Disruption Recovery within Agent Organisations in Distributed Systems

Asia Al-karkhi and Maria Fasli
School of Computer Science and Electronic Engineering, Essex University, Wivenhoe Park, Colchester CO4 3SQ, U.K.

Keywords: Disruption, Multi-agent System, Organisation, Head, Henchman Recovery Protocol, Service Provider.

Abstract: One of the challenging problems in distributed systems is dealing with agent failure. In this paper, we present an approach for task recovery in a distributed system where the agents self-organise themselves in organisations in order to execute tasks more efficiently. However, within this setting, unpredictable events can happen and agents can fail leading to task failure and weaker overall system performance. We present the Henchman recovery protocol which enables the agents within an organisation to maintain task execution and recover tasks in the event of agent failure. We show how the protocol helps to maintain the efficiency of the created organisations through a series of experiments in a simulated distributed task execution system which has been implemented in Repast Simphony. The experimental results demonstrate the robustness of the proposed solution in a number of settings.

1 INTRODUCTION

In distributed systems, it is very often the case that agents may not be able to carry out tasks because it may be busy or not active, hence delegation is one of the available options. Commonly, no agent can be designed with full knowledge about other agents in the vicinity of a network of agents. In complex agent systems, agents may form groups, coalitions or organisations under certain conditions in order to improve the execution of tasks or the utilisation of resources or improve the agent connections, (Corkill et al., 2015; Horling and Lesser, 2004; Dignum, 2009).

This work is mainly targeting data centres in the cloud to supply a more efficient service level to the customers' requests by providing both a theoretical framework and a simulated model. In our proposed environment we will recruit a multi-agent to encapsulate heterogeneous types of resources to simulate data centres that can be accessed on demand by many customers for any amount of time. Therefore, all the explanation in this paper is to demonstrate how the agents recruitment process will be carried out. Furthermore, to simulate the customer side, we have created a customer agent that asks for services by sending customers' tasks requests to the agents network using web connections.

However, one of the main challenges in open and distributed systems is the occurrence of unpredictable events which can affect the individual agent performance and in consequence the overall system performance. For instance, agents can fail and this may mean that any action(s) or task(s) that an agent has taken on would also fail to be executed. Agent failure within an organisation also creates problems as agents will fail to execute tasks and other agents may still be delegating tasks to them if they are unaware that they are no longer in operation. Hence, appropriate mechanisms are needed to handle agent failure within organisations so that their functionality and effectiveness can be maintained.

The main emphasis of this paper is on devising mechanisms that can be utilized in the context of organisations within distributed systems that would make the organisation of autonomous agents more efficient in handling failures and hence maintain their efficiency and effectiveness. The specific scenario that we are exploring is based on a scaled free network (Barabsi and Bonabeau, 2003) which is a common network topology, often found in social networks, and infrastructure networks (e.g. electricity grids, communication networks and the Internet. In the implemented network model, the agents may decide to create organisations to improve the number of executed tasks and increase their utility and make better utilisation of their resources. Hence, we show that the agent organisations can be more resilient to unexpected disruption that could affect the system performance. We

Al-karkhi, A. and Fasli, M.
Disruption Recovery within Agent Organisations in Distributed Systems.
DOI: 10.5220/0006574203710378
In *Proceedings of the 10th International Conference on Agents and Artificial Intelligence (ICAART 2018) - Volume 2*, pages 371-378
ISBN: 978-989-758-275-2

present a protocol for recovering from agent failure, called "Henchman Recovery Protocol (HRP)", whose task is to maintain the functionality of the created organisations.
The rest of the paper is organised as follows. Section 2 discusses the related work in the literature. Section 3 presents the system scenario which includes descriptions of the organisations, Members and customer agent with task message formats. Section 4 presents the HRP, and also introduces the protocol that is used by each Henchman inside the created organisation to monitor the Head. Sections 5 presents the experimental work. Finally, the paper ends in section 6 with the conclusion and proposals for future work.

