider module to obtain an opinion. Upon receiving a reply from the local decider module, it communicates the opinion along with the number of neighbours that the NA perceives to the coordinating node and waits for a global decision command. If the command is received before a timeout, it switches from the current routing algorithm to the global decision and the NA moves back to the observation stage. When a timeout occurs, the cooperator module aborts. During the course of adaptation, the cooperator module corresponds to a cohort in the two-phase commit. Any request received during an ongoing cooperation is ignored, causing the coordinating node to abort the global adaptation process. If a request is received during the coordination state, the cooperator module signals the coordinator module to back off and retry coordination. The following ASM program abstractly captures the behaviour of the cooperator module:

Cooperator Module

Cooperator ≡
  if *isState*(*Observation*) **then**
    choose *message* **in** *receivedNetworkMessages*()
          with *isCooperationRequest*(*message*) **do**
      RequestLocalDecision(*message*)
  if *isState*(*Cooperation*) **then**
    choose *message* **in** *receiveLocalMessages*()
          with *isLocalDecisionReply*(*message*) **do**
      SendCooperationReply(*message*, *weight*)
  if *isState*(*Coordination*) **then**
    choose *message* **in** *receiveNetworkMessages*()
          with *isCooperationRequest*(*message*) **do**
      SendBackoffMessage

7.4 Simulation Results

In this section we analyze the general behaviour of the proposed approach by presenting results that are extracted from various simulation runs. In Section 7.4.1, we outline the general settings in which the system is implemented, configured and tested. Section 7.4.2 details the results based on certain evaluation metrics, and provides the characterization of the overall system behaviour.

7.4.1 Testing Environment

The proposed approach is implemented on OMNET++ discrete event simulation platform using the INETMANET framework [145]. NAs are implemented as *compound modules* that consist of four *simple modules* which provide the functionality of monitor, local decider, cooperator and coordinator modules. In addition, we have implemented some convenience modules that are deployed on different layers of the network stack in INETMANET framework to assist the monitor module during the collection of observations. The convenience modules do not interfere with the existing functionality of the underlying platform; they are merely a set of extensions that access the data structures (IP tables, ARP, etc.) that hold the general information needed by our system.

The test scenarios are run using different MANET sizes ranging from 10 to 30 nodes in a grid of 500 meters by 500 meters. Each node has a radio of 7mW transmission power and -90dBm sensitivity. Also free space propagation model is used during the simulation runs. Under these settings the theoretical transmission distance is roughly 800 meters. These values are chosen to ensure that the network is fully connected. The scenarios are run using the random way point mobility model with node speed uniformly distributed between 20 meters per second and 50 meters per second, and pause times uniformly distributed between 3 seconds and 5 seconds. Our choice of the values for MANET size, grid area and the mobility model is due to their common usage in the evaluation in the former approaches, which establishes an identical general environment that facilitates comparison to the approach proposed in this chapter. Furthermore, it is important to note that the movement speed of the nodes in the scenarios are relatively high. The reason for this choice was to test the proposed solution in an extremely mobile—in a physical sense—setting, which provides an environment that is more difficult to operate in when compared to more practical and more ordinary mobility settings. Each run is limited to 18000 simulated

seconds which is longer than the runs represented in the previous studies. The reason behind selecting this duration was to increase the amount of information to be observed and collected during the simulation runs so that we had sufficient information to analyze the behaviour of the system. In addition, each scenario was run 10 times with different seeds. Note that the number of runs is relatively low, and was selected due to the length of each simulation run. It is also important to note here that the results represented in this chapter do not include the confidence intervals for the mean values collected from the simulation runs since the sample size is not sufficient to compute such information. At the beginning of each run, we randomly create pairs of nodes where a constant bit rate (CBR) traffic of 10 packets per second is generated from one node to another.

The implementation of the system is configured to use 3-second timeout bounds for collecting cooperation replies, switch command replies and switch commands, which limits each global adaptation cycle to 6 seconds. Any violation of timeouts terminates a cycle, and marks it as 'incomplete'. Furthermore, the system is configured so that the local decider modules signal coordination modules immediately after a change to the status quo is required. That is, coordination initiates without checking if the conditions that triggered the process persist over a certain period of time. The persistence periods were removed from the original implementation in order to collect more information in shorter simulation runs.

7.4.2 Test Scenarios

In the evaluation of the proposed solution, we have focused on two distinct cases. In the first case, any node in the system is eligible to coordinate the global adaptation cycles without any limitations. In the second case, we introduce a constraint to limit the number of eligible coordinators. The introduced constraint is based on the number of neighbours that a node perceives. If this number is below a certain threshold, the node is not eligible to initiate a coordination. This constraint is then tightened incrementally in different runs by increasing the threshold value. The range of threshold values used in the different runs range from *at least two neighbours* to *at least six neighbours*. Note that due to mobility, a node's eligibility is not constant throughout a run. That is a node can be eligible or ineligible to coordinate at different points in time. Both the non-constrained and incrementally constrained cases are applied in runs with all of the MANET sizes.

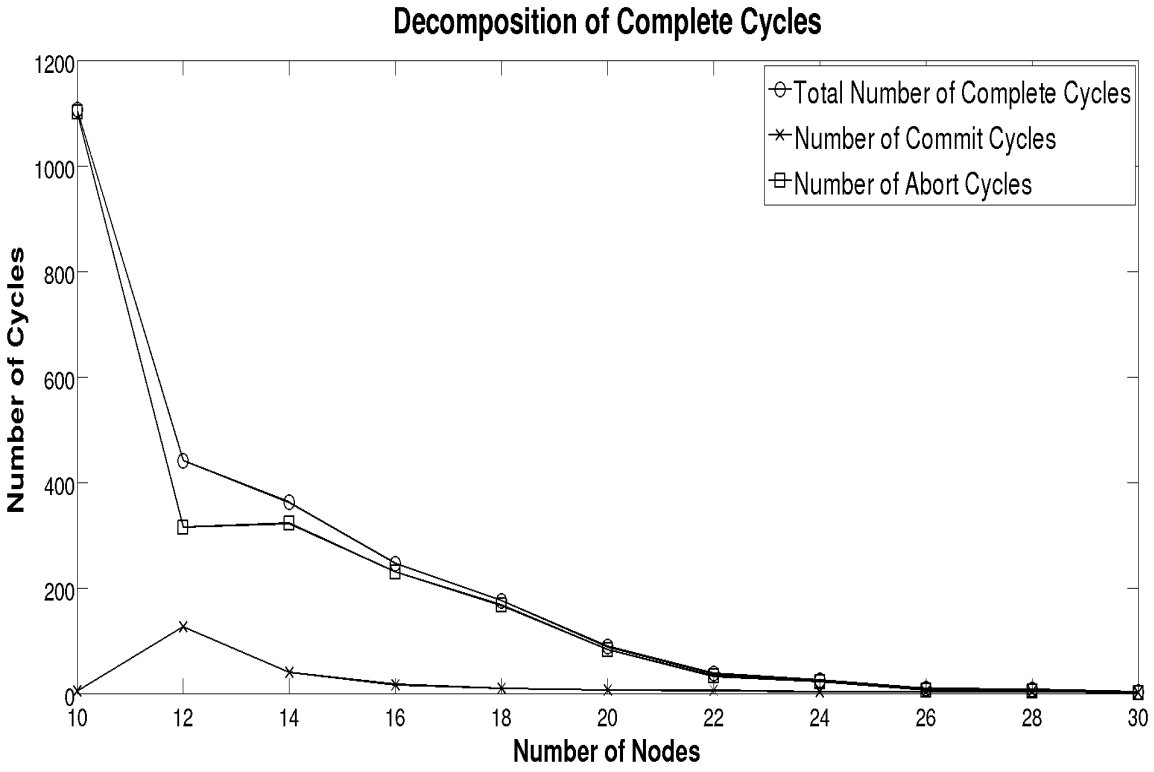


Figure 7.5: Non-constrained coordinators: The total number of global adaptation processes undertaken in a 5-hour simulation run, and its decomposition into cycles that completed with commit and abort.

