Deploying Self-Organisation to Improve Task Execution in a Multi-Agent Systems

Asia Al-Karkhi
School of Computer Science and Electronic Engineering
University of Essex, UK
aasalk@essex.ac.uk

Maria Fasli
School of Computer Science and Electronic Engineering
University of Essex, UK
mfasli@essex.ac.uk

Abstract—This paper discusses how the performance of a network of agents can be improved using a self-organisation technique. The multi-agent network performance can be improved by organizing the agents in clusters. Furthermore, principles of self-organisation can be used to create agent organisations triggered when some of the agents have high load. Hence, busy agents within the network may decide to create an organisation to receive extra support from other less busy agents in order to execute more tasks. The paper presents a simulation based on Repast Simphony that has been used to develop the proposed model and describes a set of experiments showing the performance of the system with and without the self-organisation technique.

1. Introduction

Multi-agent systems (MAS) have emanated from distributed artificial intelligence. Agents are autonomous, proactive, reactive entities and can work together with other agents. A multi-agent system (MAS) is a loosely coupled network which involves a collection of software components that work together to solve complex questions beyond the individual capabilities or knowledge of each entity [1]. Multi-agent systems have been deployed within open environments to solve complex problems in a range of areas that individual agents would not be able to solve on their own. Agents may be holding different resources, have different skills and expertise, and can perform different tasks to maximise their resource utilisation and improve the overall system performance. In this manner, multi-agent frameworks can be seen as resembling a human society where every agent has its own particular objective and works with different agents to perform tasks and accomplish individual and collective goals [2].

A self-organising system is one where the system can dynamically change at whatever point the encompassing circumstances require with no outside mediation [3]. The fields that the self-organisation frameworks emerged from are physics, social groups, and analysis of social insects. The authors in [4] have encoded some of the principles/ behaviours of these fields inside their applications. They consolidated software engineering programming with a natural self-organisation process to accomplish a decentralised self-organised system. In their work, they have described the challenges that they faced with building such an application. For further details, see also [4]. Our work involves networks of autonomous agents with their own resources capable of executing tasks. A self-organisation is carried out by agents that are busy, i.e. who have taken on more than a single task or they have been receiving tasks one after the other. Hence, the self-organisation process can be triggered when the busy agents identify an issue with their workload and start to create groups or clusters of agents. In this way, the workload will be distributed over the newly created groups. In this paper, a self-organisation process has been used to show the positive effect of a local action performed by the individual agents to the overall network to increase the task execution in the network; hence, improve the overall network performance. Creating dynamic organizations of agents and applying roles for the organization members to improve their work as societies is the target for many researchers [5]. Using organizations has been demonstrated to provide reasonable solutions for many task allocation problems in distributed environments. These solutions could be used to minimise the resource allocation costs and decrease unnecessary communications among agents during their activities within the organization [6]. In addition, the tasks have been defined to have similar structure or different complex structure, further information is in [7, 8].

The rest of the paper is organised as follows. The next section discusses the related work in the literature. Section 3 gives a description of the network of agents, the customer agent and the task delegation protocol. Section 4 presents the organization creation process, the agents' roles in the created organizations and the tasks executed by the organizations. Section 5 introduces briefly the simulator that has been used in this paper followed by the experimental work with and without the self-organisation processes. Section 6 shows the effect of agents essentially going offline. Finally, the paper ends with the conclusions and avenues for future work.

2. Related Work

Agent-based and multi-agent systems have been widely used to solve complex problems in distributed environments where agents are independent and self-interested. Using an open multi-agent system with no predefined static design is more useful as the agents in a distributed environment can be designed by different people and can satisfy different goals. Agents in such systems however can suffer from unexpected failures hence to ensure robust performance, the ability of the system to self-organize is not only desirable but essential. A self-organised system is one that can be changed dynamically without any external intervention whenever the surrounding circumstances demand. An example of a self-organised system is ad-hoc networks that can freely discover the accessibility of each other [3]. Many researchers argue that in a heterogeneous multi-agent system environment, triggering an agent organisation process to reduce the complexity impact of a large number of agents demands formal theories to design the organisation structure as well as methods for the agents' interaction, see [9]. In [10], the authors proposed a self-organised resource allocation scheme based on Decentralised Distributed Virtual Environments (DDVEs). The scheme is functioning independently from the implicit topology of P2P network. The authors presented a scheme based on the gossip protocol to identify the users' critical zone. They take the advantage of the heterogeneous peers to make the use of the potential ones which are the nodes in connection to reliability and bandwidth to smooth data distribution by breaking off the virtual peers. In our work the algorithm used by the Head to create an organisation is based on the gossip protocol to search for other potential agents in term of their state busy/ not busy to join the Head's organization. In [11], when the agents are working in a big and complex system environment, giving a role to the agents would be a better solution for them. A role adds the ability to the agents to overcome issues like event or process interruption and at the same time, gain the opportunity to maximise their interest. Other researchers use agent organisations in a distributed environment to enhance the performance of such systems. In [12], they implemented a simulator model for exploring recommendations inside the connection of a network 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 their similarity and demonstrated the ability to self-organise an overlay system. Their work shed some light on the agents' capability for decision making and agents' knowledge about connected peers which is gained during the network evolution process. In our work, a self-organisation technique has been utilized to demonstrate the beneficial outcome of a neighbourhood activity performed by the individual agents to the overall network to expand the task execution; henceforth, enhance the system performance. In addition, we have created rules to trigger the creation of agent organisations. These rules provide roles for the agents in the self-organization process which hence provide a robust enhancement schemes to the agents' network. In [13], they proposed a theoretical method to optimize the agents network. Their idea is to minimising the number of agents who are chosen to be in charge in organisations and increasing their connections to acquire network coverage. They have compared and checked different network types. However, their model is tested only on a small number of organizations.

