Task Recovery in Self-Organised Multi-Agent Systems for Distributed Domains

Asia Ali Salman Al-Karkhi

School of Computer Science and Electronic Engineering

University of Essex

A Thesis Submitted for the degree of

Doctor of Philosophy in Computer Science

August 2018

Acknowledgements

First, I would like to address my sincerest thanks to Professor Maria Fasli who gave me the opportunity to work on this project. Her professional guidance, advice, comments, criticisms and patience throughout my PhD journey were of utmost importance. Her constructive suggestions, insights and views on research style have all been integrated into the result. Without her professional guidance and help, this research could not have been completed.

I am also grateful to the Ministry of Higher Education and Scientific Research (MOHESR) of Iraq for offering me the scholarship which has helped me undertake this PhD degree, and the Computer Science Department at the University for Technology, Iraq, for nominating me to do this research.

And last but not least, I would like to express my gratitude to my family for their endless trust, support, and encouragement – as well as their critiques and advice whenever I felt lost or helpless in the course of this journey.

Abstract

Grid computing and cloud systems are distributed systems which provide substantial widely-accessible services to resources. Quality of service is affected by the issues around resource allocation, sharing, task execution and node failure. The focus of this research is on task execution in distributed environments and the effects of node failure on service provision. Most methods in the literature which provide fault tolerance, use reactive techniques; these provide solutions to failure only after its occurrence. In contrast, this research argues that using multi-agent systems with self-organising capabilities can provide a proactive methodology which can improve task execution in open, dynamic and distributed environments. We have modelled a system of autonomous agents with heterogeneous resources and proposed a new delegation protocol for executing tasks within their time constraints. This helps avoid the loss of tasks and to improve efficiency. However, this method on its own is not sufficient in terms of task execution throughput, especially in the presence of agent failure. Hence, we propose, a self-organisation technique. This is represented in this research by two different mechanisms for creating organisations of agents with a certain structure; we suggest, in addition, the adoption of task delegation within the organisations. Adding an organisation structure with agent roles to the network enables smoother performance, increases task execution throughput and copes with agent failures. In addition, we study the failure problem as it manifests within the organisations and we suggest an improvement to the organisation structure which involves the use of another protocol and adding a new role. An exploratory study of dynamic, heterogeneous organisations of agents has also been conducted to understand the formation of organisations in a dynamic environment where agents may fail and new agents may join organisations. These conditions mean that new organisations may evolve and existing organisations may change.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

With recent developments in hardware and increases in the sophistication of software, there has been a noticeable improvement in computational capacities and in network performance [1]. Grid computing has been designed to provide the same kinds of capabilities in the computing arena that power grids provide in the electrical power supply arena – as explained by Ian Foster in [2].

However, one of the issues with grid computing is job scheduling and task execution, because the resources in this environment are distributed across different locations and are diverse [3]. Also, in grid computing, higher demands are made of the distributed environment: such as, a requirement for better response times, reductions in other factors such as the traffic, and critically, having the ability to deal with malfunctions on the network and to provide fault tolerance. Therefore, and since such systems are often large and complex, resource sharing and controlling fault tolerance have been of wide interest in academia particularly enhancing resource utilization and task execution. Providing automatic failure detection is critical and this is still considered to be an open problem [4].

Controlling failure in distributed systems provides for safe services and enables work-flow to be maintained in an efficient way. However, within the dis-

tributed environment, node failure is an unpredictable and spontaneous problem. In this work, we provide methods for improving task execution and for dealing with the kinds of failures which may occur at any time in distributed environments like in the grid computing environments. We have implemented various scenarios in order to study and analyse system performance within various different parameter settings.

1.2 Research Aim and Objectives

The aims of this research are:

- To study how self-organisation as a process can help agents within the distributed networks domain achieve better utilization of resources and cope with agent failure.
- To investigate and study the disruption problems which may occur whilst a system is in operation and going through its workload and we provide reasonable solutions which recover and execute a larger proportion of the customers' tasks than would otherwise be executed - enhancing system performances in distributed networks domains.

The thesis' objectives are as follows:

- **Model a dynamic and distributed system of nodes providing services as a multi-agent system.** This is to create an open multi-agent system as a distributed network and provide its agents with heterogeneous types of resources in order that they can execute tasks in response to customers' requests, but without any central control on the agents' behaviours. Agents are connected with each other and create their own contact lists which they will use for the heuristic algorithms which have been tested here. This testing was to select an algorithm which can lead to the execution of large numbers of tasks even in the presence of disruptions. Agents always work

to increase their own utilisation values and, here, to have the capacity to accept more than one task. Furthermore, customers may request, in each cycle, various numbers of tasks, and an agent may receive more than one request in each cycle.

- **Explore and develop mechanisms for task delegation for the purpose of increasing task execution throughput and the utilisation of agents' resources.** Our objective in this objective is to provide a delegation protocol which can work in a distributed environment to increase the task execution. In network environments, even those using high performance computers, searching the whole network for a particular resource so that a task can be executed is infeasible. Therefore, we have focused on using the agents' various different statuses when operating in the network, and have come up with the idea that an agent who has received a task which does not match its own resources, should delegate. In order to delegate a task, an agent will use its partial knowledge of the other agents in the vicinity to direct the message. So, tasks will be delegated depending on agents' status, without central control or any global knowledge concerning agents in the network. Our objective here is to maintain, when failure/disruption is present in the network, the system's service level in regard to receiving and executing tasks. Service maintainability is not easy to provide in distributed environments with failure problems; reducing the effects of failure is our main aim. Having simulated failure within the environment, we have then constructed new methods to deal with this problem.

- **Explore mechanisms for the creation of organisations of agents to improve performance and utilisation.** Self-organisation and emergence were the main objectives. Various different models have been studied in order to make sure that we have selected the best model to support our further research. The objective was to ensure that the process of creating the organisations which emerge was effected without any central control. As a result, we have self-organised entities that can group themselves together

depending on their decisions/interests to maximize their utilisation at the micro level. This leads to the maximization of their functionality and the maximization of the usage of the emergent organisations at the macro level. Furthermore, in grid computing, nodes are generally underutilized even when their resources are being shared between different organisations [5]. Therefore, in our work, to simulate the agents' various different abilities and performances, the agents are designed so that they will join a number of organisations depending on the size of the network; there is a maximum number of organisations that one agent may join.

- **Develop recovery mechanisms to deal with individual agent failure within organisations.** Agent organisations mechanisms can lead to improvements in task execution throughput and to delegation within organisations. However, agents in organisations are still affected by failure. To increase task execution throughput and to deal with the failure problem a recovery protocol is an essential part of any (and this, proposed) self-organisation process. The characteristic of this protocol, here specifically, is that it facilitates an additional role for inclusion in each created organisation. Hence, the structure of the emergent organisations is enhanced by the creation and identification of another role for its organisation *Members*. The *Head* of each organisation will assign one (and only one) of its *Members* to be responsible for checking on the availability of the *Head*. So, the advantage of using this recovery protocol is that it allows for the ability to substitute the *Head* when the *Head* has failed.

- **Study the evolution of organisations in dynamic open environments.** The objective here is to study dynamic and open heterogeneous organisations wherein agents may fail permanently. Agent failure will affect the organisations' structures and stability since agents with specific roles may disappear, and this will result in the organisations disbanding. Also new agents may appear and this may lead to the emergence of new organisations. Hence, it is essential to provide a protocol that can help to maintain the environment's utilization and functionality.

1.3 Summary of Work Undertaken

In distributed systems, tasks may need to traverse several nodes in the network in order to find the resources that they need for execution. This will lead to some tasks exceeding their allocated time constraints and thus failing. Another problem which may prevent proper task execution is node failure. A failure problem within a distributed environment is difficult to detect because such will occur in one part of the system while other parts are still functioning correctly. Detecting failure in files can easily be done by computing the checksum; however, issues in a distributed environment are more difficult: e.g., the failure of an Internet server [6]. We have deployed multi-agent systems as a mean to address failure in a distributed environment and help to choose the proper mechanisms to enhance the performance of the system.

The problem of network system failure has been mentioned in relation to some of the most significant computational environments: cloud computing, cluster computing and grid computing. The failure problem and how to recover tasks being performed in distributed systems are the main targets of this present work. The following issues are the challenges/problems that this research will try to investigate and provide solutions for:

- How to create a network of nodes holding resources as a system of autonomous agents, without a centralised controller, and enable such agents to react to changes that occur within the system.
- What type of mechanisms can be applied to enable the agents to organise and improve overall task execution throughput.
- What kind of protocols/mechanisms are needed to recover from failures which may occur within organisations of agents.
- What kind of mechanisms are needed in order to recover the situation when key agents fail.

- What kind of mechanisms are needed to recover the created organisations when agents are permanently failed in dynamic environment.

Our first step was to model a network of agents with heterogeneous types of resources, simulating the situation that exists in distributed systems such as the Internet. We decided to have agents represent the nodes in the network because agents have a number of useful abilities such as autonomy, reactivity and pro-activity [7]. These capabilities have been implemented within various kinds of distributed environments and grid computing is one of these. Agents may exist in various modes: busy executing tasks, or they may be not busy (idle) and waiting to receive a task. One of the main challenges in open and distributed systems is the occurrence of unpredictable events such as a disruption which affects an individual agent's performance and in consequence the overall system's performance. In relation to this, agents will hold various types of heterogeneous resources in order to satisfy various types of tasks. We have provided a simplified description for resources and resource matching in the network that has been adopted in this work.

The second step was to have a customer agent which simulates the existence of a number of customers all of which send multiple tasks to the network of agents. Each task issued by a customer must be associated with heterogeneous types of resources, a matching value and a deadline; in the network, these must be matched with the agents' resources. Customers want their tasks to be completed within the specific time, and this reflects on the service level and the degree of efficiency of the agent network.

Moreover, since we intended to investigate task execution within distributed environments, an efficient search algorithm was vital for the purpose of delegating messages from the receiving agent to another agent in the network and so on - until the required task is executed by one of the available agents. In this work, we investigated the efficiency of a random search strategy for this application. As it turned out, random search was not found to be efficient for this task delegation process. So we developed another search algorithm, called directed search, which

is based on directing the message in the network using the agent's status (busy, not busy, failed) to direct the search around the network. In this algorithm, if the receiving agent is unable to carry out a task, then it will delegate to its direct neighbours. Hence, tasks will be executed depending on the receiving agents' status or their neighbours' status.

Our challenge was to produce an environment which can maintain the execution of the customer tasks even with the existence of undetected disruption, and which can recover tasks even when there are failed agents. In relation to this, we have studied two important concepts: the emergence phenomenon and the self-organisation technique. These concepts are applied to enhance the efficiency of multi-agent networks when disruption occurs. Agents may take one of a number of different status, and our aim was to apply two mechanisms for the creation of organisations of agents which form a virtual layer above an existent network of agents. This strategy makes use of emergence in order to demonstrate a natural means of creating organisations without any central control or human intervention. In the self-organisation process, agents will be grouped together when some triggering conditions are satisfied within the most busy agent – which is called the *Head* of the organisation which will be created. Both heterogeneous organisations and homogeneous organisation may be created. Furthermore, various different roles emerge when agents become *Members* of organisations. The kind of participation that an agent takes on will reflect its ability to accept tasks as well as its ability to facilitate the maximum functionality of the emerged organisations. Our work has shown that an environment which is equipped with a self-organisation capability can execute more tasks and thus has enhanced system performance.

Even given the improvements that we have achieved by using emergent self-organisations, the problem of failures inside the created organisation still affects the performance of the system to some extent. Therefore, to boost the performance of systems in the presence of disruption – especially that of the failure of created organisations' *Heads* – we decided to add another feature to our environments. This was to give the other agents an important role in terms of recovering tasks throughout the course of a disruption event. The *Head* agent sends a message

to any newly joined agent, asking whether the latter can be a *Henchman*. This *Henchman* role adds an important protocol to each emergent organisation, called the *Henchman* Recovery Protocol. This protocol helps to maintain the functionality of an organisation when the *Head* of the organisation fails.

Finally, we have created dynamic organisations, via our simulator, whereby agents in the created organisations may suffer permanent agent failure. An agent can be permanently offline throughout subsequent simulation cycles; this is for demonstrating the case where permanent failure occurs. This will lead to the disbanding of existent organisations and/or the creation of new organisations (when a new agent joins in). This enables us to explore the performance and stability of the created organisations in the situation where we have agents malfunctioning and others appearing. The new agents may simply become part of existent organisations or their presence may result in the emergence of new organisations.

1.4 Thesis Contribution

The main contribution of this work is to study, analyse and implement appropriate solutions which will result in increases in the task execution throughput of distributed environments in the presence of failure. The suggested solutions are applied to the recovery of customers' tasks in particular and to the maintenance of system utilisation and functionality in these dynamic environments. A number of protocols and mechanisms have been presented in this work, and a set of experiments have been constructed and analysed in order to attempt to demonstrate that our proposed algorithms are viable. The contributions of this work can be summarised as follows:

- **Presenting a framework for open and distributed multi-agent networks**. Modeling heterogeneous types of resources and creating scalable and dynamic environments is the first step in improving task execution throughput in the presence of node/agent failure.
- **Improving the task delegation protocol for use in heterogeneous net-**

works. This is mainly concerned with creating a distributed directed search algorithm that can direct the customer messages within the network of agents, depending on the agents' status. The customer's task will be assigned to an appropriate agent, depending on status.

- **Providing a mechanism for creating organisations of agents**. The self-organisation process will be imposed by the environment and depends on a trigger condition for the creation of organisations of agents. Two mechanisms for creating organisations have been presented: one for heterogeneous organisations and one for homogeneous organisations. In each mechanism, an agent in the network may join a number of organizations, but each agent can only be committed to a limited number of organizations, to reflect an agent's ability to participate in organisations, and this lead to the creation of overlapping organisations. The *Members* of the created organizations can execute the various different tasks received from the organizations they have joined, in order to fulfil their commitments.

- **Introducing roles for the agents within the created organisations**. In this work, agents' roles are an important part of the self-organisation process. The presence of roles means that structures can be established in each of the created organisations and that a separation of the agents' functionalities can be identified. These roles (*Head*, *Members*, *Henchman*) are used to coordinate the work of the organisation in order to gain more benefit from the formation of the organisations. The presence of these roles is key to the solution of the problem of recovering more tasks when failure occurs within organisations. The *Head* of an organization will send tasks to its *Members*, and if any *Members* are subject to failure, this may partially affect the process of task distribution. However, in each organisation there are heterogeneous agents, so more than one agent with matching types of resources may be available to accept the tasks.

- *Henchman* **Recovery Protocol (*HRP*)**. We have been able to demonstrate that the *HRP* is a remedy to the disruption problem. It is employed during the self-organisation process; the *Head* of each organisation selects one of

its *Members* as its *Henchman*. The purpose of the *Henchman* agent is to maintain the functionality of the organisation and its effectiveness in case of agent (specifically the *Head* agent) failure. An algorithm for checking the *Head's* availability is used by all the Henchmen in order to watch over each organisation's *Head*. Experimental work, here, has demonstrated that the *HRP* is a reliable solution for a self-organised system.

- **Deploying the *HRP* protocol to enables the agents to re-organise and deal with permanent failure**. Agents in the system are prone to permanent failure throughout the simulation cycles, and once they have failed they become unavailable to the organisations. Deploying *HRP* in such environments, where organisations' structures are changing and as agents disappear and new agents appear, is effective.

1.5 Thesis Structure

The structure of the rest of the thesis is as follows:

- Chapter Two: A survey of the literature concerning the area of interest; this includes subjects such as multi-agents, agent organizations, resource allocation, task recovery and agent failure in distributed environment domains. The types of algorithms that have been used in the literature to solve these problems, especially the self-organization methods and agent roles in self-organization multi-agent systems are also discussed.

- Chapter Three: This chapter contains the detailed description of the problem formalization. Here, the distributed environment, the multi-agent network and the customer tasks are explained in detail. The format for messages which will be sent from the customer to the network of agents is described, also the format for messages which will be sent from agent to agent in the network. Descriptions of the created organisations and the agent roles will also be provided, and the chapter will end with some detailed descriptions

concerning the simulator which has been used to simulate the proposed environment.

- Chapter Four: The main concern in this chapter is the work involved with building the simulator model and the following experiments. Examining the use of agent busy/not busy status, the agents' use of directed search and random search is described in detail. This includes showing how the system responds to executing tasks within simulation cycles. Also, the failure event and its effect on the performance of agents is explained - to show how agents can be affected by the failure problem.

- Chapter Five: In this chapter, we discuss the addition of roles to the agents, and the setting of the triggering conditions for the self-organization process. The implementation of two different mechanisms for creating organizations of agents for executing tasks is explained. Comparisons between the two suggested mechanisms are also presented in this chapter.

- Chapter Six: Organization-based recovery is suggested as a means to recover customer tasks. The *Henchman* recovery protocol is described as a means to support the system in relation to the problem of *Head* failure.

- Chapter Seven: Is a discussion of the open and dynamic agent organisations in the presence of the *Henchman* recovery protocol. In this chapter, an exploratory study is undertaken regarding the deployment of *HRP* in organizations where agents are prone to permanent failure leading to changes in organisations' structures, where agents may disappear and new agents appear.

- Chapter Eight: This concludes the work by indicating the strengths and weaknesses of the approaches taken in the research. It then goes on to identify further work that could usefully be undertaken.

1.6 Publications

The list of publications related to this thesis are shown below:
1- Asia AL-Karkhi and Maria Fasli. Deploying self-organisation to improve task execution in a multi-agent systems. Cybernetics (CYBCONF), 2017, 3rd IEEE International Conference on. IEEE, 2017. This paper is relater to chapter 4 and 5.

2- Asia AL-Karkhi and Maria Fasli. Disruption Recovery within Agent Organisations in Distributed Systems, ICAART, 2018, 10th International conference on agents and artificial intelligent. Related to chapter 6.

3-Asia AL-Karkhi and Maria Fasli. Poster presentation, 2017 CSEE Joint Workshop with Industry, Public and Charitable Sector Friday 30 June 2017, Colchester Campus.

4-Asia AL-Karkhi and Maria Fasli. Dealing with Permanent Agent Failure in Dynamic Agents Organisations, under review at Web Intelligence conference 2018, related to chapter 7.

Chapter 2

Literature Review

2.1 Overview

Here, we look at the existing research literature associated with several important areas of this work: grid computing, node failure in distributed systems and multi-agent systems. We have investigated these areas with a view to making use of agents in order to solve the failure problem. Node Failure is one of many issues that have been highlighted by experience with distributed computing systems such as clusters, grid computing and the cloud. Self-organisation and emergence techniques are topics that we have studied in particular so that we could subsequently make proper use of them. Providing fault tolerance capabilities will increase the reliability of grid computing because such enables the systems to function in an enhanced in the presence of failure [4]. In order to study a self-organised multi-agent system, it is necessary to produce an environment which can deal with critical events that are quite specifically related to dynamic systems. Understanding and conducting prior studies within this domain should help to suggest techniques relevant to the main purpose of this research.

2.2 Agents and Multi-agent Systems

Multi-agent systems (MAS) have emanated from the distributed artificial intelligence field. An agent can be defined as a computer system with autonomous, proactive, reactive capabilities to perform various types of actions to meet its design objectives [8]. In addition, agents can work together with, or possibly against, each other in order to undertake tasks , [9]. Furthermore, a multi-agent systems (MAS) is a loosely coupled network involving a collection of software components which work together in order to solve complex problems beyond the capabilities and knowledge of the individual entities [10].

Many researchers have explain various characteristic about agents, in [7] the author has shown set of properties that agents can have, an agent can communicate either with human or with other agents to achieve tasks, human has no control on agent's behaviour (actions, internal states). Agents can act reactively i.e they can preserve their environment and behave accordingly to any changes that may occur i.e. they have the ability to adapt and modify their behaviour. Furthermore, agents' ability to act according to the changes in the environment is not the only capability that they can demonstrate but also the ability to initiate an action for a goal-directed behaviour to show the pro-activity property.

In software development, one of the most powerful tools for representing complex systems is the concept of multi-agent systems. Modularity and abstraction are both considered to be characteristic of the multi-agent concept – in which agents can be used to represent elements of large, complex, and unpredictable systems such as air traffic control systems, manufacturing control systems, collecting information from the web and commercial applications (e-commerce, e-auctions) [11], [12].

Agents have been deployed within open environments in order to solve complex problems in a range of areas that involve only individual agents would not be applicable without a collective goal. Agents may hold a variety of different resources, they may have different skills and expertise, and they may be capable

of performing a variety of different tasks in order to maximise their resource utilisation and improve overall system performance.

In [13] agents are entities that are holding resources and they can interact with each other and have the decision to carry out tasks or not. Hence, agents are rational and decision makers to resolve issues or accomplish tasks. For an agent to make a decision there are number of factors that effect its decision such as the type of the task, agent's available resources, agent's role and an agent's historical information about other agents in the environment such as type of messages, an agent last action and last decisions. Hence, an agent micro level behaviour represented by an agent decision will effect the macro level of the systems [14].

2.3 Agent Organisations

In general, an organisation is a group of autonomous agents that can work and interact together to achieve specific goals/solutions, depending on system/society requirements [9]. In structured organisations, the interaction between the agents emerge from a set of negotiations that control the agents' socialization [15], [16].

Many researchers argue that in a heterogeneous multi-agent system environment triggering an agent organisation process to reduce the complexity impact resulting from having a large number of agents. Hence, this will demand formal theories to help in the design of the organisational structures – the methods whereby the agents interact, see [17]. Using organizations has been demonstrated to provide reasonable solutions for many task allocation problems in distributed environments. These solutions may be used to minimise resource allocation costs and decrease unnecessary communication among agents when they are operational within the organization [18]. Using an open multi-agent system with no predefined(static design) is the most effective option because the agents in a distributed and dynamic environment may be designed by different people and facing variety of events and traffic in order to satisfy disparate requirements. Static design can help in applications that its goal design specific and cannot

change within time.

Also, the organisations or communities created within networks may have various properties – for example, overlapping and/or hierarchical structures - and they may require special algorithms to be used in order to study them and detect their presence [19], [20], [21]. Constructing organisations of autonomous agents demands an organisational structure for the participating agents and a specification for the agents' operational processes. An organisation may dissolve or change within a limited time and this depends on the events that occur in the environment [22].

In [23], they propose a theoretical method for optimizing agents' networks. Their proposal was to minimise the number of agents who are chosen to be in charge each organisation and to maximise the connections of these which are necessary to achieve the necessary network coverage. They have investigated and compared between a number of different network types. However, their model has been tested only on a small number of organizations.

2.3.1 Self-Organisation and Emergence

Self-organisations and emergence are not new topics; both exist and indeed co-exist in distributed environments. In [24] the authors shown that, emergence phenomena have been defined in many studies as a comprehensive behaviour (macro level) which grows from cooperation among local entities (micro level) of the system, examples of such behaviours, outside the computing area, are to be found in traffic jams, flocks of birds, and ants who leave trails of pheromones. A self-organised system means a system which can change dynamically without any external intervention whenever the surrounding circumstances demand it [24]. An example of a self-organised system is an ad-hoc network that can freely discover the accessibility of its members without a router. The main fields in which self-organisation has been studied are computer science, physics and systems theory [25].

Although self-organisation is an attractive process in terms of dealing with dynamic requests to a software system, it does not always have a positive effect on the system which adopts it, as has been explained in [26], [25]. In our work, we illustrate, in chapter 5, that one of the created organisation mechanism led to negative effects on the network. Also, in [27] the researchers demonstrate the effects of creating some greedy self-organized groups as on the network has been presented using cyclic graphs.

In [28], the authors suggest a framework for a system which deploys reasoning capability in the agents. The framework provides semantic operations based on logic performed by the agents themselves in order to make the decisions instead of on installing explicit conditions within these agents. So, the agents can make organisational decisions such as to join or not to join an organization, to comply with an organisation or to leave an organisation, based on the role requested by other agents. The meta-model of the suggested system supports the idea of providing for obligations, roles and role enactment within the organisations. Their system assumes organisations to be pre-existent and independent from the agents, so the agents can decide freely to join or not to join. Part of their system's platform is implemented in Java and then this is integrated into a Jason multi-agent platform. In our work, a self-organisation technique has been utilized which demonstrates the beneficial outcome of neighbourhood activities performed by individual agents in relation to the overall network: i.e., to increase task execution and thus enhance system performance. Subsequently we designed approaches for the triggering of the creation of agent organisations. This was an upgrade to the design which implemented roles for the agents in the self-organization process and which represents a robust enhancement to the agent networks. Agents have the option to join or not depending on their preferences at the time of receiving the organisation Head's message.

One of the most important capabilities of self-organised systems is that they are highly adaptable. Such systems exhibit non-linear and complex behaviour. Hence, they have been adapted to a large range of environmental circumstances. Sometimes weak self-organisation systems are created, and this generally necessi-

tates the creation of a centralized environment. However, a system that distributes control all over its entities can be much stronger [25].

In [29], the authors have analysed and shown how the bio-inspired techniques can be deployed to provide flexible, reliable and reconfigurable solutions. The authors also explained that in nature, various complex tasks are accomplished in an effective and straightforward manner; this provides us with powerful adaptive techniques when we mimic them. One of the fields which involves bio-inspired techniques is multi-agent systems to solve complex engineering problems. Swarm intelligence can be seen in nature: e.g. in ant colonies, shoals of fish and flocks of birds. Another adaptive technique which is taken from nature is self-organisation itself. Both swarm behaviour and self-organisation are based on the individual entities' activities which emerge and then affect the operation of the whole group of entities. Swarm intelligence has been used to solve many problems in industrial fields such as the Turkish forecasting energy demand problem [30] and to solve transportations problems, especially in relation to traffic congestion [31].

In [32], the author shows how it is possible to use an emulation of bee foraging behaviour in grid computing to help the client to select dynamically the best method by which to process and execute a particular chunk of data. A metaphor of ant behaviour can be used for dynamic web page communities so that the web pages which hold particular information about authors represent pheromones – in Web terms – thus creating a foot trail. So, authors can collectively organise Web pages into communities [33].

In [34], the authors have implemented an event processing system which demonstrates a new method by which to implement recovery from failure based on rollback. The status of each operator is saved via savepoints, instead of via checkback points as were used in the original rollback algorithm [35]. When a node fails, its predecessor holds a savepoint, in a savepoint tree, that has been stored at a pre-specified time. This tree is created to save the operator states of all the operators in the path closure of the predecessor. A heart beat signal from a coordinator represents the method by which operator failure is detected, and this is used to coordinate the savepoint tree and update it when necessary. The

proposed algorithms resulted in high overheads because of the need to update the savepoint tree whenever necessary. Also, their method would have to be enhanced in order to use it in a distributed manner because a coordinator needs to be implemented for each operator in order to detect failure and control and stabilize operator topology and recovery from failure. The introduction of leadership and group membership is necessary in order to solve this problem [36]. In our work, the *Head*, the created organizations, and the *Henchman* recovery protocol constitute a viable method for implementing their system, but in a distributed manner – as we have demonstrated in this thesis.

In [37], the author has shown the effect of peer influence on activity driven networks using graph theory. In their model, each node is assigned a probability of becoming active and then the active nodes can promote their neighbours which are less active - and encourage them to become active. This will increase the probability of the neighbours becoming active, leading to the formation of a web of community structure. The limitation of their work is that the less active nodes may still receive messages even when they have started to become active and part of the community. This is in contrast to our work where, especially as described in chapter 4, and 5, the network is created by inserting agents into the context one after the other. After creating the network, peer influence is created by applying a triggering condition which is related to the context's demands: i.e., the busiest agent in the system needs some help from other agents and this will lead to the sending of message by the busy agent to its neighbours. Thus, the less busy agents will start taking part in the community or created organizations. We believe that our system corresponds to real-world networking situations.

In [38], the authors propose a self-organised resource allocation scheme based on Decentralised Distributed Virtual Environment (DDVE). The scheme functions independently from the underlining P2P network. The authors presented a scheme, based on using gossip protocol, which identify the users' critical zone. They took advantage of the presence of heterogeneous peers (clients). So, by using some information provided locally the loaded zones will be identified and the zones with less load will be the target to create the virtual peers to reduce

the load on the other regions. In our work, the algorithm used by the *Head* to create an organisation is based on the gossip protocol and will search for other agents which could potentially join the *Head's* organisation, taking note of their (the other agents') busy/ not busy statuses.

Other researchers have used agent organisations in distributed environments in order to enhance the performance of such systems. In [39], the author implemented a simulation model for exploring recommendations regarding the connection of a networked system of heterogeneous service supplier and purchaser agents in an electronic market. They introduced an agent-based model for recommendations as well as decisions, using the principle of homophilic neighbourhood choice. They implemented methods for selecting peers based on agent similarity and demonstrated the ability of such a set-up to self-organise an overlay system. Their work sheds some light on agents' capabilities in terms of decision making and the agents' knowledge concerning connected peers, gained via the network evolution process. In our work, a self-organisation technique has been utilized to demonstrate the beneficial outcome of neighbourhood activities, performed by the individual agents, to the overall network – to expand the task execution and hence to enhance the system's performance. In addition, we have created rules for triggering the emergence of agent organisations. These rules provide roles for the agents in the self-organization process which thus introduces a robust enhancement scheme to the agents' network.

2.3.2 Role in Organisations

The role concept is an important element within these self-organizations because it shows that agents are capable of abstract behaviour. The agents have to satisfy the specified constraints (skills, requirements, obligations) in order to obtain a role and accrue the benefits that an agent can expect to receive in playing that role and undertaking the duties related to it [9]. Creating organizations of agents and applying roles to its *Members* in order to improve their ability to work as societies has been the aim for many researchers [40], [41]. In relation to [42], wherein agents

are made to work in a large and complex system environment, giving a role to the agents may be a better solution for them. A role adds the capability for the agents to overcome issues like event or process interruption and at the same time, to gain the opportunity to maximise their own interests.

Also, in [43] the researchers have shown the importance of collections of roles and the connections between each other which decide the behaviour of the multi-agents in the created organisation.

In [44] the researchers proposed a method called trust-based role coordination for task oriented multi-agent systems; they claimed that sometimes heterogeneous agents have various statuses in relation to which they may be unable to receive or accomplish tasks, so processes which achieve delegation should be carried out. They developed a mechanism that enhances the decision-making process for partner selection in relation to orienting tasks. They built a role taxonomy in which it is determined which set of agents are able to execute which specific kinds of tasks. This taxonomy allows for the determining of which labelled agents are suitable for which roles. However, agents do not always follow their assigned roles especially in dynamic and distributed environment where agents may leave at any time and other agents may join in. In our homogeneous organisation, the organisations created emerge based on the ability of agents to group together to solve tasks with specific required accuracies; however this model was not as effective as the other model which we have developed where agents form organisations based on heterogeneity.

2.4 Grid Computing

Grid computing is an important case study in relation to this research. We have found a number of different problems and challenges which are implied by the use of grid computing. Researchers in academia have presented a number of solutions for various issues relating to distributed environments: such as fault tolerance, resource sharing, and resource allocation. Grid computing is considered to be

a distributed system; it has been defined in [4]as follows:“Grid is a collection of distributed computing resources available over a local or wide area network that appears to an end user or application as one large virtual computing system”. In distributed systems, availability means the network is in operational mode and reliability means providing a system that does not lose tasks; both of these concepts must be dealt with for grid computing to be viable.

The main goal of creating a grid computing system is to be able to provide services and to share heterogeneous computing resources as fast as possible, in the same way that power is shared across an electric power grid; this has been discussed by Ian Foster in [2].

Issues such as resource advertisement, resource allocation, task execution and fault tolerance have been addressed and yet more challenges are coming to light. However, grid computing has developed a-pace. Primary grid computing systems are considered to be a kind of service oriented architecture (SOA). Such a system will have its own scheduling strategy and resource allocation methods. The current grid computing architecture is called “open service oriented architecture”, [2]; this architecture has been implemented to supply resource allocation and scheduling mechanisms, using web services [45].

In many organisations and institutions, it has been found that the computational resources available are underutilized and also that the servers which exist in such organisations tend to suffer the same issue of poor utilization. Grid computing can effectively enhance the utilization of such underutilized resources, since it focuses on sharing geographically distributed resources via administrative domains; it creates virtual dynamic organisations and shares these resources between organisations [46], [5].

2.4.1 Fault Tolerance in Distributed Computing Environments

Fault tolerance is a capability that should be developed into a system: it will allow the distributed domains to continue functioning even in the presence of failure.

Failure means a system is behaving in a different way from that which it supposed to, and system errors or faults are considered the causes of system failure [4].

The probability of failure events, such as nodes failure and/or communication loss, is higher than in other, more conventional, distributed environments due to grid computing heterogeneous nature and complexity [47]. This can affect the type, quality, availability and reliability of grid services: the users may not be able to trust the service level of grid computing systems.

In the literature, researchers have developed new methods and theories in regard to this issue. In general the methods relating to fault tolerance can be divided into two types: pro-active and post-active as explained in [5], [4], [48], [49]. Where, a pro-active method is one which detects and measures the level of failure before scheduling tasks onto grid nods while a post-active evaluates the situation after scheduling tasks onto a failed grid node; most of the research is concerned with the second kind of method as the first kind is more difficult. In our study, we have implemented models that use a self-organisation technique to manage and help the system cope in the event of agent failure. The set-up of the implemented system depend on the most busy agent to trigger the self-organisation process: from this, the organisations emerge. This procedure allows the customer tasks to have a better chance of execution even in the case of failure, due to the heterogeneous type of agents which exist in the organisations. So, even if failed agents are present, other agents may be able to accept and execute the tasks.

Fault detection strategies mainly adhere to either the pull model or the push model [50], [51]. That work also recommends the proactive method and proposes a framework for failure for use in grids using autonomous agents. The agents are proactive and they also save information about grid components; hence, based on this information, the agents can work on improving the reliability and performance of the grid. The limitation to this is that they did not include failures such as network failure, response failure, node failure, etc. They only considered operating system and application failure. We think that our work is more generic even-though it operates within the application layer; we hope that we can inte-

grate it as a service in the grid middle-ware; this will be our next step (future work).

In the pull model [50] and [52], all the grid entities send periodic signals to the fault detector, if one of them has not sent a signal within a specific period, then the detector will activate a fault tolerance mechanism. In the push model, the detector is the one that sends a signal to the grid entities and detects failure. Hence, when a failure occurs in the environment and cannot be averted, the problem recovery methods then come to the fore. Agent failure detection – we investigated methods for detecting agent failure; most of them use heartbeat messages in order to check for the occurrence of failure: the push model, the pull model, the use of neural networks, and probabilistic methods - all these methods may require differing types of solutions to suit the target system.

A more recent study of agent failure in distributed MAS is [53]. There the author has addressed agent failure in situations where disaster repeatedly occurs. Consecutive or simultaneous agent failures may occur in the future as a result of the disaster event. A comparison between repair task allocation algorithms which are centralised, centralised resource allocation with repair, distributed resource allocation, distribution with repair after agent failure, and a MAS unit solution, were all looked at in order to find a viable method for decreasing the number of failed agents in a system experiencing such a critical situation. The author demonstrated the simulation of the failure of hundreds of agents, and the running of the server simulation scenario for 1000 cycles.

2.4.2 Task Recovery in Distributed Domains

Nodes in distributed systems can suffer from unexpected failure, and this means the loss of tasks and a decrease in the agents' utilization in the environment. Hence, to ensure robust performance, the ability of the system to self-organize is not only desirable but essential.

In the literature, the aim in regard to designing peer to peer networks is

to apply fair and sensible allocation of the spare resources using a number of different resource allocation methods. However, the existence of disruption and the obstacles which are sometimes associated with data transmission, such as machines offline and high user activities, may lead to severe reduction in the correct utilization of the resources [54].

The solutions for the problems caused by faults, in terms of grid computing, are fault tolerance, recovery, and removal. The main fault tolerance techniques which have been used in cluster systems and grid computing are checking points, message logging, replication and retry [55].

In a grid system like Globus, the fault detection technique which is applied in order to detect networking and the host/server failure is the heartbeat; the tolerance consists of the fact that a failed job can be resubmitted. The disadvantage of this method is that it is unable to solve errors caused by user exceptions [56].

In many studies, multi-agent systems have borrowed and applied recovery protocols from other domains. Roll-back recovery protocols are a kind of recovery method which has been applied to control disruption in another kind of distributed environment: i.e., client server, world-wide web, peer-to-peer networks and mobile ad-hoc network - when resources are limited, when there is limited access to certain types of resources or there is only (as is more likely) indirect access to such resources. Roll-back recovery protocols have been deployed in various different distributed environments, as in [35]. Those authors present a survey which helps to distinguish between different types of roll-back recovery protocols and then compare their performance. The first one looked at is the checkpoint based protocol, which is based on choosing a checkpoint and restoring the system to that point in the case of failure. The second one is the log based protocol, which is a combination of the checkpoint protocol and the use of log-in information. These protocols deal with nodes in a network as groups of processes that communicate between each other. These interactive processes access a storage appliance periodically in order to save recovery information - which will be at minimum checkpoint states for those processes passing messages to each other. This information can then be used when the nodes return to active mode

after the disruption has been processed. However, the processes might have only out-of-date information to work on after being returning to the checkpoint state, and may end up working on something which may no longer be required by the system.

In [57], the amount of information that needs to be saved for the checkpoints is large; the system must be kept updated to keep the agents up-to-date with the states pertaining to the system prior to any failure. The authors argued that their work provides a promising solution to the problem of failure in distributed agent environments. The multi-agents were designed as being Believe Desire Intention (BDI) agent oriented. In these authors' work, a single and multiple, but non neighbour agent crash or failure could be handled by the agents. They also claimed that they solved both failure causes using a distributed logical clock to update the agent's beliefs; the agents do not depend only on their own clock. To compare between messages and update the beliefs about the system held by the the agent in the recovery process, the time-stamp technique was applied. In our work, which uses an object-oriented design, we have implement the issues of neighbour agent and non-neighbour agent crashes, via a given probability value of failure. We have also implement dynamic message exchange between agents. Neither of these latter issues were addressed in [57].

In [58], it is shown that forming an organisation is with the aim of achieving either emerged or designed purposes. In their work, they showed that agents could depend on their stored values to self-organise. The first advantage of this is to give the agents more mobility to self-organise on their own account. The second advantage is to make the self-organisation process realistic in relation to the changes in the environment as represented by the agent's stored values. In our work, however, the idea is to maximize the utilization of the created organisations, as well as that of the individual *Members* of the organisations, as this will increase the performance of the organisations and their ability to satisfy the tasks requested by the customer side.

The researchers in [59], present a dynamic system recovery process, based on agents, for a distributed database system. In their work, agents are used to

buffer the initial information during the recovery stage and then they replay it. The authors have compared their work to the basic message-log protocol, see [35] and they argue that their new method generates less of an overhead and performs more efficiently. However, there is still a need for the buffered information held on the databases to be synchronised.

Other researchers have addressed the same issue of agents catching up after downtime: "Zero-delay Recovery of Agents in Production", as it is termed in [60]. In their work, they target the multi-agent systems paradigm, but without using emerged organisations, and their proposed framework is called "Mozart Spaces". It is based on a Linda-like paradigm, [61], which is used to solve the failing agents' problem by which agents are unable to catch up with the requirements of their environment. However, their method is based on using a shared memory space to store up-to-date state information – to be accessed by all agents. This could crash, and it also means that their system needs a high-capacity storage device or a relatively small number of agents. This is in contrast to our proposal, wherein agents are autonomous and are implemented with an ability to check themselves in regard to the last state, after downtime. They try to resume their tasks if the task deadlines has not been reached. It is by this means that our work attempts to provide a mechanism for solving the disruption in agent organisation systems.

This is not the case in [62], wherein the authors proposed a method to detect and identify malicious SMS messages sent to android devices. Using JADE agents, the multi-agents interacted with each other to watch and gather the existent technical features. Hence, they created a user profile on the mobile devices and transmitted this to the server so that It could analyse and detect any malicious behaviour. The authors' idea was mainly to find a correlation between the reported user profile created on the device and the malicious SMS. In our work, however, the self-created organisations will be able to detect disruption on their own and adopt a recovery process when required.