The results of the tests are analyzed using four metrics: (1) the number of complete global adaptation cycles, (2) the total number of backoffs observed during global adaptation cycles, (3) the total number of timeouts observed during global adaptation cycles, and (4) the average time-taken to complete a global adaptation cycle. The number of complete cycles is used as a measure of how many times the system attempted to change the current routing algorithm. Through this metric we observe both the cycles that finished with a commit and the cycles that finished with an abort. The commit cycles indicate that the elected routing algorithm was different from the current one, which required the network to reconfigure. Whereas, abort cycles indicate that the election produced a routing algorithm that is the same as the one in use, which does not require the final step of reconfiguration. The total number of backoffs is the sum of all the backoffs experienced by the nodes that tried to coordinate throughout a simulation run. Similarly, the total number of timeouts is the sum of all the timeouts that are perceived by the coordinators throughout a simulation run. These two metrics indicate interruptions and terminations of global

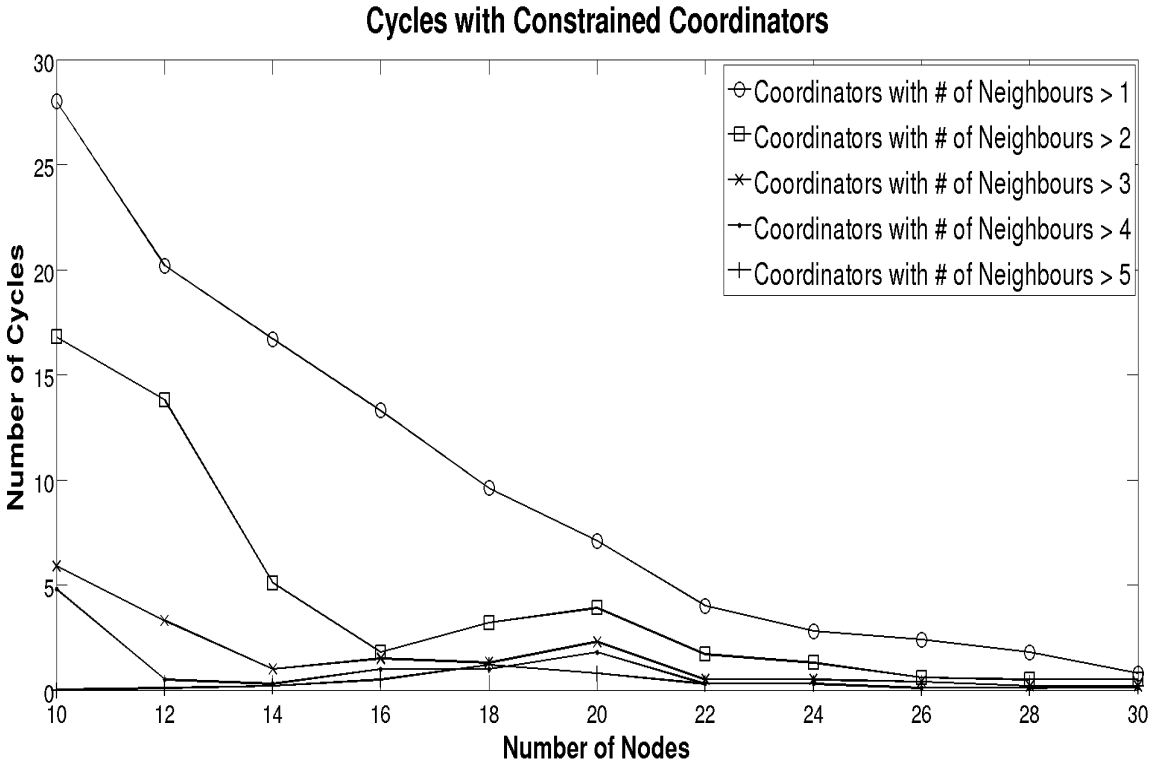


Figure 7.6: Constrained coordinators: The total number of global adaptation processes undertaken in a 5-hour simulation run, and its decomposition into cycles that completed with commit and abort.

adaptation processes, and are used to quantify the difficulties observed during the global adaptation processes throughout each run. Finally, the average time-taken is used as a metric to analyze the speed of complete global adaptation processes, and used as a metric for comparison with the previously proposed approaches.

In the first set of runs, we tested the system with no constraints on the eligibility of the coordinators. The results of these runs are used to determine the behaviour of the system in a setting where any node can attempt to initiate global adaptation processes without any limitations. Figure 7.5 illustrates the results of these runs with respect to the total number of completed global adaptation cycles and its decomposition into cycles that completed with commit and abort after weighted voting. These results indicate that the number of completed cycles is very high in the smaller MANETs. Whereas, as the MANET size is increased, this number drops significantly due to the increase in the number of simultaneous coordinators. It is also evident that the number of abort cycles is significantly higher than the commit cycles in the smaller MANETs. This behaviour in the non-constrained case stems from the fact that the

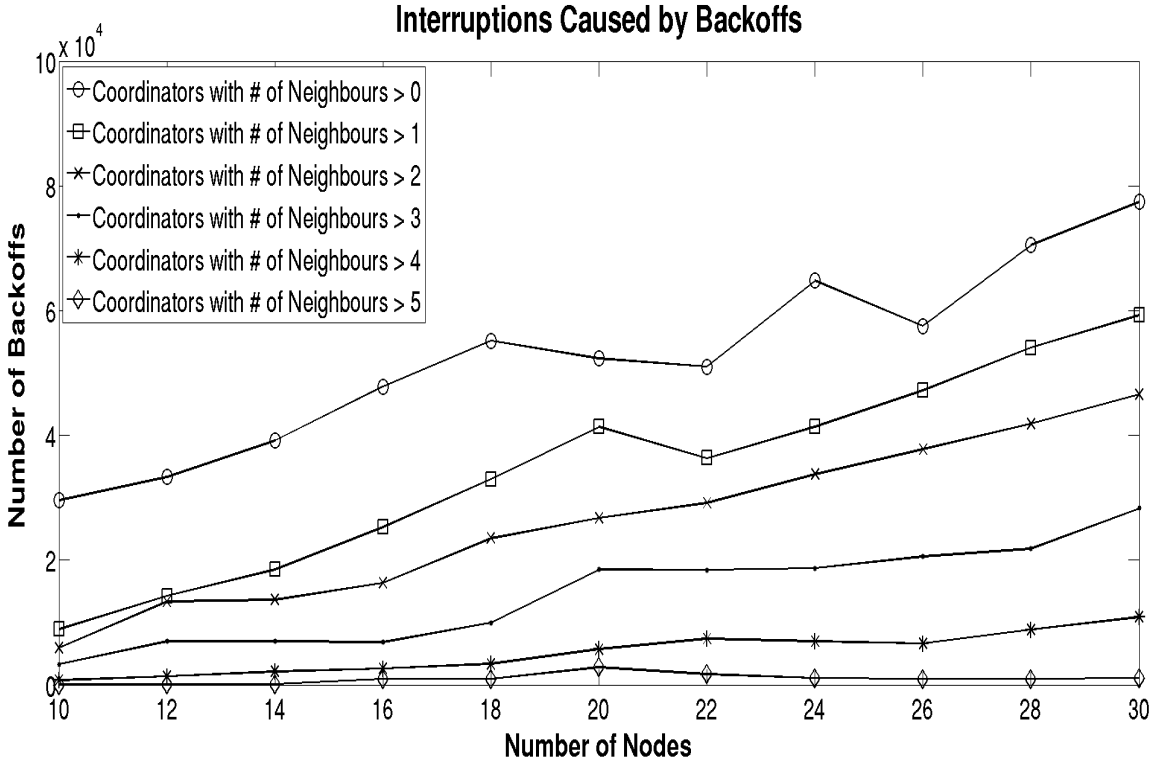


Figure 7.7: The total number backoffs observed in runs with different levels of constraints on coordinators.

coordinators initiate global adaptation processes even if their views of the conditions is very limited. Accordingly, the status quo is favoured by the majority of votes in network instead of the opinion of the coordinator, which results in aborting the cycle. As the network size is increased this difference between commit and abort cycles disappears since it becomes difficult to complete any cycle in larger MANETs. Another important result is the high number of commit cycles, although much lower than the abort cycles generally. This points to high instability where the system keeps reconfiguring the routing algorithm, which causes a considerable amount of burden on the network without producing long lasting results.

In the second set of runs, we aimed to reduce the instability through ensuring that the global adaptation cycles are initiated only by nodes with a certain level of knowledge about the network conditions. The level of knowledge is defined as the number of neighbours that a node perceives at the time that a need for reconfiguration is captured. Figure 7.6 illustrates the number of complete global adaptation cycles collected in runs where the coordination constraint is incrementally tightened by

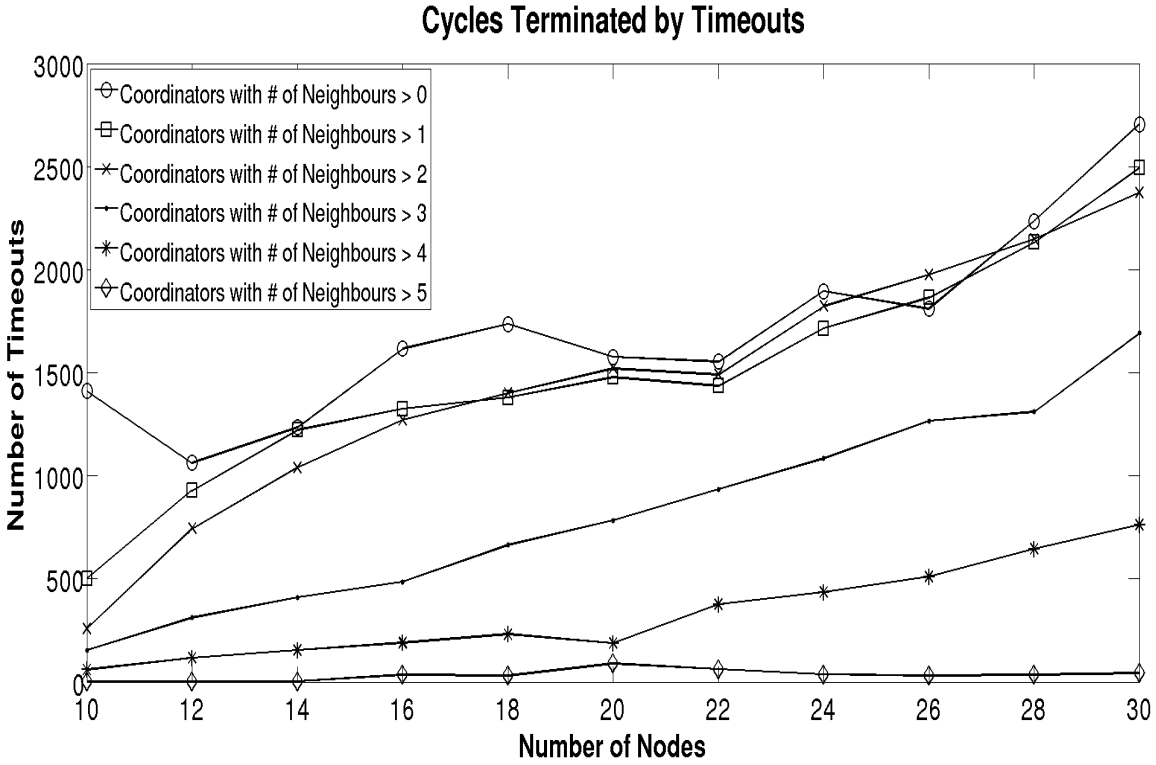


Figure 7.8: The total number timeouts observed in runs with different levels of constraints on coordinators.