3. THE NETWORK MODEL AND CUSTOMER AGENT

A scale free network has been implemented based on the Barabasi-Albert (BA) model algorithm, as explained in [14]. The designed network grows gradually to be very large to simulate networks such as the Internet, which is a scale free network, see igure (1). There are two types of agents: the customer agent and the task execution agents (simply referred to as agents from now on). The agents in the created network have heterogeneous resources and hence they have the resources to execute different types of tasks that are issued from the customer agent side. Each agent can sustain a maximum number of connections, which may be a different number from agent to agent. The implemented experiment, as will be explained later, controls the number of connections for each agent, and in this case, each agent has a number of connections with a saturation condition (M). In each cycle, the customer agent sends N messages (tasks) to a random set of agents with this format (*Task_ID*, requested resources, (TTL), deadline, required accuracy). Each task is sent to just one random agent. Where:

Task_ID: the unique identifier used to identify each task sent by the customer.

Requested Resources: the resources that are requested by the customer to execute the customer task. This has been modelled as a tuple <r1, r2, r3>, so it is a vector of three randomly selected values within a range [0-4].

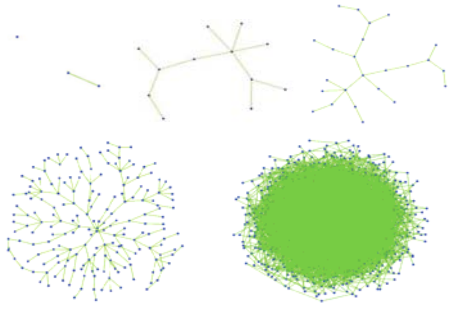

Figure 1-The implemented network of 1000 agents

TTL: Time To Live (TTL) for each task, which means the maximum number of hops that a message can make within the network and which is tracked in the search algorithms.

Deadline: in real world systems, a typical request for task execution has a deadline by which the task execution result is required by the requesting customer agent. When a customer sends a task, it wants it to be finished within this deadline. To simulate the deadline in our model, the customer sends a deadline value with each task issued to the network of agents. The deadline represents a specific value for the execution period that the randomly selected agent in the network should use for the execution of a task. It also indicates the number of cycles as to how long the customer is willing to wait for the execution result after the agent in the network has accepted the task. So, within this value, the response from the agent to the customer should be guaranteed.

Required Accuracy threshold: the matching value of the accuracy for the requested resources. The customer sends each task with a specific required accuracy value. Since, the customer resources are between [0-4] as mentioned above, we use Manhattan distance equation which produces accuracy values within (0 to 12) so it is 13 Accuracy values. When a task is received by an agent, it will use equation (1) to check its ability to execute that task in terms of resources. If the required accuracy is met, the agent will then execute the task.

$$ManhattanDistance(CustomerR, AgentnetworkR) = \sum_{i=0}^{3} |Customer(r)i - Agent(r)i| \quad \ldots\ldots\ldots\ (1)$$

Where:

Customer R= requested resources from the customer side.

$AgentnetworkR$= agent matching resource.

As an example, if the customer requests the resources <1,1,1> with a required accuracy specified to be zero, and the recipient agent resources are <1,1,1>, then an exact match has occurred when applying equation (1). In another example, if the customer asks for <4,4,4> with required accuracy 6 and the recipient agent resources are <2,2,2>, in this case this agent can execute the task. Otherwise, if the agent does not satisfy the required accuracy, then the task is going to traverse the network until it reaches an agent that can satisfy its required accuracy value as long as the TTL has not been reached yet. If the TTL has been reached, the task will be considered as having failed.