2 RELATED WORK

The main aim for designing peer to peer networks is to apply fair and sensible allocation of the spare resources using different resource allocation methods. However, the existence of disruption and other obstacles that are sometimes associated with data transmission such as machine being offline or high progression incited by user activities, may lead to minimise the correct utilisation of the available resources (Botev et al., 2015).
In grid distributed systems, wireless networks and cloud computing, failure events such as improper system service or physical failure is still under research. Hence, this work is overlapped with grid networks and cloud data centres due to the great similarity between these systems in term of concepts and heterogeneous types of resources as have been explained in (Foster et al., 2008).
Researchers (Tesauro et al., 2004) have emphasized that distributed systems should provide self-healing, self-configuration and self-protection. In their proposed solution, they provided a software solution called "Unity" that managed a data centre using an autonomous multi-agent system. They have used the agents to self-heal and self-organise the computational resources by creating 17 clusters of self-management agents in the data centre, so many customers can access the resources at the same time. Their work has been implemented only on a small size data centre and to extend their work for industry purposes, a bigger size of data centre is essential. In our created models the agent's network can create a second layer (organisations) depending on the size of the available network. (Gutierrez-Garcia and Sim, 2010), also focus on employing agents to improve the service level in clouds. They have suggested customer agents, broker agents, service provider agents and resource agents. These agents are using self-organisation technique to provide composition services from the cloud to the customers and support achieving a service level agreement. However, in their work, agents have been implemented without decision making capability, while in our work, adding such capability to the agents has enhanced the service level provided to the customers.
Multi-agent systems have been borrowing recovery protocols from other domains. For example, client server, world wide web, peer to peer network and mobile ad-hoc network. Disruption can happen when the resources are limited with limited access to certain type of resources or when there is a more likely indirect access to the resources.
Rollback recovery protocols have been recruited in different distributed environments, as in (Elnozahy et al., 2002). The authors have presented a survey to distinguish between different types of rollback recovery protocols and compared their performance. The first one is the checkpoint based protocol, which is based on choosing a checkpoint to restore the system to that point. The second one is the log based protocol, which is a combination between the checkpoint protocol and log in information protocol. These protocols deal with nodes in a network as groups of processes that communicate between each other. These interactive processes are accessing a storage appliance periodically to save recovery information which could have at least a checkpoint state for those processes during passing messages to each other. They could then be used when they would return to active after processing the disruption. However, the processes could be working on old information after returning to their original state, something which might be no longer required by the system.
(Miyashita et al., 2015a; Miyashita et al., 2015b) have claimed their work is to solve resource conflict. We observe that their work is based on the Team Formation (TF) game to create teamwork as a solution. Their system will not produce teams unless the system is very busy or there is a demand to create them. However, the TF game is not able to work with a large numbers of agents, i.e. a large network. Hence, they have not indicated the maximum network size and the number of created organisations. A team formalization process depends only on the reciprocal agents, i.e. agents which have previous knowledge about each other during the network construction time. In our work, the experimental work shows the maximum number of agents could be 5000. In addition, the busy agents are making use of other non-busy agents in the organisation creation process to execute more tasks.

3 MODEL SCENARIO

The system consists of a set of agents $A=\{a_1, a_2, ..., a_m\}$; these agents will enter the environment one after another. The number of connections between the agents will be created within a specific limit X, i.e. each agent should maintain its connections to be no more than the value of X to avoid creating a centralized network which is quite similar to the scale free network (Barabsi and Bonabeau, 2003). The network is then constructed by exchanging messages between the agents, these messages consist of $(Agent_{ID}, Resources' information)$ to be sent over each connection. An agent's connections are not fixed; later on, the agents will change their connections for better performance and utilisation. Consequently, the created network is a heterogeneous environment, i.e. agents have different types of resources to accomplish as many types of tasks as possible. During the connection process, each agent will share its contact details with its randomly selected neighbour (s), So that each agent will have known contacts in its contact list.
In addition, to simulate the semantic type of resources which already exists in the internet, the resources have been represented as a vector of integer values, where agent resources are $AR =<$ r_1, r_2, r_3 $>$, $r\_i=(0-4)$. To this end we have explained the created network of agents.