increasing the threshold from *at least two neighbours* to *at least 6 neighbours*. These results show that the number of complete global adaptation cycles decrease as the constraint is tightened incrementally making the overall behaviour much more stable. Note that the total number of complete cycles is reduced from about 1100 cycles to the range of 1-28 cycles respectively. This is a result of the global adaptation processes initiated by nodes with more knowledge of the network. The difficulty of completing cycles as the network size increases is also evident in the constrained runs. Convergence of complete cycles occurs starting with the network size of 22 in the constrained runs, whereas, in the non-constrained runs the system can complete a low number of cycles in the larger MANETs. It is also evident that the increase in the size of the network reduces the number of complete cycles regardless of the level of constraint on the coordinators.

The effects of network size and the coordination constraint can be further investigated using the backoff and timeout metrics. Figures 7.7 and 7.8 illustrate the total number of backoffs and timeouts experienced by the coordinators respectively. These

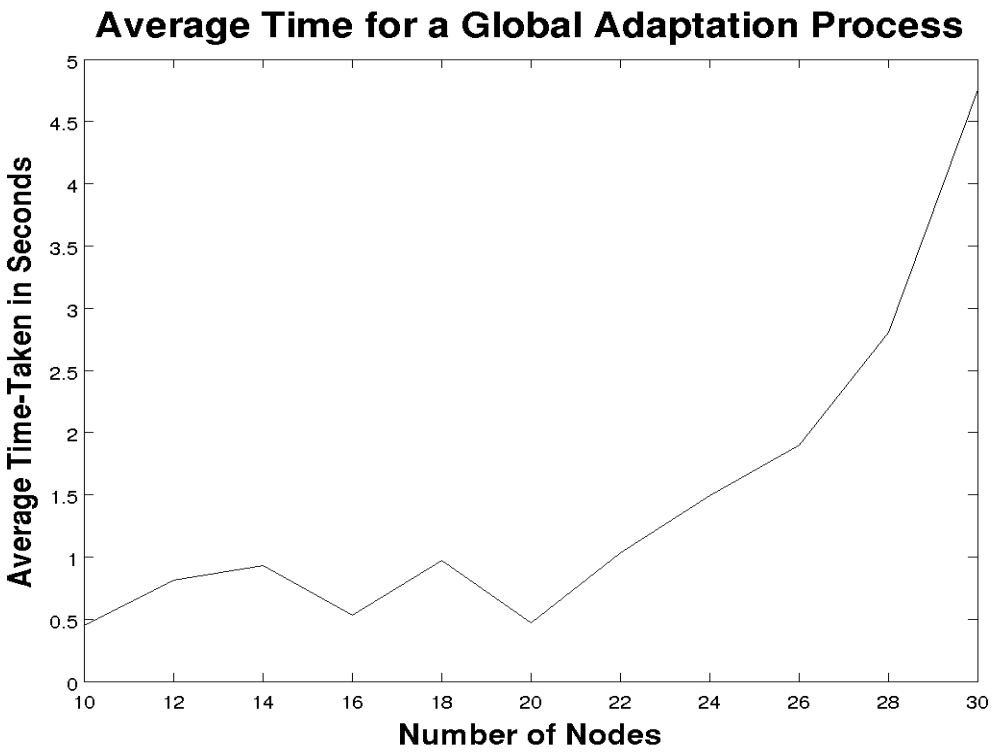


Figure 7.9: The average time-taken for global adaptation processes to complete.

results can be viewed with respect to two points: (1) tendency in the number of backoffs and timeouts within each constraint level as the MANET size increases, and (2) tendency in the number of backoffs and timeouts as the coordination constraint is tightened. These results show that both measures generally increase within each constraint level as the network size is increased. On the other hand, there is a significant drop in both measures as the coordination constraint is tightened, reducing the chance of backoffs by limiting the number of eligible coordinators. These results combined with the ones illustrated in Figure 7.6 show that by adding a coordination constraint that is incrementally tightened, the system behaviour can be enhanced in terms of increasing stability and reducing interruptions of coordination.

Finally, we have observed the system behaviour with respect to the average time-taken during the completion of global adaptation cycles. Figure 7.9 illustrates this information in relation to increasing MANET sizes. The results show that the cycles are completed in under a second in networks of size less than 22 nodes. However, as the network size increase the cycles take longer to complete, reaching roughly 5 seconds in MANETs of 30 nodes. The average time-taken is the single metric that can

currently be used to compare the proposed approach the previous research. This is mainly due to the lack of information in the former studies regarding how the problem of multiple initiators is dealt with—thus, leaving metrics such as the total number of complete cycles, backoffs and timeouts out of the comparison. In addition, the study based on MRF-MAP [61] evaluates the time-taken to complete the cycles in terms of processor cycles, where we do not possess a complete knowledge of the specifications of the hardware in use. As a result, the only method that can be compared to the proposed approach is the study based on the diffusion of innovations [83]. These results show that the proposed approach is an order of magnitude faster than the former approach where the cycles are completed in 19 to 26 seconds in MANETs of 10 to 30 nodes.

In summary, the results highlight two important points regarding the behaviour and the performance of the proposed approach. First, as long as the responsibility of coordination is limited to the nodes with a certain level of knowledge about the network conditions, the system produces stable reconfigurations in a very fast and minimally interrupted manner. Second, regardless of the level of coordination constraint, as the network size reaches above 24 nodes it becomes very hard to perform reconfigurations. These two points underline that the proposed approach is very promising in relatively small scale MANETs. Furthermore, it forms a strong basis for future research on larger scale MANETs where the network is fragmented into pockets of limited size. In those cases, the proposed approach can be used to carry out reconfigurations within the pockets.

7.5 Summary

In this chapter, we have provided the details of a new approach to the problem of adaptive routing in small scale MANETs based on the principles outlined by the proposed methodology within the context of coordinated self-management. The proposed approach focuses on leveraging the already existing routing algorithms in real-time through switching from one routing algorithm to another as the conditions change. The analyses based on the results extracted from simulations underline that the proposed approach produces stable configurations with minimal interruptions even in the presence of multiple coordinators. Furthermore, the completion of global adaptation cycles are performed significantly faster in comparison to the results outlined in the previous work.

Our future directions include: (1) a comparative study regarding the existing MANET routing algorithms to better define the conditions that cause the initiation of global adaptation cycles, (2) using the results of this comparative study to validate the correctness of the reconfigurations, (3) a comparative study on the impact of applying PROMETHEE with other parametrizations and other MCDA methods, (4) a study on the implications of different global decision methods, (5) applying the proposed approach on medium and large scale MANETs, and (6) deployment of the proposed approach on a MANET testbed.

Chapter 8

Conclusion and Future Work

In this dissertation, we have introduced a new methodology for reactive self-management in distributed systems based on MCDA. In this context, we have provided a general overview of the proposed methodology through investigating the necessary procedures that it underlines in order to build a self-managing distributed system. The proposed methodology views the adaptation process from a reactive point of view where the self-management cycles are reactions to encountering certain critical changes in conditions. Once a reaction is triggered by the conditions, the methodology uses MCDA to select the next course of action from a set of alternative actions. Accordingly, system objectives, performance indicators, critical changes, set of alternative actions and the multiple criteria decision model must be carefully determined before the system is operational. Based on the pre-operation models, autonomous agents that are designed using the methodology manage system adaptation through a continuous cycle of observe-detect-react activities.

This work also covered various MCAP based MCDA methods that could be incorporated into the reaction activity as outlined by the proposed methodology. In this sense, we have also focused on the general structure of the preference models that are employed in these methods. Moreover, the dissertation overviews some of the mainstream MCDA methods under the two categories of MAUT methods and outranking methods. The list of MCDA methods that are explained includes MUAT, UTA and AHP from MAUT methods, and ELECTRE and PROMETHEE from outranking methods. Moreover, we have also provided a list of other MCDA methods from these categories, and given an introductory explanation for each of the mentioned methods.