3.1. Task Delegation Protocol

After the agents are inserted into the environment, they start receiving tasks from the customer agent and sending tasks to each other when they cannot perform them on their own. Within this dynamic system, the various agents may become unavailable or very busy. Where, busy agent means an active agent that currently executing an accepted task.
See algorithm (1), which explains the task delegation algorithm for the tasks issued by the customer agent to the network of agents. The agent may receive tasks but cannot execute them for different reasons, such as the deadline for the task is not sufficient for the agent or it does not have the required accuracy, etc. To explain how the *task delegation protocol* process works, suppose an agent receives a task. If the agent cannot execute the task, it acts as an initiator and starts searching the network by sending the task messages which have been received in each cycle to their direct contacts using the search algorithm. The messages then will traverse the network depending on the TTL value and the network size. Also, during the search, and if there are no more recipient agents, the message will be with the last agent who received it. The search procedure fails when the TTL is reached or when the message is returned to the initiator; this means that it is a failed task.

```
Algorithm (1): Task Delegation Protocol.
Start
 Note: Task ID. Each task sent from the customer agent to
random agents in each cycle has its unique ID.
Cycle i: agent $a_x$ receives Task ID; if it cannot execute it, and
TTL> 0, then $a_x$ sends a message with Task ID to contact $a_y$ and
updates TTL, new_TTL=TTL-1. Cycle i+1: agent $a_y$ receives a
delegated task; if it can accept, it, sends a "task accepted, Task
ID" message to the customer. If it cannot accept, but TTL>0, a
message is forwarded to contact $a_z$ and new_TTL=TTL-1. If
TTL=0, sends message to customer = "task failed, Task ID"
End.
```

3.2. Individual Agent Behaviour

The agents are autonomous and self-interested with the desire to make the maximum benefit for themselves. The agents in the network have been encoded with rules that enable them to create the network. So, each agent would have partial knowledge about the network members by creating a contact list containing the contact details of all the agents it has a direct connection to. Furthermore, each agent in the network has a queue called accepted task queue contains the accepted tasks to be executed soon. Hence, when an agent in the network accepts task, it means that the agent has met all the crucial requirements for this task, and then its status will become busy and will remain at this status until task been executed will change to not busy. If its queue has tasks, then the agent will become busy to execute the tasks in its queue. After that the agent will change its status to not busy.

4. The Organization Creation Process

The purpose of creating an organization as a virtual layer above the existing agent network is to increase the number of executed tasks and minimize the time required for accepting and executing tasks. In other words, finding the exact agent that can satisfy the requested task requirement with a single hop or with as less hops as possible. To simplify matters, the explanation below will only be about the process of a single organisation. However, the system may contain more than one organisation. The emergence of agent organisations, will depend on a triggering condition resulting from the busiest agent's action. The busiest agent is a term given to a busy agent that has satisfied extra conditions which are: currently executing task, still receiving tasks from the customer and already having tasks in its accepted tasks queue. Hence, the accepted tasks in its queue may requires at least M cycles to be fully executed, where M is an overall value related to the deadline of each task, and it could be estimated within trial and error experiments. This busiest agent can decide to create an organisation and be the Head of that organization to receive more help. Hence, the Head can send a multicast message to other agents in the network to invite them to join its organisation based on the gossip algorithm to propagate the message. In algorithm (2), we used a similar method as in [15], where the author in chapter 7 has clarified the essential properties of gossip-based information dissemination and showed how the gossip approach can be utilized not only in human society but also in other domains such as networks. In our implemented experiments, inside the network environment, if a not busy agent accepts the invitation message then it will send back an acceptance message "accept to join" to the Head. If the message reaches a busy agent, then the busy agent will act as a traverser for the Head's message to reach other parts of the network, see algorithm (2). A database record will be added to the Head's database for each agent who joins the organisation. Each agent will also store the required information for any organisation it has joined. The construction of the organizations inside the agents' network environment is subject to triggering conditions. When the triggering conditions are met, the Head starts to send messages asking other agents to join its organization; any agent joining the organization must satisfy a set of norms (obligations). These norms are accepting to execute tasks or putting them in the accepted tasks queue if the required accuracy is matched and the agent is able to execute the tasks within the allocated deadline. 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 depends on its setting. However, a member of an organisation can execute additional tasks beyond those delegated by the Heads of the organisations for which it is a member of.

4.1. The Description of the Created Organizations

There are two variations in the creation of organisations which we will term Version0 and Version1. In Version0, an agent can join the created organizations depending on two conditions. The first one is a setting parameter which is a random value to specify the number of organizations the agents can join within a range from 1 to N. The second condition is that the agents can have the opportunity to decide which organization to work with, based on the resources matching process to create homogeneous organizations, see equation 1 above.