The authors in [49], have implemented a resource allocation scheme which takes into account the existence of the random failure problem. They have claimed that their methods can be applied to automatic target discrimination systems or to

observe the vital signs of medical patients. The system provides a static allocation for a batch of tasks in a heterogeneous environment. Tasks are assigned to computing resources after these complete previous tasks or recover from failure. They implemented five different methods and the most efficient one was found to be a match-making method based on the Derm-Lieberman-Ross theorem [63]. They claimed that the methods they derived improved and maximized the cumulative rewards received from the tasks finished before their deadlines.

Other researchers who have studied the failure problem in networks deploying agents are [64]. In their work they tried to find out which of their agent algorithms was a fit with which of the various different topologies of complex and distributed networks – such as small world, scale free networks and lattice networks – all with different rates of agent failure. They also suggested an algorithm for studying the effect of agent failure on the exploration process whereby agents search the network to collect and synchronize data. The suggested method was to collect data using an ant-colony based algorithm, and they demonstrated in the experimental work how this algorithm works better than the random walk algorithm for some topologies of networks. The random walk works well in scale free networks.

2.5 Research Problem and Challenges

Grid computing is a distributed computing environment which will keep providing challenges to researchers. New techniques and enhancements are constantly being added to grid computing, and this means that solutions are needed in order to allow the environment to cope with the issues arising. We focus on providing pro-active solutions to the task recovery problem by deploying self-organised agents in organisations. Most of the existing methods in the literature are post-active or, as they are often termed, reactive solutions [5].

Applying solutions which attempt to cope with the problem before its occurrence is more challenging, but also more productive. Agents and multi-agent

systems can host algorithms for fault tolerance which can be enhanced and then deployed to grid computing environments. We found a great deal of work in literature that looks at self-organisation and most of these studies use the bio-inspired method because these have demonstrated reliable performance [29]. In this work we wanted to provide some interesting heuristic methods and mechanisms for the environment in which this research problem presents itself. Mainly, this consisted of creating a trigger condition that could emerge from the individual agent's behaviour and which triggers the self-organisation process – represented by the creation of organisations. Further, combining the emergence phenomena and the self-organisation capability have attracted many researchers such as in [26]; the latter attempted to develop a system which exhibits group behaviour which emerges from individual agent behaviour. This agent behaviour was combined with an ability of the individual agents to change dynamically without any external intervention whenever the surrounding circumstances demanded, as explained in [24]. However, effectively combining the two (emergent behaviour and self-organisation) is not an easy task because this could lead to negative emergences that may affect the whole system's performance [26]. In our work, as we address in the following chapters, we adapt the methodology in order to create organisations of agents to maintain the system services using heuristic techniques. Agent roles can play a significant part in the self-organisation process inside the created organisation. The presence of roles implies the specification of constraints like skills and requirements that an agent should encompass in order to obtain a role and accrue the benefits that an agent can expect to receive in playing that role and undertaking the duties related to it [9]. However, agents do not always follow their assigned roles, especially in dynamic and distributed environments where agents may leave at any time and other agents may join; such issues need to be tackled.

Chapter 3

The Problem Formalization

3.1 Introduction

This chapter provides the formalization of the problem which has been studied in this research and includes descriptions of our environment's entities: namely, agents, organisations and roles. Notwithstanding this, the formalization of the problem will be the main focus of this chapter. In particular, here we will concentrate on the task execution and recovery problem inherent in prone-to-failure, distributed, scalable and dynamic systems. We have seen that including dummy nodes in a network is not sufficient to detect or solve the problem of agent failure [65], [66], [67]. Instead, it has been discovered that adding distributed adaptive agents provides the network with the necessary autonomous/proactive/reactive elements for this purpose [68], [69].

Networks of agents can be used to model actual networks with heterogeneous types of resources: e.g., the Internet, the cloud and grid computing networks. In addition, agents hold various different kinds of resources and can perform various different tasks, accordingly, in order to maximize their resource utilization. Furthermore, a self-organization capability can be one of the most empowering abilities that groups of agents can be endowed with.

Simulators for agent-based modeling and also dynamic network analysis

software have been widely used to study the kind of complex, adaptive and non-linear systems which we are dealing with here, see [70].

3.2 Problem Description

Disruption is a problem that almost all distributed, scalable and dynamic systems face. In this work, we study dynamic and open systems (agents may enter or leave at any time). With these systems, unexpected failure is a problem which leads to lost tasks and delayed task delivery (to the customer) and which affects the network performance. This problem has been modeled in this research using heterogeneous entities/agents with decision making capabilities. Agents can decide to carry out tasks or not depend on number of constraints. The environment generates random failure events that effect the agents performance.

This research focuses on how to use distributed multi-agent systems to solve the customer tasks recovery problem through deploying new self-organisation mechanisms to avoid bottlenecks/failure and guide the agent's individual behaviour (at the local level) as well as the system behaviour overall. We envisage that the proposed solutions, will lead to better, more effective and efficient utilisation of agents' resources and increase the throughput of the system. Thus, a solution to the disruption problem, using self-organisation (i.e. agents with the ability to self-organize into aggregations), is suggested.

We study how an agent can perform a number of different roles in the course of the self-organisation process. Furthermore, creating new roles for the agents, via the self-organization process, results in agents which act more appropriately in response to failure, and this is important in relation to the failure of a *Head*. In this research, we assign agents roles in order to enable self-organisation – we study the effect of the agents' roles on each other and on the types of relationships which may occur. The questions we are seeking to answer is how can the process of adding these roles help the process of recovering tasks i.e. increase task execution

and control disruption in the network of agents. After that, decide which of the suggested self-organisations mechanisms is better in terms of task execution. Furthermore, we look at repairing these aggregations/organizations when things go wrong. Finally, agent failure can be permanent and this will change the emerged organisations structure.

3.3 Environment Formalization

The environment used in this work consists mainly of two components: the customer agent that sends customers' tasks and the network of agents which receives these. Agents are added to the environment one after the other. Each node can have N connections to other nodes – where $N \geq 1$. The agents are connected by bidirectional links. Each agent may have a different resource set from other agents, and this makes the network heterogeneous and distributed. This network, like any network in the real world, involves the possibility that some of its agents may become inactive, or active but busy executing tasks – i.e., unavailable for a period of time. Agents exchange their resources with each other in order to obtain (partial) knowledge of their surrounding environment. The abstract functions can be seen in Figure 3.1 and a simple agent structure is shown in Figure 3.2.

Each agent has an identification (a name) by which it can be identified across the open, distributed environment and which can be used for further processes, such as exchanging its name with other agents so that agents can gain partial knowledge of their surrounding environment. Also the agents are connected together in a network, and the type of connection is bidirectional, as explained below:

- The system: is a tuple, < A, L >, where A = { a_1, a_2, a_3,....., a_n } is a finite set of agents and L is a set of links, and where each link { a_i, a_j } ∈ L indicates the bidirectional connection between agents a_i and a_j.
- Each agent is linked to N (number of neighbour) agents, and can sustain

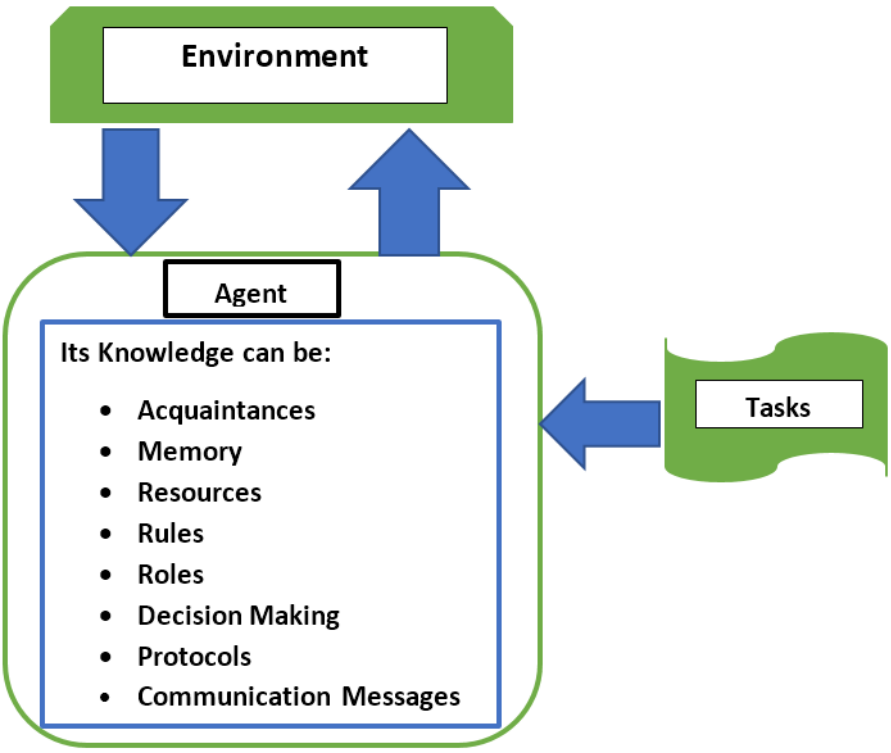


Figure 3.1: *Abstract View of an Agent*

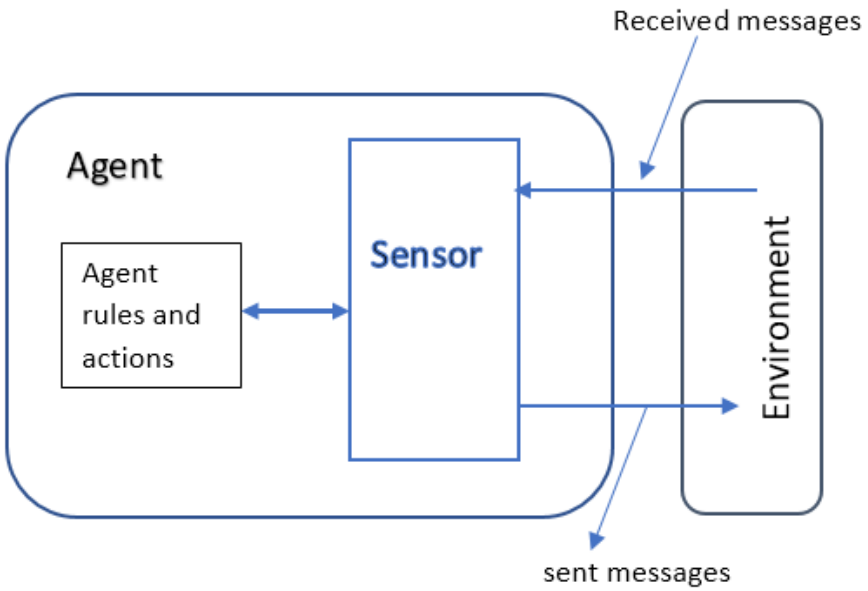


Figure 3.2: *A simple View of an Agent Structure*

a maximum number of connections, which may be different from agent to agent. An agent has a set of capabilities depending on its role in the system.

3.3.1 Agent Model Formalisation

An agent is an active entity that can communicate with other entities and form groups with them. An agent will have role/roles in the groups which are formed and may join a number of different groups simultaneously. There are no restrictions on the architecture of an agent: agents can be designed to have properties such as re-active, proactive, cognitive, mobile and others [71].

An agent should be able to provide services to other entities in the system and its services should be made accessible via the use of the agent's identification [9].

An agent can store information about its connections with its neighbours and/or about the tasks it has accepted – in terms of time and resources. The descriptions of the elements that are defined in each agent are as follows:
a_i ={ a_{ID}, *Role_i*, *N*, *Resource_i*, *TID*, *CL*, *BA*, *BT*, *PS*, *ST*, *ATQ*} where:

- a_{ID}: is an agent identifier which is used when agents send messages to each other.
- *Role_i*: specifies agent roles; each agent will have its default role as a service provider when the environment first becomes active. After the organisations emerge, various roles can be held by the agents in the various organisations, an agent's role can be defined as a tuple Role=<R_1, R_2, R_3, R_4 >.
- *N*: is the maximum number of connections for agent a_i where $N \geq 1$.
- *Resource_i*: each agent has a number of different types of resources. The resources are represented by a resource tuple Resource$_i$ <r_1, r_2, r_3>: is the resource descriptor; the resource is specified via three elements, and these are used to encode a range of resources. Each element consists of a value between [0, 4].
- *TID*: is the task received from the customer to be executed by the service provider agent; this will be described in more detail in the next section.
- *CL*: refers to the Contact List which is a list of an agent's neighbours' names

and resources – stored as < Neighbour_ID, Neighbour_Resources >, where the first parameter is the neighbour's identifier and the second parameter is the neighbour's resources.

- *BA*: Busy Agent is a Boolean value used to change the agent's status from one state to another. If its value is set to true, this means that the agent is currently actively executing a task, while if its value is set to false, this means that the agent is in an idle state, waiting to receive a task and execute it.

- *BT*: Busy Time is the number of cycles an agent will need in order to finish executing its current task. This value is dependent on the deadline that will be sent with each task. This situation will be described in detail later.

- *PS*: Probability of Failure, we have created an agent failure property which uses probability values to simulate the occurrence of failure in real-life networks.

- *ST*: Shut-down Time, agents are set as offline for a period of time to create the impression in the network that these agents are unable to accept messages or execute tasks for this period. This we describe in chapters 4, 5, and 6, while in chapter 7 we show what happens when the ST variable is removed in order to simulate the situation where the agents fail permanently.

- *ATQ*: Accepted Task Queue, each agent has queue to hold the accepted task(s) with their deadline restrictions.

3.3.2 The Customer Agent and the Task Description

The customer agent is an entity whose role it is to send tasks to the agent network. The customer agent will send its tasks to the network throughout the course of the simulation cycles. The tasks will be sent via a type of message called a customer message, *CM*, which contains the following elements:

$CM = \{CID, TID, RV, TTL, TD, RA\}$

Where:

- *CID*: Customer *ID* is a unique identification number which is given to each customer and used by the service provider agents. They use it to return the response concerning the task's status – e.g., pending (in the agent accepted task queue), executed or failed – to the appropriate customer.
- *TID*: Task *ID* is a unique identification number given to each task; each customer can send various numbers of tasks *M_Tasks* in each cycle.
- *RV*: Resource Vector represents a sequence of resources, $RV = < r_1, r_2, r_3 >$; these are requested variously by each task.
- *TTL*: Time to Live is the number of hops which the customer message can use to traverse through the network of agents.
- *TD*: Task Deadline is the deadline which represents the duration of execution allowed after an agent satisfies the required resources accuracy *RA*.
- *RA*: Required Accuracy represents the required accuracy in terms of matching the customer-required resources with the agent's resources; its value ranges between $(0 - 12)$ and is pre-agreed between the customer and the agents.

3.4 The Structure of Organisations

The term organisation in the context of multi-agent systems has been defined by many researchers one of them is in [72]: "an organisation as a collection of roles, that stand in certain relationships to one another, and that take part in systematic institutionalised patterns of interactions with other roles".

Predicting the behaviour of the overall system is a difficult task because of the possibility of an unexpected behaviour emerging [9]. In our work, creating an organisation means formalizing a group of agents with roles so that the ability of a system to cope with malfunctions is enhanced. The purpose, here, of an organisation of agents is to increase the number of tasks that can be executed (under conditions of failure) and minimize the time required for accepting and

executing tasks. This means that an agent that can satisfy the requested task requirements should be found within a single hop or in a minimal number of hops. Further, the system may contain more than one organisation and organizations may overlap.

The organisations can be defined as follow:
$Org = \{Org_1,..., Org_q\}$, where Org_p with $1 \leq p \geq q$, is the pth organisation in the system, hence, the elements in each organisation is represented by:

$$Org_p = < OrgID_p, OrgH_p, \{a_1,...a_{no}\}, HM_p, Z_p >, \text{where } 1 \leq no \leq Z_p.$$

- $OrgID_p$: a unique identifier for each organisation created.
- $OrgH_p$: is the name of the *Head* of an organisation.
- $\{a_1,...a_{no}\}$: the participants in the organisation at a specific time.
- HM_p: is the *Henchman's* (follower's) name for each organisation created.
- Z_p: is the maximum size of each organisation.

Any agent, a_i, in the network can be a *Member* in a number of organisations. The emerged organisations are intended to fulfil all the recovery requirements with minimum overhead. An agent with or without an organisation can ask for help from another agent or refuse to perform a task. Also, an agent can leave an organisation or a new agent can join in. Furthermore, if an organisation becomes ineffective/inefficient, then it needs to be re-organised; alternatively the agents' connections need to be reconsidered.

3.4.1 Agents Role Description

The idea of agents taking up roles is an important concept which we have explored in this research; a specific set of roles has been used for this research. An agent role can be defined as an agent behaviour that can affect, enhance and/or change

a system's structure. Typically a role will posit expectations, skills and duties and hence an agent which takes on a particular role must be able satisfy these requirements [9]. The taking-on of roles by the agents in a network is an essential part of the self-organisation process used in our proposed solution to the task recovery problem. Each agent a_i, at any one time, has a number of roles R_i.

- R_i is defined as a set of roles, $< R_1, R_2, R_3, R_4 >$
- An agent a_i can assume a role in relation to at least one other agent, which can be an individual or an organisation.
- The roles an agent may take on within an organisation are {*Member*, *Henchman*, *Head*}, each agent may have one of these roles or more than one, depending on a set of conditions.
- If a_i has a role it may accept consequential role(s), this means that an agent that is a *Member* in an organisation later on may accept the *Head's* message to be the organisation's *Henchman* agent.
- If Org_p and Org_q are two created organisations, then a_i can belong to both of them. In turn, any two or more organisations in the network can share a number of *Members*, so an agent can have different or the same Roles: if the overlapped organisations(Org_p, Org_q):
  (a) a_i can be *Member* in Org_p, *Member* in Org_q.
  (b) a_i can be *Head* in Org_p, *Member* in Org_q.
  (c) a_i can be *Head* in Org_p, Not *Henchman* in Org_q.
  (d) a_i can be *Henchman* in Org_p, Not *Henchman* in Org_q.
- In the emerged overlapped organisations, An agent cannot be a *Head* and a *Henchman* in the same organisation.
- The *Head* of each organisation is responsible for the activities of its agents. The presence of a *Head* in an organisation of agents is necessary in order to coordinate the work.

- A *Henchman* in an organisation will become a temporary *Head* when the *Head* of its organisation has failed.
- Agents are expected to take on and change roles while they are operating in the environment.

3.5 Agent Based Modeling and Simulation

The concept of using agents to conduct simulations has taken different guises in the literature and has been referred to as Agent-Based Modelling (ABM), Agent-Based Simulation (ABS), ABMS, or IBM (individual based modeling). In this work, we use the acronym ABMS (agent based modeling and simulation).

Complex and non-linear adaptive systems have been widely studied using agent-based modeling and simulation (ABMS) and dynamic network analysis (DNA) techniques. The ubiquity of cheap computing power has broadened the scope for research so that more intricate problems in many unexamined fields can now be looked at [73]. ABMS has been used in many fields: social, physical and biological. The growth in the use of simulation to study complex systems, such as the activity or behaviour of individual entities (humans, agents, proteins, etc.) and their actions in a group, is critical and dynamic not only at the micro-level, but also as it relates to the integrity, modification and evaluation of whole systems which can be intensively affected by such behaviours [73]. ABMS is used for simulating the continuous interactions between decentralized, heterogeneous agents with decision making capabilities over time using available scheduling methods – either continuous or discrete/time stepped. The interaction process centres around agent behaviours relating to the exchange of their information and how to reach a desired end from these interactions: [74], [75].

3.5.1 Repast Simphony Simulator

For this research, we have chosen the Repast Simphony simulation software, which is widely used, free and open source: the REcursive Porous Agent Simulation Toolkit (Repast), using Eclipse IDE (Java/Python /C++ programming) [76]. Repast is a desktop development environment for implementing Agent-Based Model Simulations (ABMS), and it is considered one of the most scalable agent based environments, depending on the complexity of the agent source code. Other simulators which share the same properties are Swarm, MASON, and anyLogic. An ABMS can be used to explore issues relating to heterogeneous environments and emergent systems, [77], [78], [76]. Repast has been extensively used for social simulation, but it can be used in any domain. Many versions of the Repast modeling toolkit are available: such as, Repast under Java, ReLogo, Repast python, Repast for Microsoft.NET and Repast Simphony. Repast Simphony allows the developer to work with networks, optionally combined with a geographic information systems (GIS), and to control the number of agents, the agents' actions and their behaviour by using different scheduling methods – which can be either continuous or discrete.

Using Repast allowed us to design and study the behaviour of multi-agent systems and to investigate agent paths, actions, and behaviour (micro level) and their effect on the final, created societies of agents (macro level). In our work we have used Repast Simphony to model a complex adaptive system to study how the self-organisation process can be used to enhance the network environment in a better way than using other network simulators. For example,using ns2 simulator would create a more complex environment and we would need to model network details that are not relevant to what we are trying to study in this work. Repast Simphony helps to produce simpler environment and provides better control to design adaptive agents. Developers can control the number of agents as well as the agents' actions and behaviours by using different scheduling methods. This facility allowed the design and implementation of a framework that deploys multi agent organisations to provide a self-adapting system. In general, Repast Simphony has been used for various complex adaptive systems such as

optimization techniques, system engineering, producer – consumer systems, and pedestrian traffic networks; other examples of its use can be found in [74].

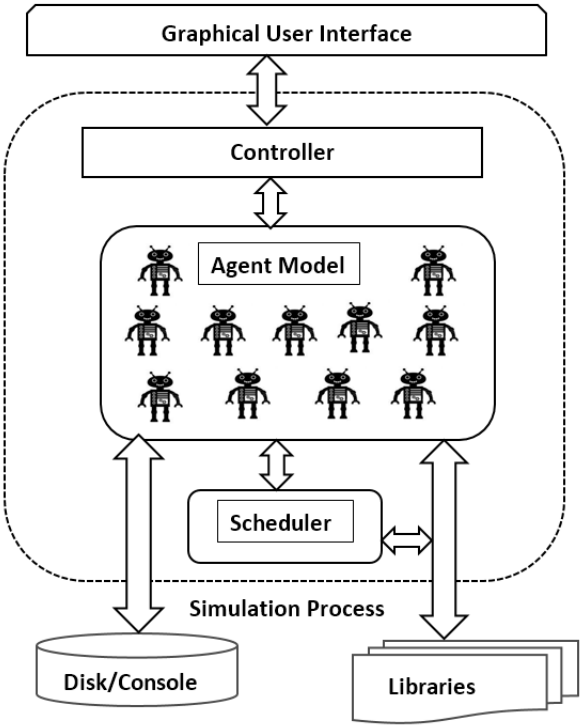


Figure 3.3: *Repast Simphony Agent Diagram*

Figure 3.3 shows an abstract representation of the agents we designed using Repast. As the diagram shows, agents were injected into the environment; the population of agents can be created or added to and will vary according to the complexity of the problem modeled. The controller is used to send the agents' interaction and behaviour to the Graphical User Interface (GUI). Moreover, there is a scheduler which is used as a basis on which to simulate the agents' actions within simulation cycles; either discrete or continuous scheduling can be used depending on the problem requirements. Repast Simphony contains a large library to support agent modeling. Input parameters to be passed to the agents can be set by the system, and the output can be files or output messages to be sent to the environment's console. Furthermore, Repast Simphony has a number of plug-in features [79], the following are some of them:

- The application framework which provides Repast Simphony's runtime interface plug-in system and user interface

- The Eclipse set provides programming tools and model specifications
- The Core set provides the main modeling simulation procedures and functions, for example scheduling.
- The ReLogo set consists of a simple language with its own library for agent based modeling.
- GIS The geographical interface system can be used to enhance for modeling and visualization
- The 2D Visualization set for viewing models with appropriate features.
- 3D Visualization set for viewing models with appropriate features.
- The Integrated Library set for supporting neural networks, Genetic algorithms, and social network
- The Freeze Dryer set provide for storage in XML and text formats.
- Re third party applications, Repast Simphony supports a number of software systems which can thus be accessed from the GUI:
    1. R statistics.
    2. Java Universal Network/Graph framework network analyses
    3. spreadsheet
    4. weka data mining
    5. Structure Query Language(SQL).

3.6 Summary

This chapter has presented the problem formalisation for our distributed scalable dynamic agent network. We have explained briefly the problem scope and how agents are deployed to provide the solution. We have provided descriptions of the agents in the network, the customer agent, organisation's structure and the agents' roles. The components described in this chapter form the basis on

which we will build the final model which will be studied and analysed over the following chapters. We have provided a description of the Repast Simphony simulator's main components, what kind of software it is, and how it can help when developing and working with complex, adaptive and scalable network systems.

Chapter 4

Task Execution and Delegation in a Distributed Environment

4.1 Introduction

This chapter describes our first strides in implementing our proposed solutions – the practical implications of our framework for multi-agent networks. Problems such as agent failure, topology change and lack of reliability all still exist in almost every distributed environment [80], [49]. Such environments are dynamic, and critical events may occur throughout the time workload is present.

However, the techniques and methods in use in academia and industry have progressed in relation to overcoming these issues. We resolved to address the problem of node/agent failure in distributed grid computing. This can affect service provision to customers and reduce resource utilization. Hence, we have suggested using the properties of a distributed adaptive multi-agent system to provide remedies to the problem of node/agent failure in relation to this environment. As we have mentioned, agents are autonomous, proactive and reactive entities. The microscopic, i.e., the agents' individual behaviours, can affect the macroscopic, i.e., the network level behaviour. Adaptive multi-agent systems may have the ability to self-heal and self-organise, and this ability can be de-

ployed in distributed open and dynamic environments in order to develop auto-organized and high-performance systems [81], [82]. Both multi-agent systems and grid computing have been used to provide various techniques and mechanisms which can control or avert problems in open, dynamic and distributed environments [83], [84]. The main motivation for constructing distributed environments is resource sharing. Resources are heterogeneous – they can be files, printers, scanners, or for instance web pages. All of these are managed by specific servers on the Internet. On the other side of the equation, the customers are users that use the web browsers to access these resources [6].

In this research, nodes in the grid environment have been represented by agents. These agents hold heterogeneous types of resources; customers will request these using their connections. We are aware that there are specially designed languages which are used to define the resources in multi-agent systems [85]. However, our system has been implemented via simulation and we have used our own method to represent the resources in order to avoid any overheads. The target in this research is to provide methods and techniques that can be deployed in distributed environments for, in particular, increasing network utilization and task execution. There are tasks which may need more than one message to be sent across the network in order for it (the task) to be completed. Hence, the contribution of the work described in this chapter is the introduction of a delegation algorithms that can increase the number of executed tasks and also agent utilization in dynamic and distributed systems. Instead of using a random search algorithm [86], we introduced the use of a directed search algorithm that will direct the search in the network depending on the agent's status. This is to find out whether the directing the search will make the system works better – improves task execution and resources utilization. The agents have been designed as objects; each object holds its own data and can perform a particular set of functions – such as, exchanging messages with other agents in the network, changing its own status, rejecting a task, etc. Several models have been implemented for studying the initial results while advancing towards acceptable solutions. The following sections will explain the system's main component and functions.

4.2 The Initial Network Formation

Within the simulation environment, there are two types of agents: the customer agent, which is used to simulate task requests emanating from the multiple customers that might exist in reality, and agents which possess the resources to execute tasks. Initial network formation occurs within the first few simulation cycles. So, the network of agents and the customer agent are constructed at this time as well. The customer is responsible for forming the messages representing the tasks to be sent to the network of agents. Using the Repast Simphony Simulator, the following steps are performed for a specific number of cycles as specified by the researcher; this is in order to construct the agent network:

- The agents start to enter the environment randomly one by one. So the network is created by adding agents. An initial scale-free network is created with an initial number of agents, as specified by a parameter, *NumberOfAgents*. During this construction process, even when the network consists of only one or two connected agents, the network still has the ability to accept tasks from customer agent because everything is created in parallel. The acceptance of tasks depends on whether the requested resources are available or not; if the required accuracy is met, a task will be executed by the receiving agent, otherwise the task will be delegated to another agent in the network.

- The experiments we have undertaken control the number of connections for each agent in the network. Each agent has a random number of connections within a maximum value N, and so each agent can sustain a maximum number of connections, and this maximum may be different from agent to agent. However, the system is still scalable – agents can be added to/deleted from the system. The N value is for simulation purposes only and to simulate circumstance that agents have a limited capacity/memory for maintaining contacts (i.e., an agent cannot 'know' the entire network). Figure 4.1 is a network of 300 agents visualized using Gephi [87] with the Force Altas2 layout which is useful to visualise Small-World and scale free networks.

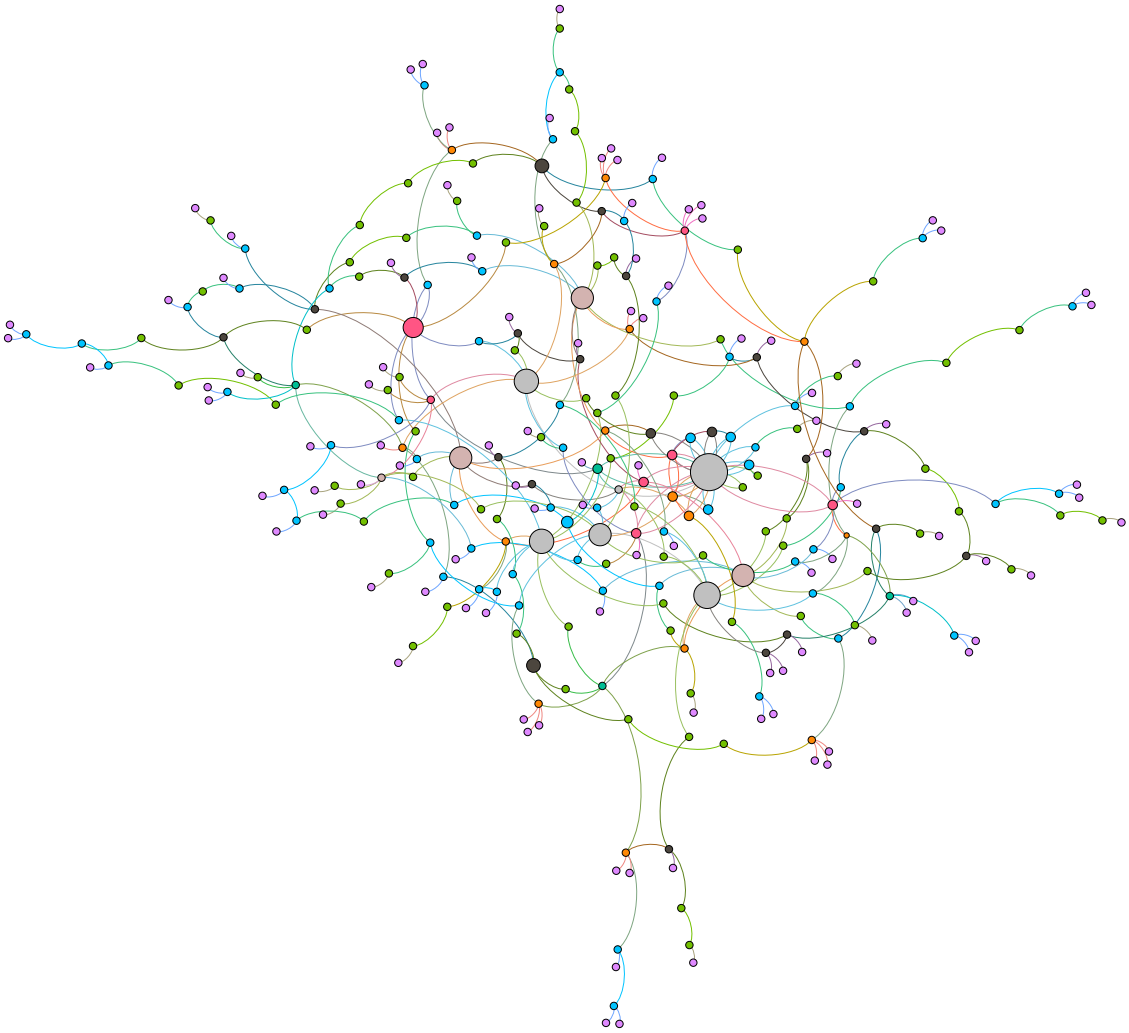
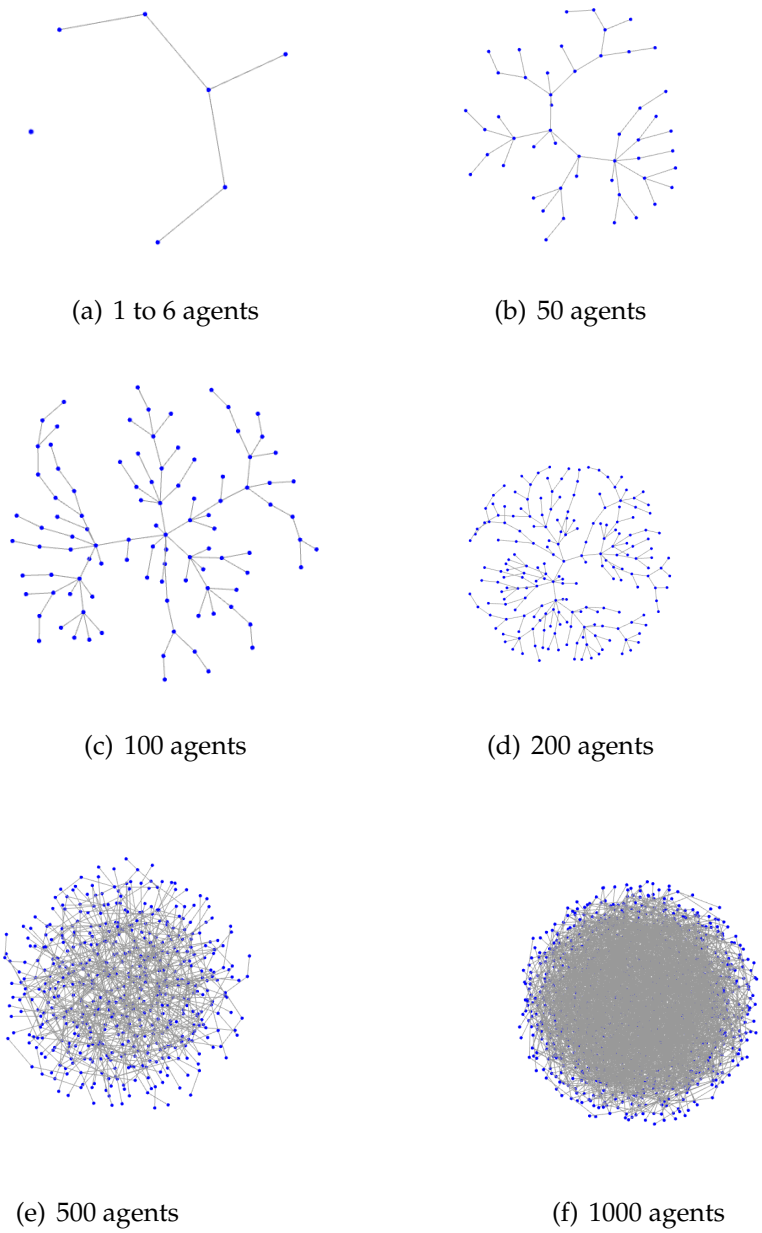


Figure 4.1: *Visualisation of a network size of 300 agents.Different colours represent the degree of each agent i.e. the number of connections for each agent. The purple=1 connection, light green=2, blue=3, black=4, orange=5, green=6,red=7, light pink=8, grey=9 or more).*

- During the connection process (as described above), each agent will send a message containing its contact details to its randomly selected neighbour(s) and the receiver(s) will respond with a message containing their contact details, depending on whether the original agent gives its consents to receive these details. So each agent maintains a list of its own known contacts. As a result, in the initial phase, each agent obtains at least one other agent's contact details. So in this phase, the agents acquire a partial knowledge of their surrounding environment.

- An agent's connections will not remain fixed. The agents may change their connections in order to achieve better performance and to create organised groups.

- The type of network which result from the simulation model which has been developed is one which is scale free and contains a number of hubs; this is typical for scale free networks [88]. So, the network created in the above way will have many agents each with a number of connections to other nodes. Some nodes may have few connections while other nodes may have a large number of connections and in this regard such a scale free network follows a power low distribution. Specifying the number of agent connections during the creation of the network minimises the possibility of having centralized nodes (agents): i.e., it prevents the system from converging to centralization, and this is important in relation to the fact that the network is scale free. A condition has been set which specifies a random number of connections for each agent. In the model used, a scale free network has been chosen because this is the form of network which exists in various relevant systems such as social networks, the Internet and biological systems [88], [89]. The following figures provide a visualization of the scale free networks involved, using a variety of network sizes – all under the Repast Simphony simulator. The networks of agents have been implemented using the algorithm proposed by Barabsi, [88]. Figures 4.2 show, in sequence, network sizes of (1, 24, 50, 100, 200, 500, 1000) agents.



(a) 1 to 6 agents (b) 50 agents

(c) 100 agents (d) 200 agents

(e) 500 agents (f) 1000 agents

Figure 4.2: *Visualisation of scale free networks of different sizes using Repast Simphony*

4.3 Implemented Model Description

The model which has been implemented mainly results in network of heterogeneous agents. The agents do not necessarily know of each other, but they attempt to process tasks and services and start making connections with each other via messages. The agents in the network are providers that have diverse resources which can execute heterogeneous tasks and provide services; these match the

requirements of the customers to various extents. On the other side, there is the customer agent, which simulates the existence of several customers which need to find services or have tasks executed.

The participant agents in our simulated network can be in different states (busy, not busy, not failed/online and failed/offline). For example, when an agent's status changes from 'busy' to 'not busy' this means that an action has occurred in the system which has meant that this agent has had to change its status. Tasks being delegated from agent to agent or from one network to another is the primary means by which agent statuses are changed. An agent may have a busy status if it has accepted a task and is currently busy executing it. Other agents may be in high demand, and they can accept more tasks only within the customer time and accuracy constraints. Hence, agents in the network have queues which hold the tasks that they must execute after finishing the current tasks. This situation is termed "Busy agent and its Accepted Task Queue (ATQ) $\geqslant K$"; this means the agent is currently busy, has just received another task from a customer and has a ATQ containing K number of accepted tasks $K \leqslant W$, where W is the maximum size of ATQ.

A customer agent is an entity which issues tasks to the network of agents. Large numbers of tasks will be sent to the network to be accepted and executed. Customer tasks are heterogeneous and so need computational resources with specific required accuracies. Agents in the network hold heterogeneous computational resources, an agent makes its resources available as part of task execution for a period of time. If an agent receives a task (becomes a receiving agent) and cannot satisfy the requested task due to a lack of time before the task deadline, unmatched required accuracies, the agent being busy, or the agent's accepted task queue being full then it will initiate a search algorithm to delegate the task to other available agents in the network.

A feedback message has been implemented. When the customer agent sends a task to an agent in the network, a report message should be sent back to the customer to give the current task status.

Two search algorithms have been investigated; this investigation was carried out in order to select which algorithm was the most applicable for use in the delegation process which takes place between agents in the network. Furthermore, the preferred algorithm will be put forward as an element of the suggested solutions in the subsequent chapters.

4.4 The Implemented Model Execution Cycles Scenario

In the scenarios we have implemented, the simulation time was divided into cycles, and the number and types of these is a parameter defined by the researcher. In each cycle, both the customer agent and the network agents perform their own execution cycles.

$Cycle_0$: constructs the network by adding the agents to the environment one by one. Via this initial phase of the simulation, a scale free network is created with an initial number of agents – as specified by the parameter, $NumberOfAgents$. Agents will start to acquire knowledge of their surrounding neighbours by exchanging messages in order to create a contact list relating to their neighbourhoods.

$Cycle_e$: (where e=1 . . . $Total\_Simulation\_Cycles$) Once the network is formed, in each simulation cycle:

4.4.1 Customer Agent Execution Cycles

In each $Cycle_e$ of the simulation scenario, the customer agent executes the following steps:

- issues a number of task requests (M_Tasks) to a random set of agents, to indicate a specific task TID_j and to indicate a specific agent a_i.

- receives task TID_j acceptance messages from agents.
- receives task TID_j rejection messages from agents.
- receives task TID_j execution results.

4.4.2 Network Agent Execution Cycles

An agent a_i execution cycle consists of two main elements:

(A) Communication of messages which covers both checking incoming messages and sending messages.

(B) Task execution (if an agent has any tasks currently to execute).