Furthermore, this work has also outlined the general structure of the simulation platforms that are built and used during the assessment of the application of the

proposed methodology. In this context, two platforms were covered. First, we have explained the working principles of a simulation platform that is designed and implemented in order to apply the proposed methodology to the problem of dynamic resource consolidation management in clouds as a representation of the independent self-management. Second, we have outlined the usage of OMNET++ in the application that represents the usage of the methodology to the problem of coordinated self-management within the context of adaptive routing in MANETs. In that sense, we have provided a set of extensions to OMNET++ so that the simulation platform supports the activities that the proposed methodology underlines for coordinated self-management. In the following chapters, we have covered the independent and coordinated self-management simulation case studies in detail with respect to metrics such as performance, feasibility and scalability.

As the first simulation case study—independent self-management, using the proposed methodology we have designed and implemented IMPROMPTU, an MCDA based distributed resource consolidation manager for clouds. In IMPROMPTU, each autonomous agent is attached to a unique physical machine in the cloud where it autonomously manages local resource utilization in a way that is independent from the rest of the autonomous agents by controlling the accommodation of the virtual machines. When the conditions impose, autonomous agents react by migrating certain virtual machines that are running on their physical machines in order to preserve a pre-determined desirable resource utilization state. The virtual machines to be migrated and their new locations are selected by the autonomous agents using MCDA. The proposed system has been investigated in detail in terms of its reactive behaviour, MCDA model, distributed architecture and performance assessment through simulated runs. Simulation results show that a reactive and distributed approach to autonomous resource consolidation management can be a solid alternative to the existing centralized architectures. Furthermore, it is shown that resource consolidation management can be substantially improved through an MCDA model and proper extensions to it. The proposed system achieves scalability through a distribution of responsibility that reduces the complexity of computing new resource distributions by localizing the management process. In addition, it produces feasible solutions with respect to the observed marginal number of migrations that are performed in order to apply new distributions of computational resources. Moreover, IMPROMPTU triggers less SLO violations by smoothly distributing the overall resource utilization over

more physical machines in a cloud. As an MCDA based distributed resource consolidation manager, IMPROMPTU proves to be a strong candidate in the domain.

In the second simulation case study—coordinated self-management, the proposed methodology is used to design and implement a new solution to the problem of adaptive routing in small scale MANETs. The proposed approach focuses on leveraging the already existing routing algorithms in real-time through switching from one routing algorithm to another as the conditions change with full compliance in the overall system. In this context, each MANET node in the system is attached an autonomous agent that continuously observes the conditions, detects critical changes and react by selecting the most suitable routing algorithm to use through MCDA. The end-product of the selection process is deemed as the local opinion of an autonomous agent and is independent from the rest of the views in the system. Since the same routing algorithm needs to be used by every MANET node, the local opinions cannot be implemented on a MANET node by its corresponding autonomous agent without the consent of the rest of the system. In order to reach a network-wide decision, each autonomous agents local opinion is collected by a coordinator. The coordinator is not an elected leader, rather, it is usually the first MANET node that detects a critical change. During the process of generating a final decision, the coordinator prompts each MANET node for a local opinion. Upon receiving a complete list of opinions regarding which routing algorithm should be used, the coordinator aggregates the collected opinions using a weighted voting scheme where nodes with more knowledge have higher influence on the outcome of the aggregation process. Once a final decision is reached, depending on whether this decision requires a change to the status quo, the coordinator commands the cohorts to change their routing algorithm to the final decision. The analyses based on the results extracted from simulations underline that the proposed approach produces stable configurations with minimal interruptions even in the presence of multiple coordinators. Furthermore, the completion of global adaptation cycles are performed significantly faster in comparison to the results outlined in the previous work.

To the best of our knowledge the two systems designed using the proposed methodology are the first solutions in their domains that adopt an MCDA based approach. The results of the two simulation case studies show that the proposed methodology can be used to design efficient self-managing systems. The work outlined in this dissertation can be further extended with respect to both improvements on the current applications of the proposed methodology in the sort-term, and application of the

proposed methodology in the context of other self-management problems in the long-term. While the short-term goals can be viewed more as enhancements to the models behind the application of the methodology in the two simulation case studies outlined in this dissertation, the long-term goals rather focus on the idea of outlining how such an methodology for reactive self-management based on MCDA can be applied in the general field of autonomous computing and multi-agent systems.

Our short-term goals include refining the application of the proposed methodology to the two simulation case studies with respect to improving the manner in which the reactions are triggered, and detailing further on the MCDA models used in the reaction processes. The future work for both simulation case studies is currently outlined as: 1) the investigation of additional indicators that can provide a more granular view of the conditions to be used in the observation and detection activities, 2) the calibration of reaction thresholds and re-definition of desirable system behaviour, 3) the investigation of the theory of fuzzy subsets in the context of threshold definitions, effects of employing such an approach on the overall system behaviour, and comparison of this approach with the current crisp definitions, 4) the investigation of additional decision criteria, their evaluation models and the effects of these new criteria on the efficiency of the self-management process, and 5) employing several other MCAP based MCDA methods to the problems at hand. Furthermore, another short-term goal is to provide an in-depth analysis and validation of the simulation platform that is designed and implemented for the assessment of IMPROMPTU.

In the long-term the proposed methodology is to be applied to other self-management problems in distributed systems of varying scales in order to provide a more general assessment of its applicability. Currently, we aim at applying the proposed methodology to the following problems: 1) domain name system (DNS) security, and 2) transmission power control in wireless networks. The first problem is another example to independent self-management, whereas the second problem can be treated as both an independent self-management and coordinated self-management problem.

In the case of the DNS security problem, the main idea is to use the proposed methodology to design and implement a light-weight DNS client—which requires no modifications to the underlying infrastructure units, such as primary and secondary DNS servers—that provides secure DNS resolutions through MCDA. The traditional DNS works in a way that is rather vulnerable to various types of attacks since it blindly trusts the responses to queries [157, 211, 4, 210]. This vulnerability was attempted to be alleviated by DNSSEC [13, 14, 15] which also had certain issues mainly

related to difficulties in deployment [157, 211]. Some of the problems that are inherently existent in DNS—both security and fault tolerance related—are addressed in a number of later studies. Accordingly, various new name resolution schemes such as CoDNS [200, 147], CoDoNS [165, 164] were proposed to deal with the issues with respect to DNS's shortcomings in fault tolerance. Whereas, other systems such as Byzantine Fault Tolerant DNS [43, 44, 4, 210] and ConfiDNS [157, 211] were introduced to deal with security related issues where the process of domain name resolution depends on responses from multiple DNS servers. In this approach, the collected responses were assessed based on historical or cross-referenced data in order to provide a trustworthy resolution that can be used by the application that requested the network address of the domain name. The system based on the proposed methodology is to adopt a similar approach with respect to collecting multiple resolutions. Differently from the former approaches, the system is to employ MCDA during the process of selecting the most suitable resolution as the final result. This requires the implementation of a light-weight client that wraps the already existing DNS client in order to facilitate the activities that are outlined by the proposed methodology.

In the case of transmission power control in wireless network, the main idea is to use the proposed methodology to design and implement autonomous units to control the transmission power on wireless nodes through MCDA in order to reduce collision rates in wireless networks. This problem can be addressed both as an independent self-management or a coordinated self-management problem. In the independent self-management view, the nodes adjust their transmission power levels without the consent of the rest of the nodes in the network. Whereas, in the coordinated self-management view, the nodes coordinate to use the same transmission power level throughout the system. This problem has also been addressed in the previous studies where mainly principles of control theory and optimization are used [112, 143]. The system to be created can be assessed on both simulated or simple physical settings [213].

Bibliography

[1] Janet Abbate. *Inventing the Internet*. MIT Press, 2000.

[2] T. Abdelhazer, Y. Diao, J. L. Hellerstein, C. Lu, and X. Zhu. Introduction to Control Theory and Its Application to Computing Systems. In *Performance Modeling and Engineering*, 2008.

[3] Shubham Agrawal, Sumit Kumar Bose, and Srikanth Sundarrajan. Grouping genetic algorithm for solving the serverconsolidation problem with conflicts. In *GEC'09: The First ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, pages 1–8. ACM, 2009.

[4] S. Ahmed. A Scalable Byzantine Fault Tolerant Secure Domain Name System. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusets Institute of Technology, 2001.

[5] Akamai. http://www.akamai.com, Retreived 2011.

[6] J. Almeida, V. Almeida, D. Ardagna, C. Francalanci, and M. Trubian. Resource Management in the Autonomic Service-Oriented Architecture. In *ICAC '06: IEEE International Conference on Autonomic Computing*, pages 84–92, 2006.

[7] Amazon. Elastic Compute Cloud (Amazon EC2), http://aws.amazon.com/ec2, Retreived 2011.

[8] Amazon. Simple Storage Service (Amazon S3), http://aws.amazon.com/s3, Retreived 2011.

[9] Eric Anderson, Michael Hobbs, Kimberly Keeton, Susan Spence, Mustafa Uysal, and Alistair Veitch. Hippodrome: Running circles around storage administration. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, FAST '02, Berkeley, CA, USA, 2002. USENIX Association.

