Department of Computer Science

Submitted in part fulfilment for the degree of MEng.

Swarm Memory

Harry Burge

Version 1.0, 2020-November

Supervisor: Simon O'Keefe

Acknowledgements

Contents

List of Figures

List of Tables

Executive Summary

1 Introductory Material

1.1 Introduction

Swarms are an increasingly important area of research for society, as the world moves towards a distributed technology future. The research of swarms within a technology setting can be broadly divided into two partitions, these are intelligence and mechanics.

Swarm intelligence can be viewed as the research into highly distributed problem solving[2, 5]. This is ever more becoming relevant as computer systems start to level out in sequential performance [7] and parallelism is embraced, satisfying the demand of the age of big data [9].

Swarm mechanics heads more towards the robotics side and can be seen as the study of practical implementation of a swarm, whether that be movement or communication within the swarm. This is on the rise in industry, as society's pace increases and manual labor is automated out. Whether its drone delivery to inpatient customers or mapping areas in dangerous environments [1].

These areas often are highly integrated rather than disjoint from each other, and are rarely seen in their pure form. An example of a pure form of swarm intelligence can be seen in [5] with network routing protocols. This project focuses predominantly on swarm intelligence to deal with a practical problem.

Most research on memory within a swarm has gone down the route of optimization on distributed problem-solving algorithms, as compared to practical applications of storage of abstract ideas as a collective. As one of the key reasons for using a swarm is redundancy, which is often assumed and boasted about, rather than proven, specifically within the memory domain.

This relative lack of research into collective memory, seems to the author to be a glaring hole in the foundations of a complex and interchangeable subject. The need for more research into collective memory can be seen in examples, like how invaluable it would be to mapping of dangerous

areas [2]. By being able to handle the loss of agents combined with the redundancy of sufficient memory policy, provides a much wider scope for swarms to be used.

An explanation for why swarm based memory management solutions are an under developed area of study is the existence of cloud-based storage research. The argument for these two subjects being partially-separated is the nature of a swarm's locality and ever-changing network style, compared to a typical server network. Reliable storage of data on an ever-changing network of devices is a hard task to complete, handling loss of connection between servers, enforced reliability of access to data or loss of services, whether from non-correlated or correlated failures [9].

Most elements to cloud storage policies at a high level of abstraction could work effectively within a swarm based environment. However as explained above, key adaptations would need to be created for an effective policy. A prime example is "SKUTE" from [10]. "SKUTE" will be the main inspiration for this project's solution, too storage on effectively a highly dynamic network.

The objective of this dissertation is to merge three areas of study into an effective/suitable storage policy for swarm-like agents in a setting with high locality and dynamic connection behaviours. Then to perform multiple analyses on the created policy using a variety of simulations to gauge its capability compared to the desired abstract behaviour.

To complete this objective, we will programmatically break down the problem described in Section 1.2 into solvable tasks. Therefore the report will be structured as follows, firstly we'll define a clear and concise problem, of which incompaces all disgruntlements described above. Secondly, bring the reader up to date with relevant literature and explain key concepts that will be required for a complete understanding of where and how the proposed solution has been derived and why said solution might be a relevant stepping stone for future research. This will be undertaken in sections, Section 2.1 of which goes into detail about current cloud based storage technologies and Section 2.2 of which will delve into more of a background behind swarms, specifically relating to the problem and possible solutions. We'll then go through the methodology of the proposed solution's design, how it is supposed to act and react in different scenarios, and the reasons for why. This leads to analysing how effective the proposed policy has kept to standards derived in the methodology section, and whether it solves the problem defined in Section 1.2.

1.2 The Problem

The problem that this report will cover is to create a storage policy for agents of a swarm, to be able to store directional abstract data as a collective, without complete duplication.

The problem can be split into two separate sub-problems. One of them is handling data duplications throughout the swarm, as to be able to control for failure of an agent, and provide a mechanic for recovery from said failures.