Within the simulation model there is a customer agent which is an entity that is in charge of sending customers' tasks to the network of agents. Each customer will send tasks to the network through each cycle of the simulation time. The tasks will be sent as a message called customer message CM which contains the following parameters: $CM = \{CID, TID, RV, TTL, RA, TD\}$, where:

- CID: The Customer ID is a unique identifier which is used to identify customers and enable communication between them and service providers regarding the status of tasks (in task queue, executed, or failed).
- TID: Task ID is a unique identification number given to each task, where each customer can send a different number of tasks in each cycle.
- RV: Resource Vector represents a sequence of simulated resources, $RV = < r1, r2, r3 >$, which may be different from task to task.
- TTL: Time to Live is number of hops for the customer message to traverse through the network of agents.
- RA: Required Accuracy represents the required accuracy for matching the customer resources with the agent's resources. Its accuracy values between $(0-12)$ are pre-agreed between the customer and the agents.
- TD: The Task Deadline represents the deadline by which the customer would expect to have the result of the task execution back.

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 in the network if it meets certain conditions that are checked by each agent. The first such condition is a matching process between the customer resource vector and the receiving agent resources using the well-known Manhattan Distance. The resulting matching value should be met with the task required accuracy, which is a specific value between $(0-12)$.
For example, if a customer RV is $<0, 0, 0>$ with a RA equal to 6, and the recipient agent RV is $<2, 2, 2>$, then when applying the Manhattan Distance equation, the match has then occurred. The second condition checked is the TTL, if $TTL = 0$, the task will be considered failed, otherwise, if $TTL > 0$ the receiving agent will check the TD of the task; if it is sufficient then it will be executed, but if its not sufficient (either because the received agent is currently executing a task or its queue of tasks has 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 mentioned above, the task will be either failed or delegated to another agent in the network and so on.
In this network environment, the agents can fail (controlled by a probability) to simulate the situation when agents can be offline and are unable to accept messages and execute tasks for a period of time. The potential of an agent failing is controlled by a probability based on a uniform distribution to simulate real-life network failure occurrences. When a customer agent sends tasks to the network, a feedback from the receiving agent should be issued in return that must be one of the following possibilities:

- "Task has been executed": If an agent has accepted the task.
- "Task in ATQ": If the receiving agent is able to execute the task but is currently executing another task and the deadline of the new task is within the time consideration of the receiving agent, then it will be added to the agent Accepted Task Queue ATQ.
- "Task has been failed": If any one of the conditions has not been met, the TD and $TTL = 0$.

3.1 Organisation Creation Process

In a network of agents, an agent is called Busy if it accepts a task and is currently Busy executing it for a period of time. We have defined another status called "Busy agent and $ATQ \geqslant K$" which means an agent that is currently Busy, it has just received another task from the customer and its ATQ contains K number of accepted tasks, $K \leqslant W$, where W is the maximum size of ATQ. An agent with these conditions is one that can start the organisation creation process and by default it is the Head of the organisation. The newly created organisations emerge based on the gossip algorithm, (Serugendo et al., 2011).

In earlier work that is presented in (Al-karkhi and Fasli, 2017), we have used a gossip algorithm where the Head is sending a multicast message to randomly selected neighbours asking the agents to join its organisation and be its service providers.

But here in Algorithm 1, we have introduced a better usage based on the gossip algorithm where the Head sends the multicast message to all directly connected neighbours to create the organisation. in both works, the decision of accepting to join or not, depends on the status of the agents of being busy or less busy at that time. The less busy agents are the most likely ones which would accept to join the organisation in order to improve their resources utilisation. An "accept to join" message will be sent to the Head with its contact details. If the Head's message encounters a busy agent, that agent will only convey the message to other agents in the network. The size of an organisation may vary based on the status of the agents that receiving the Head's message. The Head's message will continue to be transmitted over the network until it returns back to the Head or it's TTL has expired. To this end, the system has added a virtual layer "an organisation" over the network of agents. We believe that the creation of organisations will lead to better utilisation of the system resources.