[10] Eric Anderson, Mahesh Kallahalla, Susan Spence, Ram Swaminathan, and Qian Wang. Quickly finding near-optimal storage system designs. *ACM Transactions on Computer Systems (TOCS)*, 23(4):337—374, November 2005.

[11] James A. Anderson. *An Introduction to Neural Networks*. MIT Press, 1995.

[12] Gregory R. Andrews. *Foundations of Multithreaded, Parallel, and Distributed Programming*. Addison-Wesley, 2000.

[13] P. Arends, R. Austein, M. Larson, D. Masset, and S. Rose. RFC 4033: DNS Security Introduction and Requirements. Technical report, Internet Engineering Task Force, 2005.

[14] P. Arends, R. Austein, M. Larson, D. Masset, and S. Rose. RFC 4034: Resource Records for the DNS Security Extensions. Technical report, Internet Engineering Task Force, 2005.

[15] P. Arends, R. Austein, M. Larson, D. Masset, and S. Rose. RFC 4035: Protocol Modifications for the DNS Security Extensions. Technical report, Internet Engineering Task Force, 2005.

[16] S. M. Baas and H. Kwakernak. Rating and Ranking of Multiple Aspect Alternative Using Fuzzy Sets. *Automatica*, 13:47—58, 1977.

[17] Ed Barbeau. Perron's Result and a Decision on Admissions Tests. *Mathematics Magazine*, 59(1):12—22, 1986.

[18] Nils Aall Barricelli. Esempi numerici di processi di evoluzione,. *Methodos*, pages 45—68, 1954.

[19] Jim Basney and Miron Livny. Improving goodput by co-scheduling cpu and network capacity. *International Journal of High Performance Computing Applications*, 13(3), 1999.

[20] Jim Basney and Miron Livny. Managing network resources in Condor. In *Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9)*, pages 298–299, Pittsburgh, PA, August 2000.

[21] Jim Basney, Miron Livny, and Paolo Mazzanti. Utilizing widely distributed computational resources efficiently with execution domains. *Computer Physics Communications*, 140(1-2):246+, October 2001.

[22] Mohamed N. Bennani and Daniel A. Menasce. Resource Allocation for Autonomic Data Centers using Analytic Performance Models. In *ICAC '05: Proceedings of the Second International Conference on Automatic Computing*, pages 229–240, 2005.

[23] Norman Bobroff, Andrzej Kochut, and Kirk Beaty. Dynamic Placement of Virtual Machines for Managing SLA Violations. In *IM '07: 10th IFIP/IEEE International Symposium on Integrated Network Management*, pages 119–128, 2007.

[24] Jeff Boleng. Efficient Network Layer Addressing for Mobile Ad Hoc Networks. In *International Conference on Wireless Networks (ICWN'02)*, pages 271–277, 2002.

[25] Jeff Boleng. Metrics to Enable Adaptive Protocols for Mobile Ad Hoc Networks. In *International Conference on Wireless Networks (ICWN'02)*, pages 293–298, 2002.

[26] Piero P. Bonissone, Raj Subbu, and John Lizzi. Multicriteria decision making (mcdm): a framework for research and applications. *Comp. Intell. Mag.*, 4:48–61, August 2009.

[27] P.P. Bonissone. Multi-criteria decision-making: The intersection of search, preference tradeoff, and interaction visualization processes. In *Computational Intelligence in Multicriteria Decision Making, IEEE Symposium on*, page 1, april 2007.

[28] E. Börger and R. Stärk. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer-Verlag, 2003.

[29] Egon Börger. Construction and Analysis of Ground Models and their Refinements as a Foundation for Validating Computer Based Systems. *Formal Aspects of Computing*, 19(2):225–241, 2007.

[30] D. Bouyssou. Some Remarks on the Notion of Compensation in MCDM. *European Journal of Operational Research*, 26:150—160, 1986.

[31] D. Bouyssou, P. Perny, M. Pirlot, A. Tsoukias, and P. Vincke. A Manifesto for the New MCDA Era. *Journal of Multi-Criteria Decision Analysis*, 2:251—127, 1993.

[32] Denis Bouyssou and Marc Pirlot. Conjoint Measurement Tools for MCDM: A Brief Introduction. In *Multiple Criteria Decision Analysis: State of the Art Surveys*, pages 73—112. Springer New York.

[33] J. P. Brans. *L'Ingéniérie de la Décision. Elaboration d'Instruments d'Aide à la Décision. Méthode PROMETHEE',*. PhD thesis, Université Laval, Québec, Canada, 1982.

[34] J. P. Brans, B. Mareschal, and P. Vincke. How to Select and How to Rank Projects : The PROMETHEE Method for MCDM. *European Journal of Operations Research*, 24:228—238.

[35] J. P. Brans, B. Mareschal, and P. Vincke. PROMETHEE : A new family of outranking methods in MCDM. In *IFORS'84*, 1984.

[36] Jean-Pierre Brans and Bertrand Mareschal. Promethee methods. In *Multiple Criteria Decision Analysis: State of the Art Surveys*, pages 163—186. Springer New York.

[37] J.P Brans and B. Mareschal. PROMCALC & GAIA : A New Decision Support System for Multicriteria Decision Aid. *Decision Support Systems*, 12:297—310.

[38] J.P Brans and B. Mareschal. PROMETHEE V : MCDM Problems with Additional Segmentation Constraints. *INFOR*, 30(2):85—96.

[39] J.P Brans, B. Mareschal, and P. Vincke. A Preference Ranking Organisation Method : The PROMETHEE Method for MCDM. *Management Science*, 31(6):647—656.

[40] P. J. Braspenning, F. Thuijsman, and A. J. M. M. Weijters. *Artificial Neural Networks: An Introduction to ANN Theory and Practice*. Springer, 1995.

[41] Josh Broch, David A. Maltz, David B. Johnson, Yih Chun Hu, and Jorjeta Jetcheva. A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. In *ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'98)*, pages 85–97, October 1998.

[42] C. Bana e Costa. *Readings in Multiple Criteria Decision Aid*. Springer Verlag, 1990.

[43] M. Castro. *Practical Byzantine Fault Tolerance*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusets Institute of Technology, 2000.

[44] M. Castro and B. Liskov. Authenticated Byzantine Fault Tolerance without Public Key Cryptography. Technical report, Laboratory for Computer Science, Massachusets Institute of Technology, June 1999.

[45] A. Chames and W. Cooper. *Management Models and Industrial Applications of Linear Programming, Volume I.* John Wiley and Sons, 1961.

[46] Abhishek Chandra, Weibo Gong, and Prashant Shenoy. Dynamic resource allocation for shared data centers using online measurements. *SIGMETRICS Perform. Eval. Rev.*, 31(1):300–301, 2003.

[47] Ann Chervenak, Ewa Deelman, Miron Livny, Mei-Hui Su, Rob Schuler, Shishir Bharathi, Gaurang Mehta, and Karan Vahi. Data placement for scientific applications in distributed environments. In *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing (Grid 2007)*, Austin, TX, September 2007.

[48] D.M. Chess, A. Segal, I. Whalley, and S.R. White. Unity: Experiences with a Prototype Autonomic Computing System. In *2004. Proceedings. International Conference on Autonomic Computing*, pages 140–147, 2004.

[49] A. Chipperfield, J. Whidborne, and P. Fleming. Evolutionary Algorithms and Simulated Annealing for MCDM. In *Multicriteria Decision Making: Advances in MCDM Models, Algorithms, Theory and Applications*, pages 16.1—16.32. Kluwer Academic Publishers, 1999.

[50] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.

[51] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol (OLSR). Request for Comments (RFC) 3626, Internet Engineering Task Force, October 2003.

[52] Retreived 2011 CloudSim, http://www.buyya.com/gridbus/cloudsim/.

[53] Retreived 2011 CobWeb, http://www.cs.cornell.edu/people/egs/beehive/cobweb/.

[54] CoDeeN. http://codeen.cs.princeton.edu/, retreived 2011.

[55] B. Jack Copeland and Diane Proudfoot. On alan turing's anticipation of connectionism. *Synthese*, 108(3):361—377, March 1996.

[56] R. Das, J.O. Kephart, I.N. Whalley, and P. Vytas. Towards Commercialization of Utility-based Resource Allocation. In *ICAC '06: IEEE International Conference on Autonomic Computing*, pages 287–290, 2006.

[57] Samir R. Das, Charles E. Perkins, and Elizabeth M. Royer. Performance Comparison of Two On-demand Routing Protocols for Ad Hoc Networks. In *Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'00)*, volume 1, pages 3–12, March 2000.

[58] C. Van de Panne. *Methods for Linear and Quadratic Programming*. North-Holland Publishing Company, 1975.

[59] D. K. Despotis, D. Yannacopoulos, and Zopounidis. A Review of the UTA Multicriteria Method and Some Improvements. *Foundations of Computing and Decision Sciences*, 15(2):63—76, 1990.

[60] Shlomi Dolev. *Self-Stabilization*. MIT Press, 2000.

[61] Linda E. Doyle, Anil C. Kokaram, Senan J. Doyle, and Timothy K. Forde. Ad Hoc Networking, Markov Random Fields, and Decision Making. *IEEE Signal Processing Magazine*, 23(5):63–73, September 2006.