In each *Cycle_e*

(A) **Communication**
If no messages have been received, the a_i continues to (B).
If there are messages, the a_i process them one by one:
If *message* = TID_j from customer

(a) If the a_i has the required resources with the required accuracy, then check if it a_i has the time by deadline TD_j to execute the task. If it can execute the task by the TD_j, add the task to the queue of tasks to be executed. Send message to customer "Task received, TID_j".

(b) If this a_i has the required resources with the required accuracy, but cannot schedule due to ATQ being full until TD_j, then the task is sent to one of this agent's neighbours and the a_i responds with the message "Busy".

(c) If this a_i does not have the required resources with the required accuracy, and there is still Time To Live TTL, forward/delegate the message to one/random agent of the contact/neighbour agents.

If *message* = from contact with request for task TID_j

(a) If this a_i has the required resources with the required accuracy, then check if it has the time by deadline TD_j to execute the task. If this a_i can execute the task by TD_j, add task to the queue of tasks ATQ to be executed.

(b) If this a_i has the required resources with required accuracy, but it cannot schedule the task due to ATQ task being full until TD_j, then forward the message to one of this agent's contacts and return a message to the customer "Busy".

(c) If this a_i does not have the required resources with the required accuracy, and there is still TTL, forward the message to the set of contacts/neighbours.

(B) **Task Execution**

(a) If a_i = *busy* with TID_j (this means TID_j is still under execution), execute another step of it, this is meant to simulate the circumstance that each task may take a number of cycles to execute, not just one, decrease the number of steps required for this task to be completed. Send a message to the customer containing the following information:
Execution message:(TID_j, a_i, $Send\_Cycle$, TD, $Agent\_Cycle$, $Duration\_of\_Execution$, *result*) and a simple Execution message: "TID_j, result". Remove task from ATQ and Go to (c).

(b) If a_i = *idle*, look at ATQ for tasks; If there is a task scheduled TID_j, execute one step of it. If there are no tasks in the ATQ, remain *idle*.

(c) End a_i execution cycle.

4.5 Agent Communication and Messages

The customer agent and the network agents communicate with each other using a number of messages. This form the contains part of messages formulate in agent communication languages. As this is a simulation, messages are used instead of

utilising an Agent Communication Language (ACL) [90] such as that defined by the Foundation for Intelligent Physical Agents (FIPA) or the Knowledge Query and Manipulation Language (KQML) – doing this would add to the overhead of processing messages within the system. The format of the message that is sent from the customer to the agents to request the execution of a task takes the format: Message (*TID*, *TTL*, *TD*, *Cl*, *RV*, *RA*), where:

TID – is the task's unique identifier; this is used to identify each task sent by the customer.

TTL – is the Time to Live (*TTL*) for each task; this specifies the maximum number of hops that a message can make within the network and is tracked in the search algorithms.

TD – is the task deadline, which relates to real world systems; typically these require task executions to have deadlines by which task execution results must be returned to the requesting customer agent. When a customer sends a task, it generally wants this to be finished within this deadline. To simulate such a deadline in our model, the customer sends a deadline value with each task issued to the network of agents. The deadline is a specific value representing the execution period that the randomly selected agent in the network should use for the execution of the task. It means also, the number of cycles the customer is willing to wait for the execution result after the agent in the network accepts the task. So, the response from the agent to the customer should be guaranteed within this specified, simulated, time period.

Cl – is the simulation cycle in which the task has been sent.

RV – is the resources vector representing the resources that are requested by the customer for the execution of the task. The contents of the vectors may be different from task to task.

RA – This represents the required accuracy for matching the customer resources with the agent's resources. The accuracy values range between (0 – 12).

When the customer agent sends a customer task to an agent in the network, this task will only be accepted and executed by the receiving agent if certain conditions are met – as checked by each agent.

Task messages – There are various types of messages that the agents send to the

customer agent and to each other on the receipt of a task request message.

- Task acceptance – An agent which has received a task from the customer will try to provide the required resources by checking the required accuracy, and if this is met, the agent will reply with "Task accepted, *TID*" otherwise, the agent has to transfer the message to one of its neighbours.

- Task in Queue acceptance – If the agent is executing another task it means that it is busy, and if the deadline has not expired, it will send a "Task in my Task queue, *TID*" message to the customer to confirm the acceptance of the task – but only if the agent is able to execute the message within the deadline and with the required accuracy.

- Task rejection – If the Agent is busy and its Task queue is full, the task will be refused, so the agent will reply with "Busy, *TID*", and the message will be passed on to one of the agent's contacts.

- Task failure – If no agent has the required resources with the required accuracy, agents are busy, the search fails due to TTL =0, the remaining time to deadline is not sufficient, and there is no other agent to search then the last agent to receive the task request sends the message "Task Failed, *TID*" to the customer.

- Successful task execution: Once an agent in the network successfully executes a task, it sends an execution message to the customer:
  Execution message Execution message:(TID_j, a_i, *Send_Cycle*, *TD*, *Agent_Cycle*, *Duration_of_Execution*, *result*) Where:
  TID_j – is the specific task identifier which is received from the customer.
  a_i – is the identifier of the agent who has executed the task.
  Send_Cycle – is the simulation cycle in which the task has been sent.
  TD – The time by which the customer expects to receive the result from the network of agents.
  Agent_Cycle – is the cycle in which the agent received the task from either the customer or another agent in the network.
  Duration_of_Execution – is the number of cycles allowed for execution. This

is a value randomly selected but depending on the deadline value. Thus both the situation where the agent does not use all the time available up to deadline and the situation where the agent does use this is simulated; the use of a random value simulates the variation from one agent to another in regard to the period of execution.

Once an agent has sent an acceptance message to the customer, it starts to execute the task using the determined duration (once the task has been scheduled in its task queue). So, the agent keeps executing the task until it finishes its execution, as indicated by the parameter "*Duration_of_Execution*".

result – failed or success tasks.

After that, the agent sends the above message to the customer and changes its status to not busy.

4.6 Resources Matching Process

In the distributed environment, there are various different methods for representing resources – such as the Web Ontology Language (WOL) [91] and the Resource Description Framework (RDF) [92]. However, the resources associated with each task issued by the customer have been represented in a simplified way in this research. This is because we are trying to provide mechanisms that can maintain services to the requesting customer in critical agent environments. The resources in each task are represented in the following format.

$RV = < r1, r2, r3 >$.

An agent in the network that receives a task will match the task's resources with its resources. The first such condition is the result of a matching process between the customer resource vector and the receiving agent resources, which uses the well-known Manhattan Distance (Manhattan_Dis) Equation 4.1. The resulting matching value should meet the task's required accuracy; this is a specific value between $(0-12)$. Therefore, in the delegation protocol as shown in Algorithm 2 an agent accepts a task if a matching has occurred between the customer resources

and the receiving agent's resources.

For example, where a customer, RV is <0, 0, 0>, has a RA equal to 6, and the recipient agent's RV is <2, 2, 2>, then when applying the Manhattan Distance equation, the match will be positive. The second condition checked is the TTL. If $TTL = 0$, the task will be considered to have failed, otherwise, if $TTL > 0$, then the receiving agent will check the TD of the task; if this is sufficient then the task will be executed, but if it's not sufficient (either because the received agent is currently executing a task or its queue of tasks already contains a number of tasks) the agent will then delegate the task to a neighbour agent in the network. Hence, if an agent cannot satisfy at least one of the conditions given above, the task will be either failed or delegated to another agent in the network; this process will be propagated.

$$Manhattan\_Dis(RV, AR) = \sum_{x=0}^{x=2} |RV_x - AR_x| \tag{4.1}$$

Where:
RV_x : the requested resources vector for the task.
AR_x : the matched agent resources that performs task successfully.
$x = 0$ to 2: the index of the requested resource vector, $<r_1, r_2, r_3>$.

4.7 Task Delegation Protocol

To delegate tasks which have been received from the customer across the agents' network, a task delegation protocol was proposed and tested. In each cycle, the customer sends M_Tasks messages to a random set of agents. So, when the customer requests a task TID, this is sent to one agent at random. Each agent in the network has a queue; the queue is for storing the tasks which have been accepted ATQ. Before an agent accepts a task, it must check the task's deadline value, as has been explained regarding the Task deadline Algorithm 1. So, when an agent in the network accepts a task it means all the crucial requirements for the task

have been met by the agent. The agents are autonomous and self-interested and have the desire to maximise the benefit to themselves; therefore, agents will keep the accepted tasks in their ATQ for as long as they are currently busy executing task. After executing the current task the agent will check its ATQ and then start work on the tasks held on this. Hence, when the agent is not busy it will check its queue of accepted tasks; if the queue has at least one task, the agent starts to execute the first in the queue. If the agent's queue of accepted tasks has more than one task, the agent uses the First Come First Served (*FCFS*) technique to execute the tasks in sequence.

Algorithm 1 Task deadline Algorithm

Input: a_i: is a busy agent that has accepted a task TID_j.
Output: $a_i.ATQ$.insert(TID_j) and return (1) or return(0) then a_i delegates TID_j.

1: **if** (a_i.busy == *true*) and (a_i.ATQ == 0) **then**
2: a_i predicted the finish time for its current task.
3: **if** ($a_i.Duration\_of\_Execution$) < ($TID_j.TD_j$) **then**
4: $a_i.ATQ$.insert(TID_j)
5: Compute random value for the execution cycles depending on the $TID_j.TD_j$
6: "Task in my Task queue, TID_j"
7: return 1
8: **else**
9: return 0
10: **end if**
11: **else if** (a_i.busy == *true*) and ($a_i.ATQ > 0$) **then**
12: the busy agent access to its last task in ATQ to find out the $Duration\_of\_Execution$.
13: **if** ($a_i.Duration\_of\_Execution$) < ($TID_j.TD_j$) **then**
14: $a_i.ATQ$.insert(TID_j)
15: Compute random value for the execution cycles depends on the $TID_j.TD_j$
16: "Task in my Task queue, $TIDj$"
17: return 1
18: **else**
19: return 0
20: **end if**
21: **else**
22: return 0
23: **end if**
24: End

When workload is high, an agent may receive a task that it cannot execute for various reasons – such as the deadline for the task does not leave enough time

for the agent to execute it or the resources of the agent did not match with the task's required accuracy. In order to delegate such tasks, each agent has to use either a distributed directed search algorithm or a random search algorithm. In the random search algorithm technique, the search is based on randomly selecting neighbours for sending the customer message during the delegation process. The directed search algorithm has a different mechanism. An agent which has been randomly selected to receive a task from the customer but cannot execute this itself is called the initiator. The initiator will start searching the network by sending the task message which it has received (in any cycle) to its direct contacts and by this means the messages will traverse through the network, depending on the receiving agents' status – whether they are not busy or busy and whether their $ATQ = W$ or $< W$ – and on the value of the TTL and the network sizes. The task message will be considered failed if the TTL becomes zero or if the message is returned to the initiator, see Figure 4.3. This delegation protocol can work in parallel with the agent network construction process which was described in section 4.2. A further description of the task delegation protocol is embodied in the flowchart 4.4 and a simple pseudo code specification of what happens inside the network is presented in Algorithm 2. It is worth mentioning that when a_i accepts a task it means that a_i has performed the resources matching process, using the same process shown in the example described in section 4.6.

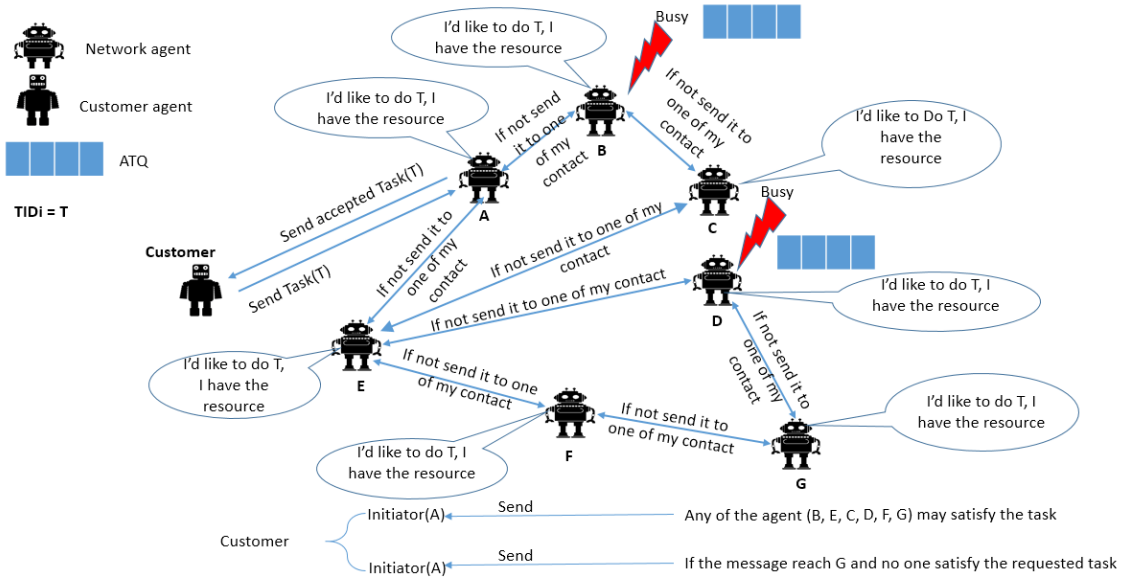


Figure 4.3: *Task Delegation Process*

Algorithm 2 Task Delegation Protocol

Input: TID_1,TID_j....,TID_M are M_Tasks: number of tasks sent from the customer in each cycle, TID_j is to show the process for a specific task.
A set of agents a_1,a_i,....,a_f in each single cycle, where $f < NumberOfAgents$.
Output: TID_j Success or Fail.

1: $Cycle_e$:
2: a_i.receive(TID_j)
3: **if** (a_i.busy == $false$) and (a_i.accept(TID_j == $true$) **then**
4: a_i.SendMessage("Task accepted, TID_j") to customer.
5: **else if** (a_i.busy == $false$) and (a_i. accept(TID_j == $false$)) and ($TTL > 0$) **then**
6: $TTL = TTL$ -1
7: // a_i delegates the task to one of its contact list
8: a_i.SendMessage(contacts,TID_j)
9: **end if**
10: $Cycle_{e+1}$
11: a_i.receive(TID_j)
12: **if** (a_i.busy == $true$) and (a_i.accept(TID_j == $true$)) and ($a_i.ATQ$ <> W) **then**
13: a_i.ATQ.insert(TID_j)
14: a_i.SendMessage("Task accepted, TID_j) to customer.
15: **else if** (a_i.busy == $true$) and ($a_i.ATQ$ == W) and (TTL >0) **then**
16: $TTL = TTL$-1
17: // a_i delegates the task to one of its contact list
18: a_i.SendMessage(contacts,TID_j)
19: **else if** $TTL = 0$ **then**
20: a_i.SendMessage("Task failed,TID_j")
21: **end if**
22: End

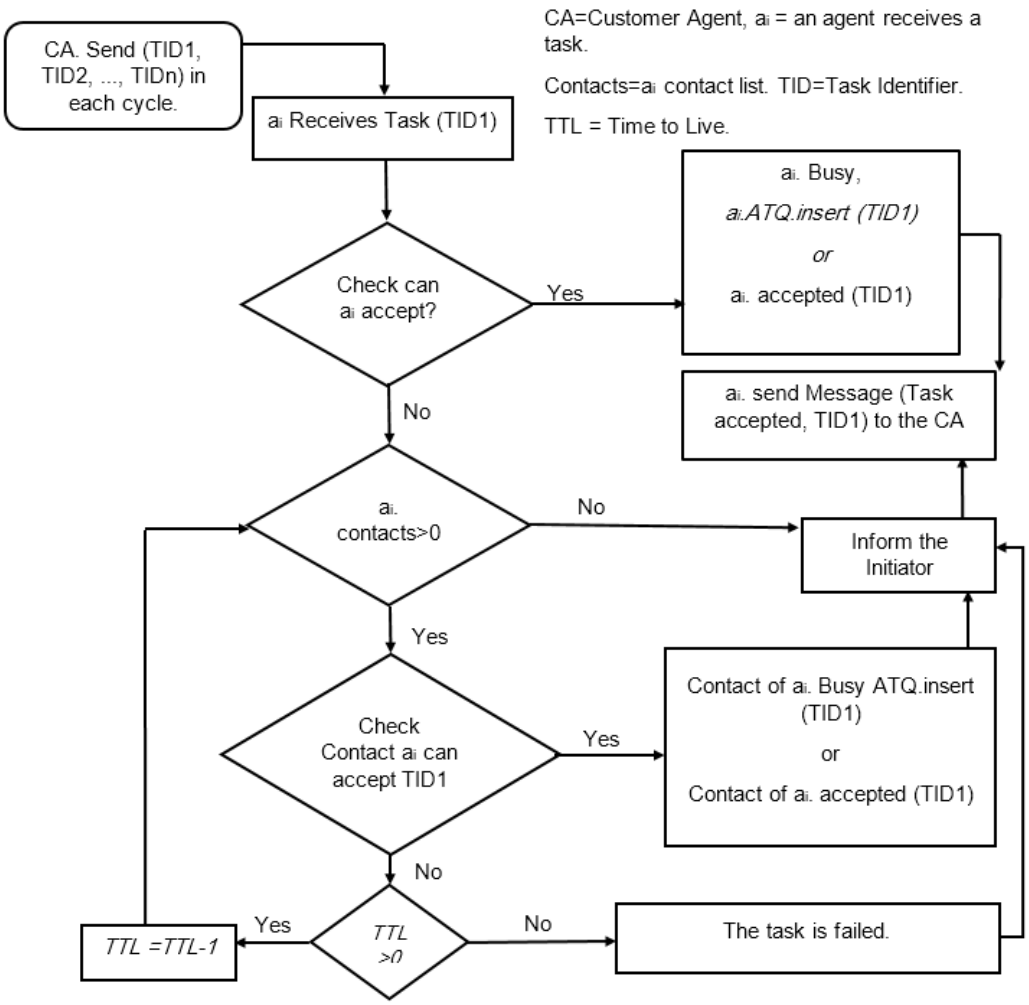


Figure 4.4: *Flowchart: Directed Search Algorithm*

4.8 Experimental Work

Our network has been implemented using the Repast Simphony simulator, an Agent Based Modeling (ABM) simulator running under Java [76]. Agents in our implemented models are objects implemented using the Java programming language. Table 4.1 shows all the parameters which have been used to set up the experiments.

In this experiment, two models have been investigated and developed, the directed search model and the random search model; this was in order to decide which one of these is more efficient for carrying out the delegation process. In regards to this, the next sections study the effects of each model on the task execution in relation to the increasing of the agents' neighbour connection size, the *TTL* values, and the total network sizes.

4.8.1 Neighbourhood Size Experiment

In this section, the maximum neighbourhood size, N, is studied in regard to its effects on the number of executed tasks and the network's performance. Agents' connections are limited to a specific number, and for this experimental work we have increased the individual agents' connections to a number of different values (10, 15, 20, 25); the effects of each value have been evaluated over ten test runs and simulator runs for 2000 cycle. Table 4.1 shows the setting parameter that has been used for the delegation processes associated with both directed and random search models. Please note that in the pie charts that follow, the different colours used represent different accuracies between 0-12 and therefore we use 13 different colours.

Table 4.1: *The input data parameters for the agent neighbourhood connections changes*

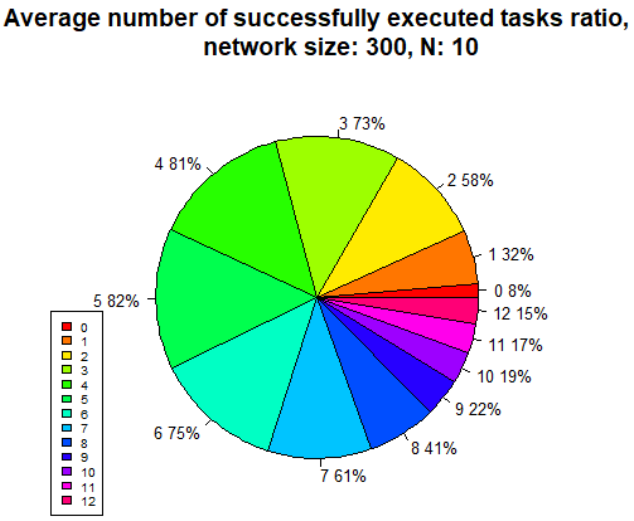
Agents	Max N	TTL	TD	Max Tasks cycle
300, 1000	10	10	(11-13)	Normal distribution*T+M(5,300)
300, 1000	15	10	(11-13)	Normal distribution*T+M(5,300)
300, 1000	20	10	(11-13)	Normal distribution*T+M(5,300)
300, 1000	25	10	(11-13)	Normal distribution*T+M(5,300)

The equation 4.2 yields the Average Number of Successfully Executed Tasks Ratio (ASETR). This is the average number of successfully executed tasks per required accuracy divided by the total number of tasks issued per required accuracy. Tasks are issued in each cycle from the customer side using a normal distribution, in order to simulate the real world network situation when different numbers of task requests are issued.

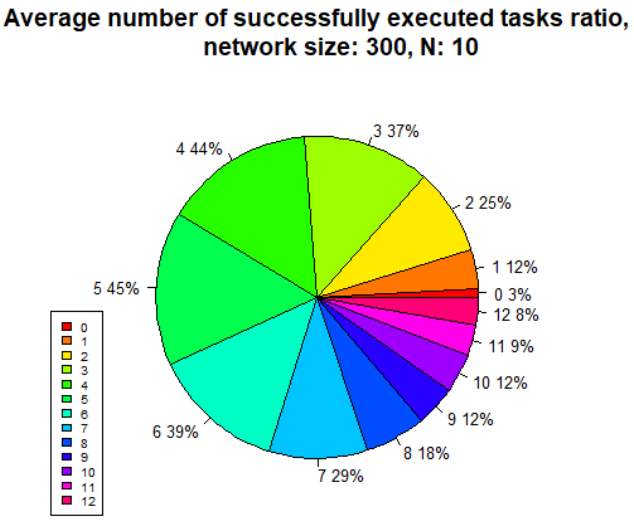
$$ANETR = \frac{STA(s)}{TTR(s)} \times 100 \tag{4.2}$$

Where:
$STA(s)$: Number of successfully executed tasks with their required accuracy (s).
$TTR(s)$: The total number of tasks issued per required accuracy (s).

Figures 4.5 a and 4.5 b show the use of a network size of 300 agents with a neighbourhood count of N =10. The search algorithms perform differently. The directed search achieves much better performance in comparison to the random search algorithm, in particular, where tasks are executed with different required accuracies. The directed search heuristic algorithm uses agents' status (busy or not busy) to direct the search around the network. This is unlike the random search algorithm which, as its name suggests, is based on random selection for its next move. With the same setting parameters, random search is unable to achieve the same level of results as the directed search: i.e., there is no significant impact on the average number of successfully executed tasks ratio when the neighbourhood size is changed and the random selection search is used.

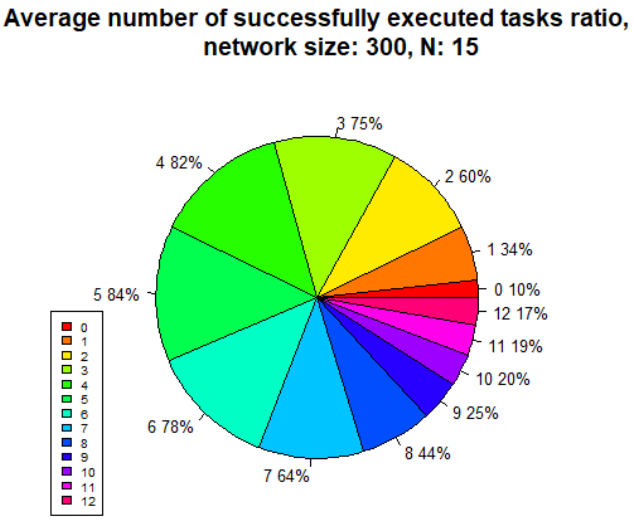


(a) Directed search

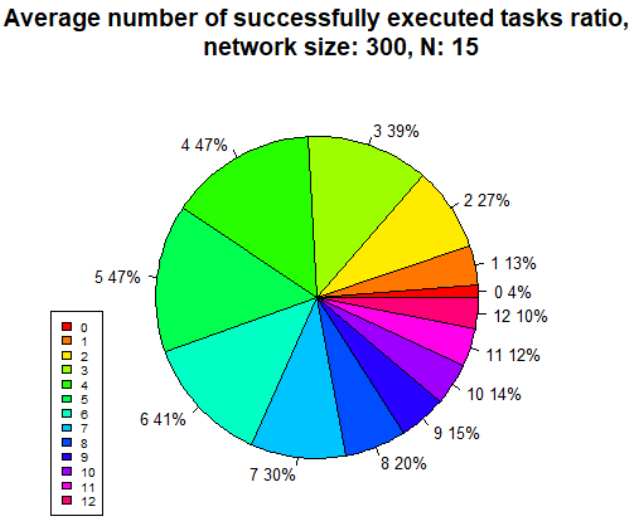


(b) Random search

Figure 4.5: *ANETR for network size:300 agents with maximum agent's neighbourhood N: 10*



(a) Directed search



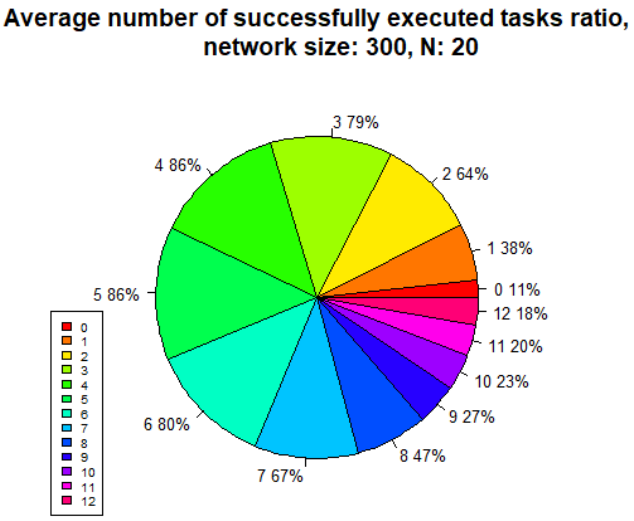
(b) Random search

Figure 4.6: *ANETR for network size: 300 agents with maximum agent's neighbourhood N: 15*

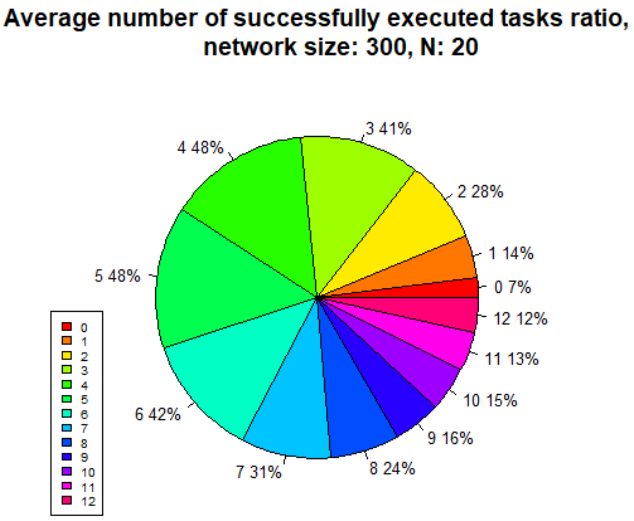
In Figures 4.6 a, 4.6 b, 4.7 a, 4.7 b, 4.8 a and 4.8 b different neighbourhood values are presented in order to show the different performances of the two algorithms when the number of individual agents' connections is changed. The average number of successfully executed tasks ratio increases with increasing connections when using either directed or random search. This is because the more connections that are available, the better placed each agent will be: i.e., having a variety of neighbourhood agents with different types of resources is advantageous and this leads to the execution of more tasks. Almost all the tasks needing a number of different types of required accuracies are executed or their execution percentages are gradually increased. However, a minority of tasks which have certain specific required accuracies are executed less than those with other required accuracies because some resources are more rare than others. Tasks that need these rare resources will be executed at a lower rate than those which do not – because the number of hops required to reach these resources is excessive.

The network size has been extended to 1000 agents in order to further investigate which of the search algorithms is better in terms of performance and utilisation of the network – as shown in Figures 4.9 a, 4.9 b, 4.10 a, 4.10 b,4.11 a, 4.11 b,4.12 a and 4.12 b, the directed search with N = 10, 15, 20, 25 shows a higher number of successfully executed tasks with different required accuracies than does the random search.

In the course of this experimental work, it was realised that, especially with the directed search, increasing the network size to 1000 agents with the number of connections set to only (N = 10), the system can satisfy more tasks than with a network size of 300 agents with the same number of connections per agent set N = 10. This is so under the same simulation parameters: i.e., the number of tasks issued in the system is the same in both cases. For example, when the network size was 300 agents and N = 10, the tasks with specific required accuracies were successfully completed at a rate of (7) = 61 per–cent. When the network size was 1000 agents with N = 10, the tasks with the same specific required accuracies were successfully completed at a rate of (7) = 94 per–cent. This is because increasing the network size, adding more agents, appears to be sufficient to increase the ratio



(a) Directed search



(b) Random search

Figure 4.7: *ANETR for network size: 300 agents with maximum agent's neighbourhood N: 20*

of executed tasks, and is more effective in this regard than increasing the number of the individual agents' connections, with a smaller overall network size. The results using random search follow the same trends as those yielded from using directed search in these terms, but, on average, the percentages yielded thus are less than those produced using directed search.

4.8.2 The Effect of Increasing Message Time To Live (TTL) Experiments

In this section, the results of increasing the messages' *TTLs* in relation to both the directed search and the random search algorithms are shown. This was done by allowing the *TTL* for the tasks to take the values 15 and then 20 while keeping the values of the agents' neighbourhood connections to a maximum of 10. This allows the tasks to have more time to traverse the network. So, the task message can be sent to more agents for possible delegation in the case of the task not being executed by the first receiving agents in the course of the search. Table 4.2 shows the setting parameters which have been used in this experiment. Various network sizes have been tested with various *TTL* values, without increasing the range of neighbourhood connections. The simulator ran for 2000 cycles and the runs were repeated 10 times.

Table 4.2: *The Input Data Parameters for the TTL Experiment*

Agents	Max N	TTL	TD	Max Tasks cycle
300, 1000	10	10	(11-13)	Normal distribution*T+M(5,300)
300, 1000	10	15	(16-18)	Normal distribution*T+M(5,300)
300, 1000	10	20	(21-23)	Normal distribution*T+M(5,300)
300, 1000	10	25	(26-28)	Normal distribution*T+M(5,300)

When the *TTL* was increased to 15 and then to 20 as shown in Figures 4.13 a, 4.13 b for *TTL* = 15 and 4.14 a, 4.14 b for *TTL* = 20 and where the network size was maintained at = 300 agents, both search algorithms showed slight increases in the ratio of executed tasks in relation to certain required accuracies. Overall though, the directed search showed better performance than the random search in relation to all the task required accuracies.

When the network size was increased to 1000 agents, applying the same changes to *TTL*, i.e., *TTL* =15 and then *TTL* = 20, did not greatly improve the ratio of executed tasks. This was due to the presence of an abundance of agents. This, on its own, led to the execution of more tasks with specific ranges of required accuracies. In general, though, increasing the *TTL* values when the network has

1000 agents, especially when using directed search, leads to the execution of more tasks than when the network size = 300 agents. To analyse this, increasing the *TTL* values means that the message can traverse more agents with various types of resources and this will increase the chance of finding most of the requested resources with the required accuracies issued from the customer. This leads to a higher percentage of tasks of specific accuracies being executed. The same is the case with random search but its performance is not as good as that of directed search.

Here we describe the experiments which we conducted to calculate the number of successfully executed tasks in each cycle of the simulation running time; which was 2000 cycles for all cases mentioned above in relation to modifying the number of neighbour connections and increasing task message *TTL* values. For this reason, the following formula 4.3 has been used.

$$ANSET = \frac{1}{N_R} \sum_{i=1}^{N_R} \sum_{x=0}^{x=2} |RV_x - AR_x| \tag{4.3}$$

Where:

N_R: number of runs = 10.

RV_x : the requested resources vector for the task.

AR_x : the matched agent resources that performs task successfully.

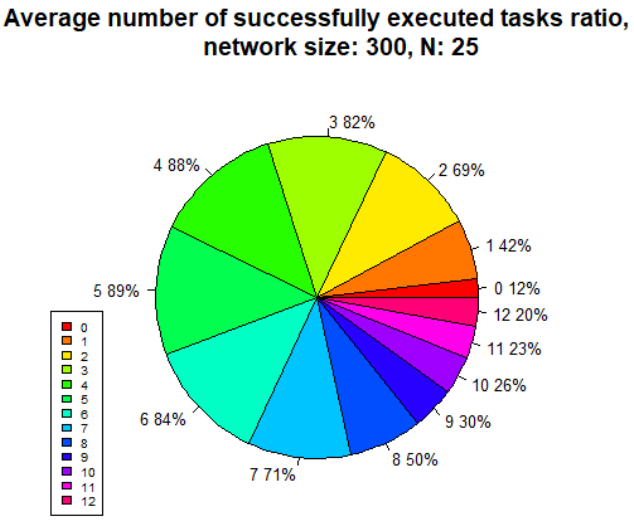
x = 0 to 2: the index of the requested resource vector, $<r_1, r_2, r_3>$.

In Figures 4.17 a and 4.17 b, both with a network size of 300 agents, the average number of successfully executed tasks in each cycle has been computed in relation to both the directed search and the random search algorithms. For both algorithms, when the number of connections was increased to 15, 20 and then 25, the number of successful task executions in each cycle also increased. This is because increasing the number of connections means increasing the chances of having a greater variety of neighbours with different types of resources. However, the directed search performed better than the random search for all of the connection values mentioned above.

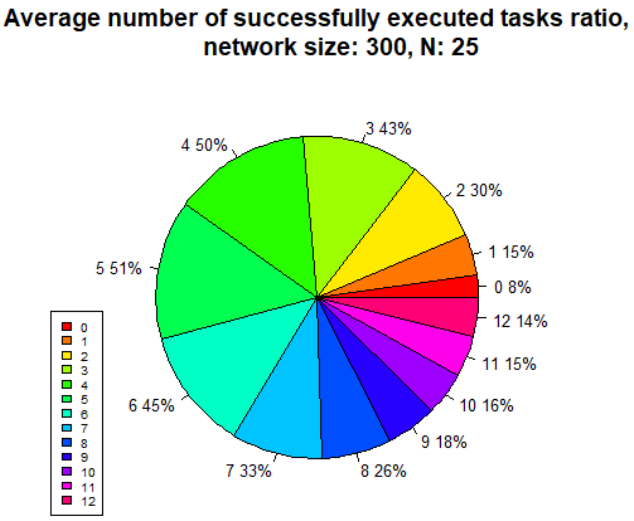
In Figures 4.18 a and 4.18 b, where the network size was 1000 agents, the average number of successfully executed tasks in each cycle for all numbers of connections was higher for the directed search than for random search for all the simulation cycles. This demonstrates that the directed search is a more promising algorithm than the random search. This is because, first, increasing the individual agent's neighbourhood connections means that the directed search can make use of this increased number of directly contactable agents to direct the search to the most proper agent in the contact list, this led to better performance than that of random search within the same simulation setting parameter. Also, a network

size of 1000 agents also contributed to the enhancement of the number of tasks executed successfully using directed search as compared to the same number in relation to random search.

The set of experiments described below were performed to determine the effects of increasing the *TTL* value on the number of successfully executed tasks in each cycle. The results of two different *TTL* values for a network size of 300 agents are shown in Figures 4.19 a and 4.19 b. A network size of 300 agents means there are fewer agents than arriving customers' tasks – customers send about 305 tasks in each cycle. As a result, increasing the *TTL* will lead to a small improvement in the number of executed tasks in each cycle. Figures 4.20 a and 4.20 b show the same results in relation to a network size of 1000 agents. With both of the search algorithms which were applied, increasing the *TTL* values increased the average number of successfully executed tasks within each cycle. This is because increasing the *TTL* means giving more time for the customers' tasks messages to be sent to more agents for delegation – especially important considering that the network size has been increased – and this will increase the probability of them being successfully executed. As can be seen, the directed search again outperforms the random search.