While, the creation of an organization in Version1, any agent can be a member of an organization if it is not currently a busy agent and its maximum allowed number of organizations to join has not been met yet. Subsequently, there is no restriction for the required accuracy to be met as in Version0. The Head has the following database information about its organization.

Head_database = {Head_ID, Max_no_member, Member_ID, Member resources, Required_Accuracy}

When an agent accepts to join an organization, it will create an entry in its database. Our work is focusing on agents that can have two roles within the self-organization process: that of the Head which can start to organize its own organisation, and that of the member which needs to have the reliability to act as a service provider agent. However, when a member within an organization receives a task from the customer and it has no ability to execute it, then the member will send the task to other organizations that the agent has joined.

Algorithm (2): "Gossip method to initiate an agent organisation based on the busiest agent(Head)"

The Head agent searches for members for its organisation using the following steps:

```
1. Transmitting to [N] random targets (agents),
   choosing [N] from the local membership contact list.
2. If [N] are non-busy agents, then they will be
   infected with the gossip message and they have the
   option to join or not to the created organisation.
3. Otherwise, the busy agents will be used only to
   transmit the message to their random peers.
4. The receiving agent in the last period will broadcast
   the gossip message to its randomly selected peers.
5. End
```

4.2. Task Execution by Organizations

Using Version0 or Version1 the received task by the Head follows the following procedure.

- The task will be executed if and only if the Head is not busy and can accept the task depending on the required accuracy matching criteria of the Manhattan distance and task deadline value.
- The task will be queued in the Head's accepted tasks queue. This occurs when the task matches the required accuracy of the Head's resources and the Head can execute the task within the pre-specified deadline, but the Head is currently busy executing other tasks.
- Otherwise, the Head will send a message to ask its Members whether they are able to accept and execute the task or not.

5. Experimental Work

5.1. Repast Simphony Simulator for Agent-Based Models

In this part, we give a brief description of the Repast Simphony simulator where the agent model has been developed to study the self-organisation process. So, a set of experiments has been designed and executed utilising Repast Simphony in Eclipse IDE using Java programming. Repast Simphony is an open source Agent-Based Simulator (ABS), which is a desktop developing environment for Agent-Based Models (ABM). An ABM can be used to explore issues in heterogeneous environments and emergent systems, for more details we refer the reader to [16-18]. There are many versions of Repast modelling toolkits such as, Repast under java, Repast python, Repast for Microsoft.NET and Repast Simphony. Repast Simphony allows the developer to control the number of agents as well as control the agents' actions and behaviour by using different scheduling methods, which can be either continuous or discrete.

5.2. Network Performance With and Without Self-Organisation

Several experiments have been implemented to demonstrate the self-organization process. We aim mainly to see the effectiveness of adding organizations on the performance of the proposed system. The experiments have been implemented on different agent network sizes (300, 500, 1000) with maximum connection for each agent up to N=10 and time to live for the task message being TTL=10. The simulator runs for 3000 cycles. The results have been collected and used to produce the graphs by repeating the run for 10 times to ensure the robustness of the results. The experiments will also show the effect of changing the task distribution on the performance of the system with different network sizes. Table (1) shows the values for tasks issued from the customer side using the normal distribution to simulate the real-world problem when variable number of tasks requests have been issued. Equation 2 has been used to produce Figures (2-14) except for Figure (5) which has been produced using equation 3. The Average Number of Successfully Executed Tasks Ratio (ASETR) is the average number of successfully executed tasks with their required accuracy divided by the total number of tasks issued per required accuracy.

ASETR$=\frac{STA(i)}{TTR(i)} \times 100$ .... (2)

Where: $i = 0 - N,\ where\ N \leq 12$

$STA(i)$: successfully executed tasks per required accuracy.

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

Table -1- Task Distribution values

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

Figure (2) shows the performance of the organization Version0 and Version1 compared to the model without organization. In general, the number of executed tasks with organizations show better values compared to without organization for all the required accuracy from (0-12). In organization Version0, each created organization has the same required accuracy specified by its Head, hence the number of executed tasks has shown slight improvement in relation to the agents' network without organization. While in organization Version1, each created organization is a cluster of agents which can satisfy tasks with different required accuracy. Hence, the results from this version shows more tasks being executed than in Version0 for all ranges of the required accuracies.