[62] Xavier Dutreilh, Aurlien Moreau, Jacques Malenfant, Nicolas Rivierre, and Isis Truck. From Data Center Resource Allocation to Control Theory and Back. In *IEEE CLOUD: 2010 IEEE 3rd International Conference on Cloud Computing*, 2010.

[63] J. S. Dyer and R. A. Sarin. Measurable Multiattribute Value Functions. *Operations Research*, 22:810—822, 1979.

[64] James Dyer. Maut—multiattribute utility theory. In *Multiple Criteria Decision Analysis: State of the Art Surveys*, pages 265—292. Springer New York.

[65] Carlos Bana e Costa, Jean-Marie De Corte, and Jean-Claude Vansnick. On the mathematical foundations of macbeth. In *Multiple Criteria Decision Analysis: State of the Art Surveys*, pages 409—437. Springer New York.

[66] W. Edwards. How to Use Multiattribute Measurement for Social Decision Making. *IEEE Transactions on Systems, Man and Cybernetics*, 5:326—340, 1977.

[67] W. Edwards and F. H. Barron. SMARTS and SMARTER: Improved Simple Methods for Multiattribute Utility Measurement. *Organizational Behaviour and Human Decision Process*, 60:306—325, 1994.

[68] EGEE. Enabling Grids fo E-Science, http://www.eu-egee.org/, Retreived 2011.

[69] Emulab. http://www.emulab.net, Retreived 2011.

[70] Sholomo Engelberg. *Series in Electrical and Computer Engineering—Vol. 2: A Mathematical Introduction to Control Theory*. Imperial College Press, 2005.

[71] D.H.J. Epema, M. Livny, R. van Dantzig, X. Evers, and J. Pruyne. A worldwide flock of Condors: Load sharing among workstation clusters. *Future Generation Computer Systems*, 12:53–65, 1996.

[72] R. Farahbod, V. Gervasi, and U. Glässer. CoreASM: An Extensible ASM Execution Engine. *Fundamenta Informaticae*, pages 71–103, 2007.

[73] R. Farahbod and U. Glässer. Semantic Blueprints of Discrete Dynamic Systems: Challenges and Needs in Computational Modeling of Complex Behavior. In *New Trends in Parallel and Distributed Computing, Proc. 6th Intl. Heinz Nixdorf Symposium, Jan. 2006*, pages 81–95. Heinz Nixdorf Institute, 2006.

[74] Peter H. Farquhar. A Survey of Multiattribute Utility Theory and Applications. In *Multiple Criteria Decision Making: Volume 6 of TIMS Studies in the Management Sciences*, pages 59–90. 1977.

[75] Jacques Ferber. *Multi-Agent System: An Introduction to Distributed Artificial Intelligence*. Harlow: Addison Wesley Longman, 1999.

[76] FidoNet. http://www.fidonet.org/, Retreived 2011.

[77] José Figueira, Vincent Mousseau, and Bernard Roy. Electre methods. In *Multiple Criteria Decision Analysis: State of the Art Surveys*, pages 133—153. Springer New York.

[78] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[79] P. C. Fishburn. Independence in Utility with Whole Product Sets. *Operations Research*, 13:205—217, 1965.

[80] P. C. Fishburn. *Utility Theory for Deision Making*. John Wiley and Sons, 1970.

[81] Sally Floyd and Vern Paxson. Difficulties in Simulating the Internet. *IEEE/ACM Transactions on Networking*, 9(4):392–403, 2001.

[82] Timothy K. Forde, Linda E. Doyle, and Donal O Mahony. Self-Stabilizing Network-Layer Auto-Configuration for Mobile Ad Hoc Network Nodes. In *IEEE International Conference on Wireless And Mobile Computing, Networking And Communications (WiMob'05)*, volume 3, pages 178–185, August 2005.

[83] Timothy K. Forde, Linda E. Doyle, and Donal O Mahony. Ad Hoc Innovation: Distributed Decision Making in Ad Hoc Networks. *IEEE Communications Magazine*, 44(4):131–137, April 2006.

[84] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud Computing and Grid Computing 360-Degree Compared. In *GCE: Grid Computing Environments Workshop*, pages 1–10, 2008.

[85] Freifunk. http://start.freifunk.net, Retreived 2011.

[86] L. Gardenier and D. Vanderpooten. Interactive Multiple Criteria Procedures: Some Reflections. In *Multicriteria Analysis*, pages 290—301. Springer Verlag, 1997.

[87] Sukumar Ghosh. *Distributed Systems: An Algorithmic Approach*. Chapman & Hall/CRC, November 2006.

[88] Joseph C. Giarratano and Gary D. Riley. *Expert Systems: Principles and Programming*. Course Technology, 2004.

[89] Bill Godfrey. A Primer on Distributed Computing. http://www.bacchae.co.uk/docs/dist.html, Retreived 2011.

[90] S. Greco, B. Matarazzo, and R. Slowinski. The Use of Rough Sets and Fuzzy Sets in MCDM. In *Multicriteria Decision Making: Advances in MCDM Models, Algorithms, Theory and Applications*, pages 14.1—14.59. Kluwer Academic Publishers, 1999.

[91] David Alan Grier. Someday, You Will Understand. *Computer*, 43:11–13, 2010.

[92] Adel Guitouni and Jean-Marc Martel. Tentative Guidelines to Help Choosing an Appropriate MCDA Method. *European Journal of Operational Research*, 109(2):501—521, 1998.

[93] Kevin Gurney. *An Introduction to Neural Networks*. CRC Press, 1997.

[94] Fabien Hermenier, Xavier Lorca, Jean M. Menaud, Gilles Muller, and Julia Lawall. Entropy: A Consolidation Manager for Clusters. In *VEE '09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pages 41–50, 2009.

[95] Elisa Heymann, Miquel A. Senar, Emilio Luque, , and Miron Livny. Adaptive scheduling for master-worker applications on the computational grid. In *Proceedings of the First IEEE/ACM International Workshop on Grid Computing (GRID 2000)*, Bangalore, India, December 2000.

[96] Elisa Heymann, Miquel A. Senar, Emilio Luque, , and Miron Livny. Evaluation of an adaptive scheduling strategy for master-worker applications on clusters of workstations. In *Proceedings of the 7th International Conference on High Performance Computing (HiPC 2000)*, Bangalore, India, December 2000.

[97] John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[98] C. L. Hwang and K. Yoon. *Multiple Attribute Decision Making—Methods and Applications: A State-of-the-Art Survey*. Springer Verlag, 1981.

[99] IBM. Autonomic Computing: IBM's Perspective on the State of Information Technology. Technical report, IBM Research, 2011.

[100] INETMANET. https://github.com/inetmanet/inetmanet Retrieved, 2011.

[101] Peter Jackson. *Introduction to Expert Systems*. Addison Wesley, 1998.

[102] E. Jacquet-Lagrèze and Y. Siskos. Assessing a Set of Additive Utility Functions fo Multicriteria Decision Making: The UTA Method. *European Journal of Operational Research*, 10(2):151—164, 1982.

[103] D. Johnson, Y. Hu, and D. Maltz. The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4. Request for Comments (RFC) 4728, Internet Engineering Task Force, February 2007.

[104] D. Kahneman and A. Tversky. Prospect Theory: An Analysis of Decisions under Risk. *Econometrica*, 47:276—287.

[105] R. L. Keeney and H. Raiffa. *Decision with Multiple Objectives: Preference and Value Tradeoffs*. Cambridge University Press, 1993.

[106] Jeffrey O. Kephart and David M. Chess. The Vision of Autonomic Computing. *Computer*, 36:41–50, January 2003.

[107] G. Khanna, K. Beaty, G. Kar, and A. Kochut. Application Performance Management in Virtualized Server Environments. In *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, pages 373–381, 2006.

[108] Aaron Kimball. Problem Solving on Large-Scale Clusters, http://code.google.com/edu/submissions/mapreduce-minilecture/listing.html, Retreived 2011.

[109] A. Kochut. On Impact of Dynamic Virtual Machine Reallocation on Data Center Efficiency. In *Modeling, Analysis and Simulation of Computers and Telecommunication Systems, 2008. MASCOTS 2008. IEEE International Symposium on*, pages 1 –8, 2008.

[110] George Kola, Tevfik Kosar, and Miron Livny. A fully automated fault-tolerant system for distributed video processing and off-site replication. In *Proceedings of the 14th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2004)*, Kinsale, Ireland, June 2004.

[111] George Kola, Tevfik Kosar, and Miron Livny. Run-time adaptation of grid data-placement jobs. *Parallel and Distributed Computing Practices*, 2004.

[112] A. Konig, M. Hollick, and R. Steinmetz. On the Implications of Adaptive Transmission Power for Assisting MANET Security. In *Distributed Computing Systems Workshops, 2009. ICDCS Workshops '09. 29th IEEE International Conference on*, pages 537 –544, june 2009.

[113] Timo Kosch, Christian Adler, Stephan Eichler, Christoph Schroth, and Markus Strassberger. The Scalability Problem of Vehicular Ad Hoc Networks and How to Solve It. *IEEE Wireless Communications Magazine*, 2006.

[114] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[115] J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, 1994.