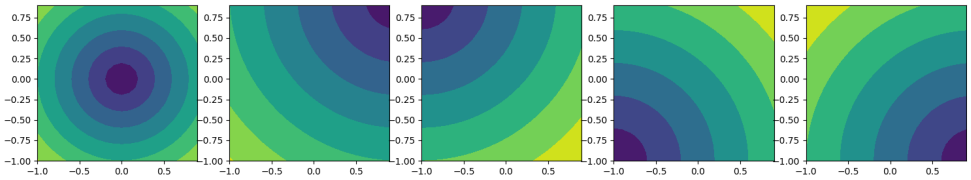

Figure 1.1: Data duplication density based off distance to datas desired location

The second is to control where data is duplicated, to give directional characteristics of the spread of said data. A visualization of this can be seen in Figure 1.1. Which is the density of duplications within an area based on distance from the desired position of that data. As the reader will see, the further we get from the desired point we should have less duplicates of that data.

A real life example of where this type of problem arises, is in mine field operation [2]. In our example, agents would need to map out where potential mines might be and record their location. Once an agent locates a mine it wants to spread the word of its location. However, memory restrictions in agents can only permit the knowledge of a minimal amount of mines. Therefore agents closer to the mine should be prioritized in knowing local mine locations.

In a practical solution, agents could fail individually for example running out of battery, or fail as a group from a mine going off. The swarm doesn't want to lose any collective data from these failures occurring.

With this increase in complexity, the solution needs to be able to withstand fluctuations of the swarm. With reliable redundancy to handle correlated (Mine detonation) and non-correlated (Power loss) failures.

1.3 Approach and Justification

An initial solution was to use an adaptive version of RAID parities. However, a better concept solution became apparent after conducting more research into cloud based storage.

Within the adaptive RAID design, if a subject agent has two agents within its locality with different data, a XOR parity of both data points would be recorded alongside a record of which agents they came from. If one of those agents left the subject locality then the parity would be used to reconstruct the lost data. This approach was disregarded because of these four factors:

- Implementation of directionality would be tricky and not consistent
- If the two agents die/leave the locality at the same time then the parity cannot be restored
- The algorithm relies on a swarm that breaks connections often to spread data
- There is no duplication reduction, therefore over time the data point would spread to all agents

These drawbacks highlighted the need for a new approach, cloud based-storage policies. The proposed solution takes heavy inspiration from “SKUTE” [10]. The design of a replication policy lends itself to heuristic based control, and allows us to implement duplication density and direction to the spread of data.

A distributed/homogeneous style of control is used, partially because there is an average power loss over the swarm compared to a leader based control, but mainly because of the methodology of a homogeneous swarm. A hybrid approach, if not accounting for power loss, would almost certainly be better in every single way, due to its global view. A hybrid implementation would be very simple and deterministic compared to its homogeneous counterpart. Both methods could not guarantee data loss will not occur, however, a hybrid approach would be better equipped to handle correlated failures.

With an extensive list of drawbacks to using a homogeneous control over a hybrid control scheme, it would be deemed inappropriate to use said homogeneous control. This disgruntlement is valid only when talking about swarms that are partially static in nature. When the swarm is highly volatile, in terms of movement, then we might spend more time assigning a leader rather than doing our actual task. Scaling of a hybrid system is harder because of the partitioning of agents to leaders. This justifies the authors choice to focus on distributed homogeneous replication policies.

1.4 Motivation

In sections 1.1 and 2.2, we have painted a clear picture as to why the study of swarms is becoming an integral part for leading us to the future. With sequential computation reaching its limits [7], and the world becoming increasingly data rich, the need for distributed problem solving is heavily required. As technology gets more efficient, the ability to make swarms become exponentially viable, especially with recent advances in nano technology/biology.

Swarm robotics is limited by the technology of its time. The use of research is limited and far between in our day and age. We research this for the future where technology can support and utilize the fullest potential that swarm behaviour can offer.

The main uses nowadays are within surveillance [21, 18], delivery or military [1]. However, to see the full scope of what man-made swarms could do, we look to the future. Whether it be space exploration [20], nano-robots in medicine or in the parallel/distributed software domain [19].

It is for these reasons that I have decided to contribute my part to this extensive and breakthrough field, and will hopefully see it grow to its fullest potential.

2 Litreture Review

2.1 Cloud/Backup storage policys/schemes

Like most things in computer science, cloud storage started off relatively simple (In nowadays terms). As the years have progressed so has the demand and use of the cloud, varying from everyday people storing files to large businesses storing harvested data. This led to the need for much sophisticated storage solutions, which can handle the increasing file size and frequency of use. A component that increased this complexity was the Legal Services Act. 2007 [11]. This enforces that cloud storage suppliers must provide reliable and fast data collection for users. Not only that but to also provide near guaranteed longevity of stored data. In this section we will focus on cloud-based storage policies and some background terminology needed for this report.