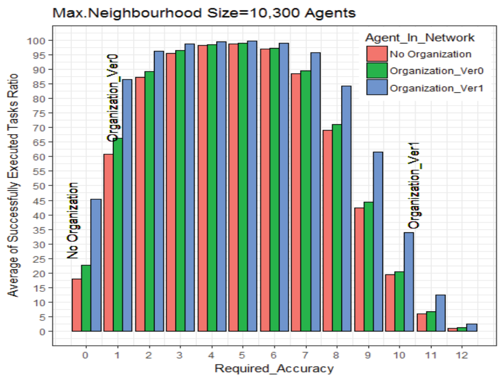

Figure 2- The ASETR for 300 Agents

By increasing the number of agents up to 500 agents as shown in Figure (3) and keeping the same number of tasks, the system shows that the organization Version0 and Version1 gave higher number of executed tasks than the system without organizations especially for Version1. Now, it is worth mentioning that when the size of the network has increased to 1000 agents, nearly all the tasks have been executed with the required accuracies from (3 to 7) as it represents the highest demanded accuracies from the customer side. And for the rest of the accuracies, the system shows less executed tasks because less demand had originally been issued. Since the aim of the work is to create organizations in order to address the inflation problem of the heavy requests from the customer(s) in a system that does not have a very high number of agents, increasing the number of agents slowed down the impact and the effectiveness of the created organizations within the system with the same demand from the customer side, and Figure (4) shows nearly similar performance of the system with the existence of organizations for both versions and without organizations. To compute the number of successfully executed tasks within the simulation cycles the following formula has been used.

ANSET $= \frac{1}{R}\sum_{1}^{R}\sum_{j=0}^{2}\left|x_j^r - x_j^m\right|$ .... (3)

Where:

ANSET: is the Average Number of Successfully Executed Tasks within simulation cycle.

R: Number of runs = 10.

Figure (5) shows the performance of the system with the number of successfully executed tasks within cycles by applying equation 3. The organization Version1 shows a higher efficiency in executing tasks than the other two models. This is due to the variety in the agents' resources in each organization, so when any agent becomes unavailable there are other agents that can accept and execute the tasks.

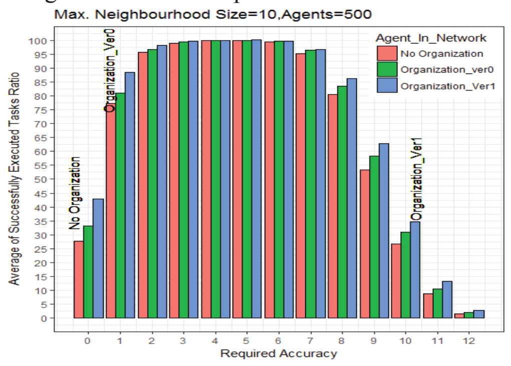

Figure 3-The ASETR for 500 Agents

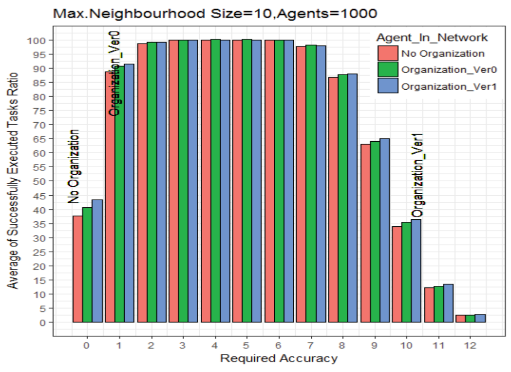

Figure 4-The ASETR for 1000 Agents.

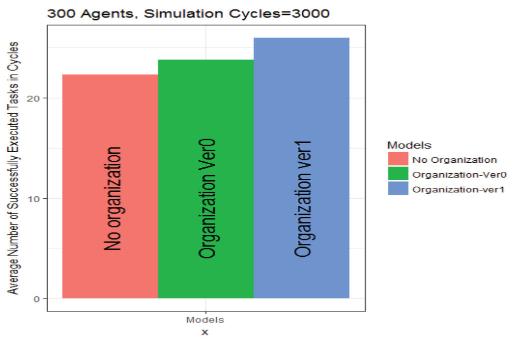

Figure 5-The ANSET for 300 Agents

6. Dealing with Disruption in Agent Service Provision

This part of the work is examining how the system would be able to deal with the disruption that would occur by agents in the network losing their connectivity, i.e. being offline. To model this, agents are equipped with a parameter that enables them to be switched on/off for a period of time, in essence creating the impression in the system that they are offline and unable to execute tasks or respond to messages. This is applied to both Version1 and Version0. The on/off parameter has been applied using probability values of 0.9, 0.5 and 0.2.

6.1. Experimental Work

This experiment involves the creation of the organizations using Version0 and Version1 as well as adding the offline features to the agents with probability values of 0.9, 0.5 and 0.2 and with changing the agents' network size being 300, 500 and 1000 accordingly. Figure (6) shows that Version1 has improved the system performance and effectively deals with the offline event due to having a variety of agent organizations that can satisfy high percentage of requested tasks. In contrast with Version0, which shows a negative effect in the network because its organization structures are based on specific required accuracies that could not easily overcome the issue of agents being offline which leads to more failed tasks than even the system without organizations.