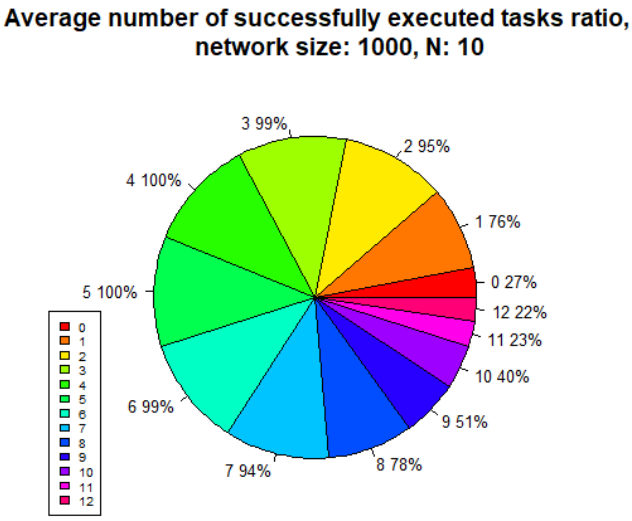


(a) Directed search

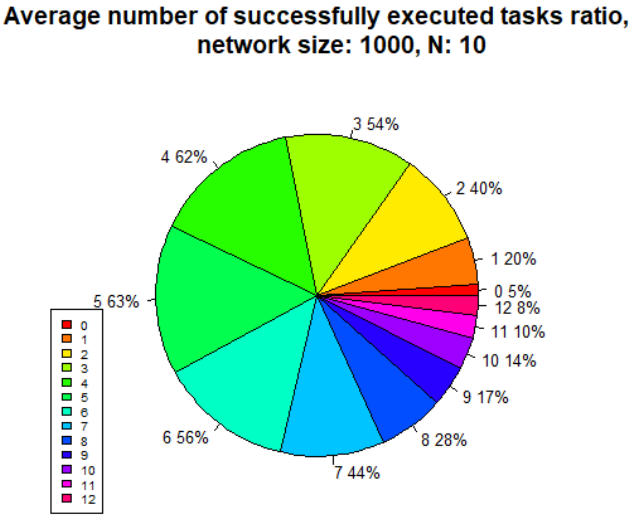


(b) Random search

Figure 4.8: *ANETR for network size: 300 agents with maximum agent's neighbourhood N: 25*

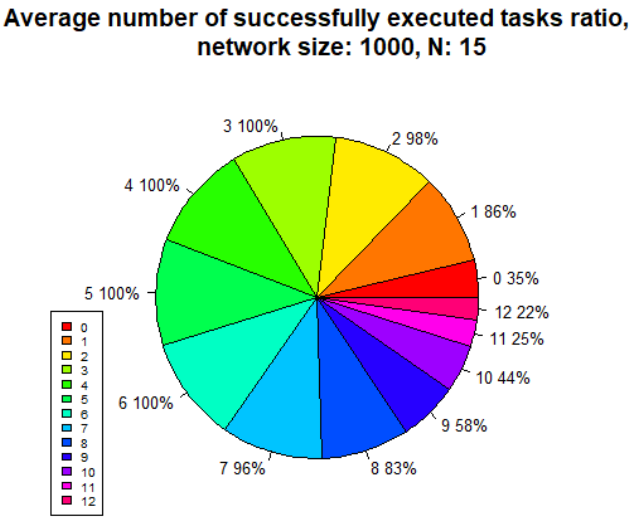


(a) Directed search

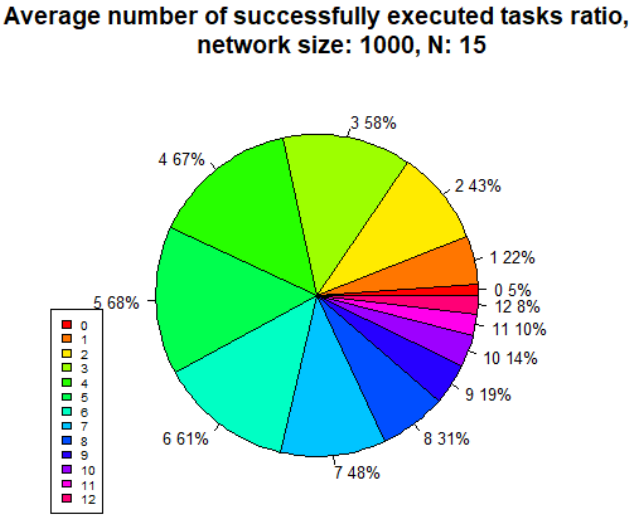


(b) Random search

Figure 4.9: *ANETR for network size: 1000 agents with maximum agent's neighbourhood N: 10*

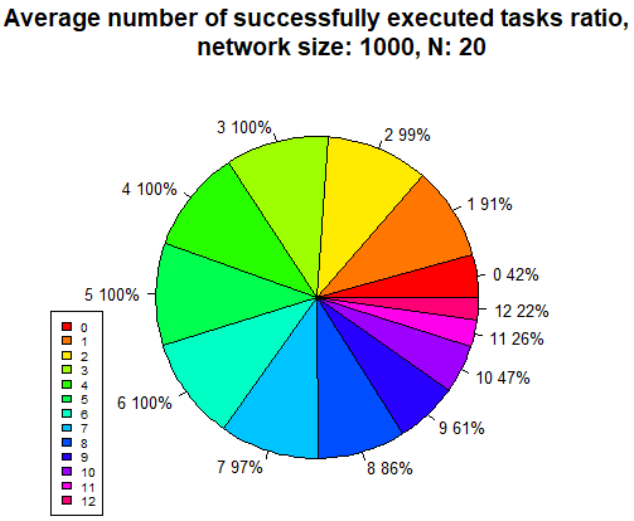


(a) Directed search

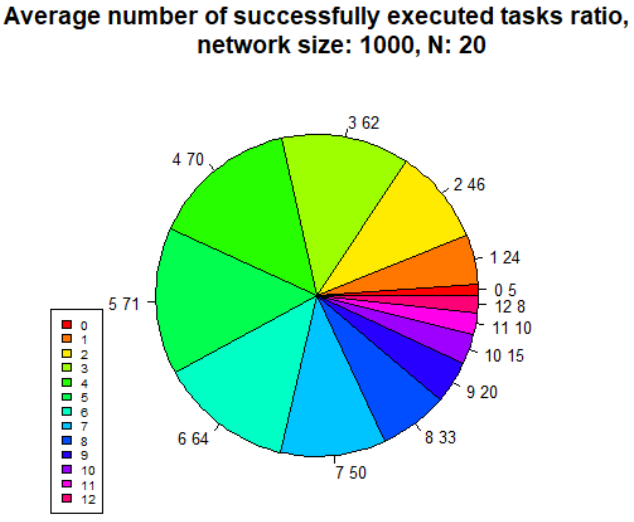


(b) Random search

Figure 4.10: *ANETR for network size: 1000 agents with maximum agent's neighbourhood N: 15*

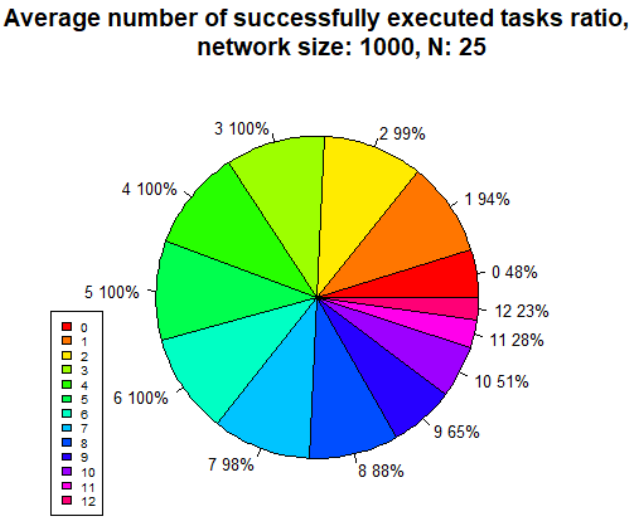


(a) Directed search

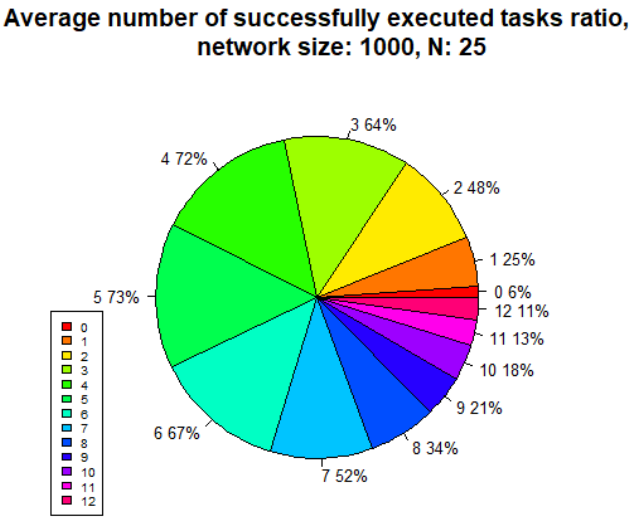


(b) Random search

Figure 4.11: *ANETR for network size: 1000 agents with maximum agent's neighbourhood N: 20*

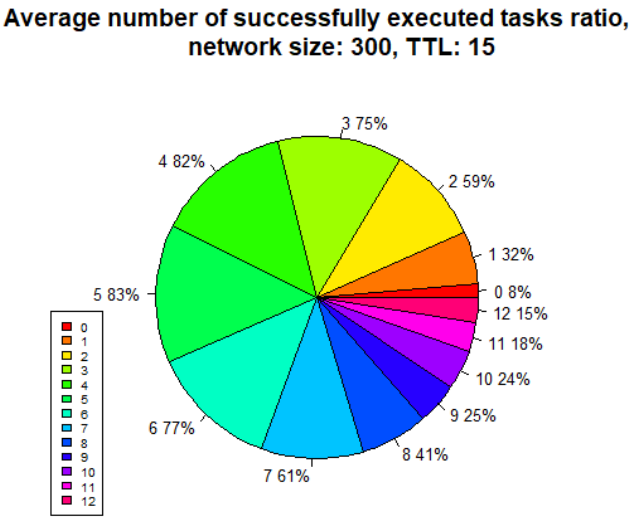


(a) Directed search

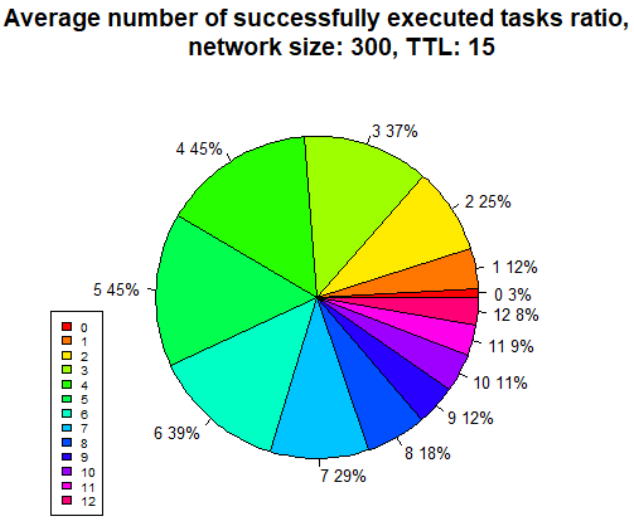


(b) Random search

Figure 4.12: *ANETR for network size: 1000 agents with maximum agent's neighbourhood N: 25*

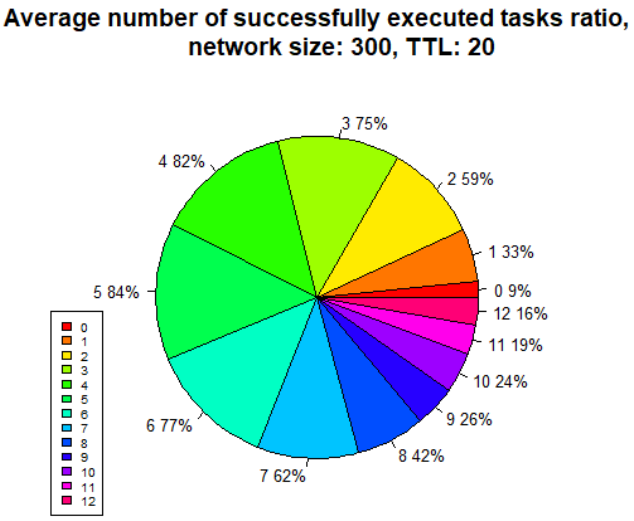


(a) Directed search

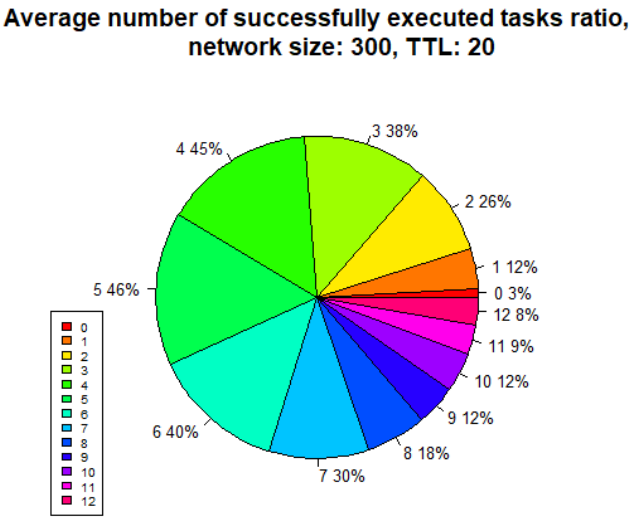


(b) Random search

Figure 4.13: *ANETR for network size: 300 agents with TTL value: 15*

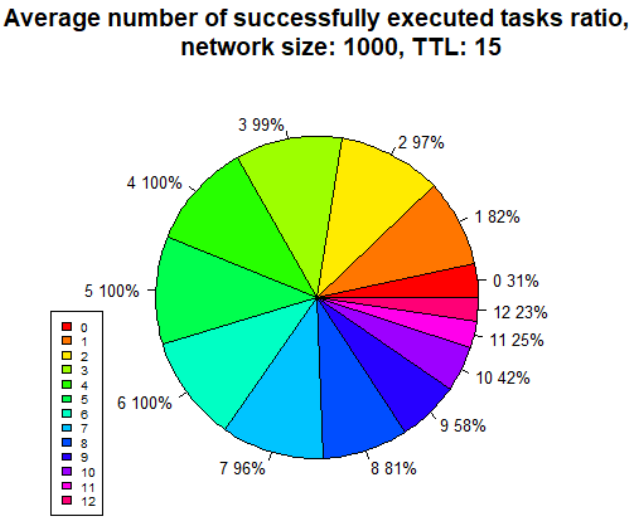


(a) Directed search

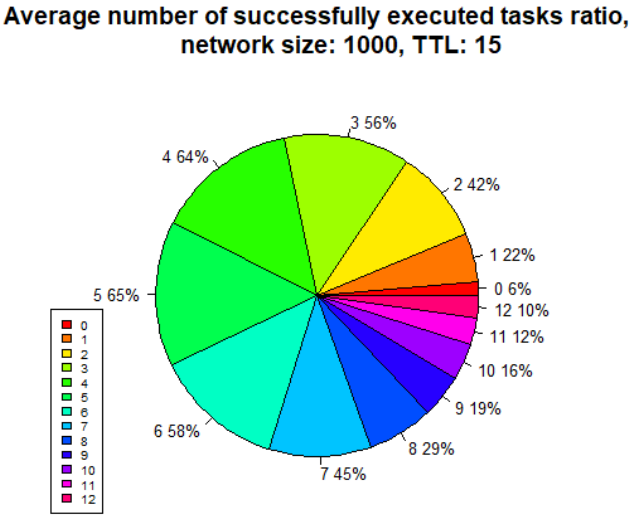


(b) Random search

Figure 4.14: *ANETR for network size: 300 agents with TTL value: 20*

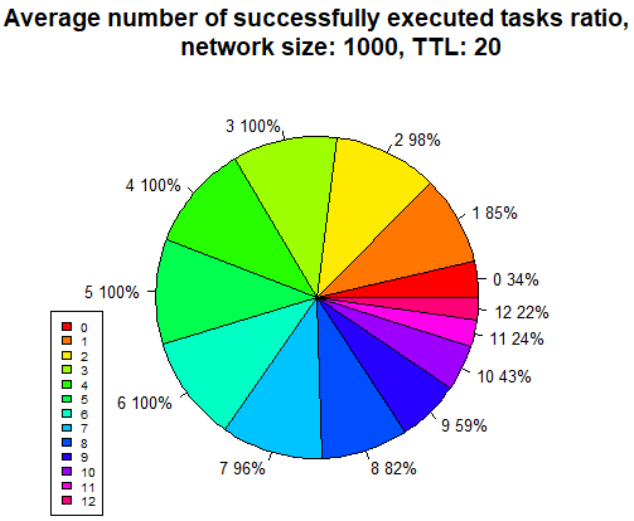


(a) Directed search

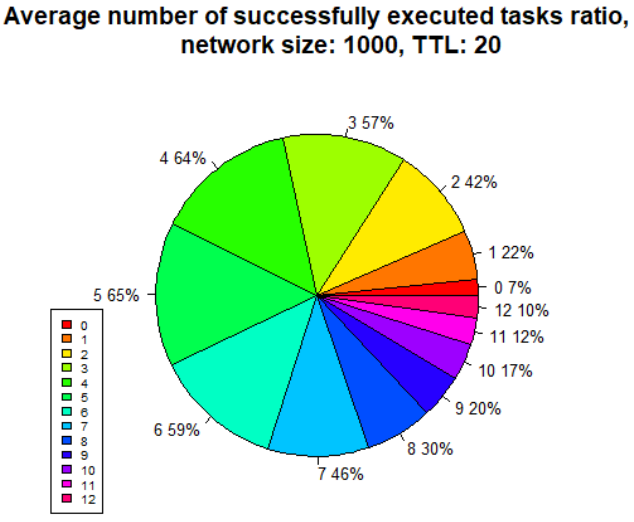


(b) Random search

Figure 4.15: *ANETR for network size: 1000 agents with TTL value: 15*

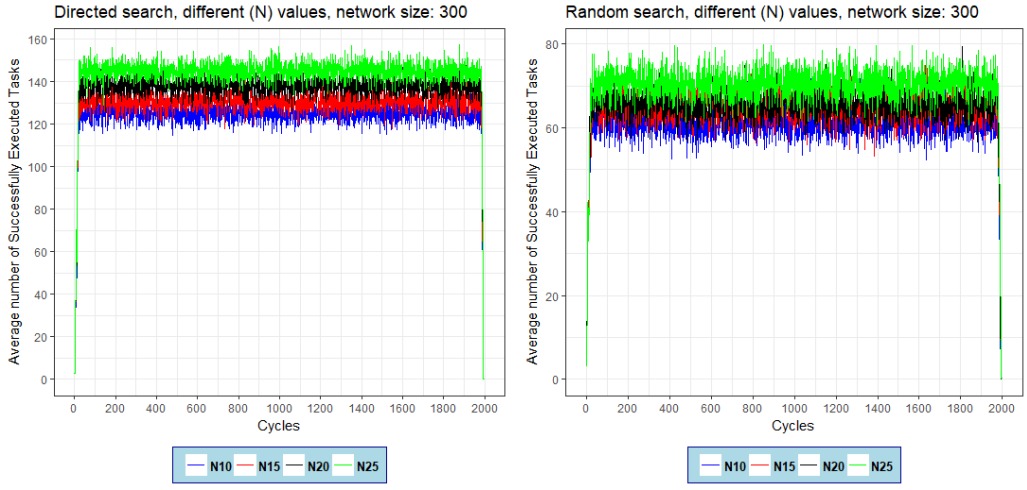


(a) Directed search



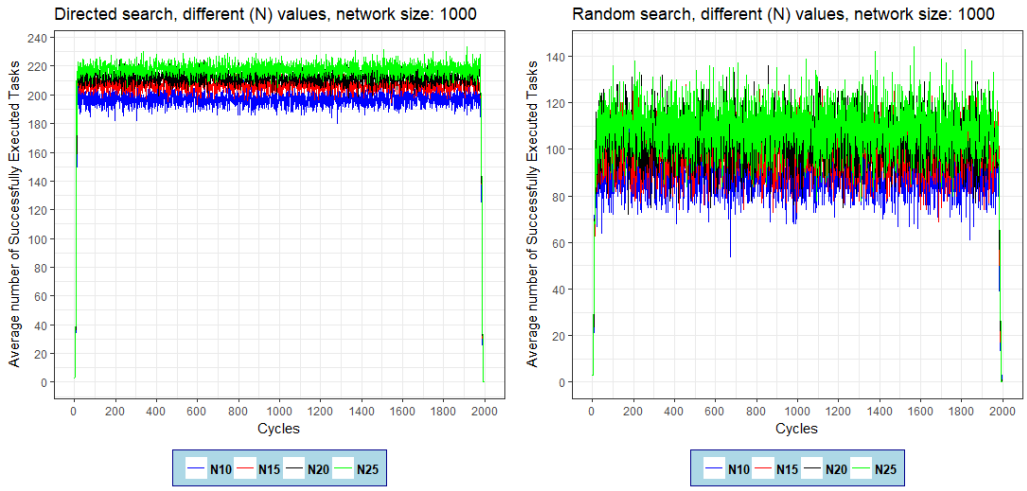
(b) Random search

Figure 4.16: *ANETR for network size: 1000 agents with TTL value: 20*



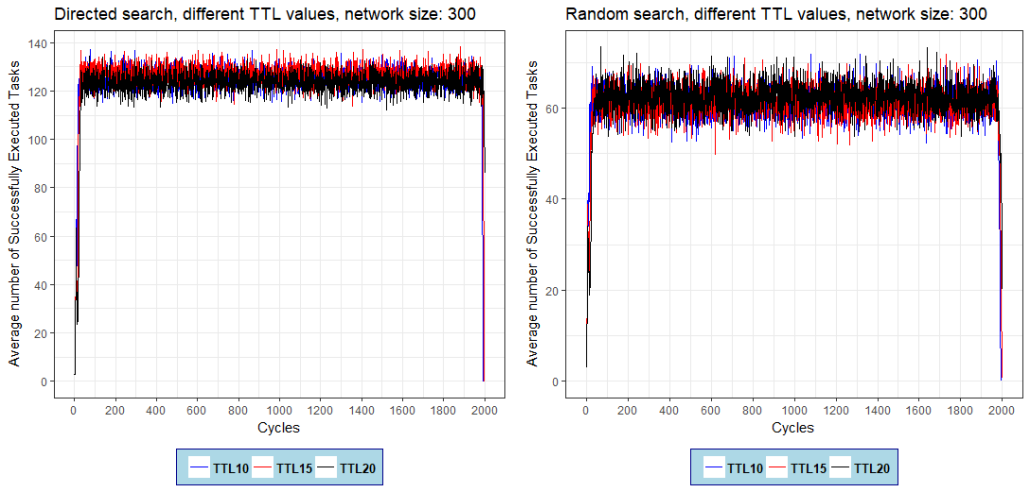
(a) Directed search (b) Random search

Figure 4.17: *Successfully executed tasks in cycles, network size = 300 agents, different N values*



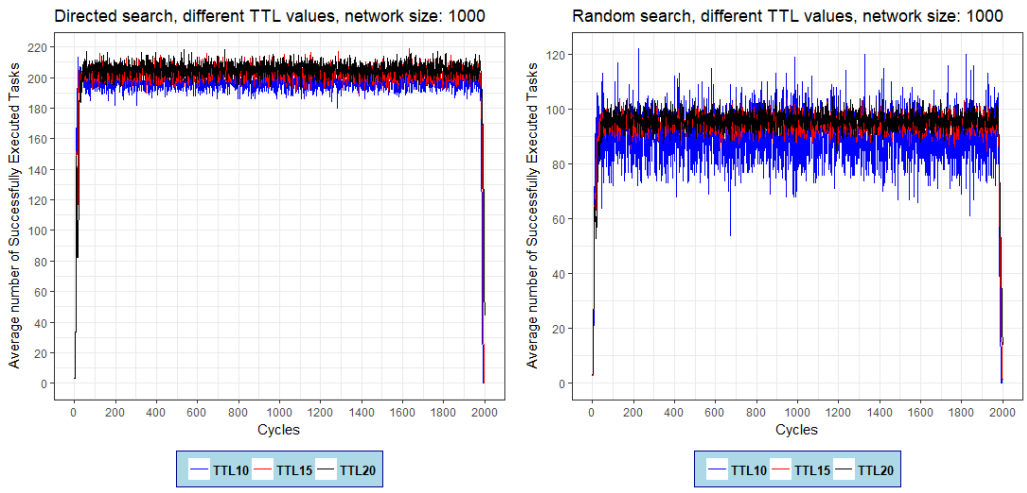
(a) Directed Search (b) Random Search

Figure 4.18: *Successfully executed tasks in cycles, network size =1000 agents, different N values*



(a) Directed search (b) Random search

Figure 4.19: *Successfully executed tasks in cycles, network size = 300 agents, different TTL values*



(a) Directed search (b) Random search

Figure 4.20: *Successfully executed tasks in cycles, network size = 1000 agents, using different TTL values*

4.8.3 Message Traffic in The Network

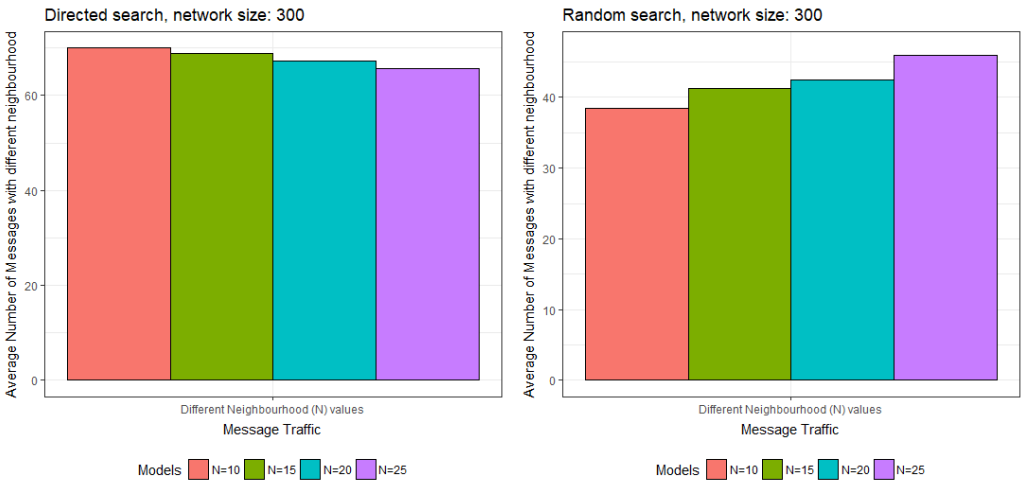
In this section, the amount of message traffic occurring as a result of the delegation process is discussed – in relation to increases in the neighbourhood connections and the messages' Time to Live *TTL* values. Changes to these change the traffic status of the network, as does the use of the directed search algorithm as opposed to the random search algorithm. To give a brief description of the tasks issued by the customer, the number of customer tasks which will be issued in each cycle is a value generated from a normal distribution with variance = 5 and mean = 300; hence, roughly the maximum number of tasks issued in each cycle is around 300.

We have computed the traffic in the network, $MsgTraffic$, using the following formula:

$$MsgTraffic = \frac{\text{NoMsgs}}{\text{TRT}\times\text{TTL}} \times 100 \tag{4.4}$$

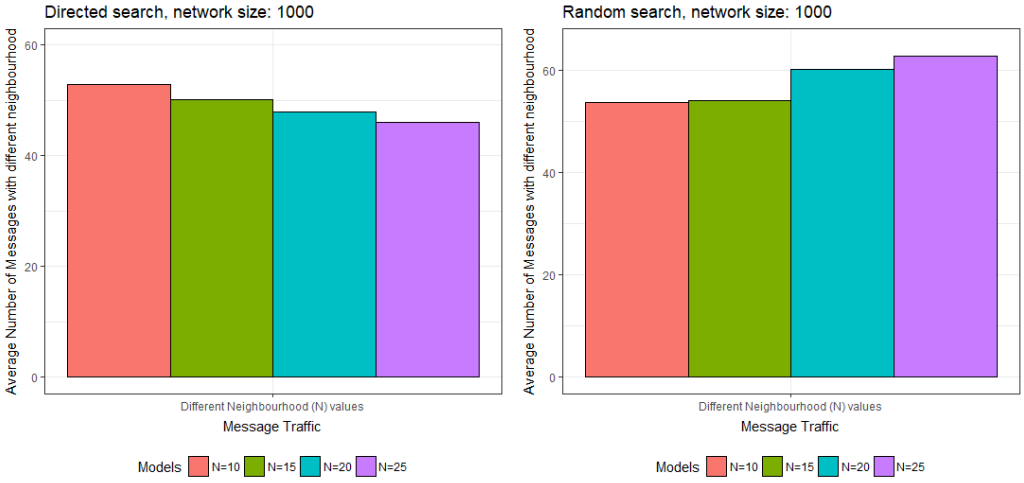
Where:
NoMsgs: The accumulated number of messages in the network for all the received tasks.
TRT: The total number of received tasks.
A number of experiments were conducted to investigate the effect of increasing the neighbourhood connections on network traffic while keeping the value of *TTL* = 10; the results of these experiments are shown in Figures 4.21 a, 4.21 b,4.22 a and 4.22 b. On average the message traffic when the system uses the directed search algorithm with network sizes of either 300 or 1000 agents decreases with increases in agents' neighbourhood connections; this is to be considered a good sign, and with random search, there are increases of network traffic with increases in the agents' neighbourhood connections.

Looking at the comparison between networks of size 300 agents and networks of size 1000 agents as shown in Figures, on average, when applying the directed search algorithm, the networks of size 300 agents have more traffic than the networks of size 1000 Agents. This is because when the network size is 300 agents, agents are more likely to be busy than when the network size is 1000 agents. The busy agents will delegate the receiving tasks to other agents in the network and



(a) Directed search (b) Random search

Figure 4.21: *The message traffic generated by 300 agents with different N values*



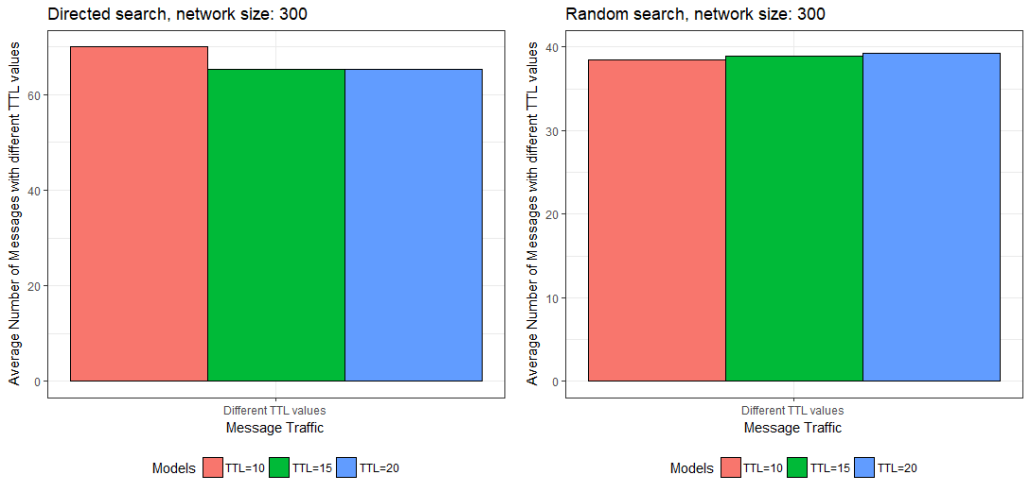
(a) Directed search (b) Random search

Figure 4.22: *The message traffic generated by 1000 agents with different N values*

so on – leading to a high volume of traffic. The number of tasks issued by the customers also plays a role in this behaviour as it is the same number of tasks for both network sizes. When random search is applied, it is noticeable that the traffic increases when the network size is increased to 1000 agents. This is because the technique of random search is based on randomly selecting neighbours for the receipt of the customer messages during the delegation process and as the neighbourhood increases the delegation requests increase accordingly.

After analysing the effects of changing the neighbourhood value, N, on the message traffic in the network, the TTL value was then investigated. This was

done by increasing its value and then looking at how this affected the number of messages (i.e., the traffic) produced with network sizes of 300 agents and the size of the neighbourhood is 10 and *TTL* =(10,15, 20). In Figures 4.23 a and 4.23 b. We note that the network traffic increased with increasing *TTL* values in random search and with directed search the traffic, on average, started to decreased. This means that the agents in random delegation algorithm trying to use all the task's *TTL* value with the hope to execute the task until its *TTL* expired. Meanwhile, in directed search the traffic starts to decreased in the network when the task's *TTL* value has increased this is because the directed search may not use all the *TTL* to find the proper resources to accomplish a task.



(a) Directed search (b) Random search

Figure 4.23: *The message traffic generated by 300 agents with various TTL values*

Figures 4.24 a and 4.24 b show the directed and random search with network size = 1000 agents, N=10 and different *TTL* values. When the random search was applied, the increase in traffic was greater when compared to the situation with directed search. This can be seen from the average values shown in the figures.

(a) Directed search (b) Random search

Figure 4.24: *The message traffic generated by a 1000 agents network with various TTL values*

4.9 Disruption in Agent Service Provision

From the analyses that have been carried out in the previous sections, it is clear that the directed search was the algorithm that could be counted on to delegate the customers' tasks through the network and the one that could provide the customers with the required level of successfully executed tasks – with greater efficiency than the random search algorithm. Therefore, from this section on, only the directed search will be adopted to delegate tasks in the network.

In this section agent failure problem has been applied to the system to study the effect of disruption on task execution and system performance. Agents in the network may lose their connectivity, i.e., they may go offline. To model this, agents are equipped with a parameter that enables them to be switched on/off for a period of time, and when they are off they, in essence, create the impression to the system that they are offline and unable to execute tasks or respond to messages. We have skewed a probability distribution between[0,1] such that a lower (nearer 0) probability value produces fewer failed agents and a higher probability value (nearer 0.9) will produce a larger number of failed agents in the system. During the simulation initialization step, all agents are assigned the same probability of failure. For example, if we have 300 agents and the probability is set as p=0.2, then all the agents will be assigned the probability p=0.2 when the system imposes the

probability that in one time could be the same as the currently active agent, that agent will fail for a random number of cycles to demonstrate that it can not accept tasks or run its currently task. After returning to operational mode, the agent checks the workload that it already has. It will start executing tasks if there any in its workload have not yet reached their deadlines.

Applying a high failure rate simulates the harsh operational conditions which exist in some systems: i.e., it progressively increases the chance of failure within the system.

After an agent has been offline for a period of time, it will check its current task and its *ATQ*, if the deadlines for its tasks have not been reached, then the agent will try to execute them. Otherwise, the agent will declare the tasks to have failed. The on/off parameter has been applied using the probability values of 0.9, 0.5 and 0.2. Algorithm 3 shows the steps that have been carried out to implement this.

Algorithm 3 Agent Failing Algorithm

Input: Initialize the individual agents with a failure probability value ($ex.prob$ = 0.6), an agent a_i where i = 0 to M, M is a maximum number of the available agents in the system. match_probability= $(0.9 - 0.1)$

Output: a_i.status= failed(true) or not failed agent(false)

```
1: Cycle_K
2: match_probability =Random (0,1)
3: if a_i.prob == match_probability and a_i.status ≠ true then
4:    a_i.fail=random_cycles(1,m)
5:    a_i.status =True
6: else
7:    a_i carry on executing task(s)
8: end if
9: Cycle_K+1
```

4.10 Experimental Work

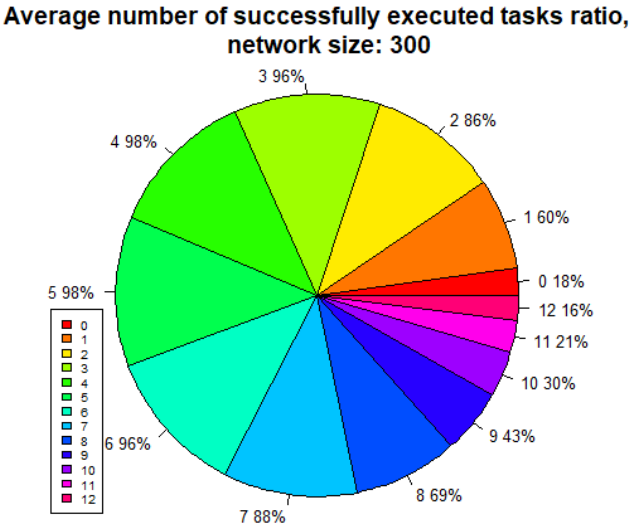
The experimental work in this section implements a simulation of the problem of agent disruption and its effects on system performance. To carry out the experiments, two network sizes have been tested with differing failure probability

values and the same task distribution has been applied here. Figure 4.25 a shows the situation with a network size of 300 agents, without failure, while the other Figures 4.25 b, 4.26 a, and 4.26 b show the situation with the same network size but with the effects of agents failure on the performance of the network. It can be seen that increasing the failure probability leads to the loss more tasks with various types of required resources.

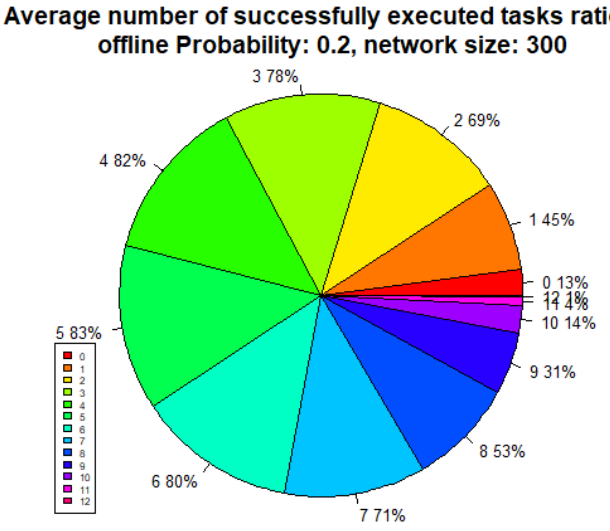
Figure 4.27 a shows the situation that pertains with a network size of 1000 agents, and without a failure probability being applied, and Figures 4.27 b, 4.28 a and 4.28 b show the situation with the same network size but with failure probability values being applied. These figures show that even when the size of the network is increased, agent failure can have a negative effect on the network's performance and the agents' utilization. This means that the system cannot always provide the customer with successfully executed tasks.

The average number of successfully executed tasks in each cycle has been calculated for both network sizes (300 and 1000 agents) as shown in Figures 4.29 a and 4.29 b. The average number of successfully executed tasks in each cycle is decreased when the probability of failure is increased – as more agents fail.

In this section, we have presented introductory work which examines the failure problem in the distributed system. This shows that the deployment of autonomous agents in dynamic and distributed systems is not sufficient to maintain these systems at an appropriate level of functionality without providing these entities with special capabilities and mechanisms which allow them to be self-motivated, adapting themselves to the disruption problem in the distributed environment. The distribution of such a scenario has been achieved by creating self-organised organisations in the system environment that help the system to keep functioning even with the existences of failure issues and this what we will demonstrate in chapters 5, 6, and 7.

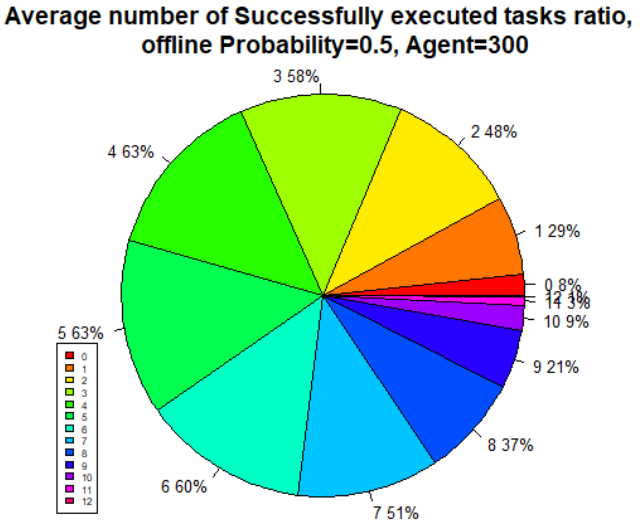


(a) Directed search p=0.0

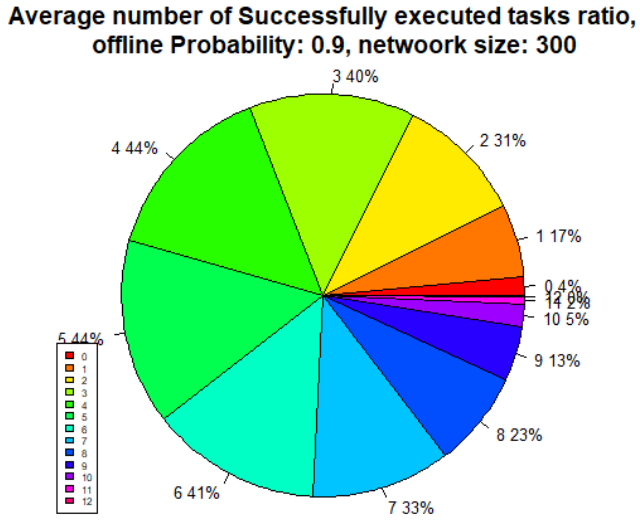


(b) Directed search p=0.2

Figure 4.25: *ANETR for network size=300 agents with probability of failing p=0.0 and p=0.2*

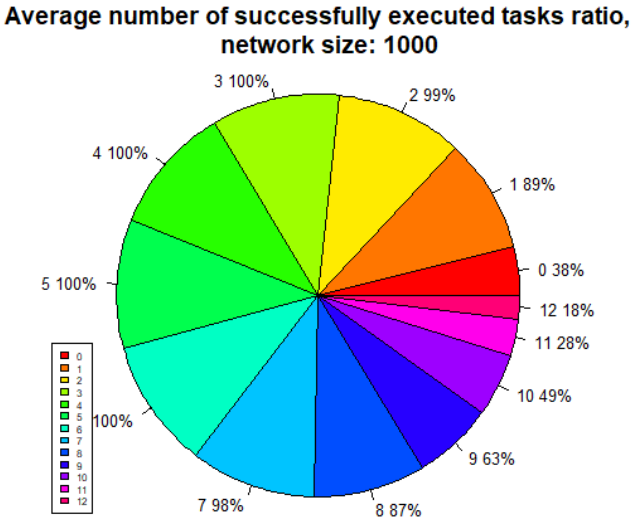


(a) Directed search p=0.5

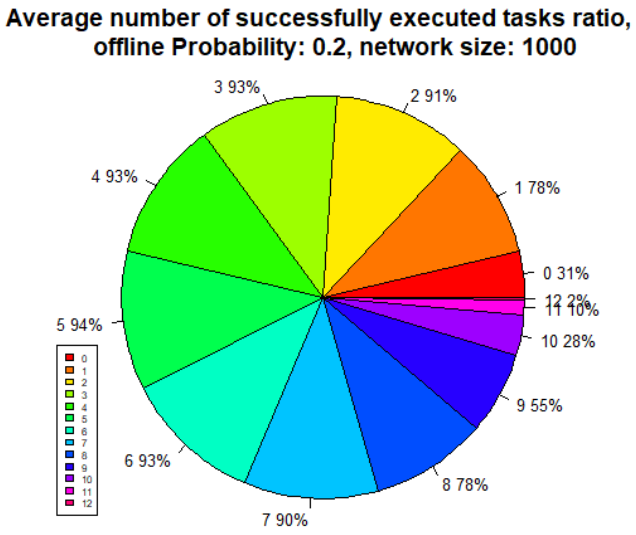


(b) Directed search p=0.9

Figure 4.26: *ANETR for network size=300 agents with probability of failing p=0.5 and p=0.9*

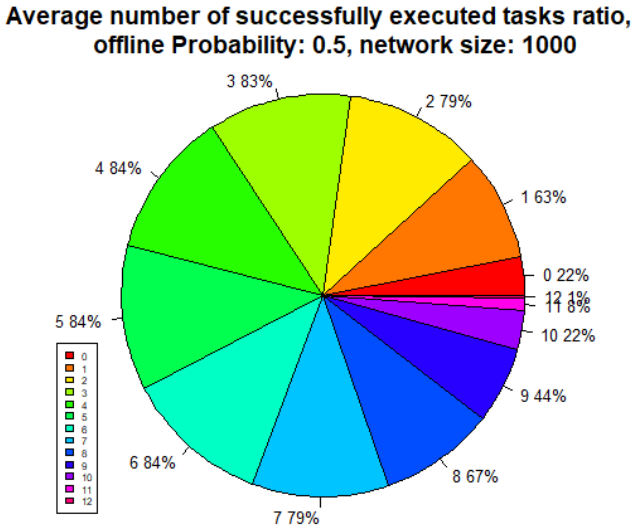


(a) Directed search p=0.0

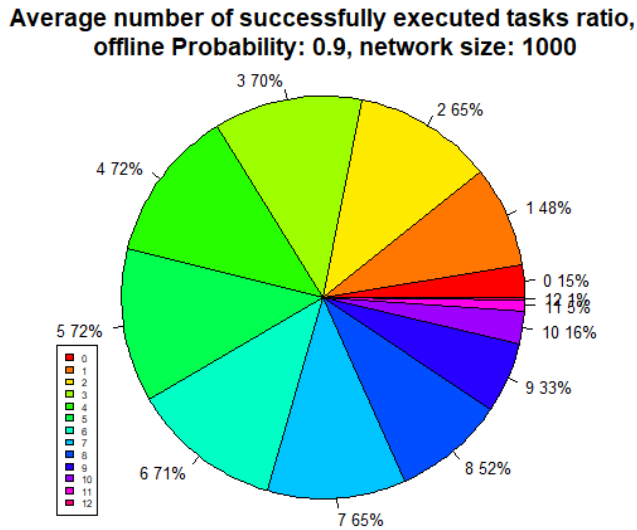


(b) Directed search p=0.2

Figure 4.27: *ANETR for network size=1000 agents with probability of failing p=0.0 and p=0.2*

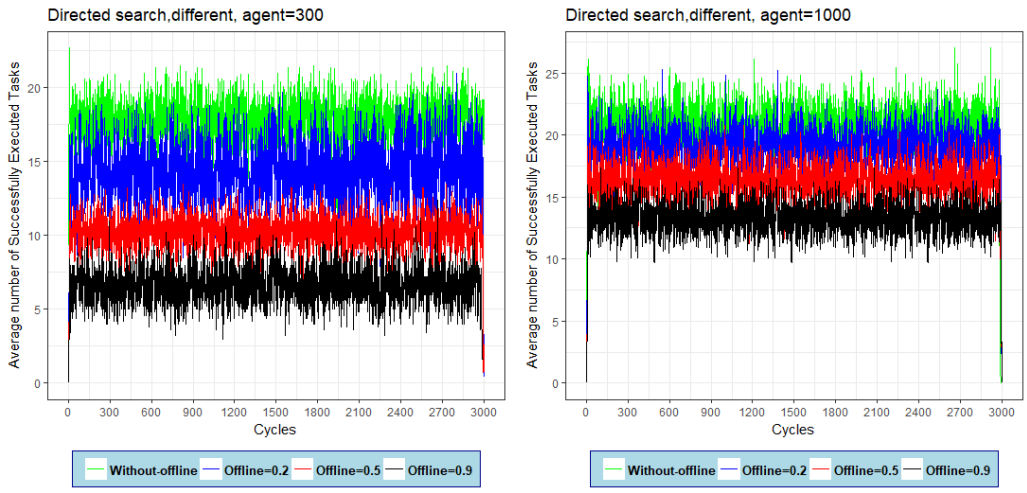


(a) Directed search p=0.5



(b) Directed search p=0.9

Figure 4.28: *ANETR for network size=1000 agents with probability of failing p=0.5 and p=0.9*



(a) 300 agents (b) 1000 agents

Figure 4.29: *ANSET for network size = 300 and 1000 agents with probabilities of failing*

4.11 Chapter Summary

This chapter has investigated the main components of the simulated network system which has been constructed from autonomous agents. Network agents are the entities which can adopt the mechanisms which can deal with the failure problem in the grid computing environment.