The grid and cloud computing are focusing on providing a powerful range of resource sharing in their distributed environment. But, it has been found that in an organisation that is part of a grid computing system, a computational node can be underutilised and not meet its full power. i.e it can be only utilised or be busy less than 5% of the time (Haider and Nazir, 2016). We are aiming to increase the agent's resources utilisation by allowing an agent to join more than one organisation. This lets us execute more tasks and makes the system more resistant to the failure. 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 had joined (since it can join more than one organisation). The Head will send the task to its Members if one of them can not accept the task, the Head can send the task to another Member that can satisfy the required resources and accuracies and so on for all other joined organisations. If the task cannot be executed inside any joined organisations, then the Member will work as if there is no organisation i.e the task will be delegated to one of the Member's neighbours with the TTL and deadline constraints.

However, there is still a number of constraints on the resulting self-organised system and on the resulting task execution values such as the existing network size, the number of tasks being issued in each cycle and the simulation running time. So, in our work we have invented a mechanism named the HRP as an added solution for the creation process of agent organisations. The first part of this protocol has been implemented during runtime of the gossip algorithm by calling "BeMyHenchman" which will be described in section 4.1 in this paper. The second part is explained in section 4.2.

Algorithm 1 : Gossip Protocol.

```
Select an agent a_i
while true do
  cycle_i
  a_i.TransmitTo ([N] targets (agents)), [N] is the
  local membership contact list of agent a_i.
  if [N] are non-Busy agents then
    a_i.Infect (gossip message)
      They have the option to join or not to the
      created organisation.
      call "BeMyHenchman" to select an agent to
      be Henchman.
  else
      The Busy agents will be used only to transmit
      the message to their peers.
  end if
  cycle_i=cycle_i+1
  The receiving agent in the last period broadcasts
  the gossip message to its peers.
end while
```

3.2 Model Description

This section presents the description for the second layer (organisations) above the existing agent's network.

Every created organisation in the system has the following format:$Org = \{Org_{ID}, Org_H, \{a_1, ..., a_j\}, HM, Z\}$,where:$Org_{ID}$: a unique identifier for each created organisation. Org_H: is the name of the Head for an or-

ganisation. $\{a_1, ..., a_j\}$: the Members in the organisation at a specific time. HM: is the Henchman name, the Head's follower in each created organisation. Z: is the maximum number of accepted Members to join in each organisation which is a setting parameter for all the organisations but the number of Member joined in each organisation may vary from one organisation to another as mentioned in section 3.1.

3.3 Model Implementation

The following are the principles for the implemented model.

- a_i is an agent that can join f number of organisations as a maximum but it can decide to join only a random number g of f, where $g \leqslant f$.
- The role an agent a_i in an organisation (Org) is a service provider, where: $Org \neq \emptyset$ and $1 \leqslant$i$\leqslant$j.
- The roles for a_i within an organisation Org can be {Head, Service Provider, Henchman}.
- A Head in each organisation is responsible for forwarding the message to its Members bearing the customers task message. The function of a Head in an organisation is to coordinate the work in its organisation. The Head may also act as Service Provider to maximise its resources utilisation.

4 HENCHMAN RECOVERY PROTOCOL (HRP)

After implementing the organisation's model in the previous section, we have seen that the number of executed tasks has been increased, and we have demonstrated that with different network sizes in the depicted Figures in section 5 below. The black line in the figures indicates the organisation's model performance. However, the environment is dynamic, so that we need more powerful management to be imposed on the structure of the organisations to provide an efficient service to the customers, mainly to detect the Head failing.

Our method focuses on providing the Head of each organisation with a follower called Henchman.

The Head is the one which is responsible for distributing the tasks to its Members in the organisation, so that when the Head has failed, the Henchman will act as a substitute to the failed Head providing self-organised capability to the system to maintain the functionality of the created organisation. Since the customer is sending tasks to the network, there is a risk from sending the tasks to a failed Head, and here the Henchman's role in the recovery protocol is to prevent losing the tasks.

4.1 Henchman Recruitment

The Head agent sends a message to the first agent a_i, where $1 \leqslant$i$\leqslant$j that joins its organisation asking whether it can be its Henchman (HM) and sends the same message to other agents that join subsequently until it assigns a Henchman for its organisation. The Head has the responsibility to synchronize its database (DB) with the Henchman which contains the contact details of the Members of the organisation, and also for every newly joined agent. This means that the Henchman DB will always be identical with the Head's database. When the Head is offline, the Henchman will immediately replace that Head to maintain the functionality of the organisation.