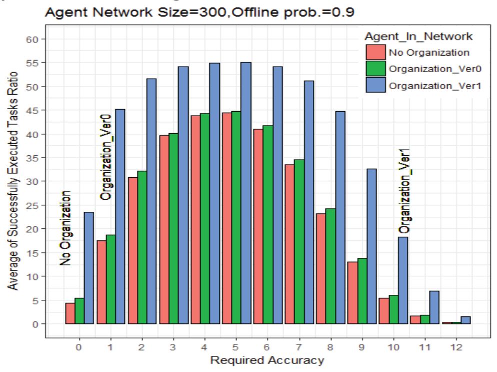

Figure 6-The ASETR with offline prob.0.9-300Agents.

By increasing the number of agents to 500, more tasks have been executed especially with organization Version1 as shown in Figure (7). Then, when increasing the network size to 1000 agents, the system shows moderate reaction for the models as shown in Figure (8) which can be understood as the network size needing more requested tasks to benefit from the facility provided by the created organizations during the event of agents being offline. Figures (9-11) illustrate the runs of the system where the offline probability has been set to 0.5 with different agent network sizes (300,500,1000) respectively. These Figures show nearly the same trends as in the previous Figures (6-8). While when the offline probability is 0.2, in Version1, generally, the performance of the system has increased and it can effectively cope with the case of agents being offline due to having agent organizations that can satisfy higher percentages of requested tasks. In contrast to Version0, and due to its construction criteria, it shows a negative effect in the network. It could not overcome agents being offline, and that leads to more failed tasks than even the system without organization, see Figures (12-14). To show the number of created organizations with different network size agents (300,500,1000), the network size with 300 agents has been chosen as a representation case as shown in Figure (15). Overall for the organization Version1 without agents going offline (normal case, green colour), it is noticeable that the average number of organizations rose dramatically during the first chunk of cycles, then it levelled off at a certain value where there is no demand to create more organizations for the remaining simulation time cycles. while with the presence of agent offline with different probability values (0.9,0.5,0.2), the average number of created organizations increased steadily with time and nearly follow the same trends for all the simulation time cycles, thus the average number of created organizations decreased as the probability of offline increased. This behaviour is due to less agents being able to satisfy the triggering condition with each time cycle to formulate organizations.

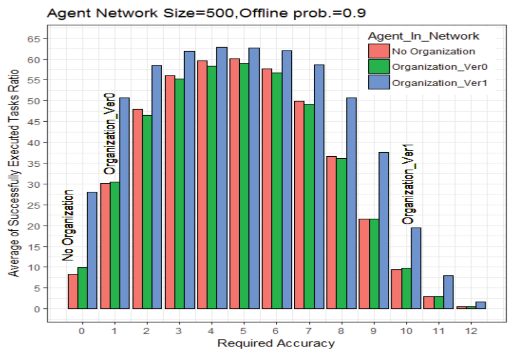

Figure 7-The ASETR with offline prob.0.9-500Agents.