The contribution which has been presented in this chapter is the implementation of networks which allow for the use of autonomous agents to encapsulate the heterogeneous computational resources and to provide a better service for simultaneous and multiple customers' requests. Furthermore, two delegation algorithms, one using directed search and the other random search have been investigated in relation to the requirement to distribute tasks in case of the receiving agent not being able to accept and execute tasks. These two algorithms have been implemented and compared against each other: a number of experiments have been implemented to study which algorithm is more applicable.

The directed search algorithm has been shown to result in better performance in relation to task execution in the network than the random search algorithm. The Average Number of Successfully Executed Task Ratio (*ASETR*) and the Average Number of Successful Executed Tasks in each cycles (*ANSET*) for the directed search is higher than those same measures in relation to random search. The directed search algorithm is based on a heuristic choice for the next agent to navigate the message. Hence, directing the task message across the network using agents' statuses; busy or not busy decreases the possibility of losing tasks as compared to random search. The technique of random search is based on randomly selecting neighbours for the receipt of the customer message during the delegation process. For both algorithms, each agent maintains its own queue to increase its benefit and its resource utilisation. The resources have been modeled as a vector of resources with specific required accuracies to simulate the semantic representations for resources in heterogeneous networks. In all of the experimental work, the directed algorithm has been compared against the random search algorithm. For this comparisons, various different setting parameters have been applied in-

cluding various network sizes, neighbourhood values N, and TTL values. The comparisons have shown that the directed search outperforms random search in term of the $ASETR$, $ANSET$ and message traffic generated in the network.

Some resources are rarer than others and tasks requiring them have a lower rate of completion, in terms of percentages. This effect is dependent on the require accuracies issued by the system and the availability of such resources in the network. Furthermore, after selecting the directed search for use here, another set of experiments were undertaken to simulate disruption in the system. This was done by applying a probability of failure to the agents in the network, whereby a number of probability values were selected, such as (0.2, 0.5 and 0.9), and studying the effects of failure on the agents' performances as well as on the network's performance as a whole. The experimental work has shown that agent failure can significantly affect customer task execution and the utilization of the agents in the system. This problem will be addressed, broadly, and suggested mechanisms for its solution will be described in the next chapters.

Chapter 5

Task Delegation through Agent Organisations

5.1 Introduction

Grid computing is one of the types of distributed systems with which virtual organisations are a good fit; sharing resources is one of the most important concepts associated with the topic of grid computing [93], [5]. A grid computing system consists of heterogeneous resources; it is crucial to discover usable methods for ensuring that such resources can be utilized with maximum efficiency. The use of emergent organizations has been demonstrated to be capable of providing reasonably effective solutions for many of the task allocation problems found in distributed environments [94], [95]. These solutions can be used to minimise resource allocation costs and reduce unnecessary communication among agents within organizations [18]. In this chapter, we present the mechanisms we propose for the maximization of resource utilization, the improvement of task execution and for delegation, as well as those aimed at dealing with the problems caused by agent failure – via the creation of virtual organisations of agents. We attempt to demonstrate that the performance of a network of agents can be improved by the use of a self-organisation technique whereby agents can self-organise into emergent clusters/organisations. The principle of self-organisation has been used

to create agent organisations; this activity is triggered when some of the agents experience excessive loading. Hence, busy agents within the network may decide to create an organisation so that they can receive extra support from other, less busy, agents so that, in turn, more tasks may be executed. By this means, the workload will be distributed over the newly created groups. Here, a self-organisation process has been used to show the positive effects of local action performed by individual agents, intended to increase the task execution throughput of the network and hence to improve overall network performance. The target of this research is the creation of dynamic organizations of agents whose *Members* may take on roles so that their work as societies is enhanced. Other researchers, such as [40], [96], have explored this area. However, most of the methods proposed in the existent literature present large networks as separated communities, while most real-world large networks are composed of significantly overlapped and nested communities [20]. The agents presented in this work may participate in different communities/organisations, and this ability varies from agent to agent – so demonstrating the autonomous agents' differing abilities.

In this chapter, two different mechanisms have been explored whereby organisations may be created. The first creates homogeneous organisations, all of whose *Members* have semantically similar resources. The second method creates heterogeneous organisations from the various dissimilar agents that are available in the network. In relation to both of these suggested mechanisms, a task delegation protocol is proposed, based on delegating tasks to the *Members* of the organisation instead of to isolated individual agents in the network. In each of the created organisations, there is a *Head* agent which will act as the decision maker in terms of delegating tasks to the *Members* of its organisation.

In chapter 4, we presented two delegation protocols: the directed search and the random search. Directed search demonstrates better performance as compared to random search. Directed search improves the delegation process by taking into account the status of agents (busy/ not busy) when choosing the next agent to receive a task allocation message. In contrast, random search delegates the task message purely randomly. Also, we show that there are disruptions which

can affect the system's performance as demonstrated by resource utilisation and task delegation across the network. Therefore, the introduction of organisations here is needed because we want to improve system performance – by the delegation of tasks, by increasing the throughput of tasks, by reducing the number of messages and as a means to deal with disruption in the system.

5.2 Creating Organizations

The network design presented in chapter 4 allows for a network to grow gradually as agents connect to each other randomly and thus create a large network. Agents in the network created possess heterogeneous resources and so are able to execute many different types of tasks as issued by the customer agent. Each agent needs at least one connection with another agent in the network and each can sustain up to a maximum number of connections; this can be a different number from agent to agent. Each agent maintains partial knowledge of the network *Members* by creating a contact list containing the contact details of all the agents to which it has a direct connection. The tasks are delegated to agents based on their status, and task messages may need to traverse many agents in order to either succeed or fail. The agents are autonomous and self-interested and have the goal of creating the maximum benefit for themselves. Manifesting the way in which agents are self-interested, each agent in the network has a queue ATQ containing the tasks it has accepted for imminent execution. Hence, when an agent in the network accepts a task, this means that the agent has met all the crucial requirements specified by this task, and so its status will become busy and will remain at this state until the task has been executed when it will be changed to not busy. If its queue has tasks, then the agent will become busy while it executes the tasks in its queue.

However, we are still left with problems which lead to lost tasks and decreased agent utilization. The first problem is that tasks may require more hops for their execution and this leads to there being more messages in the network which, in turn, will create more traffic in the network. Second, agents may receive tasks

which they cannot execute for a number of different reasons, such as the task deadline allowance is not sufficient for the receiving agent or the agent does not have the required accuracy, etc. Third is the problem of disruption caused by agent failures; this may lead to more tasks being lost.

In response to these circumstances, here we propose the idea of creating organizations of agents; this mechanism is to be based on gathering autonomous agents in groups as a virtual layer above the existing agent network in order to enhance the task delegation process. We assume that this will lead to the recovery of more tasks, an increase in the number of executed tasks and a minimization of the time required for accepting and executing tasks. This will be effected using a new task delegation process, within each organisation. This delegation process aims to find an agent which can most precisely satisfy the requirements of the requested task with a single hop or with a few hops only.

To simplify matters, the explanation here will focus on only one organisation. However, the system as a whole may well contain more than one. The emergence of agent organisations will depend on a triggering condition resulting from the most-busy agent's activity. The most busy agent is an agent that can satisfy the following conditions:

(a) Currently executing a task.

(b) Still receiving tasks from the customer.

(c) Already having tasks in the accepted tasks queue. $ATQ \geqslant K$; ATQ contains K accepted tasks – $K \leqslant W$, where W is the maximum size of ATQ. Hence, the accepted tasks in the queue will require at least Y cycles to be completed, where Y is a random value related to the deadline of each task.

The created organisations can be defined as: $Org = \{Org_1,..., Org_q\}$, where Org_p with $1 \leq p \geq q$, is the pth organisation in the system, hence the elements in each organisation are:

$Org_p = < OrgID_p, OrgH_p, \{a_1,...a_{no}\}, Z_p >$, where $1 \leq no \leq Z_p$.

- $OrgID_p$: a unique identifier for each organisation created.
- $OrgH_p$: is the name of the *Head* of an organisation.
- $\{a_1,...a_{no}\}$: the participants in the organisation at a specific time.
- Z_p: is the maximum size of each organisation.

The most busy agent can decide to initiate the process of creating a new organisation and to become the *Head* of that organization – in order to receive more help. The *Head* will then send a multicast message to its contact list neighbours in order to invite them to join its organisation. This is performed as shown in Algorithm 4 which is based on the multi-cast "Push Gossip" algorithm [25] which, for purposes of brevity we call the gossip algorithm. This action will propagate the message across the network. The authors of [25], in their chapter 7, clarified the essential properties of gossip-based information dissemination and showed how the various gossip processes can be utilized, not only in human society, but also in other domains such as computer networks. The push gossip protocol may produce slightly elevated numbers of messages when spreading the gossip message across a network. This is because an agent that receives the gossip message may send it to other random agents that may already have received it. However, the overhead that is generated by this process is not unacceptably high. Several gossip protocols have been explained in [25] and they have been used in many studies in the literature which look at self-organising agent societies [97], [98]. In this work, we study the gossip protocol and show that this is the most applicable method in terms of the conditions that must be met. The *Head* will spread its message to the agents in the network to create diverse (in terms of resources and agents) organisation with various agents; this set-up can help to execute more tasks within a shorter duration, so our approach was to look in more detail at this network structure and at what the gossip protocol can provide.

Within this network environment, if a not-busy agent accepts an invitation message (to join an organisation) then it will respond with an acceptance message "accept to join" to the *Head*. If, on the other hand, the message reaches a busy agent, then the busy agent will act as a transmitter so that the *Head's* message

can reach other parts of the network; the gossiping process stops when the *Head* message returns back to the *Head* or the *TTL* for the message has expired. Hence, the size of an organisation may vary based on the status of the agents that receive the *Head's* message. In this way, the system has added a virtual layer "an organisation"over and above the network of agents. The resulting organisations are shown in Figure 5.1.

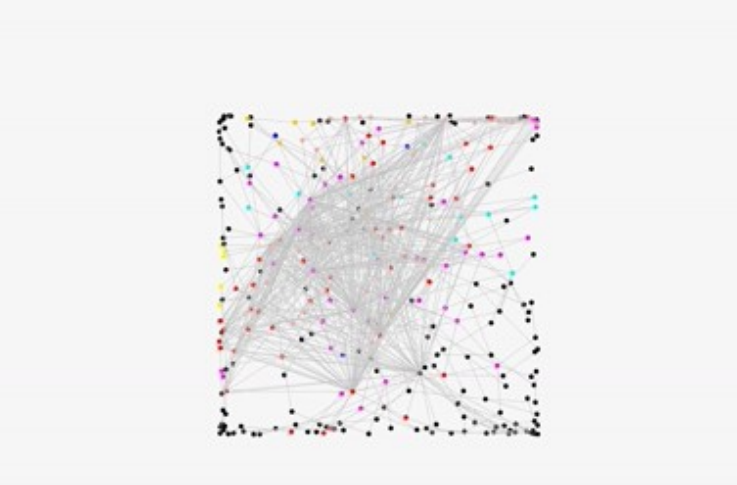


Figure 5.1: *The Created Organizations*

A database record will be added to the *Head's* database for each agent who joins the organisation. The *Head* holds the following information on its database about its organization.

$H\_DB = \{OrgID_p, OrgH_p, a_{ID}, Resource\_i\}$

In the literature, grid and cloud computing are shown to be focused on providing a powerful range of resource sharing facilities for their distributed environment. However, it has been found that in an organisation that is part of a grid computing system, a computational node can be underutilised and not fulfil its potential; i.e., it may be utilised (busy) only 5% or less of the time [5]. This motivated us to create overlapped organisations; we aimed to increase the agents' resource utilisation by allowing agents to join more than one organisation. However, in practice, depending on how busy it is, an agent can only be committed to a limited number of organizations at any one time and this number depends on its setting parameter – which varies from agent to agent. So, this will lead to the creation of overlapping communities/organisations as shown in Figures 5.2

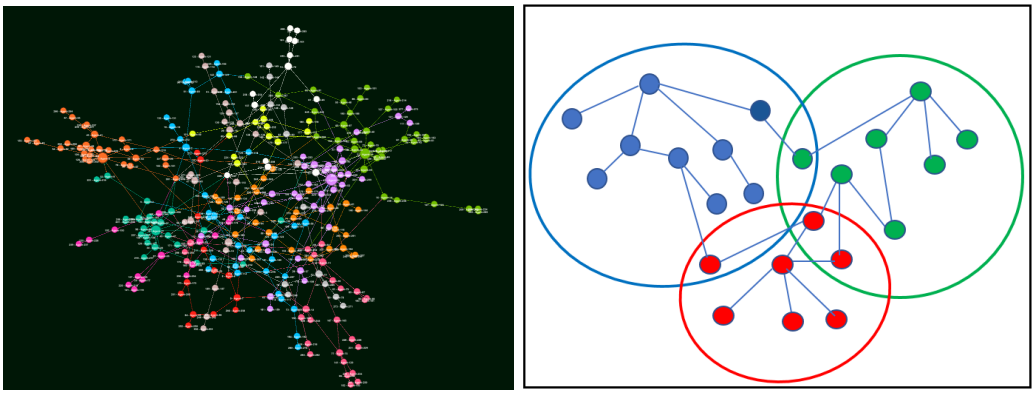
Algorithm 4 Gossip Protocol

Input: a_i: is the most busy agent (*Head*) that will create an organisation, a_l is one of the a_i contacted list neighbour, $TTL > 0$.
Output: a_i completed organisation.

```
1: Cycle_e
2: a_i.SendMessage(Contactlist(1,N))
3: if a_l ! = busy then
4:    a_l infected with the gossip message of a_i
5:    a_l has the option to join or not to the created organisation.
6: else if a_l == busy then
7:    a_l.SendMessage(1,Random(N))
8: end if
9: if TTL > 0 then
10:   TTL = TTL − 1
11: end if
12: Cycle_e + 1
13: End
```

a and 5.2 b. For example, if a *Member* agent receives a task and cannot execute it, then it will send the task initially to the *Head* of the first organisation that the agent joined – since it can join more than one organisation. The *Head* will send the task to its *Members* and if none of them can accept the task, the *Head* may then send the task to another *Member* which can satisfy the resource requirements/ accuracies; it will continue to do this for all the organisations it has joined. If the task cannot be executed by any agent of any of the organisations that either the *Head* or the original *Member* belong to, then the *Member* will operate as if there are no organisations: i.e., the task will be delegated to one of the *Member's* neighbours which can comply with the *TTL* and deadline constraints.

Each agent will hold the required information for all the organisations it has joined. Any agent joining an organization must satisfy a set of rules, as follows: agreeing to execute tasks or put them in the accepted task queue if the required accuracy is matched and the agent is able to execute the task within the allocated deadline. The following section will illustrate the two different mechanisms which we have used to create organizations of agents.



(a) Several overlapped organisations structure.

(b) Simplified structure, an example of overlapping organisations where q=3

Figure 5.2: *Illustration of the concept of overlapping organisations. In Figure (b), the Blue organisation overlaps with the Green and the Red organisations in terms of a single node, whereas the Green overlaps with the Red in relation to two nodes. These overlapping regions are in the intersection of the large circles.*

5.2.1 The Creation of Homogeneous Organizations

The mechanism we first used to create organisations utilises autonomous and heterogeneous agents in the network to create homogeneous organizations in which the agents are clustered together depending on similarities between their resources in relation to predefined required accuracies.

- Agents will have the opportunity to decide which organization to work with based on a semantic checking process related to resources.
- Any agent can join an organization if its resources match the required accuracy specified by the *Head* of that organization, provided it (the agent) is not currently busy.
- Each agent makes its own decision in terms of whether to join or not depending on its preferences; if these are met, then it will join the organization.
- Each agent has a setting parameter that specifies the number of organizations it can join; this may be different from agent to agent, within the simulation, to demonstrate the different abilities of agents in the network.

5.2.2 The Creation of Heterogeneous Organizations

The second mechanism we investigated for creating organizations of agents with heterogeneous types of resources was one in which agents were gathered in groups without any predefined threshold; the only condition they had to meet was that agents less busy to help the *Head* of the organisation with executing more tasks. Any agent can be a *Member* in an organization, provided it is not currently a busy agent and the maximum number of organizations that it can join has not yet been reached. There is no requirement that the required accuracy be met – as there is with the first mechanism. Via either mechanism, the process of creating an organization takes time. During this time, the network of agents is still receiving tasks from the customer even when the network consists of only two agents. The tasks can be executed by individual agents with the aid of the directed search algorithm.

5.2.3 Agent Roles within Organizations

This work focuses on adding roles for the agents within the self-organization process. These roles, from an abstract viewpoint, have been defined in chapter 3. Each agent is a service provider in the network. Creating organisations of agent leads to the emergence of new roles.

- The most busy agent is the *Head* of the organisation; this agent is responsible for distributing tasks to its *Member* as well as for accepting and executing tasks itself.
- The *Members* are agents in the network that have responded positively to the *Head's* message inviting them to join its organisation and they all operate as service providers within the organisations they have joined.
- Any *Member* of one organisation can be a *Head* or a *Member* of another organisation (overlapped organisations), so an agent can have a number of roles $R\_i = < R_1, R_2, R_3 >$, each agent may have just one of these roles or more

than one.

- *Members* who join the organization to accept and execute tasks have the role of service provider agents.
- Any *Member* within an organization that receives a task from the customer but has no ability to execute it, will send the task to other organizations that the agent has joined.
- An agent can have the *Member* role in many organisations. However, it can be a *Head* of only one organisation.

5.3 Task Execution and Delegation in Organizations

Task execution and delegation is the same in both mechanisms described above. However, the method follows different directions in the homogeneous organisations than it does in the heterogeneous organisations. This is because the *Heads* in homogeneous organisations request specific accuracies to accept tasks. The following steps occur in both mechanisms.

- **If the task received by the *Head* of an organisation.**
  (a) The task will be executed if and only if the *Head* is not busy and can accept the task, depending on there being a match in terms of the Manhattan distance required accuracy and the task deadline value.
  (b) The task will be queued if its required accuracy matches the *Head* preferences but the *Head* is executing another task when the message arrives.
  (c) Otherwise, the *Head* will check which of its *Members* can execute the task and delegate the task to one of these.
  (d) If the *Head* cannot find any of its *Members* who can execute the task, then the *Head* will delegate the task using its contact list.
- **If the task has been received by $\mathbf{a}_i$ where $\mathbf{a}_i$ is a *Member* in an organisation.**

(a) The task will be executed by a *Member* if it (the member) is not busy, and can accept the task; this depends on the Manhattan distance accuracy matching criteria requirement being met and on the task deadline value.

(b) If the *Member* cannot accept the task and cannot put it in its *ATQ*, the *Member* agent will send the task to its *Head* and the *Head* will accept it or delegate it as indicated above.

(c) A *Member* agent may be a *Member* of more than one organisation, so a *Member* agent will repeat step (b) to send the task to the next organisation that it joined, and so on until the task succeeds.

(d) If the task $TTL > 0$, and there is no agent (*Members* or *Heads*) in any of the a_i emerged organisations that can accept and execute the task, then the a_i will delegate the task, using the neighbours in its contact list.

5.4 System Performance with and without Organisations

We have designed and implemented a three different models in order to demonstrate the effects of creating organisations.

In the figures below, we have pointed out and compared between the three models which were developed, both with organisations and without organisations.

- The homogeneous organisation model as (Organisation_Ver0).
- The heterogeneous organisation model as (Organisation_Ver1).
- The delegation protocol (the directed search, as described in chapter 4) model, named (No Organisation).

Agent Network Size	Task distribution	Simulation Time
300	Mean=30,variance=8	3000 Cycles
500	Mean=30,variance=8	3000 Cycles
1000	Mean=30,variance=8	3000 Cycles

Table 5.1: *Network Size and Task Distribution*

5.4.1 Experimental Results

The evaluation of the suggested organisation models and of their performance has been carried out using a simulation environment. The experiments were conducted using a number of different agent network sizes (300, 500, 1000) – we have included the additional network size of 500 agents in order to demonstrate network/size performance as it exists in between the other two sizes. The maximum connection for each agent is up to $N = 10$ and the time to live for the task messages is $TTL = 10$. The simulation time has been increased to 3000 cycles from 2000 cycles in order to gain more accurate results relating to the self-organisation process. The results have been collected by repeating the run 10 times to ensure their robustness; these results were then used to construct the graphs. The experiments show the effects of changing the task distribution on the performance of the system with different network sizes and how task execution can be increased using a self-organisation process. Table 5.1 shows the values representing tasks issued from the customer side. These are generated using a normal distribution in order to simulate the real-world situation whereby variable numbers of tasks requests are issued. Figures 5.4 a to 5.11 have been produced using Equation 5.1 which represents the Average Number of Successfully Executed Tasks Ratio (ASETR). All these values, except for those in Figure 5.3 have been produced using Equation 5.2 – which computes the number of successfully executed tasks within the allocated number of simulation cycles (*ANSET*).

$$ANETR = \frac{STA(s)}{TTR(s)} \times 100 \tag{5.1}$$

Where:

$STA(s)$: The number of successfully executed tasks with their required accuracy

(s).

$TTR(s)$: The total number of tasks issued per required accuracy (s).

$$ANSET = \frac{1}{N_R} \sum_{i=1}^{N_R} \sum_{x=0}^{x=2} |RV_x - AR_x| \tag{5.2}$$

Where:

N_R: The number of runs = 10.

RV_x : The requested resources vector for the task.

AR_x : The matched agent resources that performs task successfully.

x = 0 to 2 : The index of the tuple that represent the requested resource vector.

The first collection of experiments was conducted in order to analyse the performance of the two mechanisms for creating organisations with different setting parameters and the effects of these variations on system performance and agent utilisation. Also, we have compared between the different performance of the emerged organisations and the directed search. Figure 5.3 shows the performance of the system in terms of the number of successfully executed tasks within the cycles; this statistic has been extracted by applying equation 5.2. Organization_Ver1 shows a higher efficiency in terms of executing tasks than the other two models. This is due to the variety of agents' resources in each organization; thus, when any agent becomes unavailable there are other agents that can accept and execute the tasks received. Figure 5.4 a shows the performance of Organization_Ver0 and Organisation_Ver1 as compared to the No Organization model. In general, the number of executed tasks when models with organizations are used is better than the numbers of executed tasks yielded by a No Organization model for all the required accuracies from (0-12). In the Organization_Ver0 model, each emerged organization has the same accuracy requirement as specified by its *Head*; here, the number of executed tasks showed slight improvements as compared to the same measure for the agents' network NO Organizations. On the other hand, in Organization_Ver1, each created organization is a cluster of heterogeneous agents which can satisfy tasks with a number of different required accuracies. The results from this version show more tasks being executed than were executed by Organisation_Ver0, for all ranges of required accuracies. By

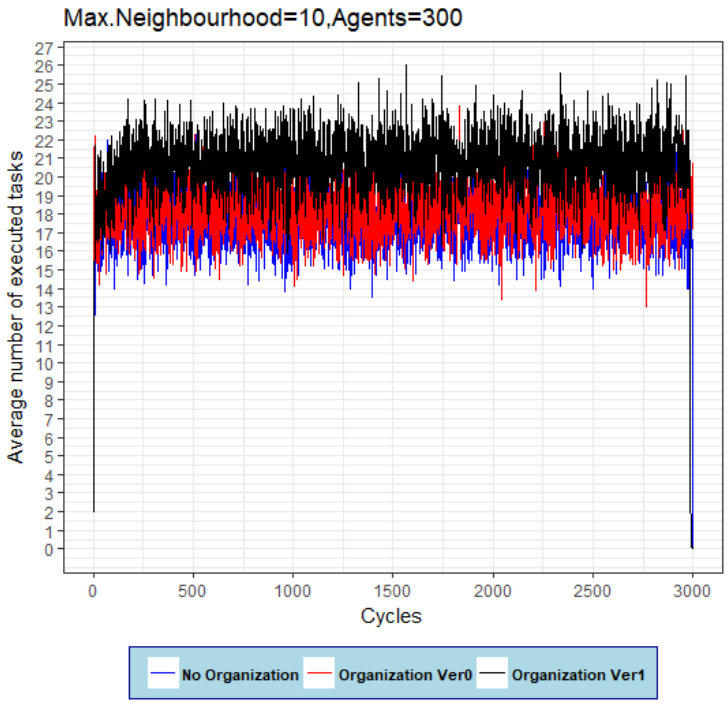
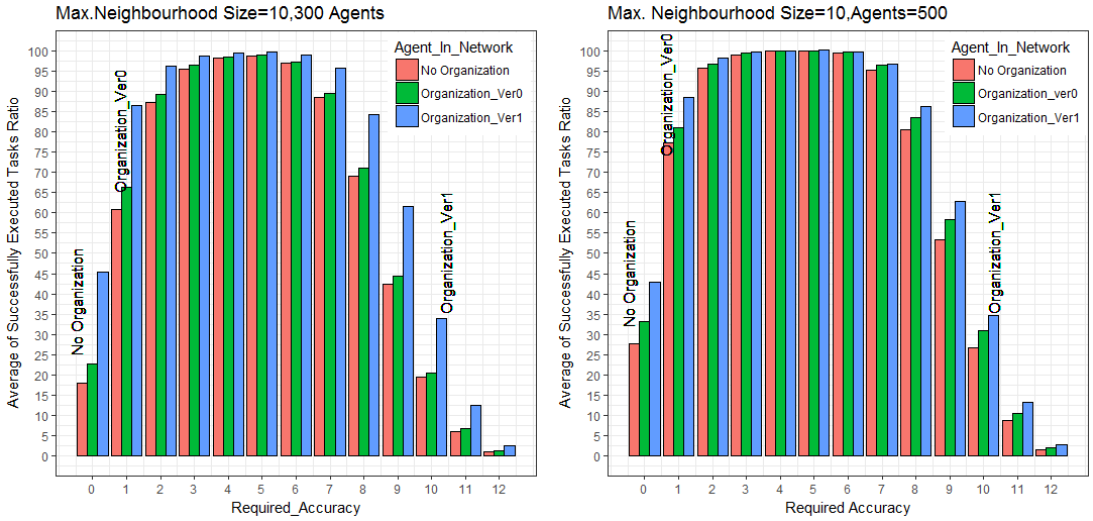


Figure 5.3: *The successful executed tasks in cycles for Network Size: 300 Agents*

increasing the number of agents to 500 agents, as shown in Figure 5.4 b, and by maintaining the number of tasks at the same level, we showed that Organization_Ver0 and Organisation_Ver1 yielded higher numbers of executed tasks than did the system with No Organizations; the results were particularly improved when using Organisation_Ver1.

Now, it is worth mentioning that when the size of the network was increased to 1000 agents, nearly all the tasks were executed with the required accuracies (from 3 to 7) as more resources are available for these accuracies; these tasks represent those with the highest requested accuracies as specified by the customer side. A minority of tasks which have certain required accuracies are executed less frequently than those with other required accuracies because some resources are more rare than others. This leads to fewer tasks which need these rare resources being executed. The reason behind this is that the number of hops required to reach the necessary resources is excessive. In relation to this, nearly similar performance was achieved by both with-organization models and by the No Organization model. The aim of the emergent organizations is to address the requests from the customer(s) where these represent a heavy workload in a system



(a) (b)

Figure 5.4: *ANETR for Network Size: 300 and 500 Agents*

which does not have a very high number of agents, increasing the number of agents reduced the impact and effectiveness of the emerged organizations in relation to the same demand from the customer side.

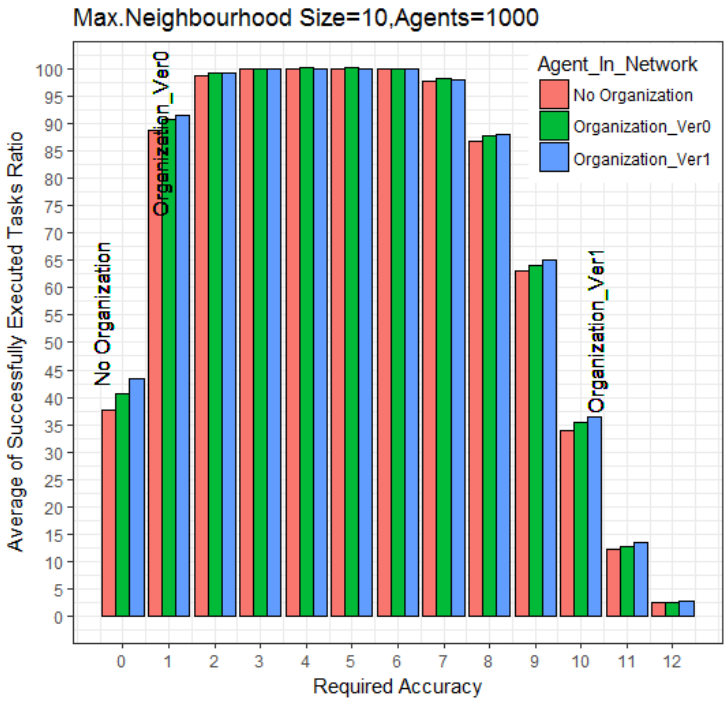


Figure 5.5: *ANETR for Network Size: 1000 Agents*

5.5 Dealing with Disruption in Agent Service Provision

Within distributed and dynamic systems, fault tolerance techniques must be made available because failure rates will affect the level of service provided by the system. In such systems, there is uncertainty, and this means that nodes may fluctuate between an operational and a failed state randomly. An example of this situation is represented by a dynamic group of nodes which may join and leave the system at random times in an ad-hoc network [49].

In this section of this work, we examine how our system can deal with the disruption that may occur as a result of agents losing their connectivity: e.g., by being offline. Agents prone to severe or mild failure will degrade overall system performance. To simulate this, agents were equipped with a parameter that enables them to be switched off for a period of time, in essence creating the impression that they are offline and unable to execute tasks or respond to messages – as has been presented in chapter 4. The on/off parameter was applied using

probability values of p= 0.9, p= 0.5 and p=0.2. It is certain that an environment which has a failure problem needs pre-determined mechanisms which can deal with this problem. In this section, we describe the scenario whereby a self-organisation technique is relied on as a means to recover customer tasks.Two approaches to this were taken, represented by Organisation_Ver0 and .

Only the delegation protocol implemented via directed search will be used here for the purposes of these investigations. The directed search algorithm deals with failure as follows: when an agent, called the initiator, delegates a task to an agent in its contact list, if no response from that agent has been received after a pre-specified number of cycles (e.g., 3 cycles), then the initiator will delegate the task to another agent on its contact list and so on until the task is executed or *TTL* =0. If this process proves to be unable to deal with the failure and the situations leads to the *TTL* value being consumed, then the task can only be delegated to two or three agents and this means that the probability of a failed task is increased. Therefore, these two mechanisms have both been presented here.

5.5.1 Experimental Work

Our experiments examine the advantages and disadvantages of the emergent organisation mechanisms by comparing them with a selected delegation protocol as represented by the directed search – which was presented in chapter 4. Hence, the failure function described there – the Agent Failing Algorithm outlined in section 4.9 at the page 87 – has been added to the mechanisms as follows:

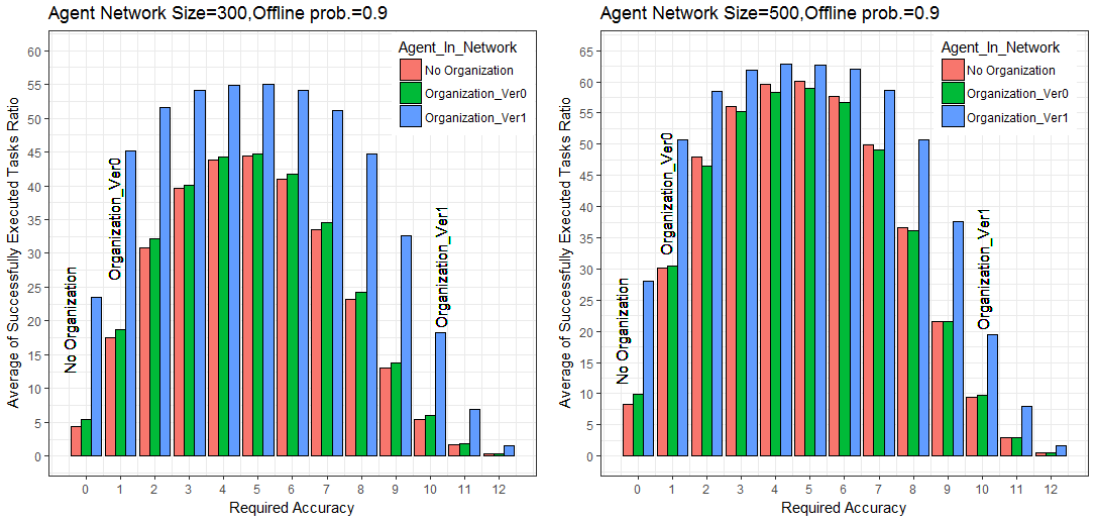
- The homogeneous organisations as (Organisation_Ver0) model with failure problem.
- The heterogeneous organisation as (Organisation_Ver1) model with failure problem.
- The directed search as (No Organisation) model with failure problem.

The simulation experiments present the three models: Organisation_Ver0, Organ-

isation_Ver1 and No Organisation with three different probability values for agent failure. The selected probability values (P) are: 0.9, 0.5 and 0.2. The same setting parameters shown in Table 5.1 are applicable here. The study looks at three different network sizes: 300, 500 and 1000 agents. The output of the experiments has been used to compute the *ANETR* and *ANSET* for all three models

Figure 5.6 a shows the three models – Organisation_Ver0, Organisation_Ver1, and No Organisation – implemented on a network size of 300 agents and with a failure probability of p= 0.9. The simulation results show that the *ANETR*, using Organisation_Ver1, is the highest of the three across all the required accuracies with different degrees, as compared to the other two models. The use of Organisation_Ver1 improved the system performance and coped with the presence of offline events; this was due it having a variety of heterogeneous agent organizations which, together, are able to satisfy a higher percentage of the requested tasks as compared to Organisation_Ver0. Also task delegation requires fewer hops, using this organisation, as the *Head* has knowledge of its *Members* agents. In contrast, in the No Organisation model, the process of delegation and of checking agents consumes the *TTL* value, and as a result more tasks will fail. Both the No Organisation model and the Organisation_Ver0 model yield a lesser convergence evaluation in term of performance as compared to the Organisation_Ver1 model. The reason behind the relatively good performance achieved via Organisation_Ver0 is that the organization structures in this model is based on specific required accuracies (homogeneous organisations) whereby each *Head* of these organisations has a specific required accuracy, and this may not match with the required accuracies of many of the receiving tasks. This leads to the delegation of the tasks to different organisations or, in other words, simply to the agents in the receiving agents' contact lists (if the *TTL* value has not yet been reached). This behaviour is similar to that of the No Organisation model where task delegation occurs only when agents are unavailable or have unpredictably dropped out. In the latter model, there is no other, more efficient, strategy that can be followed in order to deal with agent failure.

In Figure 5.6 b, increasing the number of agents to 500 and the failure prob-



(a) (b)

Figure 5.6: *ANETR for Network Size: 300 and 500 Agents With Failure Problem p: 0.9*

ability to p= 0.9, resulted in more tasks being executed, especially via Organization_Ver1. The latter can be explained by the fact that the probability of finding resources for the requested task that match the issued required accuracies is higher than in the other two models. However, with some of the higher required accuracies, it is still difficult to find the appropriate resources. In relation to the Organisation_Ver0 model, with the increase in the number of agents more organisations were created but at a rate that was still unable to cope with the accuracy requirements of the incoming tasks since the organisations were based on the *Heads*' specific required accuracies. This meant that the tasks lost more of their *TTL* searching for an organisation that could match their required accuracies. In addition to this, due to the fact that each organisation had to comply with its *Head's* required accuracy, fewer agents were able to join more than one organisation. Therefore, for some accuracies, the system performance dropped down to even less than that achieved by the No Organisation model.