[116] D. H. Krantz, R. D. Luce, P. Suppes, and A. Tversky. *Foundations of Measurement. Volume I: Additive and Polynomial Representations*. Academic Press, 1971.

[117] D. M. Kreps. *A Course in Microeconomic Theory*. Princeton University Press, 1990.

[118] Byung-Jae Kwak and Nah-Oak Song. A Mobility Measure for Mobile Ad Hoc Networks. *IEEE Communication Letters*, 7(8):379–381, August 2003.

[119] J. P. Leclerc. Propositions dExtension de la Notion de Dominance en Présence de Relations dOrdre sur les Pseudo-critères: MELCHIOR. *Mathematical Social Sciences*, 8:45—61, 1984.

[120] Alexander Lenk, Markus Klems, Jens Nimis, Stefan Tai, and Thomas Sandholm. What's Inside the Cloud? An Architectural Map of the Cloud Landscape. In *ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pages 23–31, 2009.

[121] Antoni Ligeza. *Logical Foundations for Rule-Based Systems*. Springer, 2010.

[122] Henry Linger. *Constructing The Infrastructure For The Knowledge Economy: Methods and Tools, Theory and Practice*. 2003.

[123] Michael Litzkow. Remote unix - turning idle workstations into cycle servers. In *Usenix Summer Conference*, pages 381–384, 1987.

[124] Michael Litzkow, Miron Livny, and Matthew Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.

[125] Miron Livny, Jim Basney, Rajesh Raman, and Todd Tannenbaum. Mechanisms for high throughput computing. *SPEEDUP Journal*, 11(1), June 1997.

[126] Miron Livny and Myron Melman. Load balancing in homogeneous broadcast distributed systems. In *Proceedings of the Computer Network Performance Symposium*, College Park, Maryland, April 1982.

[127] Miron Livny and Rajesh Raman. High-throughput resource management. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.

[128] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[129] Deying Li Maggie Xiaoyan Cheng. *Advances in Wireless Ad Hoc and Sensor Networks*. Springer, 2008.

[130] Sai Rajesh Mahabhashyam. Dynamic Resource Allocation of Shared Data Centers Supporting Multiclass Requests. In *ICAC '04: Proceedings of the First International Conference on Autonomic Computing*, pages 222–229. IEEE Computer Society, 2004.

[131] B. Mareschal. Aide a la Decision Multicritere: Developpements Recents des Methodes PROMETHEE. *Cahiers du Centre d'Etudes en Recherche Operationelle*, pages 175–241, 1987.

[132] J. M. Martel and B. Matarazzo. Other outranking approaches. In *Multiple Criteria Decision Analysis: State of the Art Surveys*, pages 197—259. Springer New York.

[133] Matthew L. Massie, Brent N. Chun, and David E. Culler. The Ganglia Distributed Monitoring System: Design, Implementation And Experience. *Parallel Computing*, 30, 2003.

[134] L. Maystre, J. Pictet, and J. Simos. *Les Méthodes Multicritères ELECTRE*. Presses Polytechniques et Universitaires Romandes, 1994.

[135] John McCarthy. What is Artificial Intelligence?, http://www-formal.stanford.edu/jmc/whatisai/, Retreived 2011.

[136] Daniel A. Menasce and Mohamed N. Bennani. Autonomic Virtualized Environments. In *ICAS: Proceedings of the International Conference on Autonomic and Autonomous Systems*, pages 28–37. IEEE Computer Society, 2006.

[137] Patrick Meyer and Marc Roubens. Choice, ranking and sorting in fuzzy multiple criteria decision aid. In *Multiple Criteria Decision Analysis: State of the Art Surveys*, pages 471—503. Springer New York.

[138] Cary Millsap. Thinking Clearly About Performance, Part 2. *Communications of the ACM*, 53(10):39–45, 2010.

[139] P. Mockapetris. RFC 1034: Domain Names—Concepts and Facilities. Technical report, Internet Engineering Task Force, 1987.

[140] P. Mockapetris. RFC 1035: Domain Names—Implementation and Specification. Technical report, Internet Engineering Task Force, 1987.

[141] Gordon E. Moore. Cramming More Components onto Integrated Circuits. *Electronics*, 38(8), April 1965.

[142] Erik G. Nilsson and Ketil Stolen. Ad Hoc Networks and Mobile Devices in Emergency Response—A Perfect Match? In *Ad Hoc Networks*, volume 49 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 17–33. Springer Berlin Heidelberg, 2010.

[143] R. Oda, T. Ohta, and Y. Kakuda. An Adaptive Transmission Power Control for Hierarchical Routing in Mobile Ad Hoc Networks. In *Distributed Computing Systems Workshops, 2009. ICDCS Workshops '09. 29th IEEE International Conference on*, pages 312 –317, june 2009.

[144] OLPC. One Laptop Per Child, http://one.laptop.org/, Retreived 2011.

[145] OMNET++. http://www.omnetpp.org Retrieved, 2010.

[146] Frans Osinga. *Science Strategy and War, The Strategic Theory of John Boyd*. Routledge.

[147] Kyoungsoo Park, Vivek S. Pai, Larry Peterson, and Zhe Wang. CoDNS: Improving DNS Performance and Reliability via Cooperative Lookups. In *In Proceedings of the Sixth Symposium on Operating Systems Design and Implementation (OSDI*, 2004.

[148] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, 2000.

[149] C. Perkins, E. Belding-Royer, and S. Das. Ad Hoc On-Demand Distance Vector (AODV) Routing. Request for Comments (RFC) 3561, Internet Engineering Task Force, July 2003.

[150] Dmitri D. Perkins, Herman D. Hughes, and Charles B. Owen. Factors Affecting the Performance of Ad Hoc Networks. In *IEEE International Conference on Communications (ICC'02)*, volume 4, pages 2048–2052, April 2002.

[151] P. Perny and J.Ch. Pomerol. Use of Artificial Intelligence in MCDM. In *Multicriteria Decision Making: Advances in MCDM Models, Algorithms, Theory and Applications*, pages 15.1—15.43. Kluwer Academic Publishers, 1999.

[152] P. Perny and D. Vanderpooten. An Interactive Multiobjective Procedure for Selecting Medium-Term Counter Measures After Nuclear Accidents. *Journal of Multi-Criteria Decision Analysis*, 7(1):48—60, 1998.

[153] Ian Peter. Ian Peter's History of the Internet. http://www.nethistory.info/History of the Internet, Retreived 2011.

[154] PlanetLab. http://www.planet-lab.org, Retreived 2011.

[155] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A Field Guide to Genetic Programming*. Lulu ENterprises, 2008.

[156] David Poole, Alan Macworth, and Randy Goebel. *Computational Intelligence: A Logical Approach*. Oxford University Press, 1998.

[157] Lindsey Poole and Vivek S. Pai. ConfiDNS: Leveraging Scale and History to Detect Compromise. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 99–112, Berkeley, CA, USA, 2008. USENIX Association.

[158] Gerald J. Popek and Robert P. Goldberg. Formal Requirements for Virtualizable Third Generation Architectures. *ACM Communications*, 17(7):412–421, 1974.

[159] James C. Pruyne. *Resource Management Services for Parallel Applications*. PhD thesis, University of Wisconsin-Madison, 1996.

[160] Jim Pruyne and Miron Livny. Managing checkpoints for parallel programs. In *Workshop on Job Scheduling Strategies for Parallel Processing (IPPS '96)*, Honolulu, HI, April 1996.

[161] Rajesh Raman, Miron Livny, and Marvin Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC7)*, Chicago, IL, July 1998.

[162] Rajesh Raman, Miron Livny, and Marvin Solomon. Resource management through multilateral matchmaking. In *Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9)*, pages 290–291, Pittsburgh, PA, August 2000.

[163] Ram Ramanathan and Jason Redi. A Brief Overview of Ad Hoc Networks: Challenge and Directions. *IEEE Communications Magazine, 50th Anniversary Commemorative Issue*, May 2002.

[164] Venugopalan Ramasubramanian and Emin Gün Sirer. Beehive: O(1)lookup performance for power-law query distributions in peer-to-peer overlays. In *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1*, pages 8–8, Berkeley, CA, USA, 2004.

[165] Venugopalan Ramasubramanian and Emin Gün Sirer. The design and implementation of a next generation name service for the internet. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols*

for computer communications, SIGCOMM '04, pages 331–342, New York, NY, USA, 2004.

[166] Susan Rea and Dirk Pesch. Multi-Metric Routing Decisions for Ad Hoc Networks Using Fuzzy Logic. Technical report, Cork Institute of Technology, 2000.

[167] Chet Richards. *Certain to Win: the Strategy of John Boyd, Applied to Business*. 2004.

[168] B. Roy and V. Mousseau. A Theoretical Framework for Analysing the Notion of Relative Importance of Criteria. *Journal of Multi Criteria Decision Analysis*, 5:145—159, 1996.

[169] Bernard Roy. Paradigms and challenges. In *Multiple Criteria Decision Analysis: State of the Art Surveys*, pages 3—24. Springer New York.

[170] Bernard Roy. Classement et Choix en Présence de Points de Vue Multiples (La Méthode ELECTRE. *RIRO*, 8:57—75, 1968.