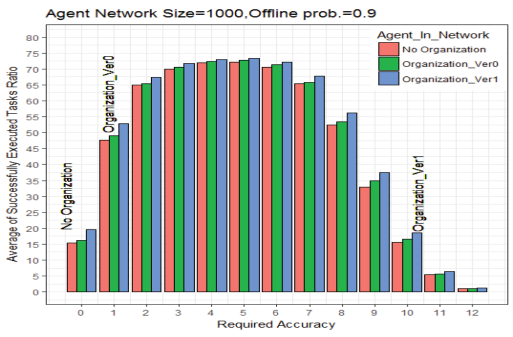

Figure 8-The ASETR with offline prob.0.9-1000Agents

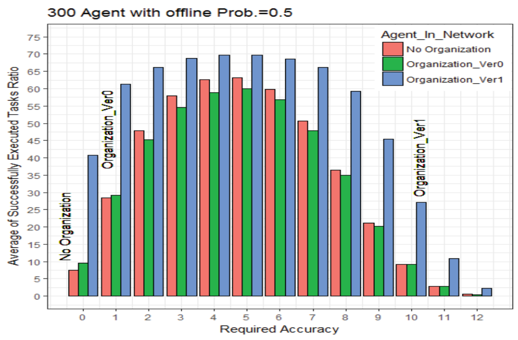

Figure 9-The ASETR with offline prob. 0.5-300Agents

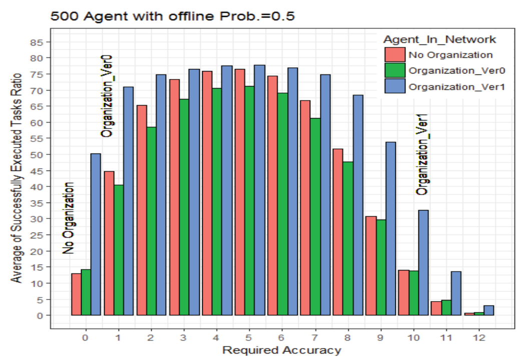

Figure 10-The ASETR with offline prob .0.5-500Agents

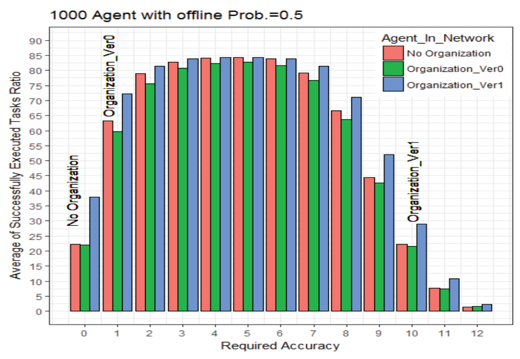

Figure 11-The ASETR with offline prob. 0.5-1000Agents

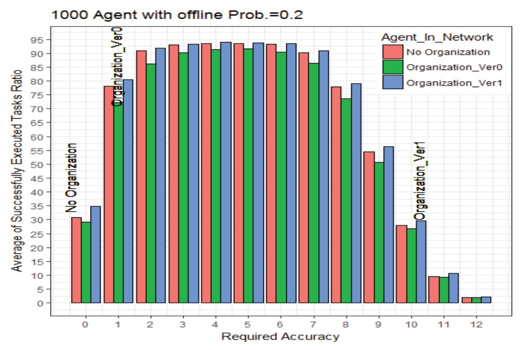

Figure 12-The ASETR with offline prob. 0.2-1000 Agents.

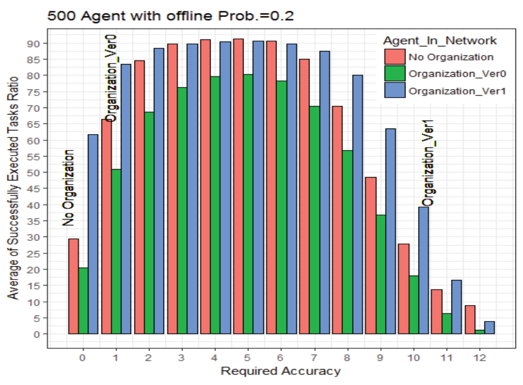

Figure 13-The ASETR with offline prob.-0.2-500Agents

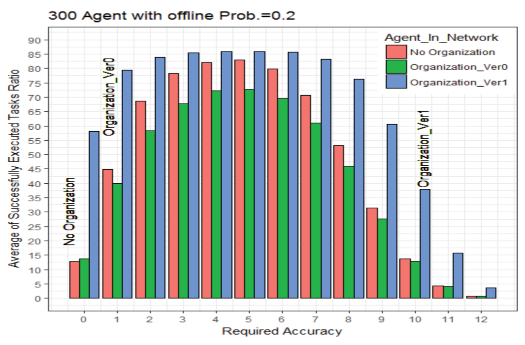

Figure 14-The ASETR with offline prob. 0.2-300Agents

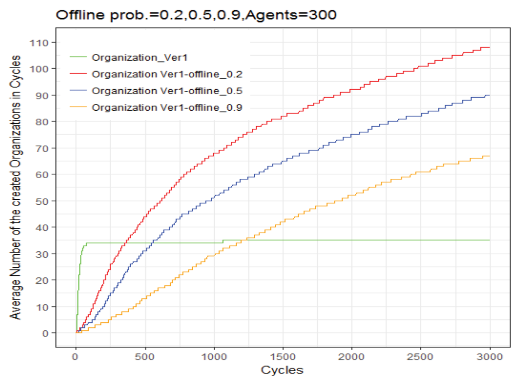

Figure 15- number of created organizations for 300 Agent with different probability