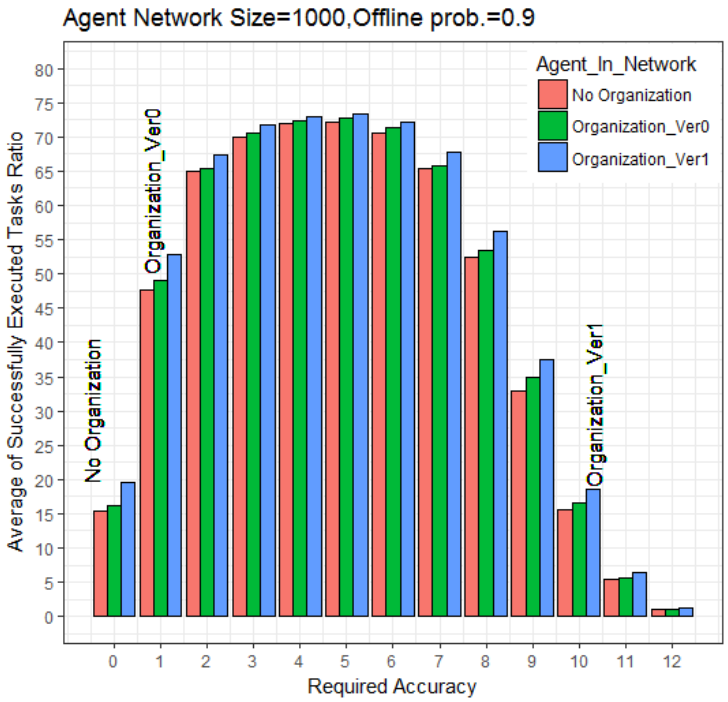
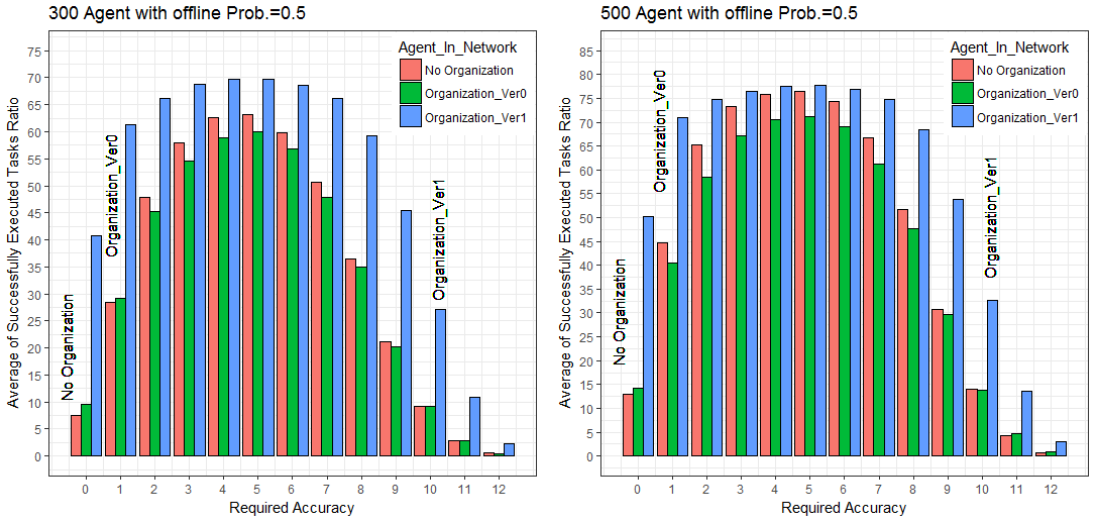


Figure 5.7: *ANETR for Network Size: 1000 Agents With Failure Problem p: 0.9*

When the network size was increased to 1000 agents, all three models showed better ratios than with the lower network sizes of (300 and 500) agents, as shown in Figure 5.7. This can be understood as a larger network requiring more requested tasks before it can benefit from the facility provided by the created organizations in the event of agents being offline.



(a) (b)

Figure 5.8: *ANETR for Network Size: 300 and 500 Agents with Failure Problem p: 0.5*

Figures 5.8 a, 5.8 b and 5.9 illustrate the runs of the system where the offline probability has been set to 0.5 with different agent network sizes (300,500,1000). On average, the *ANETR* increased for all the required accuracies for all the three models, due to the reduction of the probability of failure. The highest *ANETR* is still achieved by Organisation_Ver1 with all of the network sizes presented. With Organisations_ver0, the consumption of the *TTL* within the emerged organisations leads to worse performance as compared to that achieved by the No Organisation model. These Figures show higher *ANETRs* compared to Figures 5.6 a, 5.6 b and 5.7.

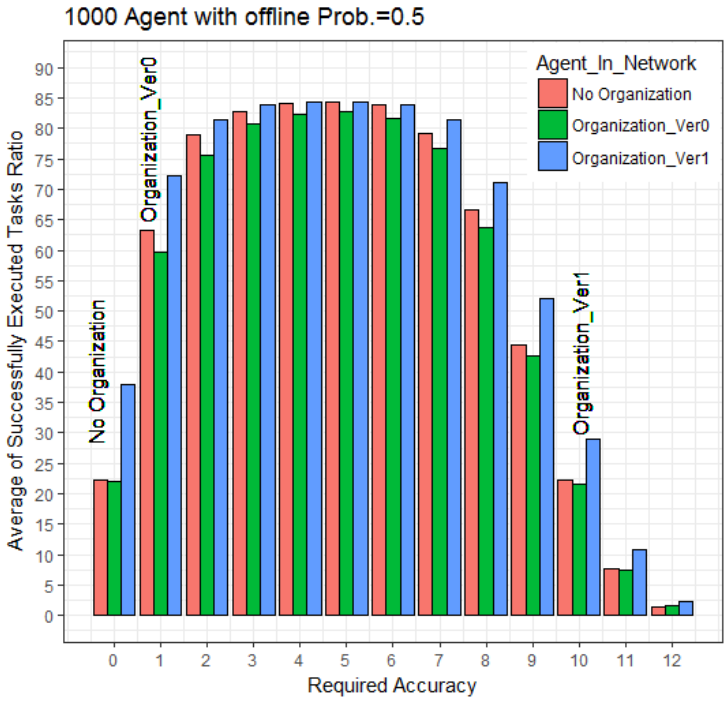
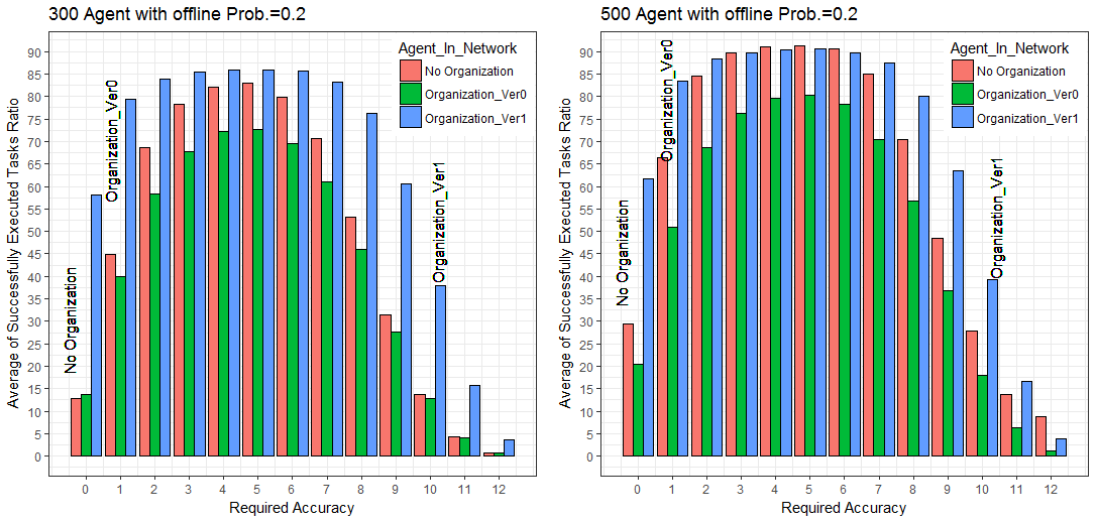


Figure 5.9: *ANETR for Network Size: 1000 Agents with Failure Problem p: 0.5*



(a) (b)

Figure 5.10: *ANETR for Network Size: 300 and 500 Agents with Failure Problem p: 0.2*

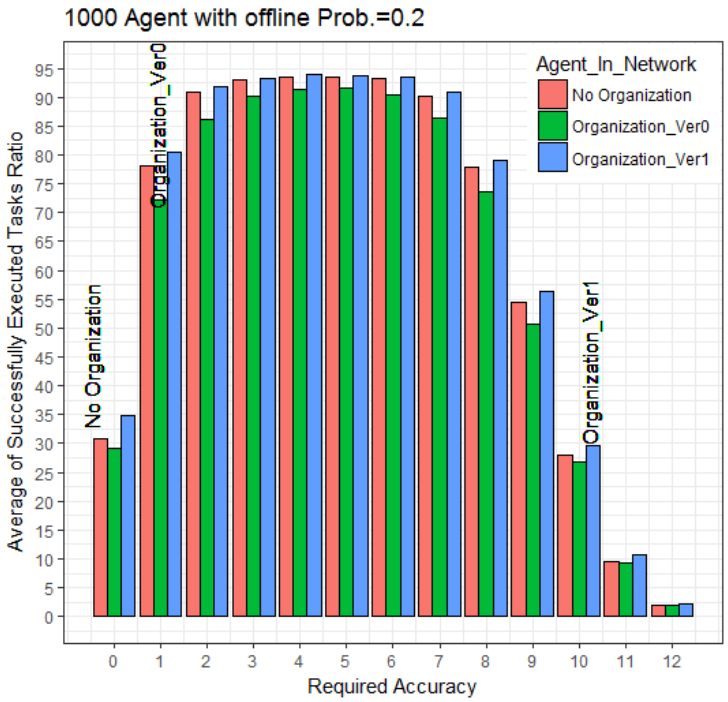


Figure 5.11: *ANETR for Network Size: 1000 Agents with Failure Problem p: 0.2*

When the probability of an agent going offline is 0.2, the effects vary according to the mechanism used. Using Organisation_Ver1, generally, the performance of the system increased and it demonstrated that it can effectively cope with the case of agents being offline; this was due to it possessing agent organizations which can satisfy high percentages of the requested tasks. In contrast, Organisation_Ver0, due to its construction criteria, shows negative effects in the same situation. It cannot overcome the problems caused by having specific accuracies in its created organisations; this leads to more failed tasks than even with the No Organization model, see Figures 5.10 a, 5.10 b and 5.11.

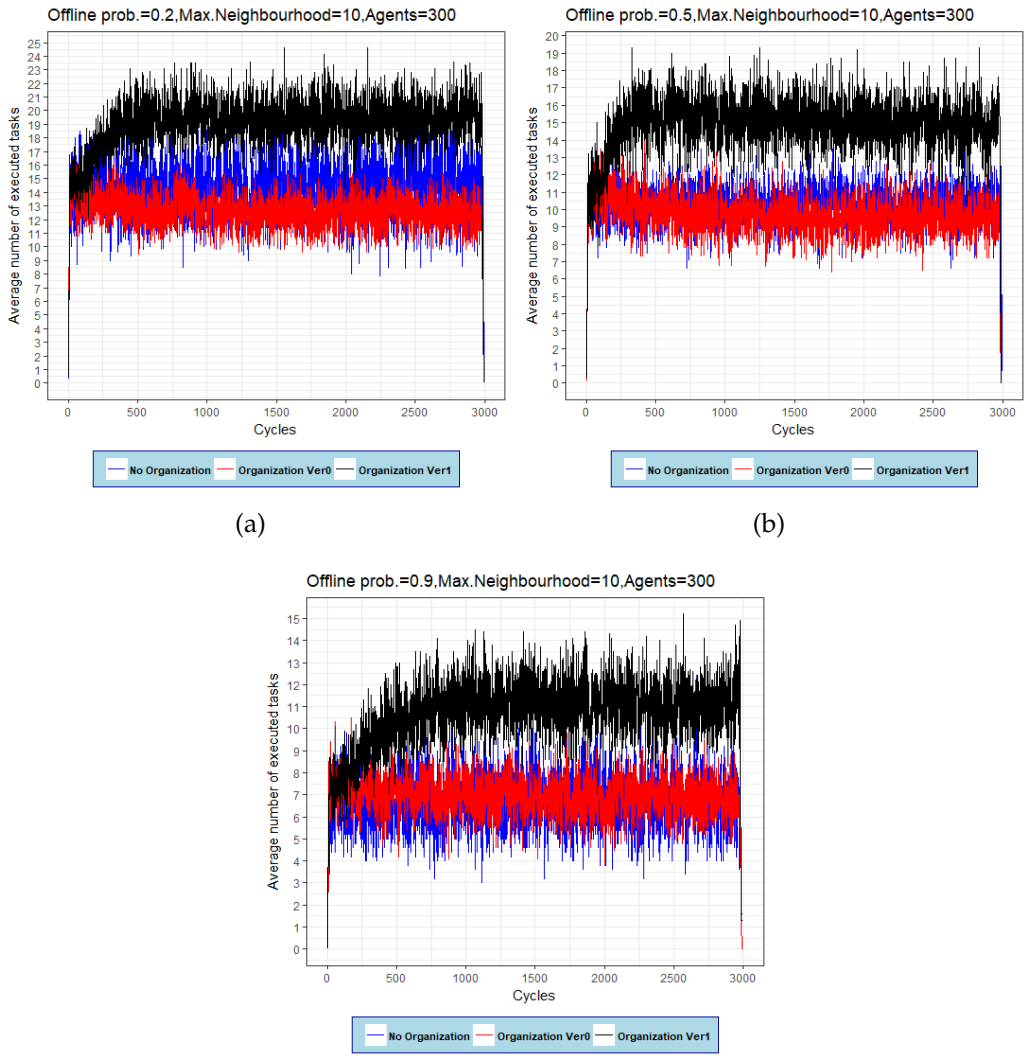


Figure 5.12: *ANSET for network of 300 Agents with Probabilities (0.2,0.5,0.9)*

We also computed the Average Number of Successfully Executed Tasks (*ANSET*), representing the system throughput within the cycles. This we did for network sizes (300, 500, 1000) and with probabilities of failure of (0.2,0.5,0.9). Organisation_Ver1 executes more tasks as shown in Figures 5.12 a, 5.12 b, and 5.12 c. Organisation_Ver1 executes more tasks than Organisation_Ver0 and the No Organisation model even when the probability values were increased: Organisation_Ver1 achieves higher throughput than the two other models. This means that self-organising heterogeneous agents in organisations lead to increases in the agents' opportunities to accept more tasks and this situation enhances the system's performance.

These experiments show that Organisation_Ver1 is a more stable and effective model than the Organisation_Ver0 model. Furthermore, the use of Organisation_Ver1 results in higher throughput across all the applied probability values especially with a network size of 300 agents. This indicates that the Organisation_Ver1 model can result in a system with better performance when small number of agents exists in the network. Increasing the network size showed that Organisation_Ver1 achieved more stable throughput than Organisation_Ver0. However, increasing the network size does not show the effectiveness of having organisations, because the same number of tasks arrive are issued from the customers i.e. we need to increase the number of customer tasks in each cycle to demonstrate the better performance and usage yielded by the organisations created in the system.

A further experiment was conducted; this time, to show the differing numbers of emerged organizations which result from having differing network sizes (300,500,1000 agents) but applying the same task distribution and setting parameters as per Table 5.1. The following explanation focuses purely on a network size of 300 agents; this has been chosen as a representative case, as shown in Figure 5.13. Due to the poor performance results obtained from Organisation_Ver0, the explanation here is only concerned with Organisation_Ver1.

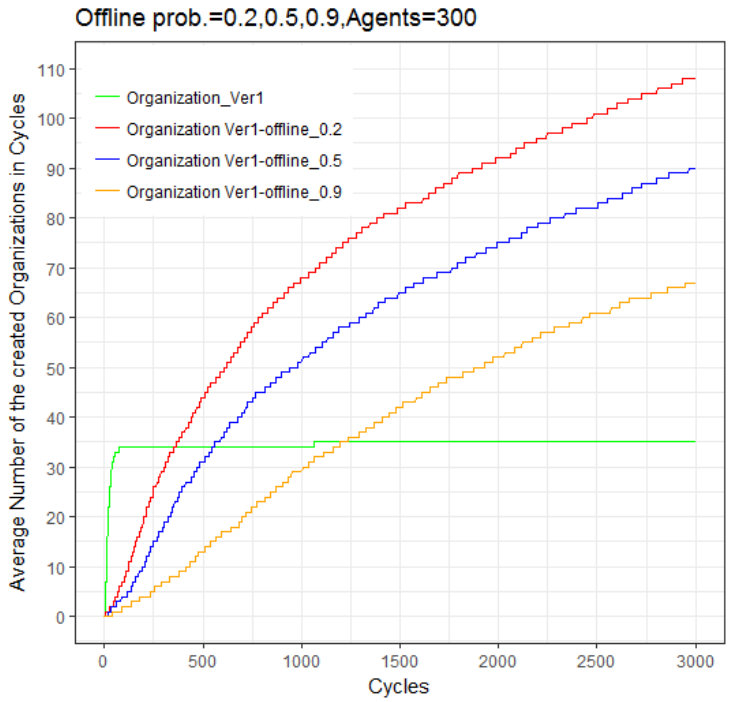


Figure 5.13: *Number of created organizations for 300 agent with different probabilities*

Overall, with Organisation_Ver1 and without agents going offline (normal case, green colour), it is noticeable that the average number of organizations rose dramatically in the first few cycles, then it levelled off at a point where there was no demand to create more organizations for the remaining simulation time because there were no critical issues (failures) and most of the agents were either busy and their $ATQ < W$ so they could accept more tasks or were not busy. With the presence of the agent offline condition as governed by a number of different probability values (0.9,0.5,0.2), the average number of created organizations increased steadily with time and nearly followed the same trends in all the simulation time cycles. Thus, the average number of emerged organizations decreased as the probability of there being offline agents increased. Hence, with p=0.9 the system created

fewer organisations (around 66), with p=0.5 it created around 90 organisations and with a failure probability of only p=0.2, the number of created organisations was about 108. Increasing the probability of failure makes the system unstable and some agents become very busy. However this does not lead to the creation of more organisations.

Increasing the number of tasks issued by the customer may lead to increases in the number of emerged organisations. However, this number is bounded by the size of the network, the triggering condition that declares an agent as the most busy and as needing help from other agents, the availability of agents, and most importantly, the fact that an agent can only be a *Head* for one organisation.

5.6 Chapter Summary

This chapter has presented the idea of using self-organisation techniques in order to enhance task execution throughput and system performance. Two self-organisation mechanisms have been presented: the homogeneous organisation mechanism and the heterogeneous organisation mechanism. The organisations emerged to manage the problem of agents being busy for long periods of time – the condition that their *ATQ* become forms a trigger condition which initiates the organisation creation process. Such agents will be considered the *Heads* of the organizations, so they start to send multicast messages to other agents asking the less busy agents to join in their organizations and provide services. In the literature [5], it is reported that computational nodes may be underutilised and not meet their full potential: i.e., they may be utilised or busy less than 5% of the time.

We aimed to increase the agents' resource utilisation by allowing an agent to join more than one organisation (overlapped organisations). This lets us execute more tasks and makes the system more resistant to failure. A comparative study using a set of experiments has been conducted which shows the relationship between these two mechanisms and the delegation protocol represented by the

directed search which was presented in chapter 4.

Heterogeneous organisations demonstrated high performance in comparison with both homogeneous organisations and directed search. This is because the emerged organisations consist of heterogeneous types of agents and a new method for task execution and delegation was used with these organisations, as presented in section 5.3 at the page 106. These factors enabled the tasks to be executed and delegated inside the organisation instead of using agents' contact lists. Also, in the Organisation_Ver1 model, agents have heterogeneous resources, and this allows the organisations to accept and execute various different types of tasks. This is in contrast with Organisation_Ver0 where the agents are grouped in organisations only if they have satisfied the *Head's* accuracy requirements; this leads to the creation of homogeneous organizations. This mechanism represents part of the contribution of this work relating to task execution using organizations of heterogeneous agents in distributed domains.

Another contribution is that the agent can enact various different roles in the system. The fulfilling of this contribution requirement led to the creation of *Heads* of organisations and *Members* both of which are service providers in the emerged organisations. Agents can adopt more than one role due to the creation of overlapped organisations. An agent can be a *Member* of one organisation but the *Head* of another organisation. Hence, the system supports two roles: one is the *Head* and the other is the *Members*, and both are service providers. This makes our mechanisms unlike the proposals in other related works, [99] where agents are able to join an organization only at specific times in their life time and have to change their behaviour in order to join an organization and to match the sole role that they have been requested to perform.

Further, we applied agent failure to networks using both of these mechanisms as well as to networks using the delegation protocol described in chapter 4. As we have mentioned before, in chapter 2, fault tolerance techniques for modern distributed systems have been explored by many researchers. However, most of the methods available for such dynamic and open systems are reactive techniques. These types of technique provide solutions for node failure only after its occur-

rence [100], one of the few exceptions is [101]. Here, we have implemented models that use a self-organisation technique to manage and prepare the system for failure, before it occurs. This approach means that the customer tasks have a better chance of execution even in the case of failure, due to the heterogeneous types of agents which exist in the organisations. So, even if failed agents are present, other agents may be able to accept and execute the tasks they would otherwise have executed. As illustrated by the experiments, the model using heterogeneous organisations outperforms the other two methods. In the No Organisation model, the network performance was weak, as shown in the experimental results, where the *TTL* value was frequently consumed. It is predictable that this issue can lead to the loss of tasks and can affect the network performance. The implemented work shows that the proposed protocols have delivered models that can cope with the failure problem in distributed domains. This indicates that self-organisation is a promising mechanism for improving task execution throughput in heterogeneous systems that face random agent failure.

Chapter 6

Henchman Task Recovery Protocol

6.1 Introduction

One of the most challenging problems in distributed systems is dealing with agent failure; examples of complex systems where this is an issue are the Internet, the World Wide Web and peer-to-peer networks [102]. Distributed systems are heterogeneous, scalable and have to maintain the availability of their services even in the presence of disruption. This means that supplying distributed systems with fault tolerance is a very significant goal [6]. For instance, agents can fail and this may mean that any action (s) or task (s) that such agents have taken on will fail to be executed.

In chapter 5, we explored how agent organisations can limit the effects of disruption, caused by agent failure, via a better mechanism than that which we presented in chapter 4. An agent can decide to create organisations in order to improve the number of executed tasks, to increase their own utility and to make better use of their resources. Hence, we show that agent organisations can be resilient to unexpected disruption which would otherwise significantly affects a system's performance. However, the occurrence of an organisation *Head* failure still leads to have failed tasks. *Head* failure can also lead to consume the *TTL* values as *aMember* in an organisation may delegate tasks to other *Heads* of

organisations it has joined. Appropriate mechanisms are needed to handle *Head* failure within organisations so that their functionality and effectiveness can be maintained within these parameters.

We present a protocol for recovering from agent failure which occurs within the emerged organisations, called the "Henchman Recovery Protocol (*HRP*)". This protocol has been added to the chosen self-organisation mechanism (which uses heterogeneous organisations). This mechanism has been chosen because it demonstrated higher task execution throughput than the other mechanism (homogeneous organisation). We have studied and analysed both mechanisms via the experimental work described in chapter 5, and determined that this mechanism is the better one.

The *HRP* enables the agents within an organisation of heterogeneous agents to maintain task execution throughput and recover tasks in the event of agent failure. We show how the protocol helps to maintain the efficiency of the emerged organisations. This enhancement is a remedy for the situation where a *Head* agent fails and its effect is to recover the number of tasks executed by the emerged organisations.

6.2 Henchman Recovery Protocol (*HRP*)

This section describes the situation wherein we add the new protocol to the heterogeneous organisation mechanism. We have seen that the numbers of executed tasks were increased to a greater extent by this as compared to homogeneous organisations mechanism and we have demonstrated this effect with different network sizes.

In the literature, especially in the network environment, there are some methods that can deal with fault tolerance similar to the *HRP* like the Master (MAS) /standby server (SBS) model [103]. Either term will generally get the point across, but theres some differences between them. *HRP* has structured organisations with specific roles to the participating agents in the self-emerged organisation. This

enables the HRP to react more effectively in term of fault tolerance.

In any of these organisations, the *Head* is the agent which is responsible for distributing the tasks to its *Members* in the organisation, so that if the *Head* fails, the tasks will also fail, and this will affect the performance of the system as a whole and lead to the loss of more tasks. To avoid this loss of tasks and enhance the performance of the system, an enhancement has been applied to the heterogeneous organisation mechanism; a new role is introduced to the structure of the emerged organisations. We suggest that this will assist the organisation to provide an efficient service to the customers. This will be because agents having the new role will detect the failure of their *Head* and then take the necessary action to avoid losing tasks. The new role is a *Head* follower, called the *Henchman*; this role is to be created in each organisation structure. As we have explained before, the *Head* is the one which is responsible for distributing the tasks to the *Members* of its organisation; when a *Head* fails, the role of the *Henchman* is to act as a substitute to the failed *Head*, providing a further self-organised capability to the system, maintaining the functionality of the emerged organisation. Each *Member* has the ability to decide to accept a new role or to reject it. The following describes the mechanism of the *Henchman* role.

The first stage of the *HRP* is performed during the runtime of the gossip algorithm – by calling "BeMyHenchman". Here, the algorithm which is used for creating heterogeneous organizations has been applied alongside the agent self-switching (on/off) capability. The gossip protocol has been modified. In the previous chapter, we used a gossip algorithm whereby the *Head* sends a multicast message to randomly selected neighbours, asking the agents to join its organisation and be its service providers. But here, in Algorithm 5, we have introduced an improved usage of the gossip algorithm whereby the *Head* sends the multicast message to all its directly connected neighbours to create the organisation. In both situations, the decision as to whether to join or not, depends on the status of the agents in terms of them being busy or not. The less busy agents are the most likely to accept the invitation to join the organisation – in order to improve their resource utilisation. If the *Head's* message encounters a busy agent, that agent

will merely pass the message on to other agents in the network. We believe that this way of creating organisations will lead to a better utilisation of the system resources.

Algorithm 5 Gossip Protocol Updated

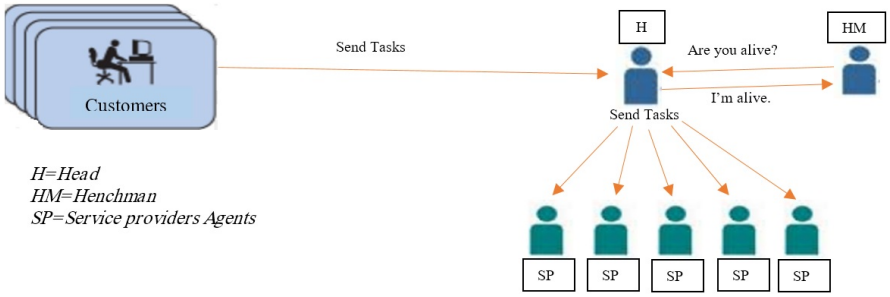
Input: a_i: is the most busy agent (*Head*) that will create an organisation, a_x is one of the a_i contacted list neighbour, $TTL > 0$.
Output: a_i completed organisation.

1: $Cycle\_e$
2: a_i.SendMessage ($Contactlist$(1,N))
3: **if** (a_x ! = busy) **then**
4: a_x infected with the gossip message of a_i
5: // a_x has the option to join or not to the created organisation.
6: call "BeMyHenchman" to select an agent to be *Henchman*.
7: **else if** (a_x == busy) **then**
8: a_x.SendMessage (1,(N))
9: **end if**
10: **if** ($TTL > 0$) **then**
11: $TTL = TTL - 1$
12: **end if**
13: $Cycle\_e + 1$
14: End

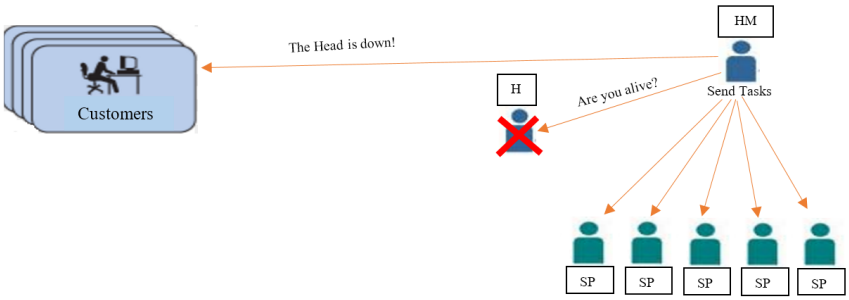
6.2.1 Henchman Appointment in Organisation

In this section, we describe an appointment of the *Henchman* agent in an organisation. The process is started by the *Head* agent sending out a message to the first agent a_i that joined the *Head's* organisation, asking the agent whether it will agree to be its (the *Head's*) *Henchman* (*HM*). The *Head* will send the same message to all the other agents that joined in sequence until one of the *Members* accept the *Head's* message and becomes the organisation's *Henchman*. All the *Members* contacted have the ability to take the decision either to accept or to reject the *Head's* message.

After assigning a *Henchman* for its organisation, the *Head* has the responsibility to synchronize its database *DB* with that of the *Henchman*; this contains the contact details of the *Members*, and will, of every newly joined agent. This means that the *Henchman's DB* will always be maintained to be identical to the *Head's DB*. When the *Head* goes offline, the *Henchman* will immediately replace the *Head* to maintain the functionality of the organisation.



(a) The interaction process between the customers with Head



(b) The interaction process between the Henchman and the customers

Figure 6.1: *Abstract system interaction using HRP*

If the *Henchman* does not receive any acknowledgement back from the *Head*, the *Henchman* will declare to all the other *Members* and the customer agent that the *Head* has failed and the new *Head* is the *Henchman*; this is in order to redirect the traffic to itself instead. When the *Head* recovers, i.e. the *Henchman* receives an "I'm alive" messages again from the *Head*, the *Henchman* will instantly inform the organisation *Members* as well as the customer that the *Head* is now alive and the organisation should be back to its normal condition. This will be explained in more detail in the Algorithm 6 called "Monitoring Head Availability". However, there is a small chance that a disruption will effect both the *Head* and its *Henchman* at the same time. Here, if the customer agent sends tasks while this is the case, these tasks will fail – until the *Head* and/or the *Henchman* return to an active state after being offline for a random number of cycles. Figures 6.1 a and 6.1 b are included to demonstrate how the system works, using the *HRP*.

6.2.2 Monitoring Head Availability

After the assignment of the *Henchman* role to the desired *Member* agent, in Algorithm 6, this *Henchman* will check the availability of the *Head* at random numbers of cycles. At this point, the task messages received by the *Head* are automatically seen by the *Henchman* in order to keep the *Henchman* continuously updated about the customer's *CIDs*.

Algorithm 6 Monitoring Head Availability Algorithm

```
1:  while (true) do
2:    Henchman.SendMessage (Head, "Are you alive.")
3:    Head.SendMessage (Henchman,"I'm alive")
4:    if (Henchman.SendMessage (Head, "Are you alive")= fail) then
5:      // Redirect the customer and the Members to the Henchman.
6:      Henchman.SendMessage (Customer, "Head has failed")
7:      Henchman.SendMessage (Members, "Head has failed")
8:    end if
9:    if (Henchman.SendMessage (Head, "Are you Alive")= success) then
10:     // Redirect the customer and the Members to the Head.
11:     Henchman.SendMessage (Customer, "Head is alive")
12:     Henchman.SendMessage (Members, "Head is alive")
13:   end if
14:   // keeps on checking the Head, select a random cycles
15:   cycle_e=cycle_e+Random(1, No_cycles)
16: end while
```

6.3 Revised Organisation Structure and Roles

This section presents the structure description for the second layer (for heterogeneous organisations) above the existing agent network, after adding the *Henchman* role into the emerged organisations.
Now every created organisation in the system has the following format:

$Org = \{Org_1,..., Org_q\}$, where Org_p with $1 \leq p \geq q$ is the pth organisation in the system; hence the elements in each organisation are:

$Org_p = < OrgID_p, OrgH_p, \{a_1,...a_{no}\}, HM_p, Z_p >$, where $1 \leq no \leq Z_p$.

- $OrgID_p$: a unique identifier for each organisation created.
- $OrgH_p$: is the name of the *Head* of an organisation.
- $\{a_1,...a_{no}\}$: the participants in the organisation at a specific time.
- HM_p: is the *Henchman's* (follower's) name for each organisation created.
- Z_p: is the maximum size of each organisation.

The following are the updated principles of the *Member's* role in the emerged organisations, after adding the *HRP*.

- R_i is a set of roles, $< R_1, R_2, R_3, R_4 >$
- The roles an agent may take on within an organisation are {*Member*, *Henchman*, *Head*}, each agent may have one of these roles or more than one, depending on a specific set of conditions.
- If Org_p and Org_q are two emerged organisations, then a_i can belong to both of them. In turn, any two or more organisations in the network can share a number of *Members*, so an agent can have a number of different or the same Roles: if the overlapped organisations are (Org_p, Org_q):
  (a) a_i can be *Member* in Org_p, *Member* in Org_q.
  (b) a_i can be *Head* in Org_p, *Member* in Org_q.
  (c) a_i can be *Head* in Org_p, Not *Henchman* in Org_q.
  (d) a_i can be *Henchman* in Org_p, Not *Henchman* in Org_q.
- An agent cannot be a *Head* or a *Henchman* of more than one organisation.
- Agents may change their roles during runtime. When organizations start to emerge, a *Member* of one organisation can be a *Head* of another organization, a *Henchman* of another and a *Member* of yet another, or perhaps, for example, a *Henchman* of one organization and the *Head* of a different organization.

- The *Head* of each organisation is responsible for the activities of its agents. The presence of a *Head* in an organisation of agents is necessary to coordinate the work.
- A *Henchman* of an organisation can be a temporary *Head* when the *Head* of its organisation has failed.

6.4 Task Recovery in Organisations

At this point, the task messages received by the *Head* are automatically also seen by the *Henchman* – in order to keep the *Henchman* informed about the customers. During simulation activity, when the *Henchman* checks the *Head's* availability, in a number of cycles which is randomly determined, the *Henchman* may detect that the *Head* has gone offline, so the *Henchman* will then directly report this to the customer side as well as to the other agents in its organisation. So, both sides will send or receive the tasks to and from the *Henchman* instead of to and from the *Head* while the latter is absent. During this time, the *Henchman* will also be receiving tasks and trying to execute them: if it can do so within their required accuracies and deadlines. If this is not the case, it will send such received tasks to the *Members* of its organisation. After the *Head* returns to work, the *Henchman* will inform the customer side and its organization's agents that the *Head* has returned and that they should redirect tasks to the *Head* again.

Sometimes there is a case, although the incidence is very small, that an organization may lose both its Head and Henchman which leads the member agents to lose their connections with the both. In such a case the system will take the advantage from the created overlapped organisations. Hence, the *Member* will send its task to the second *Head*; it will continue to do this for all the organisations it has joined. If the task cannot be executed by any agent of any of the organisations that a *Member* already joined, then it will act as if there are no organisations and will delegate tasks to one of its neighbours which can comply with the tasks *TTL* and deadline constraints.

6.5 Experimental Set Up

In the following experiments, differing assigned values have been used to show the differing statuses of the *Head* and the *Henchman* while the *Head* is offline as a result of differing probability values. Table 6.1 shows the setting parameters which have been used to set up the experiments, where (*ND* = Normal Distribution, and (T and M) are given the values of the mean and the standard deviation; also *H* = *Head*, *HM* = *Henchman*). The number of experimental runs which had to be

Agents	Run Time	Tasks	Prob.offline(1)	Prob.offline(2)	Offline cycles
5000	5000	Normal distribution*T+M	H=0.9 ,HM=0.6	H=0.6, HM=0.2	(2-5)HM+ Menber= (10-11)
2500	5000	Normal distribution*T+M	H=0.9 ,HM=0.6	H=0.6, HM=0.2	(2-5)HM+ Member= (10-11)
500	5000	Normal distribution*T+M	H=0.9 ,HM=0.6	H=0.6, HM=0.2	(2-5)HM+ Member= (10-11)

Table 6.1: *Henchman recovery protocol setting parameters*

carried out in order to obtain a reliable result was determined to be 10, and each run took from 55 to 70 minutes, so in total it took more than 11 hours before we could process the data and extract the relevant figures using the R tool software. Results from the three models (*HRP*, Organisation_Ver1 and No Organisation) are shown in the figures below, the runs used the setting parameters are as shown in Table 6.1. The simulation was run for 5000 cycles to produce the result for the model using *HRP* and we were able to increase the network size=5000 and 2500 agents, and re-execute for 10 times with the setting parameter shown in Table 6.1; the same process was undertaken for the other two models and with the various network sizes already mentioned.

Equation 6.1 is to compute *ANSET*, the average number of successfully executed tasks completed during the simulation cycles (*ANSET*).

$$ANSET = \frac{1}{N_R} \sum_{i=1}^{N_R} \sum_{x=0}^{x=2} |RV_x - AR_x| \tag{6.1}$$

Where:

N_R: thenumber of runs = 10.

RV_x : the requested resources vector for the task.

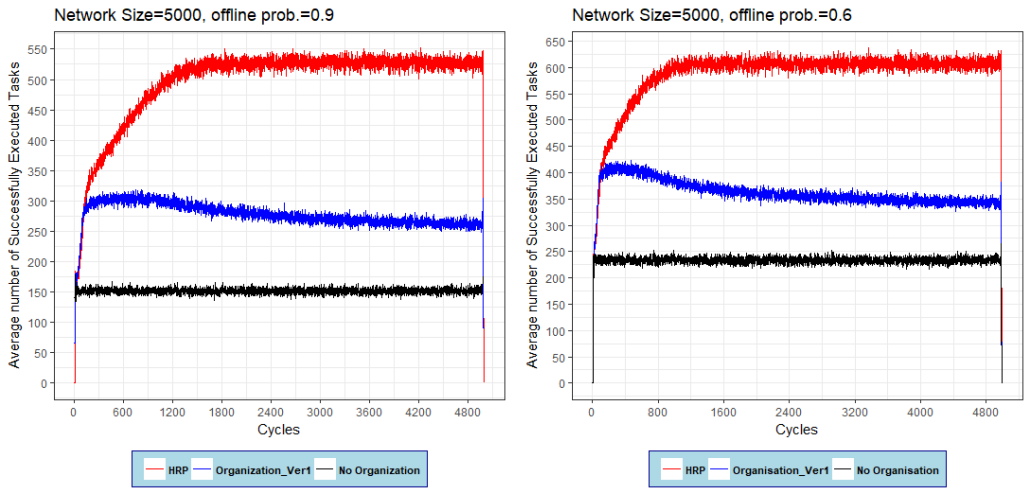
AR_x : the matched agent resources that performs task successfully.

$x = 0$ to 2 : the index of the tuple that represent the requested resource vector $<r_1, r_2, r_3>$.

The Figures below demonstrate the differences in the average numbers of successfully executed tasks across each of the three models (*HRP*, Organisation_Ver1 and No Organisation) and for different network sizes. With a network size of 5000 agents, our tests revealed that the setup with *HRP*, in comparison to the results for the other two models, had a higher percentage of executed tasks across all the different offline probability values (0.9, 0.6, 0.2); this result applies to all the types of agents, whether *Head*, *Henchman* or *Members*, as shown in Figures 6.2 a and 6.2 b. Also similar results were achieved with a network size of 2500 agents, as shown by Figures 6.3 a,6.3 b. The same positive performance effect, resulting from the use of *HRP*, was also encountered with a network size of 500 agents – in comparison with the other two models. Further, with a network size of 500, it was very difficult for the agents in both the (Organisation_Ver1 and No Organisation) models to cope with a non-zero failure probability as well as a very high number of tasks issued from the customers in each cycle, as shown in Figures 6.4 a and 6.4 b.

The most noteworthy output in the depicted Figures is from the *HRP* model. Clearly, having a *Henchman* within each organisation has advantages due to its integrated roles: it is a temporary *Head* and also a *Member*. In addition, the fact that there exists heterogeneous *Members* in the organisations plus the *HRP* means that the organisation can effectively deal well with the presence of failure. This demonstrates the idea that generating heterogeneous organisation structures and imposing roles for the agents in the organisations will improve system performance and throughput – because more tasks can be recovered.