The Henchman is a Member as well as it is responsible for checking the availability of the Head by sending a heartbeat message "are you alive" to the Head in random cycles. This will be explained in more detail in the Algorithm 2 called the "Heartbeat". If the Henchman does not receive any acknowledgement back from the Head, the Henchman will declare to all other Members and the customer agent that the Head has failed and the new Head is the Henchman in order to redirect the traffic to itself instead. When the Head recovers, i.e. the Henchman receives "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.

However, there is a small chance that the disruption may effect both the Head and its Henchman at the same time. In this case if the customer agent sends tasks during that time, tasks will be considered failed, until the Head and/or the Henchman return(s) back to active after being offline for a random number of cycles.

4.2 Henchman Heartbeat Algorithm

After assigning the desired Henchman, in Algorithm 2, the Henchman will check the availability of the Head after a random number 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 2 : Henchman Heartbeat Algorithm

```
while (true) do
    Henchman.SendMessage (Head, "Are you Alive.")
    Head.SendMessage ("I'm Alive", Henchman)
    if (Henchman.Receive (Head, "no message") then
        Henchman.SendMessage(Customer, Members) // To redirect them to the Henchman.
    else
        The Head is online again, redirect everything back to the Head
        Henchman.SendMessage (Customer, "Head is alive")
        Henchman.SendMessage (Members, "Head is alive")
        in case they would like to send tasks or receive tasks to/from the Head.
    end if
    keeps on checking the Head in random cycles
    cycle_i=cycle_i+Random(1, N)
end while
```

5 EXPERIMENTAL WORK

The network has been implemented in Repast Simphony which is an Agent Based Modeling (ABM) simulator using Java programming (Macal and North, 2014). Table1 shows the setting parameters which have been used to set up the experiments, where (*ND* = Normal Distribution, (T and M) are given values of the mean and the standard deviation, $H = Head$, $HM = Henchman$, $SP = ServiceProvider$).

Table 1: Henchman Recovery Protocol.

Agents	Tasks	Prob.offline(1)	Prob.offline(2)	Offline cycles
5000	ND*T+M	H=0.9, HM=0.6	H=0.6, HM=0.2	H(2-5), HM+SP=(10-11)
2500	ND*T+M	H=0.9, HM=0.6	H=0.6, HM=0.2	H(2-5), HM+SP=(10-11)
500	ND*T+M	H=0.9, HM=0.6	H=0.6, HM=0.2	H(2-5), HM+SP=(10-11)

The three models (HRP, With organisation and Without organisation) as shown in the figures below, have been run for 5000 simulation cycles with the setting parameters shown in Table 1. The simulation has been run for 5000 cycles to produce the result for the model with HRP and network size=5000 agents, and re-executed for 10 times with the setting parameter revealed in Table 1, and the same process has been applied for the other two models. The experiments have shown the operation of all the three models to different network size $(5000, 2500, 500)$ and with different offline probability values $(0.9, 0.6)$ for each network size. These experiments measure the effect of the simulated models on the system performance using Equation 1. The equation has been used to calculate the average number of successfully executed tasks during the simulation cycles $(ANSET)$, the absolute value is the Manhattan Distance, used to produce the value of the matching process between the requested resources issued by the customer for each task and the matched resources from the agents for all the tasks which are executed successfully.

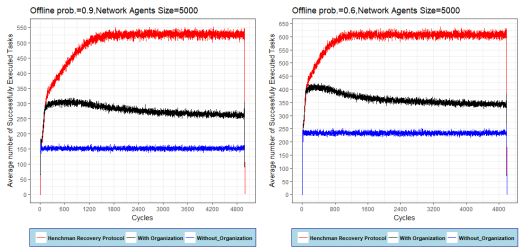
$$ANSET = \frac{1}{R} \sum_{i=1}^{R} \sum_{j=0}^{j=2} \left| RV_j - AR_j \right| \qquad (1)$$

Where:
R: number of runs = 10.
RV_j : the requested resources vector for the task j.
AR_j : the matched agent resources that performs task j successfully.
j = 0 to 2 : the index of the tuple that represent the requested resource vector.