[171] Bernard Roy. The Outranking Approach and the Foundations of ELECTRE. *Theory and Decision*, 31:49—73, 1991.

[172] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003.

[173] Thomas Saaty. Analytic Hierarchy and Analytic Network Processes for the Measurement of Intangible Criteria and for Decision Making. In *Multiple Criteria Decision Analysis: State of the Art Surveys*, pages 345—405. Springer New York.

[174] Thomas Saaty. A Scaling Method for Priorities in Hierarchical Structures. *Journal of Mathematical Psychology*, 15:234—280, 1977.

[175] Thomas Saaty. *The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation*. McGraw-Hill, 1980.

[176] Thomas Saaty. Ranking According to Perron: A New Insight. *Mathematics Magazine*, 60(4):211—213, 1987.

[177] Leonard Savage. *The Foundations of Statistics*. John Wiley and Sons, 1954.

[178] Mark Silberstein, Dan Geiger, Assaf Schuster, and Miron Livny. Scheduling mixed workloads in multi-grids: The grid execution hierarchy. In *Proceedings of the 15th IEEE Symposium on High Performance Distributed Computing (HPDC-15)*, Paris, France, June 2006.

[179] Y. Siskos. A Way to Deal with Fuzzy Preferences in Multicriteria Decision Problems. *European journal of Operational Research*, 10(3):314—324, 1982.

[180] Y. Siskos. Analyse de Systemes de Deciosion Multicritere en Univers Aleatoire. *Foundations of Control Engineering*, 10(3-4):193—212, 1983.

[181] Y. Siskos and D. Yannacopoulos. An Ordinal Regression Method for Building Additive Value Functions. *Investigacao Operacional*, 5(1):39—53, 1985.

[182] Yannis Siskos, Evangelos Grigoroudis, and Nikolaos Matsatsinis. UTA Methods. In *Multiple Criteria Decision Analysis: State of the Art Surveys*, pages 297—334. Springer New York.

[183] R. Slowinski. *Intelligent Decision Support: Handbook of Applications and Advances of the Rough Sets Theory*. Kluwer Academic Publishers, 1992.

[184] T. Steward. Concepts of Interactive Programming. In *Multicriteria Decision Making: Advances in MCDM Models, Algorithms, Theory, and Applications*, pages 7.1—7.29. Kluwer Academic Publishers, 1999.

[185] R. Subbu, P. Bonissone, N. Eklund, Weizhong Yan, N. Iyer, Feng Xue, and R. Shah. Management of complex dynamic systems based on model-predictive multi-objective optimization. In *Computational Intelligence for Measurement Systems and Applications, Proceedings of 2006 IEEE International Conference on*, pages 64 –69, july 2006.

[186] Andrew S. Tanenbaum and Maarten Van Steen. *Distributed Systems: Principles and Paradigms*. Pearson Prentice Hall, 2006.

[187] Todd Tannenbaum, Derek Wright, Karen Miller, and Miron Livny. Condor – a distributed job scheduler. In Thomas Sterling, editor, *Beowulf Cluster Computing with Linux*. MIT Press, October 2001.

[188] G. Tesauro, R. Das, W.E. Walsh, and J.O. Kephart. Utility-Function-Driven Resource Allocation in Autonomic Systems. In *Autonomic Computing, 2005.*

ICAC 2005. Proceedings. Second International Conference on, pages 342–343, 2005.

[189] G. Tesauro, N.K. Jong, R. Das, and M.N. Bennani. A Hybrid Reinforcement Learning Approach to Autonomic Resource Allocation. In *ICAC '06: Proceedings of the 2006 IEEE International Conference on Autonomic Computing*, pages 65–73. IEEE Computer Society, 2006.

[190] Gerald Tesauro. Online Resource Allocation Using Decompositional Reinforcement Learning. In *AAAI'05: Proceedings of the 20th National Conference on Artificial Intelligence*, pages 886–891. AAAI Press, 2005.

[191] Alan Turing. *Collected Works of A. M. Turing: Mechanical Intelligence*. Elsevier Science Publishers, 1992.

[192] David G. Ullman. *Making Robust Decisions: Decision Management For Technical, Business, and Service Teams*. Trafford Publishing.

[193] Usenet. www.usenet.com/, Retreived 2011.

[194] Hien Nguyen Van and Frederic Dang Tran. Autonomic Virtual Resource Management for Service Hosting Platforms. In *ICES '09: Proceedings of the International Conference on Software Enginnering Workshop on Software Engineering Challenges of Cloud Computing*, pages 1–8. IEEE Computer Society, 2009.

[195] D. Vanderpooten. The Interactive Approach in MCDA: A Technical Framework and Some Basic Conceptions. *Mathematical and Computer Modelling*, 12:1213—1220, 1989.

[196] Laurent Viennot, Philippe Jacquet, and Thomas Heide Clausen. Analyzing Control Traffic Overhead versus Mobility and Data Traffic Activity in Mobile Ad-Hoc Network Protocols. *Wireless Networks*, 10(4):447–455, 2004.

[197] P. Vincke. L'aide Multicritère à la Dècision. *Éditions de l'Université de Bruxelles*.

[198] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behaviour*. Princeton University Press, 1947.

[199] William E. Walsh, Gerald Tesauro, Jeffrey O. Kephart, and Rajarshi Das. Utility Functions in Autonomic Systems. In *ICAC '04: Proceedings of the First International Conference on Autonomic Computing*, pages 70–77. IEEE Computer Society, 2004.

[200] Limin Wang, Kyoung Soo Park, Ruoming Pang, Vivek Pai, and Larry Peterson. CoDNS: Masking DNS Delays via Cooperative Lookups. Technical Report TR-690-04, Princeton University Computer Science Department.

[201] XiaoYing Wang, DongJun Lan, Gang Wang, Xing Fang, Meng Ye, Ying Chen, and QingBo Wang. Appliance-Based Autonomic Provisioning Framework for Virtualized Outsourcing Data Center. In *Autonomic Computing, 2007. ICAC '07. Fourth International Conference on*, pages 29–29, 2007.

[202] Yongwei Wang, Venkata C. Giruka, and Mukesh Singhal. Truthful Multipath Routing for Ad Hoc Networks with Selfish Nodes. *Elsevier Journal of Parallel and Distributed Computing*, 68(6):778–789, June 2008.

[203] Julie Ward, Michael OSullivan, Troy Shahoumian, and John Wilkes. Appia: Automatic storage area network fabric design. In *Conference on File and Storage Technology*, 2002.

[204] Gerhard Weikum. *Transactional Information Systems*. Elsevier, 2001.

[205] Jan Wessnitzer and Chris Melhuish. Collective Decision-Making and Behaviour Transitions in Distributed Ad Hoc Wireless Networks of Mobile Robots: Target-Hunting. *Robotics and Autonomous Agents*, pages 893–902, February 2004.

[206] Wayne L. Winston. *Operations Research: Applications and Algorithms*. Duxbury Press, 1994.

[207] Rolf Winter, Jochen H. Schiller, Navid Nikaein, and Christian Bonnet. CrossTalk: Cross-Layer Decision Support Based on Global Knowledge. *IEEE Communications Magazine*, 44(1):93–99, January 2006.

[208] Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. Black-Box and Gray-Box Strategies for Virtual Machine Migration. In *4th USENIX Symposium on Networked Systems Design and Implementation*, pages 229–242, 2007.

[209] Michael Wooldridge. *An Introduction Multiagent Systems.* John Wiley and Sons, 2009.

[210] Z. Yang. Using a Byzantine Fault Tolerant Algorithm to Provide a Secure DNS. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusets Institute of Technology, 1999.

[211] Yağız Onat Yazır. On the Virtues and Liabilities of ConfiDNS: Can Simple Tactics Overcome Deep Insecurities? Master's thesis, University of Victoria, 2007.

[212] Yağız Onat Yazır, Roozbeh Farahbod, Adel Guitouni, Sudhakar Ganti, and Yvonne Coady. Adaptive Routing in Mobile Ad Hoc Networks Based on Decision Aid Approach. In *Proceedings of the 8th ACM international workshop on Mobility management and wireless access*, MobiWac '10, pages 1–10, New York, NY, USA, 2010. ACM.

[213] Yağız Onat Yazır, K. Jahanbakhsh, S. Ganti, G.C. Shoja, and Y. Coady. A low-cost realistic testbed for mobile ad hoc networks. In *Communications, Computers and Signal Processing, 2009. PacRim 2009. IEEE Pacific Rim Conference on*, pages 671 –676, aug. 2009.

[214] Yağız Onat Yazır, Chris Matthews, Roozbeh Farahbod, Adel Guitouni, Stephen Neville, Sudhakar Ganti, and Yvonne Coady. Dynamic and Autonomous Resource Management in Computing Clouds through Distributed Multi Criteria Decision Making. Technical Report DCS-334-IR, University of Victoria, Department of Computer Science.

[215] Yağız Onat Yazır, Chris Matthews, Roozbeh Farahbod, Stephen Neville, Adel Guitouni, Sudhakar Ganti, and Yvonne Coady. Dynamic Resource Allocation in Computing Clouds Using Distributed Multiple Criteria Decision Analysis. In *IEEE CLOUD: 2010 IEEE 3rd International Conference on Cloud Computing*, 2010.