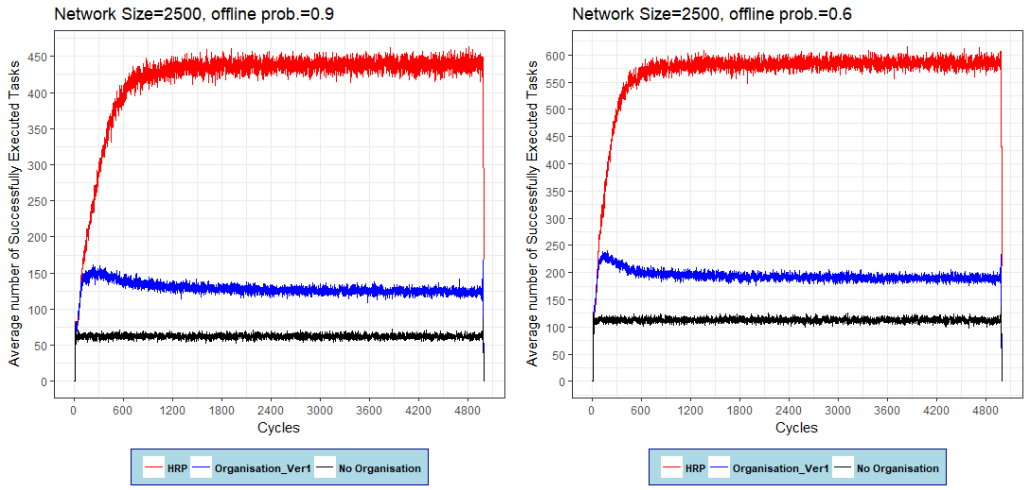
The roles that have been created together provide a method whereby tasks can be recovered. If the *Head* fails, then the *Henchman* can work as a temporary *Head*, also, the *Members* have the role of accepting and executing tasks as well as delegating tasks to the organisations that they are *Members* of. The *Head* can access its *Members* to delegate tasks across the organisation while consuming less of the *TTL* value, all these factors led to the good performance of the *HRP* model.



(a) (b)

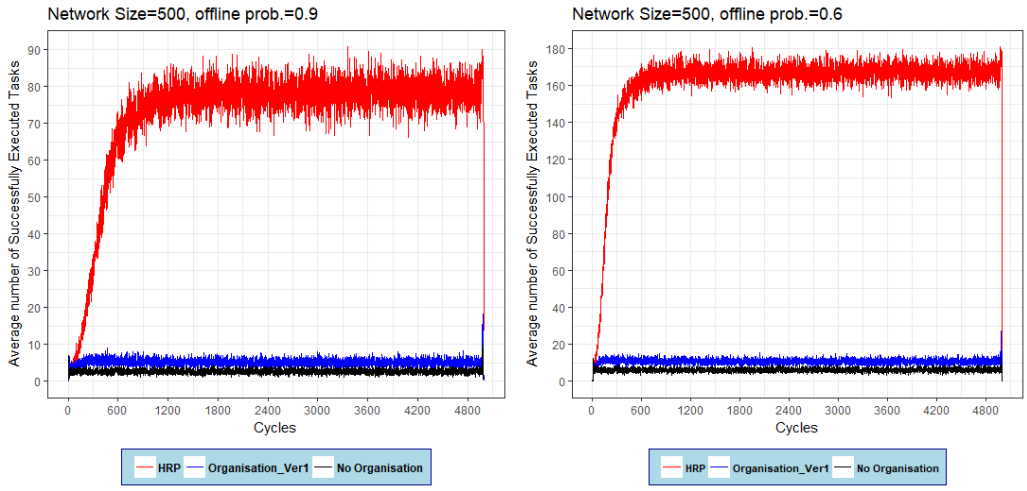
Figure 6.2: *Average Number of Successfully Executed Tasks for 5000 agents and offline probability* 0.9 *and* 0.6

Moreover, in the models (*HRP* and Organisation_Ver1), even where no *Members* are able to execute a task, this task would still have a chance of being executed by being delegated to one of the neighbour agents. Thus, the Organisation_Ver1 model has a performance which is in between that of the No Organisation model and the *HRP* model, and this implies that even with the existence of the organisations, the model is still prone to lose tasks due to the high number of tasks issued from the customer agent and the probability of agent failure. However, using the Organisation_Ver1 model, the system shows better performance than it does when the No Organisation model is used, for all the network sizes and offline probabilities which were presented to it. This is because the No Organisation model depends only on task delegation via directed search when the receiving agent cannot accept a given task. Hence, this is not an adequate solution, largely due to the presence of agent failure.



(a) (b)

Figure 6.3: *Average Number of Successfully Executed Tasks for 2500 agents and offline probability* 0.9 *and* 0.6

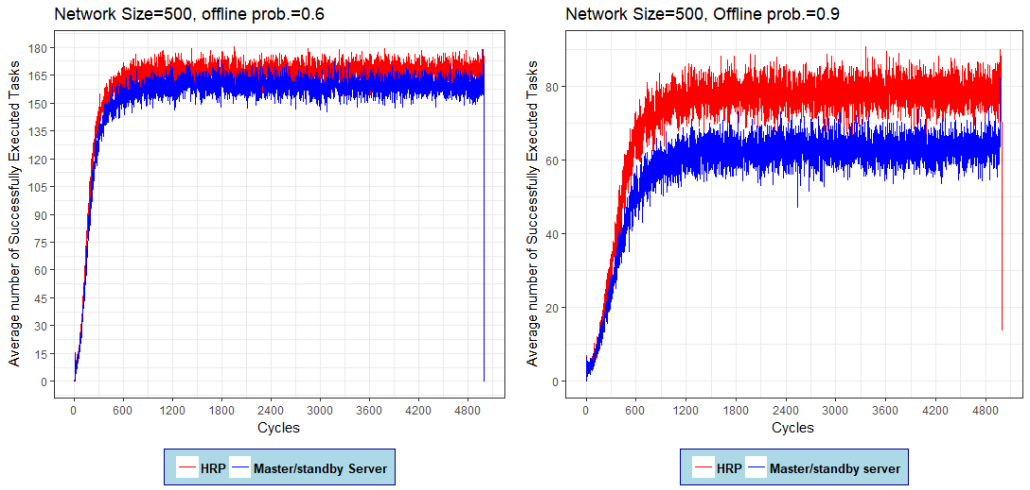


(a) (b)

Figure 6.4: *Average Number of Successfully Executed Tasks for 500 agents and offline probability* 0.9 *and* 0.6

6.6 Comparison and Evaluation of *HRP*

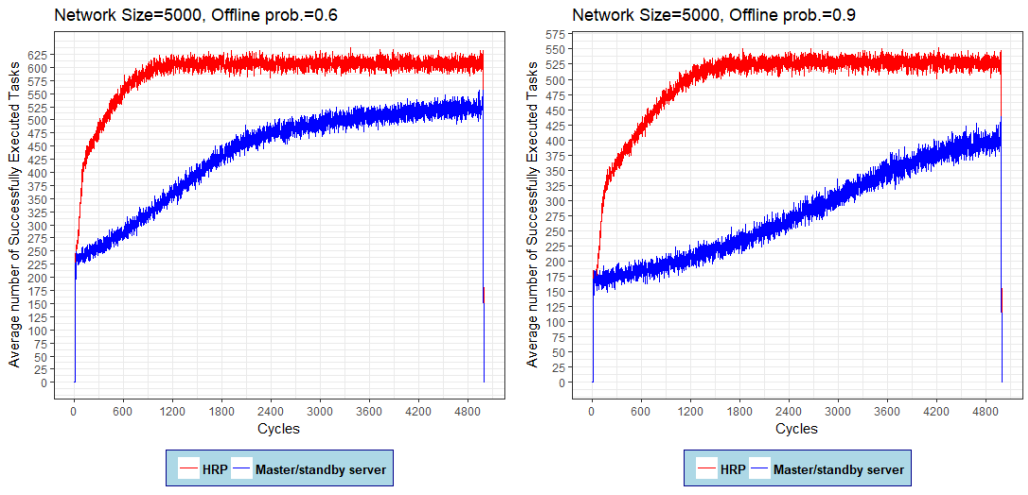
In this section of our study, we compare the efficiency of the protocol that we have suggested, *HRP*, and the heterogeneous emerged organisation structure with the Master/standby fault tolerance server system [103]. In the master/standby server system, both of the servers must be available from the beginning of the network creation process, and the connecting link must be initiated by the Master server which sends messages requiring acknowledgement to the standby server so that this can indicate its availability. Thus, to create that scenario, but for our organisations, in each organisation we have specified the *Head* to act as a MAS (server) called the Master Head (MAH) and also specified another agent in the organisation to act as a SBS called the Standby Head (SBH). The SBH will only respond to the customer messages when the MAH of the organisation is down. In our work, it is the *Henchman's* responsibility to check the *Head* availability through the heartbeat messages. In addition, the organisation structure (*Head*, *Henchman*, *Members*) with its roles leads to the emergence of self-organised groups that can overcome the failure issue, as explained in subsection 6.3 above, this is unlike the MAS/SBS method whereby the SBS can only switch to active mode when the MAS has failed. This experiment has been implemented with the setting



(a) (b)

Figure 6.5: *Comparison Between Henchman Recovery Protocol and The Master/standby Server,with network size 500 and probability* 0.6 *and* 0.9

parameters that have been presented in Table 6.1 for task distribution, and the various offline probability values (0.9, 0.6, 0.2) have been set for the agents (*Head*, *Henchman*, *Members*). Two network sizes have been used, 500 and 5000 agents, for the purpose of verifying our system as shown in Figures 6.5 a and 6.5 b. When the *HRP* system and the Master/standby server system had the probability of failure set at p = 0.6, the *HRP* demonstrated a slightly higher number of executed tasks in each cycle than did the MAS/SBS approach. Increasing the value of failure to p= 0.9, increased this difference in favour of *HRP*. This is because, in our scenario, the *Henchman* has a role in the self-organisation process whereby it acts as a *Member* which can receive and execute tasks as well as a follower of the *Head* of the organisation. In contrast, in the MAS/SBS approach, the SBS is only a backup server and is only turned to active mode when the MAS has failed. Increasing the network size to 5000 agents, as illustrated in Figures 6.6 a and 6.6 b, results in the *HRP* demonstrating that it can provide the network system with a self-organisation capability which leads to an enhancement in the number of executed task, over and above that achieved by the MAS/SBS approach.



(a) (b)

Figure 6.6: *Comparison Between Henchman Recovery Protocol and The Master/standby Server, with network size 5000 and probability* 0.6 *and* 0.9

6.7 Chapter Summary

This chapter has mainly focused on describing a new technique we have implemented and tested for dealing with *Head* failure within heterogeneous emerged organisations. We have provided the emerged organisations with a protocol which is integrated into their structure. A new role has been added as a result of deploying this new protocol, *HRP*. We have highlighted the importance of solving the disruption problem using a self-organised multi-agent system as well as of providing a solution in which organisations emerge without the requirement for central control. We have managed to demonstrate the *HRP* as a remedy for the disruption problem within multi-agent systems. The purpose of the *Henchman* agent is to maintain the functionality of the organisation and its effectiveness in the case of the failure of the *Head*. Monitoring the availability of the *Head* by the *Henchman* is achieved via a periodic message which is sent by each *Henchmen* to the *Head* of its organisation. The experimental work demonstrated that the *HRP* is a reliable solution for a self-organised system. Moreover, we have compared *HRP* against the Master/standby server system [103]. Our suggested system performed better in terms of the number of successfully executed tasks in each of the simulation cycles. The test was carried on two different network sizes: namely, 500 and 5000. In the next chapter, we explore dynamic organisations that may face permanent agent failure and how tasks can be executed in such a dynamic environment in a way which leads to the maintenance of the service level to the customers. Furthermore, we study the organisation formation structure's stability in the circumstance of new agents joining and existed agent failing.

Chapter 7

Agent Failure in Dynamic Organisations

7.1 Introduction

Self-organising agents can be considered a reliable platform for deployment in various autonomic computing systems because such agents can provide these systems with the ability to manage themselves [104]. Deploying self-organising agents in dynamic and distributed systems leads to the evolution of new structures in such systems which can enhance task execution and delegation due to changes in the structural relations between agents; these may lead to better performance.

Real world networks can consist of multiple-layers, all of which can be affected by unpredictable changes (disruptions) in their structure [105]. Dynamic systems can be changeable over time – i.e., an agent may unexpectedly fail, or a new agent may join in; at such times, the ability to continue to deliver the requested customer services may be very critical. In our scenario, the environment can receive parallel and distributed messages (tasks) in each cycle. The customer agent is the part of the system that sends messages which contain customers' tasks to be executed by randomly selected agents in the network. Also, service provider agent may fail at any cycle while it is of active status, leading to the loss

of its assigned tasks. An agent failure will almost certainly affect the stability and performance of the network.

In chapter 6, we presented the *HRP* as a recovery protocol which can be deployed in the heterogeneous organisation formations to enhance the organisations' performance and minimize the effects of the agent failure problem.

In this chapter, we explore dynamic organisations that may face permanent agent failure and how tasks can be executed in such a dynamic environment in a way which leads to the maintenance of the service level to the customers. Furthermore, we study the organisation formation structure's stability in the circumstance of new agents joining which may lead to the creation of a new set of connections within the organizations or the new agents presence may result in the emergence of new organisations.

7.2 Abstract scenario

Agents are autonomous and have their own decision making behaviours. An agent may decide to join more than one organisation and this ability varies from agent to agent. In relation to working in a dynamic environment, we have investigated the tolerance to failure of dynamic organisation systems and we have created three different models: the first is a model representing a straightforward network of agents; the second model implements a virtual layer of heterogeneous organisations which provides a self-organised multi-agent scheme and the application of roles and protocols for the agents; and the third is the *Henchman* Recovery Protocol, *HRP*, model. The following explains further:

- Agents are autonomous and hold heterogeneous types of resources. Each agent in the system has been designed so that it has its own accepted-task queue.
- An agent or multiple agents may fail randomly and permanently. In such cases, the failed agent's profile will be removed from the system. Agents in

the network are connected to a number of neighbours. An agent can detect other failed agents through the task delegation process. When an agent delegates a message to another agent, if it does not receive any feedback from that agent for a number of cycles, then that agent will be considered failed. Therefore, the sender agent will delete its contact information related to that failed agent. The failure problem can lead to the loss of agents and leave a number of agents suffering from a lack of connections. Hence, the network could eventually collapse and, indeed, gradually disappear. However, new agent(s) may appear to make up for the lack of agents in the system and so restore system performance. A new agent will send messages to randomly-selected other agents – to obtain at least partial knowledge of its surrounding environment – and start to connect with them. When a receiving agent, in this model, cannot execute a customer task, it will use just the delegation protocol in order to acquire assistance for the satisfaction of the requested task's required resources with the attached accuracies and Time To Live (*TTL*) values as requested by the customer.

- The second model is called the heterogeneous organisational model. Agents within any of the organisations which have been created may fail. This model consists of two layers; agents may participate in organisations. The failed agent could be a *Head* or a *Member*, if it is a *Head*, then when the *Members* wants to send messages to the *Head* and there is no response from the *Head* after a number of attempts, the *Members* will consider the *Head* to have failed. For the other way around, when a *Member* agent has failed the *Head* will detect this after a number of attempts at sending a messages; in this latter circumstance, the *Head* will remove that *Member* from its *DB*. To maintain the stability of the emerged organisations in the face of agent failure problems, new agent(s) will be injected and may accept roles inside one or more organisations and/or create a new organisation.

- The third model we have used is one which includes the *Henchman* Recovery Protocol, *HRP*; this is, in fact, the same as the second model except for the addition of an *HRP* in each organisation. All the above descriptions are applicable in this model as well. But, in addition, if the failed agent is one

which has a *Henchman* role in one of the existing organizations, the *Head* will remove it from its DB and the *Members* DBs will be informed by the *Head* that the *Henchman* is no longer functioning. The *Head* agent will then use the algorithm, "BeMyHenchman", as described in chapter 6, in order to obtain a new *Henchman* for its organisation.

- If both the *Head* and then the *Henchman* of the organisation have failed, the organization will have neither a *Head* nor a *Henchman*, and in such cases, when the *Members* want to access them by sending messages they will receive no reply, and after a number of attempts the *Members* will consider the organisation to have disbanded.

- Each task requires a random set of resources; these may be different from one task to another. Each task should be completed within a specific deadline, *DL*. If task T_i has not completed successfully by time *DL*, this means that agent a_i has failed during the execution time, and the received task's *DL* is also considered to have been exceeded. Sometimes tasks have failed because there was no agent available to execute the task within the *DL* – because either the agents were all busy executing tasks and/or their accepted task queues were all full or task Time To Live *TTL* value has expired.

7.3 Recovery Mechanisms in the Created Organisations

Agents in the organisations are prone to fail at any cycle, and once they have failed they are no longer available to the system. A suitable process is required in order to prevent the organisations from collapsing and to demonstrate how the organisations can cope with such failures and how agents must work with and respond to environmental changes. So, the failed agents could be *Head*, *Member*, or *Henchman*, and their failure will be detected in the following ways:

- If the agent is a *Member*, then its failure will be detected by the *Head* and the

Head will remove the failed agent from its DB and inform the *Henchman* to do the same.

- If the agent is a *Head*, then the *Henchman* will detect that and stand-in as a temporary *Head*.
- If the agent is a *Henchman* and the *Head* has other *Members*, then the *Head* will send messages to its *Members* asking them to be the new *Henchman*. But if no agent is able to comply then the organisation will have to depend solely on the *Head*.
- If the *Head* and the *Henchman* have both failed, then the organisation will disband. And this means that the *Members* will have detected the failure after a number of failed attempts to access the *Head* and the *Henchman*.

Furthermore, introducing a new agent will also affect the stability of the organisation's structure. New agents may try to join the available organisations, depending on their decision as to whether to cooperate with these organisations or not. Later on, a new agent may become one of the most-busy agents and therefore take on a role: i.e., it may satisfy the trigger conditions for creating its own organisation and so become the *Head* of this organisation. Another possible role is that the new agent may receive an invitation message from an existing *Head* to perform a role: this role could be that of *Member*, initially, and after it has accepted such an invitation, it could be then asked to become a *Henchman*.

7.4 The Experimental Work

We have developed three models which all use the setting parameters, as in 7.1. In relation to these three models, we have shown the results of various network sizes, task distributions (the task distribution is used to specify the number of tasks sent from customer to the network of agents in each cycle) and simulation times. For all of the three models, the number of tasks sent from the customer agent follows a normal distribution with specific values for the mean and variance. In this work,

we have found that we need to increase the number of simulation cycles used in order to give the self-organisation process represented by organisation_Ver1 and *HRP* more time to demonstrate better outcomes. However, in the interests of a fair test, we needed to increase the simulation time for all of the three models, hence, the simulation time was set at 7000 cycles.

Table 7.1: *Experimental Setting Parameters*

Agent Network Size	Task distribution	Simulation Time
500	Mean=1000,variance=10	7000 Cycles
2500	Mean=1000,variance=10	7000 Cycles
5000	Mean=1000,variance=10	7000 Cycles

Figures 7.1 and 7.2 show the average number of successfully executed tasks, *ANSET*, within the cycles, for each of the three models which have been implemented. Note that the two models (Organisation_Ver1 and HRP) have a specific structure in terms of their organisations and also have graduated responsibilities for their agents: agents have different roles(*Head*, *Henchman*, *Member*). However, the roles of agents may change over the simulation time and agents disappear and new agents appear; these changes will affect the execution of tasks and the system's ability to schedule tasks.

The No Organisation model consistently delivers lower numbers of successfully executed tasks over the simulation time (than the other two). So, using this model, the system loses a significant number of tasks due to frequent agent failures and the restrictions imposed by the messages' Time To Live (*TTL*) and the tasks' deadlines. In this model, the resources are distributed and the delegation for the task message may take more time to reach a desired agent that can accept the task. Hence, the other two models, Organisation_Ver1 and *HRP*, show better performance; in these cases, fewer hops are required in order to find agents which can accept the tasks because, even though there is a probability of failure, the organisations are created with heterogeneous resources and therefore, in the event of failure, other agents are readily available to execute the tasks. The *ANSETs* of these two models fluctuate due to the presence of the probability of failure, which causes disruption in the system such that agents will start to fail inside the created organisations leading to significant changes in the execution rates of the

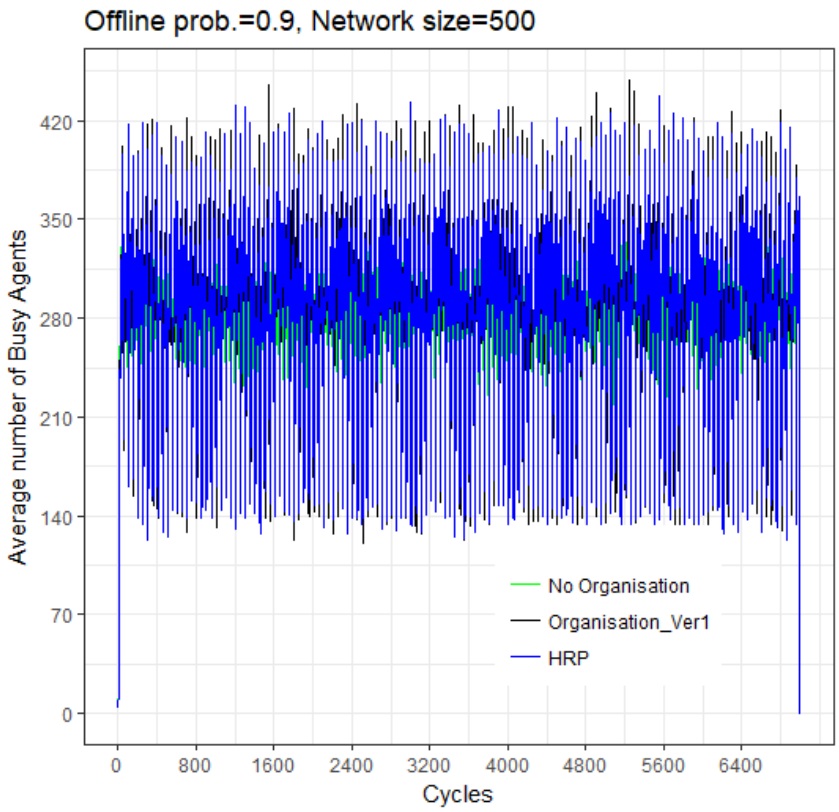


Figure 7.1: *Average numbers of successfully executed tasks computed across 7000 simulation cycles, p*= 0.9

tasks. The No organisations model shows a smaller percentage of variation in its *ANSET* as compared to the other two models. It is clear that the No Organisation model executes tasks within the same ranges of time, on average, in each and every simulation cycle. This is because the model contains no structure and new agents may connect randomly with other existent individual agents. Where the task may be executed is dependant only on delegation across the entire network, and this quickly consumes the tasks' *TTL* values. Hence, the Organisation_Ver1 and *HRP* models perform better on average than does the "No Organisation" model.

In Figures 7.3 a and 7.3 b., it is noteworthy that, with increases in the network size, the system activity becomes more stable, showing less fluctuation in the *ANSET* values than in a network size of 500 agents. Even in the presence of significant numbers of failures, the *HRP* model achieves the highest average number of successfully executed tasks. This is because, first, the higher the number of agents in the system the more organisations can be created with a

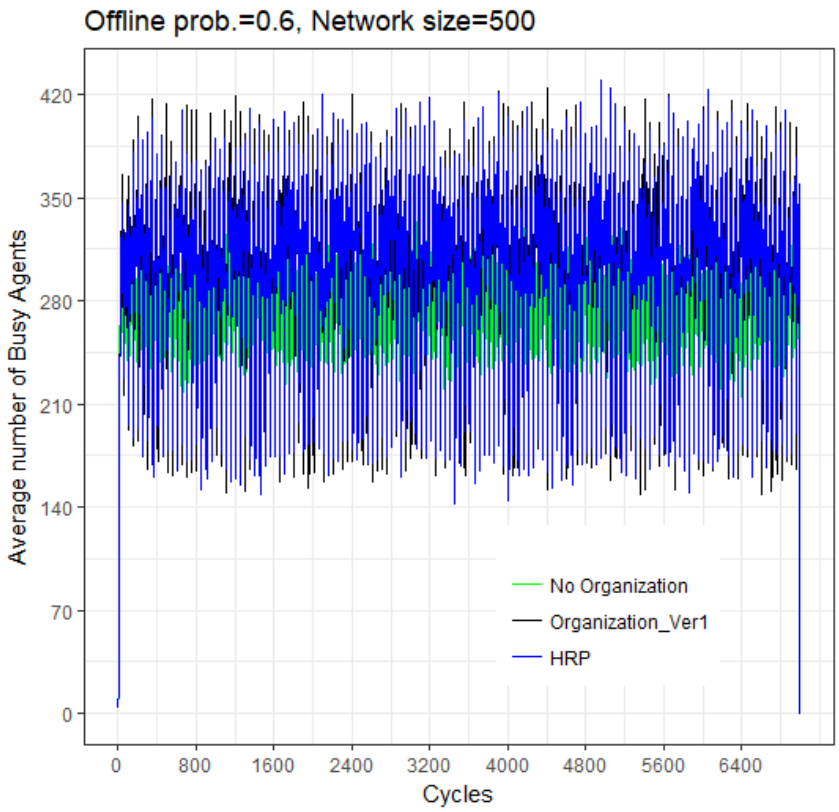
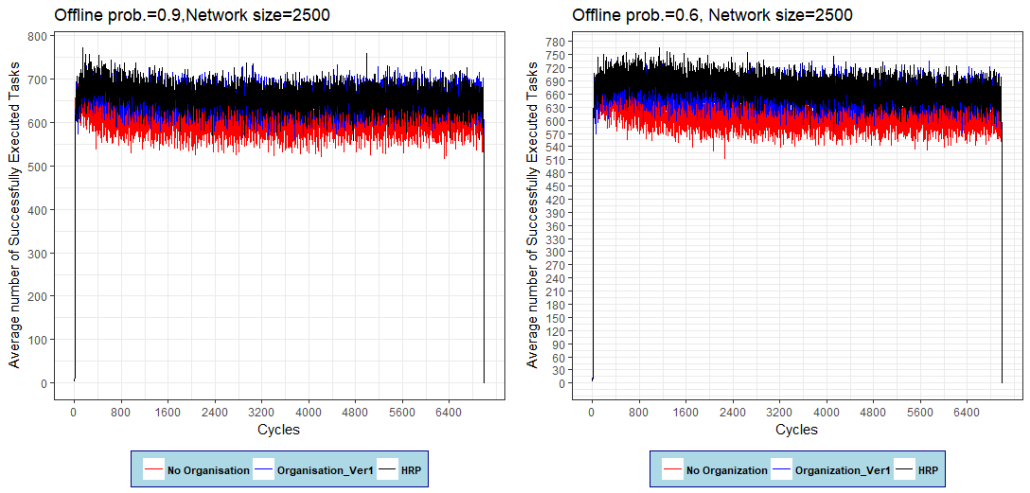


Figure 7.2: *Average numbers of successfully executed tasks computed across 7000 simulation cycles, p*= 0.6

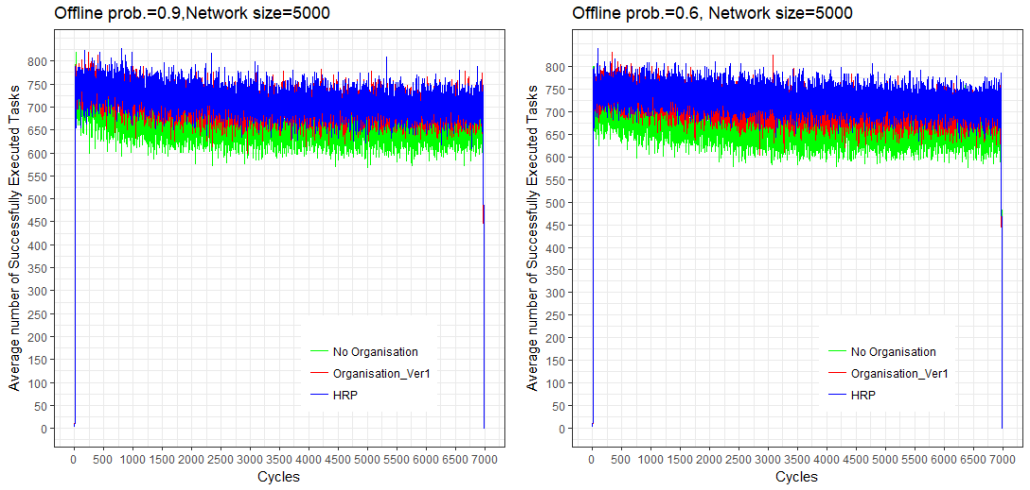
greater variety of resources. Second, when failures occur in the system, there are other agents which are able to join the network structure and so can also join the existing organisations.

The same explanation is applicable to 7.4 a, and 7.4 b, where increasing the network size leads to there being higher numbers of tasks executed within the simulation cycles: the *ANSET* is more than with the other two network sizes (2500, 500). In models Organisation_Ver1 and *HRP*, the created organisations consist of a large number of heterogeneous agents, so even with the failure of an agent the *Head* will probably still able to find a *Member* that can satisfy the required resources. Moreover, the existence of the *Henchman* in the *HRP* has added benefit to the system.



(a) (b)

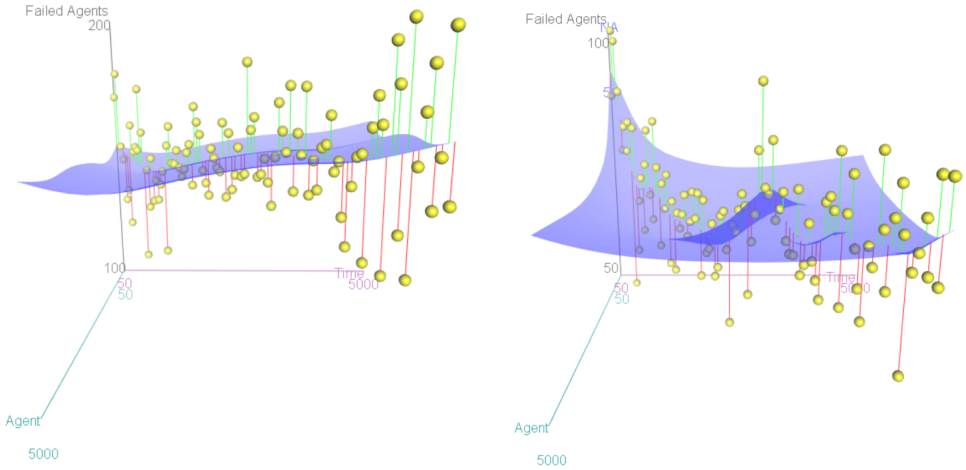
Figure 7.3: *Average numbers of successfully executed tasks computed across 7000 simulation cycles, p= 0.9 and p= 0.6*



(a) (b)

Figure 7.4: *Average numbers of successfully executed tasks computed across 7000 simulation cycles, p=0.9 and p=0.6*

Figures 7.5 a, and 7.5 b show the *HRP* model which has been used to demonstrate the number of failing agents in the system with a network size of 5000 agents. The figures show that the number of failing agents is between 135 to 187 (nodes) every 50 cycles – with a probability value of 0.9. With a probability value of 0.6 the number of failing nodes is between 55 to 87 – also every 50 cycles. In Figure 7.5 a, the average number of failing agents is between 3.0 to 3.5 every 50 cycles. Hence, in the latter case, the number of failures in the environment can be considered very small in relation to the size of the network. Also, a large number of emerged



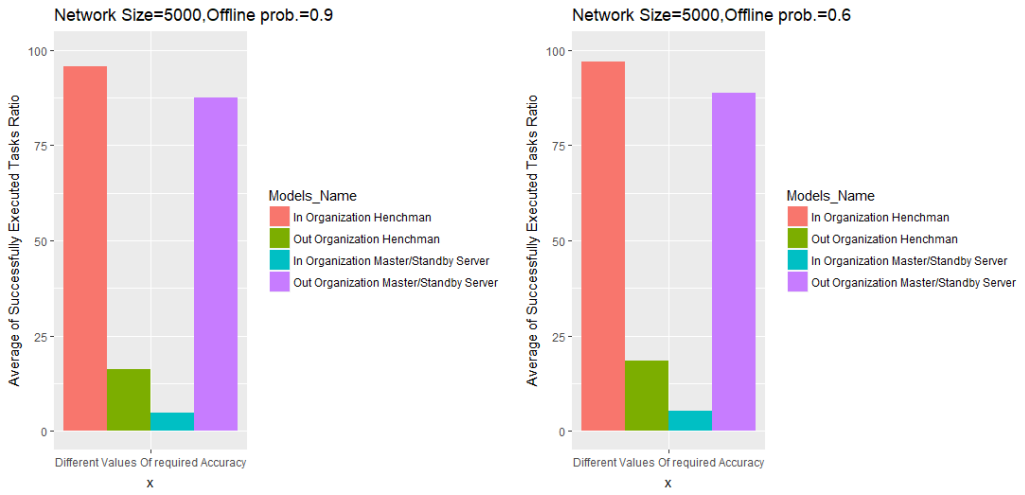
(a) 5000 Agents,p=0.9 (b) 5000 Agents,p=0.6

Figure 7.5: *Average numbers of Failed Agents computed across 7000 simulation cycles, p*= 0.9 *and p*=0.6

organisations is demonstrated here which means that the failures have less effect on the system. This shows the advantages of using the HRP model: it leads to good system performance. The same explanations are also applicable to Figure 7.5 b with a probability p = 0.6.

7.4.1 Evaluation HRP Model

We have attempted an empirical comparison between *HRP* with other methods from the literature. The results from the *HRP* model with network sizes (500, 2500, 5000) were compared against the master-standby system [103]. The Master (MAS)/standby (SBS) model is a fault tolerance model in which the system has two servers. The first one is called the MAS and all the clients are connected to it. The second is called the SBS – the clients are only connected to this when the MAS has failed. Furthermore, there is a checking message transmitted between the MAS and the SBS which enables the SBS to switch to active mode and serve the clients' requests in place of a failed MAS. We have implemented this architecture within our simulation set-up, and the results are depicted in the following Figures:

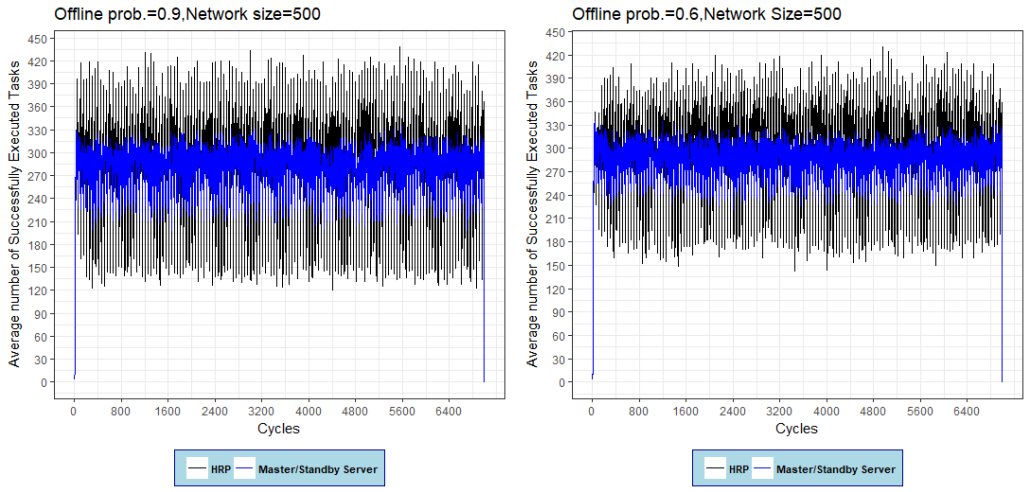


(a) (b)

Figure 7.6: *Average number of successfully executed task ratio in/out organisations with probability p*= 0.9 *and p*= 0.6

In Figures 7.6 a, and 7.6 b, we have computed the number of executed tasks ratio, *ANETR*, in relation to the following: various different required accuracies; execution inside and outside of organisations; a network size of 5000 agents; and two probabilities of failure – 0.9 and 0.6. As shown, a network where *HRP* operates within organisations outperforms the MAS/SBS model. This is because the MAS/SBS model depends for its operation on its delegation process, and a smaller number of tasks have been executed inside its organisations as a result of the fact that this model has no structural roles such as those in the *HRP* model's

organisations. *Members* inside the organisations have roles and they can delegate tasks to the *Head's* of organisations that they are part of. Also, the *Head* and the *Henchman* of an organisation each have a significant role to perform within it. However, in MAS/SBS the nodes delegate tasks to other nodes in the emerged organisations, and if no agent can accept the task the task will be delegated across the network which consumes its *TTL* values.

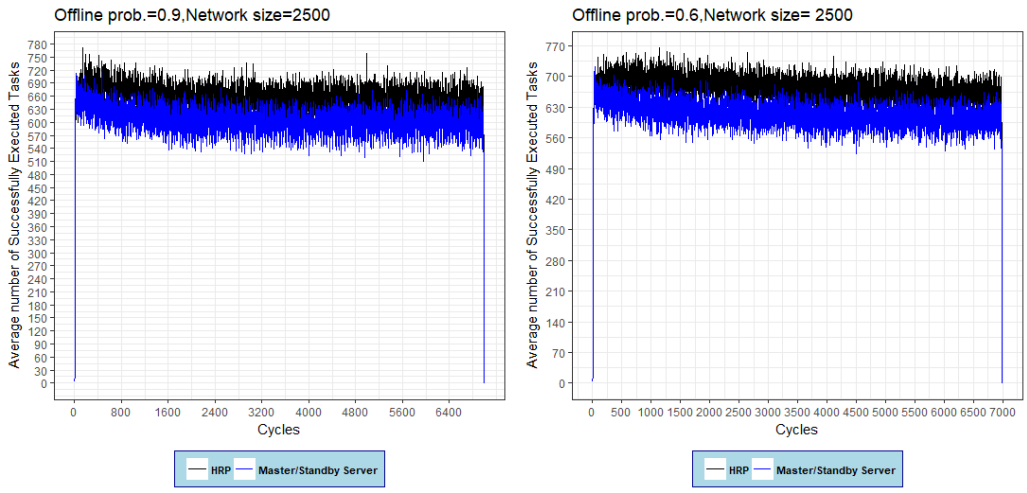


(a) (b)

Figure 7.7: *Average numbers of successfully executed tasks computed across 7000 simulation cycles, p= 0.9 and p= 0.6*

Figures 7.7 a, and 7.7 b show that the *HRP's* performance has been effected by the agents which have failed within the organisations, but on average the *ANSET* is higher than that yielded by the MAS/SBS model server. The *HRP* model with a network size of 500 agents showed a fluctuation in its *ANSET* because of losing the agents with roles in the organisations – which affects the system performance. The *ANSET* from the MAS/SBS model is almost the same as that of the No organisation model because it uses the delegation across the network.

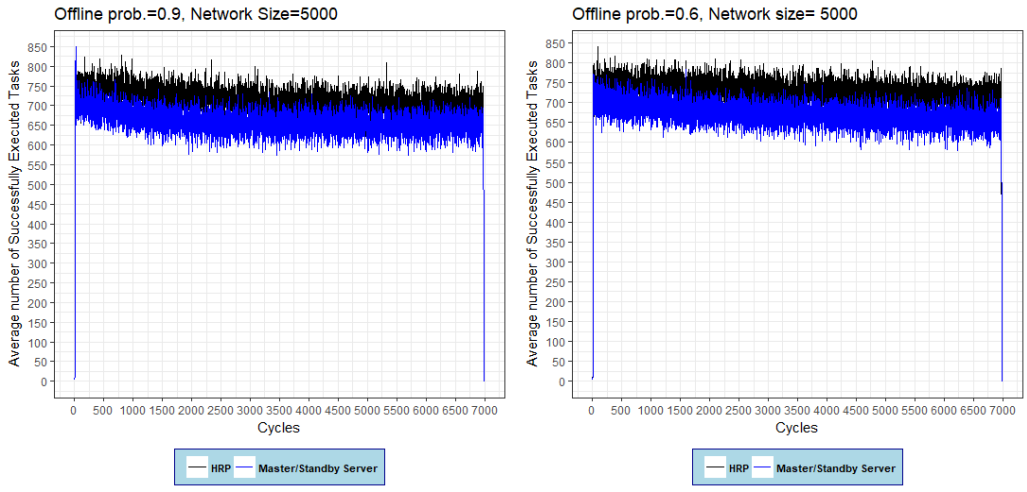
Figures 7.8 a, 7.8 b, and 7.9 a, 7.9 b, show the models' performance, in terms of *ANSET*, within the simulation cycles. It is clear that the *HRP* model performs better than the MAS/SBS model in terms of task execution throughput. The system performance increases when the network size increases to 2500 and then 5000 agents. The *HRP* demonstrates, on average, a better performance in each cycle than the MAS/SBS model. This is because, in our scenario, the *Henchman* has a



(a) (b)

Figure 7.8: *Average numbers of successfully executed tasks computed across 7000 simulation cycles, p = 0.9 and p= 0.6*

role in the self-organisation process where it acts as a *Member* that can receive and execute tasks as well as being a follower to the *Head* of the organisation. In contrast, in the MAS/SBS approach, the SBS is only a backup server and is only switched to active mode when the MAS fail. Moreover, in relation to both probabilities of failure and also in relation to all the various network sizes, the *HRP* demonstrates better performance.