7. Conclusions and Future Work

This paper has not only shown the system performance in a distributed environment when agents are being delegated and undertake tasks and are busy, but also has shown the performance when agents within the system become unavailable or drop out within a range of probability values.
The implemented models show that the proposed protocol and ideas have delivered a model that could cope with the traffic in distributed domains.
The first contribution of our work is in demonstrating the creation of organizations of multiple agents to improve the performance of a network of agents within which they may unpredictably drop out or become unavailable; the proposed protocols enable the system to avoid such disruption that may occur when agents lose their connectivity. The second contribution is that our work considers agents that have heterogeneous resources in the created organizations. This is in contrast with other works that have considered choosing agents with similar ability to create homogeneous organizations. As part of the experimental work, we show that the average number of executed tasks in Version1 has been improved in comparison to Version0 and the other related work, see [19]. Hence, the developed organisation methods and protocols that have been deployed in this paper to create the organization based on the multi-cast gossip protocol can be used as a solution for task recovery using organizations of heterogeneous agents in distributed domains. The third contribution is in providing the means for the distributed system to create organizations that can emerge depending on

system demands. In our work, the emergence of organisations is used to manage the problem of agents being busy for long periods of time and this forms the triggering condition for the organisation creation process. Agents can join many organizations so that they can interact with others within these to satisfy the request tasks. The forth contribution is that the roles are a result of the triggering condition. The busiest agent is the Head of the organization, so it starts to send a multicast message to other agents in the network to join its organization and provide services. Hence, the system contains two roles one is the Head and the other is the Members and service providers. In other related work as in [20], the agents are able to join an organization at specific time of their life time and can change their behaviour to join an organization to match the requested role. As part of our work we have implemented a set of experiments to compare the task delegation protocol described in section (3.1) against the standard random search algorithm. Although we do not include the results of these experiments here due to lack of space, we note that Algorithm 1 shows higher number of executed tasks than the random search. This is because the technique of random search is based on randomly selecting neighbours for sending the customer message during the delegation process, while the delegation protocol (Algorithm 1) is based on heuristic choice for the next agent to navigate the message, which decreases the possibility of failing tasks compared to random search.

Our current and future work is to improve the performance of the agents within the organizations. Hence, when agents are offline or not responding within an organisation a new protocol has been designed to tackle the problem of losing tasks within the network.

8. References

1. Bezek, A. and M. Gams, Comparing a traditional and a multi-agent load-balancing system. Computing and Informatics, 2012. **25**(1): p. 17-42.
2. Metawei, M.A., et al., Load balancing in distributed multi-agent computing systems. Ain Shams Engineering Journal, 2012. **3**(3): p. 237-249.
3. De Wolf, T. and T. Holvoet. Emergence versus self-organisation: Different concepts but promising when combined. in International Workshop on Engineering Self-Organising Applications. 2004. Springer.
4. Sudeikat, J., et al., Systematically engineering self-organizing systems: The SodekoVS approach. Electronic Communications of the EASST, 2009. **17**.
5. Dignum, V., et al. 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. 2002.
6. Corkill, D.D., D. Garant, and V.R. Lesser. Exploring the effectiveness of agent organizations. in International Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems. 2015. Springer.
7. Miyashita, Y., M. Hayano, and T. Sugawara. Self-organizational reciprocal agents for conflict avoidance in allocation problems. in Self-Adaptive and Self-Organizing Systems (SASO), 2015 IEEE 9th International Conference on. 2015. IEEE.
8. Miyashita, Y., M. Hayano, and T. Sugawara. Formation of Association Structures Based on Reciprocity and Their Performance in Allocation Problems. in International Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems. 2015. Springer.
9. Dignum, V., The role of organization in agent systems. Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models, 2009: p. 1-16.
10. Botev, J. 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. 2010. IEEE.
11. Fasli, M. On commitments, roles, and obligations. in International Workshop of Central and Eastern Europe on Multi-Agent Systems. 2001. Springer.
12. Neville, B., M. Fasli, and J. Pitt, Utilising social recommendation for decision-making in distributed multi-agent systems. Expert Systems with Applications, 2015. **42**(6): p. 2884-2906.
13. Zbieg, A., D. Batorski, and B. Żak, How to Select Change Agents in Organizations? A Comparison of the Classical and Network Approaches. Problemy Zarzadzania, 2016. **14**.
14. Barabási, A.-L. and E. Bonabeau, Scale-free networks. Scientific American, 2003. **288**(5): p. 50-59.
15. Serugendo, G.D.M., M.-P. Gleizes, and A. Karageorgos, Self-organising software: From natural to artificial adaptation. 2011: Springer Science & Business Media.
16. NORTH, M., et al., THE REPAST SIMPHONY DEVELOPMENT ENVIRONMENT. 2005. Paper extracted from Proceedings of the Agent 2005 Conference on Generative Social Processes, Models, and Mechanisms.
17. North, M.J. and C.M. Macal, Managing business complexity: discovering strategic solutions with agent-based modeling and simulation. 2007: Oxford University Press.
18. Macal, C. and M. North. Introductory tutorial: Agent-based modeling and simulation. in Proceedings of the 2014 Winter Simulation Conference. 2014. IEEE Press.
19. Al-Asfoor, M.J.J., Resource discovery in self-organising distributed systems. 2014, University of Essex.
20. Dastani, M., 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. 2003. ACM.