The Figures below will demonstrate the differences in the average number of successfully executed tasks within cycles for the models (HRP, With organisation and Without organisation) for different network sizes. With network size of 5000 agents, Our tests revealed that the HRP in comparing with other two models has positively executed higher percentage of tasks with different offline probability values $(0.9, 0.6)$, as shown in Figures 1a and 1b. Also similar results have been achieved with network size of 2500 agents, Figures 2a,2b. The same performance from the HRP has also achieved with network size of 500 agents in comparison with the other two models. However, with network size of 500 it was very difficult for the agents in both (With organisation and Without organisation) models to cope with the failing probability with the high number of tasks issued from the customers in each cycle, as shown in Figures 3a and 3b.
The most noteworthy output in the depicted Figures is from the HRP. Clearly, the Henchman has added advantages when it has been recruited in each organisation due to its integrated roles: temporary Head, follower, and a Member. In addition, the fact that the existence of heterogeneous Members in the organisations can operate in conjunction with the HRP to cope with organisation utilisation when some Members being offline. This demonstrates the idea that generating heterogeneous organisations structure and imposing roles for the agents in the system will improve the system performance as more tasks can be recovered.
Moreover, in the models (the HRP and With organisation) even if no Members were able to execute the task, a task still have the chance to be executed by delegating it to one of the neighbour agents. So that,

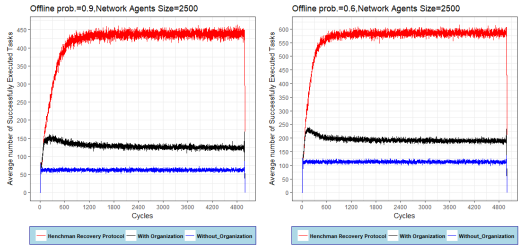
With organisation model shows in between performance and that implies even with the existence of the organisations, the model is still prone to loose tasks due to the high number of tasks issued from the customer agent and the probability of agent failing. However, With organisation model, the system shows better performance than the system Without organisation for all presented network sizes and offline probabilities. This is because the Without organisation model has no organisation and depends only on task delegation if the receiving agent cannot accept the given tasks. Hence, this is not an adequate solution due to presence of agent failure.

To compare and evaluate our work against other work in the literature, mainly the work that deals with fault tolerance in the networks, the master/standby server (MAS/SBS) has been investigated (Chen, 2007). Initially, for the configuration and setting purposes, both of the servers should be available from the beginning of the network creation process, and the connection link should be initiated by the MAS to send acknowledgement messages to the SBS showing its availability. Therefore, to create that scenario in our work, in each organisation we have specified the Head to act as MAS called Master Head (MAH) and also specified another agent in the organisation to act as a SBS called 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 send the heartbeat messages to the Head to check its availability. In addition, the organisation structure (Head, Henchman, Members) with its roles leads to achievement of self-organised groups that can overcome the failure issue as explained in subsection 3.1 above, unlike with the MAS/SBS method where the SBS can only switch to active mode when the MAS has failed. This experiment has been implemented with the same setting parameters for the same task distribution that has been presented in Table 1. However, the network size has been tested with 500 agents as a testbed to verify our system, Figures 4 a and 4 b with different offline probabilities (0.9, 0.6). Our system has demonstrated on average a better performance in each cycle compared to the MAS/SB server. This is because, in our scenario, the Henchman has a role in the self-organisation process where it acts as a Member that can receive and execute tasks as well as a follower to the Head of the organisation. In contrast in the MAS/SBS approach, the SBS is only a backup server and only turned to active mode when the MAS is failed.



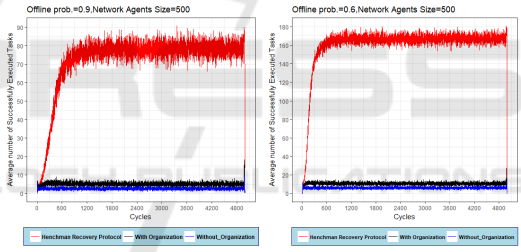
(a) (b)

Figure 1: Average Number of Successfully Executed Tasks for 5000 agents and offline probability 0.9 and 0.6.