(a) (b)

Figure 7.9: *Average numbers of successfully executed tasks computed across 7000 simulation cycles, p= 0.9 and p= 0.6*

We now present the amount of message traffic that each of the four models generates during their simulation cycles in order to demonstrate the amount of *TTL* usage for each model. The message traffic information presents the *TTL* value which is attached to each task. We have computed the *TTL* values for each task which is transferred via messages within the environment. To compute the traffic in the network, $MsgTraffic$, the following formula has been used:

$$MsgTraffic = \frac{\text{NoMsgs}}{\text{TRT}\times\text{TTL}} \times 100 \tag{7.1}$$

Where:

NoMsgs: The accumulated number of messages in the network for all the received tasks.

TRT: The total number of received tasks.

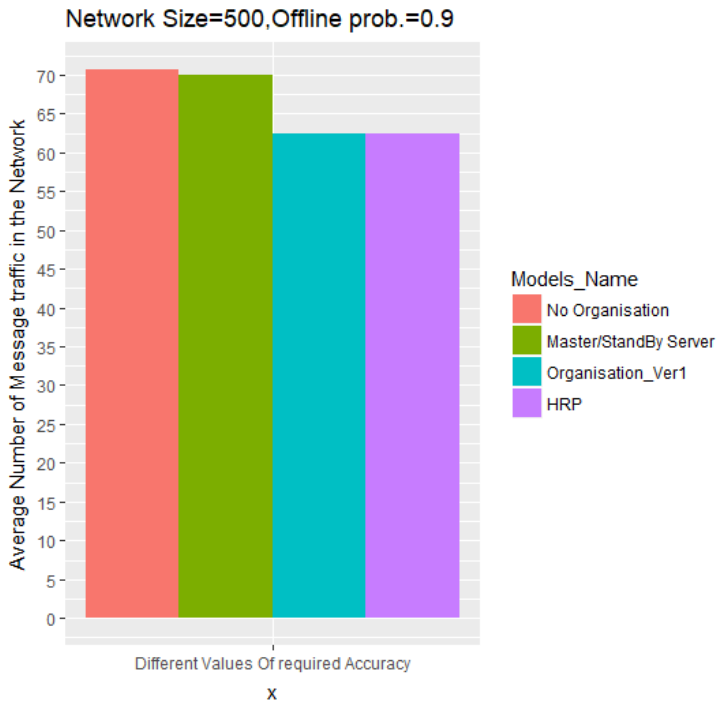


Figure 7.10: *Message Traffic in the implemented Model 7000 simulation cycles,* 500 *agents, p=* 0.9

Figure 7.10 shows that the *TTLs'* usage reached its highest, on average, in the No Organisation model due to the higher number of hops that the messages took

in order to be executed; the second highest *TTL* usage resulted from the adoption of the Master/standby server model. In contrast, using the models Organisation_Ver1 and the *HRP* model, the traffic is significantly less. This is because the latter two models contain organisations that, due to their structures, use fewer numbers of hops to traverse the agents which are within them. In addition, the *HRP* model uses even less message traffic than the Organisation_Ver1 model. This is because the *Henchman* of an organisation will work as a *Head* in the case of the *Head's* failure, so tasks can still be executed within the organisation and do not need to be delegating across all the agents in the network. So, the existence of the *HRP* in each organisation leads to a decrement in the *TTL* usage beyond that achieved by the use of the organisation model only.

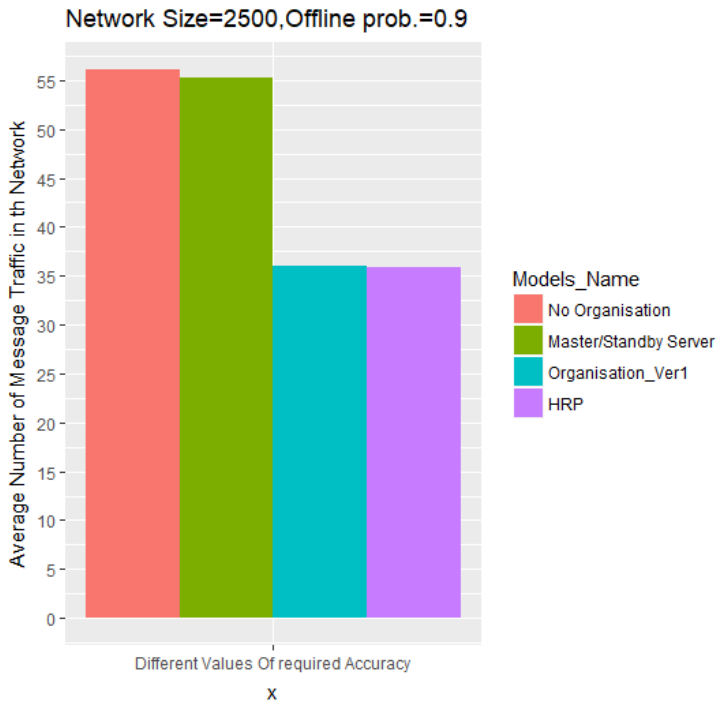


Figure 7.11: *Message Traffic in the implemented Model 7000 simulation cycles,* 2500 *agents, p=* 0.9

Figures 7.11 and 7.12 show that the increases in the network sizes to 500 and then 5000 lead to the production of less traffic in all the four available models (No Organisation, Server/standby server, Organisation_Ver1, *HRP*): Figure 7.10. In the Organisation_Ver1, and *HRP* models, the traffic in the system is less than the

traffic produced by the No Organisation and MAS/SBS model because less of the *TTL* value was used to reach agents which could accept and execute tasks – due to the organisational structure present in these two models.

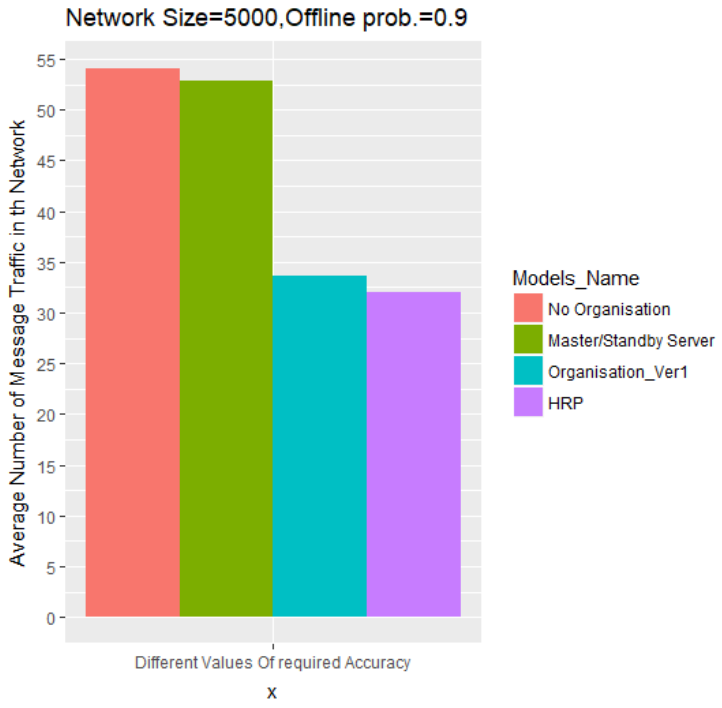


Figure 7.12: *Message Traffic in the implemented Model 7000 simulation cycles*, 5000 *agents*, *p*= 0.9

7.5 Chapter Summary

This chapter focused on studying the creation of open and dynamic agent organisation formations which can provide services to requesting customers in the presence of failure. In chapter 6, we presented the *HRP* to minimize the effects of the agent failure problem. However, in this chapter, the motivation is to explore agents in organizations when they are prone to permanent failures such as occur in any distributed environment. This can lead to the loss of tasks and to decreases in the effectiveness and utilisation of agent networks. We have presented a framework whereby agents and organisations can be counted on to provide remedies which can avert these kinds of disruptions. Our aim was to deploy the *Henchman* Recovery Protocol. *HRP*, within each organisation; this is a viable solu-

tion for maintaining the functionality of the organisations. After that, to preserve the stability of the emerged organisations, new agents will be injected into the environment to create new connections (and so the organisation structures may change). Weeding out failure from distributed systems is not something that can be done automatically on a totally predictable basis, and the problem presented by failures requires sound theories and efficient solutions that can be applicable to critical domains in order to maintain their stabilities [106], [5]. Grid computing is the target domain for this work because it can provide researchers with a suitable environment in which to apply our virtual organisations as well as in which to study node failure. Our solution is to apply a heuristic protocol, *HRP*, in order to recover customer tasks and preserves the organisations' formation structure. *HRP* has been shown to have a more acceptable performance as compared to the MAS/SBS model. The existence of roles inside the heterogeneous organisations plays an important role in the self-organisation of the systems and provides a pro-active technique for dealing with failure. The experiments have shown that the *HRP* produces fewer traffic messages than the other models: (No Organisation, Organisation_Ver1, MAS/SBS).

Chapter 8

Conclusion and Future Work

8.1 Conclusions

For systems that are dynamic and complex, there is often no straight forward mathematical equation which is able to solve them. In such circumstances, simulation can be the best approach for estimating the likelihood of various outcomes [107]. This work has aimed to address some of the challenges and limitations associated with dynamic, open and distributed network systems and most particularly those relevant to the field of grid computing. We have used the Repast Simphony simulator for these purposes. Here, we have provided methods which preserve system functionality by maintaining, in the presence of node failure, network/system effectiveness. Furthermore, we have studied the effects of node/agent failure within the environment in question, and addressed how to minimize these effects on the network and on customers' tasks execution. We have designed a number of models in order to study and analyse various outcomes resulting from the failure problem; these models helped us to adopt the necessary solutions. The novelty of our approach lies in suggesting a number of mechanisms and protocols for distributed environments. Agents in the network possess various statuses such as busy, not busy or failed; the latter was included in order to represent a dynamic environment in which a fault tolerant solution is essential in order to recover customers' tasks – as demonstrated in chapter

4. Where customer tasks cannot be executed in one hop, the resources used in distributed systems are heterogeneous and can be shared in order to speed-up services provision and maintain the quality of service to the users. Therefore, a distributed task delegation algorithm has been utilized, and within this a directed search algorithm has been implemented – to distribute the customers' tasks across the network according to agents' statuses. In the presence of failure, the system has shown that, to some extent, it has the ability to provide the customers tasks with the requested services.

The problem of failure needs to be met with effective techniques for minimising its effect on the adequate accomplishment of the customers' requested tasks. Thus, we have investigated self-organisation/self-healing methods which are appropriate to multi-agent systems. We have presented a number of self-organisation algorithms in chapters 5, 6 and 7 which are based on the gossip protocol concept: organisations emerge, i.e., organised groups of agents are created. These are either heterogeneous organisations or homogeneous organisations, and these helps to avoid the disruptions that could otherwise occur over the simulation time and recover customers' tasks. Here, we have preferred the heterogeneous types of organisation; these are unlike the organisations found described in some works in the literature – which are concerned with creating homogeneous organisations [14].

The use of this heterogeneous type of organisation has resulted in an enhancement in the performance of our example system – in terms of raising the average number of successfully executed tasks and the average number of tasks executed in each cycle. Also, we have created here a new self-organisation protocol, which we have termed the *Henchman* Recovery Protocol (*HRP*). This introduces a new role within the self-organisations. Each *Head* will try to find a *Member* that accepts the role of follower or *Henchman*. The *Henchman* helps in preserving the services delivery effected by the emerged organisations. In addition, the findings suggested that permanent agent failure was another situation which we should analyse and attempt to solve in order to try to minimize its negative effects on environments. For this scenario, we have implemented algorithms which address

the problem of permanent failure. In the following sections we will present the strengths and limitations of the methods involved. After this we will make some suggestions for future work.

8.2 Contributions Summary

This study contributes to the research which has been undertaken in regard to dealing with task recovery when disruption occurs within distributed environments. The first contribution we made was the design of a framework in which to deploy a multi-agent system and thus to make beneficial use of the autonomous agents' behaviour. This work aimed at using agents with heterogeneous types of resources as the nodes of the network. The network can benefit from autonomous agents in order to increase its functionality and performance. An agent can communicate with, and be contacted by, the surrounding environment only via its contacts. We demonstrated that agent/node status may vary: a node may be busy, not busy or failed. It should be made clear that we have deliberately focused on these particular status, here, as they represent the ones most frequently encountered; there are other node/agents' statuses such as pending and running (these are taken to mean busy for most intents and purposes). In addition, the status "lost node" is taken to mean the same as "failed agent" by the recovery processes.

Moreover, we have provided a new delegation protocol based on directing the task messages across the network; in this new delegation algorithm, a directed search based on the use of agents' statuses to direct the messages is used. This has been compared against the random search algorithm and has demonstrated better performance than the random search. The "failed" status can affect the task delegation process: e.g., if the receiving agent cannot execute a task, then it will try to delegate this to one of its neighbours; if that neighbour does not respond within a certain number of cycles, the original agent will try to delegate the task to another neighbour and then after a number of such agent delegation attempts, the task will fail because the *TTL* value associated with it will have been reached. However, a directed search algorithm which is based on checking the

agents' statuses first has demonstrated the ability to perform a higher number of customers' tasks, as compared to the random search algorithm. This was the first of our solutions that we put to the test; it has been shown that directed search results in better outcomes.

One of the most interesting ideas to come to light as a result of this research was that of prompting a network to cause virtual structures called organisations to emerge from the network environment as a second layer. Hence, we are enabling, through the use of various protocols and mechanisms, the emergence of organisations to support the effective operation of the network. This was one of the solutions that we have presented for the enhancement of task execution in distributed environments. The forming of organisations means that a task message will take fewer hops before it is executed by a receiving agent. Triggering conditions have been set which result in the creation of organisations. These are: an agent is currently busy, has just received another task from the customer and its accepted task queue contain tasks. These triggering conditions result in the creation of an organisation and the enacting of roles by agents. Thus, the original agent, by default, becomes the organisation's *Head*, and subsequently any agent which accepts the *Head*'*s* message to join-in its organisation will become a *Member*. Thus the self-organisation or self-healing technique helps to maintain the environment's efficiency even when disruption occurs without any outside intervention. Since the arriving customer tasks request various values in terms of required accuracies, in this research, both heterogeneous and homogeneous organisation models have been explored and we have compared these two.

The contribution here is that the roles are a result of the triggering conditions. The most busy agent becomes the *Head* of an organization and then sends a multi-cast message to other agents in the network inviting them to join its organization and provide services. Hence, the system contains two roles one is *Head* and the other is *Members*. In other related work such as [99], the agents are able to join an organization only at a specific point in their life time and must change their behaviour to join the organization and match the requirements of the requested role. The experiments we have carried out have shown that a system which in-

tegrates heterogeneous organisations produces a greater number of successfully executed tasks than a system which includes only homogeneous organisations. This is because having more than one type of agent, in terms of resource types, in an organisation means that various required accuracies (associated with tasks) can be satisfied. Also, this situation will help to reduce the impact of having failed agents inside the organisation. In contrast, in homogeneous organisations the process that enables agents to join an organisation is based on a matching between the agents' resources and the required accuracy that has been specified by an organisation's *Head*. This requirement means that all the *Members* can satisfy only one specific type of required accuracy while the arriving customer tasks, as mentioned above, request different values for their required accuracies. This means that not all tasks can be executed by a homogeneous type of organisation.

Another important achievement of this study is the development of what we have termed the "*HRP*". The idea here is to stop *Heads* of the organisations from losing tasks. This is done by deploying a *Henchman* Recovery Protocol, or *HRP*. The function of the *Henchman* in each organisation is to monitor the *Head's* availability via a heartbeat algorithm. The *HRP* facilitates the recovery of customer tasks when a *Head* fails. This is demonstrated in the experimental work.

Our experimental work has shown that the existence of roles (*Head*, *Member*, *Henchman*) in a self-organised environment can improve system utilisation and performance as compared to that achieved by another approach described in the literature [103], the master/standby server approach, when dealing with fault tolerance in a distributed system. Different probability values for failing have been tested with different network sizes in order to implement the master/standby server: the standby server only operates when the master server has failed. The main contributing factor to the relative success of the *HRP* is the ability of the *Henchman* agent to monitor the *Head* and process tasks whenever the latter cannot. This produces an improvement across all the required accuracies and network sizes. Also, an organizational structure which includes the main roles (*Head*, *Henchman*, *Members*) leads to the emergence of self-organised groups which can overcome the random failure issue. *HRP* thus contrasts with the master/standby

server method whereby the standby server can only switch to active mode when the master server has entirely failed.

In this thesis, we have also explored permanent agent failure by designing three models. We have developed a network model, an organisations model and an *HRP* model. These models have each been presented with permanent agent failure which is made to occur using random values. The experimental work has shown that the viable solution among these models, for maintaining the functionality of the environment, is the *Henchman* Recovery Protocol *HRP* model, whereby the *HRP* is maintained within each organisation. A further feature which has been implemented within the system is the injection of new agents into the environment to maintain the stability and the services provided by the emerged organisations. Moreover, we have compared our work to that achieved by another approach described in the literature [103].

Some areas of improvement in regard to this work are: (1) in terms of the description of the resources in the simulated models, it might be better to find another method to represent the semantic resources (e.g., by implementing them via a real-world representation such as one using Ontology); (2) another limitation is that, due to lack of resources, we could not apply the proposed work on a real grid system; (3) in chapter 7, an improvement to the implemented work is to find a new method that provides the system with the ability to deal with the probability that a failed agent may return back to active mode.

8.3 Future work

In this thesis, we have highlighted the agent failure problem as it occurs in distributed environments and its effect on task execution, and we have suggested a number of heuristic solutions to this problem. However, there is always the possibility that other approaches can be looked at which may increase a system's ability to survive; these approaches would constitute further research. One such avenue is to extend our design so that it can act as the basis of a normative system.

The roles of *Head*, *Henchman* and *Members* can be extent by adding a number of rules for their behaviour ; this would be the first attempt to create a normative system, wherein these rules could include obligations and permissions in regard to the *Members*, and by which the *Heads* of the organisations will apply either encouragement to their *Members* by increasing their utility, or sanctions, if they are not committed to the *Head*'s organisation.

Another avenue would be to attempt to design hierarchical organisations, because such an attempt might be useful for determining the type of organisational structures which are appropriate to such an environment. Studying such structures might help to determine which are optimal – by comparing between hierarchical network designs and the self-organisation design used in our model.

In term of further work, more research would be beneficial if an organisation can have both abilities to self-heal from failure that has been explained in chapter 6 and permanent failure that has been explained in chapter 7.

Another possible technique could be used alongside the organisation models, instead of *HRP*: the *Members* of the organisation could be the ones that invoke the algorithm for checking the *Head*'s availability – checking the *Head*'*s* in random cycles. If this were done, when a *Head* fails, a voting algorithm could be used by the *Members* to decide which one of them will be selected to be the new organisation's *Head*. Their preferences for selecting the new *Head* would be based on the various candidates' utility values and/or on using only the most busy *Members* to participate as candidates while other *Members* would be voters. After this, the agent with the highest number of votes would act as the *Head* of the organisation.

Another direction for further investigations, in relation to the *HRP* model, is the idea of a leadership competition between the *Head* and the *Henchman*. This would be to see which one of them was more suitable for the leadership role. For example, if the *Head* has been offline more frequently, then the *Henchman* could undertake the leadership role and exclude the *Head*. Another idea, for the case of *Henchman* failure, is that the organisation could determine that it needed another

Henchman: i.e., an organisation could be constructed with a *Head*, *Members*, and dual Henchmen.

We think that our work is more generic than the fact that our design operates within the application layer implies. We hope that we can integrate it as a service in the grid middle-ware; this will be our next step.

References

[1] M. Baker, R. Buyya, and D. Laforenza, "Grids and grid technologies for wide-area distributed computing," *Software: Practice and Experience*, vol. 32, no. 15, pp. 1437–1466, 2002.

[2] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a new computing infrastructure*. Elsevier, 2003.

[3] J. Vasques and L. Veiga, "A decentralized utility-based grid scheduling algorithm," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pp. 619–624, ACM, 2013.

[4] M. łgorzata Steinder and A. S. Sethi, "A survey of fault localization techniques in computer networks," *Science of computer programming*, vol. 53, no. 2, pp. 165–194, 2004.

[5] S. Haider and B. Nazir, "Fault tolerance in computational grids: perspectives, challenges, and issues," *SpringerPlus*, vol. 5, no. 1, p. 1991, 2016.

[6] G. F. Coulouris, J. Dollimore, and T. Kindberg, *Distributed systems: concepts and design*. pearson education, 2005.

[7] M. Wooldridge, "Agent-based computing," *Interoperable Communication Networks*, vol. 1, pp. 71–98, 1998.

[8] M. Wooldridge, *An introduction to multiagent systems*. John Wiley & Sons, 2009.

[9] J. Ferber, O. Gutknecht, and F. Michel, "From agents to organizations: an organizational view of multi-agent systems," in *International Workshop on Agent-Oriented Software Engineering*, pp. 214–230, Springer, 2003.

[10] A. Bezek and M. Gams, "Comparing a traditional and a multi-agent load-balancing system," *Computing and Informatics*, vol. 25, no. 1, pp. 17–42, 2012.

[11] N. R. Jennings and M. Wooldridge, "Applications of intelligent agents," in *Agent technology*, pp. 3–28, Springer, 1998.

[12] P. R. Varakantham, S. K. Gangwani, and K. Karlapalem, "On handling component and transaction failures in multi agent systems," *ACM SIGecom Exchanges*, vol. 3, no. 1, pp. 32–43, 2001.

[13] S. J. Russell and D. Subramanian, "Provably bounded-optimal agents," *Journal of Artificial Intelligence Research*, vol. 2, pp. 575–609, 1995.

[14] M. J. J. Al-Asfoor, *Resource discovery in self-organising distributed systems*. PhD thesis, University of Essex, 2014.

[15] M. Dignum, Virginia, J.-J. Meyer, H. Weigand, and F. Dignum, "An organization-oriented model for agent systems," AAMAS, 2002.

[16] V. Dignum and F. Dignum, "Modelling agent societies: Co-ordination frameworks and institutions," *Progress in artificial intelligence*, pp. 7–21, 2001.

[17] V. Dignum, "The role of organization in agent systems," *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, pp. 1–16, 2009.

[18] D. D. Corkill, D. Garant, and V. R. Lesser, "Exploring the effectiveness of agent organizations," in *International Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems*, pp. 78–97, Springer.

[19] Y.-Y. Ahn, J. P. Bagrow, and S. Lehmann, "Link communities reveal multiscale complexity in networks," *nature*, vol. 466, no. 7307, pp. 761–764, 2010.

[20] G. Palla, I. Derényi, I. Farkas, and T. Vicsek, "Uncovering the overlapping community structure of complex networks in nature and society," *Nature*, vol. 435, no. 7043, pp. 814–818, 2005.

[21] T. P. Peixoto, "Hierarchical block structures and high-resolution model selection in large networks," *Physical Review X*, vol. 4, no. 1, p. 011047, 2014.

[22] M. Premm and S. Kirn, "Autonomous agents in multiagent organizations," in *Proceedings of the 9th International Conference on Agents and Artificial Intelligence - Volume 1: ICAART,*, pp. 121–128, INSTICC, SciTePress, 2017.

[23] A. Zbieg, D. Batorski, and B. Zak, "How to select change agents in organizations: A comparison of the classical and network approaches," *Problemy Zarzadzania*, vol. 14, 2016.

[24] T. De Wolf and T. Holvoet, "Emergence versus self-organisation: Different concepts but promising when combined," in *International Workshop on Engineering Self-Organising Applications*, pp. 1–15, Springer.

[25] G. D. M. Serugendo, M.-P. Gleizes, and A. Karageorgos, *Self-organising software: From natural to artificial adaptation*. Springer Science & Business Media, 2011.

[26] J. Kantert, S. Tomforde, M. Kauder, R. Scharrer, S. Edenhofer, J. Hähner, and C. Müller-Schloer, "Controlling negative emergent behavior by graph analysis at runtime," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 11, no. 2, p. 7, 2016.

[27] F. Ahmed and O. Tirkkonen, "Topological aspects of greedy self-organization," in *Self-Adaptive and Self-Organizing Systems (SASO), 2014 IEEE Eighth International Conference on*, pp. 31–39, IEEE, 2014.

[28] A. S. Jensen, V. Dignum, and J. Villadsen, "A framework for organization-aware agents," in *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pp. 1133–1134, International Foundation for Autonomous Agents and Multiagent Systems, 2016.

[29] P. Leitão, J. Barbosa, and D. Trentesaux, "Bio-inspired multi-agent systems for reconfigurable manufacturing systems," *Engineering Applications of Artificial Intelligence*, vol. 25, no. 5, pp. 934–944, 2012.

[30] P. Miller, "The genius of swarms," 2007.

[31] D. Teodorović, "Swarm intelligence systems for transportation engineering: Principles and applications," *Transportation Research Part C: Emerging Technologies*, vol. 16, no. 6, pp. 651–667, 2008.

[32] S. Y. Ko, I. Gupta, and Y. Jo, "Novel mathematics-inspired algorithms for self-adaptive peer-to-peer computing," in *Self-Adaptive and Self-Organizing Systems, 2007. SASO'07. First International Conference on*, pp. 3–12, IEEE, 2007.

[33] G. W. Flake, S. Lawrence, C. L. Giles, and F. M. Coetzee, "Self-organization and identification of web communities," *Computer*, vol. 35, no. 3, pp. 66–70, 2002.

[34] B. Koldehofe, R. Mayer, U. Ramachandran, K. Rothermel, and M. Völz, "Rollback-recovery without checkpoints in distributed event processing

systems," in *Proceedings of the 7th ACM international conference on Distributed event-based systems*, pp. 27–38, ACM, 2013.

[35] E. N. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson, "A survey of rollback-recovery protocols in message-passing systems," *ACM Computing Surveys (CSUR)*, vol. 34, no. 3, pp. 375–408, 2002.

[36] M. Franceschetti and J. Bruck, "A leader election protocol for fault recovery in asynchronous fully-connected networks," 1998.

[37] M. Wölbitsch, S. Walk, and D. Helic, "Modeling peer influence in time-varying networks," in *International Workshop on Complex Networks and their Applications*, pp. 353–364, Springer, 2017.

[38] J. Botev and I. Scholtes, "A self-organized resource allocation scheme for decentralized distributed virtual environments," in *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2010 6th International Conference on*, pp. 1–7, IEEE, 2010.

[39] B. Neville, M. Fasli, and J. Pitt, "Utilising social recommendation for decision-making in distributed multi-agent systems," *Expert Systems with Applications*, vol. 42, no. 6, pp. 2884–2906, 2015.

[40] V. Dignum, J.-J. Meyer, H. Weigand, and F. Dignum, "An organizational-oriented model for agent societies," in *Proc. Int. Workshop on Regulated Agent-Based Social Systems: Theories and Applications (RASTA'02), at AAMAS, Bologna, Italy*.

[41] M. S. Fagundes, F. Meneguzzi, R. Vieira, and R. H. Bordini, "Interaction patterns in a multi-agent organisation to support shared tasks," in *International and Interdisciplinary Conference on Modeling and Using Context*, pp. 364–370, Springer, 2013.

[42] M. Fasli, "On commitments, roles, and obligations," in *International Workshop of Central and Eastern Europe on Multi-Agent Systems*, pp. 93–102, Springer.

[43] B. Horling and V. Lesser, "A survey of multi-agent organizational paradigms," *The Knowledge engineering review*, vol. 19, no. 4, pp. 281–316, 2004.

[44] R. Hermoso, H. Billhardt, and S. Ossowski, "Trust-based role coordination in task-oriented multiagent systems," *Knowledge-Based Systems*, vol. 52, pp. 78–90, 2013.

[45] J. Tang and M. Zhang, "An agent-based peer-to-peer grid computing architecture: convergence of grid and peer-to-peer computing," in *Proceedings of the 2006 Australasian workshops on Grid computing and e-research-Volume 54*, pp. 33–39, Australian Computer Society, Inc., 2006.

[46] J. Yu and R. Buyya, "A taxonomy of workflow management systems for grid computing," *Journal of Grid Computing*, vol. 3, no. 3-4, pp. 171–200, 2005.

[47] B. Nazir, K. Qureshi, and P. Manuel, "Replication based fault tolerant job scheduling strategy for economy driven grid," *The Journal of Supercomputing*, vol. 62, no. 2, pp. 855–873, 2012.

[48] Y. Zhang, A. Mandal, C. Koelbel, and K. Cooper, "Combined fault tolerance and scheduling techniques for workflow applications on computational grids," in *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on*, pp. 244–251, IEEE, 2009.

[49] V. Shestak, E. K. Chong, A. A. Maciejewski, and H. J. Siegel, "Probabilistic resource allocation in heterogeneous distributed systems with random failures," *Journal of Parallel and Distributed Computing*, vol. 72, no. 10, pp. 1186–1194, 2012.

[50] R. Garg and A. K. Singh, "Fault tolerance in grid computing: state of the art and open issues," *International Journal of Computer Science & Engineering Survey (IJCSES)*, vol. 2, no. 1, pp. 88–97, 2011.

[51] M. T. Huda, H. W. Schmidt, and I. D. Peake, "An agent oriented proactive fault-tolerant framework for grid computing," in *e-Science and Grid Computing, 2005. First International Conference on*, pp. 8–pp, IEEE, 2005.

[52] M. Besta, M. Podstawski, L. Groner, E. Solomonik, and T. Hoefler, "To push or to pull: On reducing communication and synchronization in graph computations," in *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*, pp. 93–104, ACM, 2017.

[53] H. Hayashi, "Comparing repair-task-allocation strategies in mas," in *Proceedings of the 9th International Conference on Agents and Artificial Intelligence - Volume 1: ICAART,*, pp. 17–27, INSTICC, SciTePress, 2017.

[54] J. Botev, S. Rothkugel, and J. Klein, "Socio-inspired design approaches for self-adaptive and self-organizing collaborative systems," in *Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2015 IEEE International Conference on*, pp. 19–24, IEEE.

[55] S. Haider and N. R. Ansari, "Temperature based fault forecasting in computer clusters," in *Multitopic Conference (INMIC), 2012 15th International*, pp. 69–77, IEEE, 2012.

[56] S. Hwang and C. Kesselman, "A flexible framework for fault tolerance in the grid," *Journal of Grid Computing*, vol. 1, no. 3, pp. 251–272, 2003.

[57] A. Bansal, K. Ramohanarao, and A. Rao, "Distributed storage of replicated beliefs to facilitate recovery of distributed intelligent agents," *Intelligent Agents IV Agent Theories, Architectures, and Languages*, pp. 77–91, 1998.

[58] V. Dignum and F. Dignum, "Value-sensitive design of self-organisation," in *Self-Adaptive and Self-Organizing Systems (SASO), 2015 IEEE 9th International Conference on*, pp. 156–161, IEEE.

[59] Y. Wang and X. Liu, "Agent based dynamic recovery protocol in distributed databases," in *Proceedings of the Second international conference on Parallel and distributed computing*, pp. 274–280, IEEE Computer Society.

[60] E. Kuhn, R. Mordinyi, M. Lang, and A. Selimovic, "Towards zero-delay recovery of agents in production automation systems," in *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology-Volume 02*, pp. 307–310, IEEE Computer Society.

[61] D. Gelernter, "Generative communication in linda," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 7, no. 1, pp. 80–112, 1985.

[62] A. J. Alzahrani and A. A. Ghorbani, "Towards android malware detection using intelligent agents," in *Agent, Multi-Agent Systems and Robotics (ISAMSR), 2016 2nd International Symposium on*, pp. 1–8, IEEE.

[63] C. Derman, G. J. Lieberman, and S. M. Ross, "A sequential stochastic assignment problem," *Management Science*, vol. 18, no. 7, pp. 349–355, 1972.

[64] A. Rodríguez, J. Gómez, and A. Diaconescu, "Exploring complex networks with failure-prone agents," in *Mexican International Conference on Artificial Intelligence*, pp. 81–98, Springer, 2016.

[65] J. M. Simmons, *Optical network design and planning*. Springer, 2014.

[66] M. Grötschel and S. Orlowski, *Local and global restoration of node and link failures in telecommunication networks*. PhD thesis.

[67] S. S. Al-Majeed, C.-L. Hu, and D. Nagamalai, *Advances in Wireless, Mobile Networks and Applications: International Conferences, WiMoA 2011 and ICCSEA 2011, Dubai, United Arab Emirates, May 25-27, 2011. Proceedings*, vol. 154. Springer, 2011.

[68] Z. Rehena, R. Mukherjee, S. Roy, and N. Mukherjee, "Detection of node failure in wireless sensor networks," in *Applications and Innovations in Mobile Computing (AIMoC), 2014*, pp. 133–138, IEEE, 2014.

[69] H. Wang, X. Ding, C. Huang, and X. Wu, "Adaptive connectivity restoration from node failure (s) in wireless sensor networks," *Sensors*, vol. 16, no. 10, p. 1487, 2016.

[70] T. Frantz and K. Carley, "Agent-based modeling within a dynamic network," pp. 475–505, 01 2011.

[71] N. Meskaoui, D. Gaiti, and K. Kabalan, "Intelligent features within the j-sim simulation environment," in *Intelligence in Communication Systems*, pp. 143–150, Springer, 2004.

[72] M. Wooldridge, N. R. Jennings, and D. Kinny, "The gaia methodology for agent-oriented analysis and design," *Autonomous Agents and multi-agent systems*, vol. 3, no. 3, pp. 285–312, 2000.

[73] T. L. Frantz and K. M. Carley, *Agent-Based Modeling Within a Dynamic Network*, p. 475¢505. Cambridge University Press, 2008.

[74] C. M. Macal and M. J. North, "Agent-based modeling and simulation," in *Winter simulation conference*, pp. 86–98, Winter simulation conference, 2009.

[75] M. D. Rouleau, "Normsim: An agent-based model of norm change," in *Complex Systems (WCCS), 2015 Third World Conference on*, pp. 1–8, IEEE, 2015.

[76] C. Macal and M. North, "Introductory tutorial: Agent-based modeling and simulation," in *Proceedings of the 2014 Winter Simulation Conference*, pp. 6–20, IEEE Press.

[77] M. NORTH, T. HOWE, N. COLLIER, and J. VOS, "The repast simphony development environment," vol. Paper extracted from Proceedings of the Agent 2005 Conference on Generative Social Processes, Models, and Mechanisms, 2005.

[78] M. J. North and C. M. Macal, *Managing business complexity: discovering strategic solutions with agent-based modeling and simulation*. Oxford University Press, 2007.

[79] M. J. North, N. T. Collier, J. Ozik, E. R. Tatara, C. M. Macal, M. Bragen, and P. Sydelko, "Complex adaptive systems modeling with repast simphony," *Complex adaptive systems modeling*, vol. 1, no. 1, p. 3, 2013.

[80] M. Koster, "Reliable multi-agent system for a large scale distributed energy trading network," *Master's thesis, University of Groningen*, 2011.

[81] M. Amin, "Toward self-healing energy infrastructure systems," *IEEE Computer Applications in Power*, vol. 14, no. 1, pp. 20–28, 2001.

[82] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *TAAS*, vol. 4, pp. 14:1–14:42, 2009.

[83] T. Freeman, K. Keahey, I. Foster, A. Rana, B. Sotomoayor, and F. Würthwein, "Division of labor: Tools for growing and scaling grids," in *International Conference on Service-Oriented Computing*, pp. 40–51, Springer, 2006.

[84] M. Kim, J. Kim, I. Park, S. Lee, and S. Park, "Agent-based software analysis method in distributed environment," in *Fuzzy Systems Conference Proceedings, 1999. FUZZ-IEEE'99. 1999 IEEE International*, vol. 1, pp. 321–325, IEEE, 1999.

[85] A. Ricci, M. Piunti, and M. Viroli, "Environment programming in multi-agent systems: an artifact-based perspective," *Autonomous Agents and Multi-Agent Systems*, vol. 23, no. 2, pp. 158–192, 2011.

[86] L. Rastrigin, "The convergence of the random search method in the extremal control of a many parameter system," *Automaton & Remote Control*, vol. 24, pp. 1337–1342, 1963.

[87] M. Bastian, S. Heymann, M. Jacomy, *et al.*, "Gephi: an open source software for exploring and manipulating networks.," 2009.

[88] A.-L. Barabsi and E. Bonabeau, "Scale-free networks," *Scientific American*, vol. 288, no. 5, pp. 50–59, 2003.

[89] R. Albert, "Scale-free networks in cell biology," *Journal of cell science*, vol. 118, no. 21, pp. 4947–4957, 2005.

[90] B. Chaib-draa and F. Dignum, "Trends in agent communication language," *Computational intelligence*, vol. 18, no. 2, pp. 89–101, 2002.

[91] G. K. Saha, "Web ontology language (owl) and semantic web," *Ubiquity*, vol. 2007, no. September, p. 1, 2007.

[92] J. J. Carroll and G. Klyne, "Resource description framework ({RDF}): Concepts and abstract syntax," 2004.

[93] A. Das and A. De Sarkar, "On fault tolerance of resources in computational grids," *International Journal of Grid Computing & Applications*, vol. 3, no. 3, p. 1, 2012.

[94] S. Cammarata, D. McArthur, and R. Steeb, "Strategies of cooperation in distributed problem solving," in *Readings in Distributed Artificial Intelligence*, pp. 102–105, Elsevier, 1988.

[95] G. Anders, J. Botev, and M. Esch, "Third international workshop on self-adaptive and self-organising socio-technical systems (sasost 2015),"

[96] O. Kazık, "Role-based approaches to development of multi-agent systems: A survey," 2010.

[97] S. Savarimuthu, M. Purvis, M. Purvis, and B. T. R. Savarimuthu, "Gossip-based self-organising agent societies and the impact of false gossip," *Minds and Machines*, vol. 23, no. 4, pp. 419–441, 2013.

[98] S. Voulgaris *et al.*, *Epidemic-based self-organization in peer-to-peer systems*. PhD thesis, 2006.

[99] M. Dastani, V. Dignum, and F. Dignum, "Role-assignment in open agent societies," in *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pp. 489–496, ACM.

[100] K. P. Birman, "Reliable distributed systems," 2005.

[101] S. Dhakal, M. M. Hayat, J. E. Pezoa, C. T. Abdallah, J. D. Birdwell, and J. Chiasson, "Load balancing in the presence of random node failure and recovery," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pp. 10–pp, IEEE, 2006.

[102] J.-L. Guillaume, M. Latapy, and C. Magnien, "Comparison of failures and attacks on random and scale-free networks," in *International Conference on Principles of Distributed Systems*, pp. 186–196, Springer, 2004.

[103] C.-W. Chen, "Dual redundant server system for transmitting packets via linking line and method thereof," 2007.

[104] R. Kota, N. Gibbins, and N. R. Jennings, "Self-organising agent organisations," in *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pp. 797–804, International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[105] M. De Domenico, A. Solé-Ribalta, S. Gómez, and A. Arenas, "Navigability of interconnected networks under random failures," *Proceedings of the National Academy of Sciences*, vol. 111, no. 23, pp. 8351–8356, 2014.

[106] L. Bao and J. J. Garcia-Luna-Aceves, "Topology management in ad hoc networks," in *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, pp. 129–140, ACM, 2003.

[107] A. Bagdasaryan, "Systems theoretic techniques for modeling, control and decision support in complex dynamic systems," *Artificial Intelligence Resources in Control and Automation Engineering*, pp. 15–72, 2012.