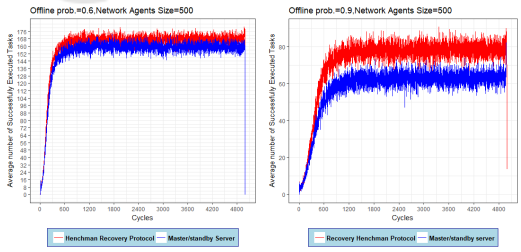
(a) (b)

Figure 2: Average Number of Successfully Executed Tasks for 2500 agents and offline probability 0.9 and 0.6.



(a) (b)

Figure 3: Average Number of Successfully Executed Tasks for 500 agents and offline probability 0.9 and 0.6.



(a) (b)

Figure 4: Comparison Between Henchman Recovery Protocol and The Master/standby Server, probability 0.6 and 0.9.

6 CONCLUSION

This paper has provided a framework for proposed data centres, mainly focusing on how to recruit agents

as part of the process of creating organisations. We highlighted the importance of solving the disruption problem using a self-organised multi-agent system as well as providing a solution in which the organisations of agents emerge depending on how busy agents get and this requires no central control. We have managed to demonstrate the HRP as a remedy for the disruption problem. It is employed during the self-organisation process when the Head of each organisation decides to have one of its Members as a Henchman. The purpose of the Henchman agent is to maintain the functionality of the organisation and its effectiveness in case of agent failure. A heartbeat algorithm has been used by each Henchmen to watch each organisation's Head. Experimental work has demonstrated the HRP as a reliable solution for a self-organised system. Our future work is to extend our system and implement one of the leadership competition protocol between the Head and the Henchman. And we can employ another Henchman inside the organization to maintain the organisation functionality in case the Head and the Henchman are both being offline.

REFERENCES

Al-karkhi, A. and Fasli, M. (2017). Deploying self-organisation to improve task execution in a multi-agent systems.

Barabsi, A.-L. and Bonabeau, E. (2003). Scale-free networks. *Scientific American*, 288(5):50–59.

Botev, J., Rothkugel, S., and Klein, J. (2015). 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*, pages 19–24. IEEE.

Chen, C.-W. (2007). Dual redundant server system for transmitting packets via linking line and method thereof.

Corkill, D. D., Garant, D., and Lesser, V. R. (2015). Exploring the effectiveness of agent organizations. In *International Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems*, pages 78–97. Springer.

Dignum, V. (2009). The role of organization in agent systems. *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, pages 1–16.

Elnozahy, E. N., Alvisi, L., Wang, Y.-M., and Johnson, D. B. (2002). A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys (CSUR)*, 34(3):375–408.

Foster, I., Zhao, Y., Raicu, I., and Lu, S. (2008). Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10. Ieee.

Gutierrez-Garcia, J. O. and Sim, K.-M. (2010). Self-organizing agents for service composition in cloud computing. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 59–66. IEEE.

Haider, S. and Nazir, B. (2016). Fault tolerance in computational grids: perspectives, challenges, and issues. *SpringerPlus*, 5(1):1991.

Horling, B. and Lesser, V. (2004). A survey of multi-agent organizational paradigms. *The Knowledge Engineering Review*, 19(4):281–316.

Macal, C. and North, M. (2014). Introductory tutorial: Agent-based modeling and simulation. In *Proceedings of the 2014 Winter Simulation Conference*, pages 6–20. IEEE Press.

Miyashita, Y., Hayano, M., and Sugawara, T. (2015a). 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*, pages 262–281. Springer.

Miyashita, Y., Hayano, M., and Sugawara, T. (2015b). Self-organizational reciprocal agents for conflict avoidance in allocation problems. In *Self-Adaptive and Self-Organizing Systems (SASO), 2015 IEEE 9th International Conference on*, pages 150–155. IEEE.

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

Tesauro, G., Chess, D. M., Walsh, W. E., Das, R., Segal, A., Whalley, I., Kephart, J. O., and White, S. R. (2004). A multi-agent systems approach to autonomic computing. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 464–471. IEEE Computer Society.