Firstly the reader will need to understand the difference between correlated and non-correlated failures. A non-correlated failure is when a device fails independently and with no relation to other failures with the system as a whole. An example of this within cloud based computing, a server node can shut down due to a software failure, therefore we lose connection and access to the node's data, typically this is completely independent of other servers in the rack. A correlated failure as the name suggests, is when a device fails with other devices with relation linking why they have failed. A typical example of this in cloud computing would be mass restarts because of a power surge from a storm. The power surge event is what links the failures together, therefore making it correlated. To be correlated doesn't mean they need to be geographically close together, however within our swarm solution will mean geographically close failures.

To control for these two types of failures there are many different solutions providing different features. Locally what is typically used is a RAID system for local failures within a node, and then a replication policy [9] for internode duplication. These in tandem provide stable node storage and redundancy for when a possible node failure acquires.

The most common forms of replication policies will take a piece of data, decide whether the data needs to be duplicated and if so will completely

copy the data onto another storage device. This provides a backup in case of failure on either one of those devices. A simplistic approach would be a random replication policy, where data is randomly chosen to be duplicated, typically duplicated within the same datacenter, so on a neighboring rack. This is an efficient design policy for handling non-correlated errors, however, lacks the robustness against correlated errors, and without a tracking of global duplications can lead to over used storage. We can mitigate for correlated errors by allowing for duplications to happen over data centers, however, this leads to downsides which will be explained below. An algorithm like random replication, is substantial for long-term storage where popularity of data and distance to users are averagely the same for all data items.

Two key concepts of availability and popularity are not taken into account when using a random replication policy. When cloud based storage transitioned from a semi-local backup system to a worldwide daily driver. Random replication can't withstand the variability of how users interact with files nowadays. An example of why these concepts are needed in today's age, are videos. If a video is hosted in one country and replicated within said country then international viewers might have delays to their streaming. If we then tried to compensate for that by hosting it in multiple countries, those other countries might not view the video as much. This therefore means we are wasting storage space of which we could use for other more popular videos. This therefore leaves us trying to maximise for both situations, and a new replication policy is required.

We will be looking into two different algorithm concepts, of which try to handle the maximisation problem. Both withstand non-correlated failures well because of the nature of replication, so we will only be looking into the other effects. The algorithms have been abstracted from papers on handling "Distributed key-value store" [14], where you have key-value pairs on multiple devices on a network where duplication only leads to more fault tolerance of the data stored.

The first approach uses a privileged level of control where it uses its global knowledge to make decisions about whether and how to duplicate items [9, 12]. This doesn't have to just be data replication, but the same principles can be seen within schema changes [13]. Due to the nature of having a privileged user, the control of the policy is a lot easier to make specific behaviours be exhibited and to be understood. This means we have higher guarantees for correlated failures unless on the master node, however this can be handled dynamically by assigning another master, touched upon in Section 2.2. Availability and popularity are handled by the policy coded onto the master node.

Having an approach that uses a master, doesn't work effectively for a

swarm. This is because the change from a server network to a swarm is quite a drastic change. Servers running over network generally have complete connectivity e.g. Global scope. Also servers have a constant power supply compared to the average swarm agent. When restricting the masters scope we have to rely on messages over other agents which will first of all reduce processing capability and power loss will be more significant to agents closer to the master, possible leading to a cut off from command [1]. It also doesn't fit into the ideology of a swarm, this will be talked about in Section 2.2.

The second approach doesn't rely on a privileged member and can be adapted for locality. We follow a distributed control approach where each node (In the case described below its each key-value pair) has its own controller. Following the approach used with "SKUTE" as proposed in [10], it can make four decisions per data item. These decisions are; Migration, Suicide, Replication, and Nothing.

Migration is the move to a lower cost or more redundant servers. Suicide is the removal of itself, this is usally be because of to many duplicates. Replication is when the data decides that it needs to be duplicated and sent to another node and Nothing is as the name indicates.

Because of the highly distributed nature of said approach, when coming to suicide we need to handle the edge case where the only two nodes with duplicates make the decision to suicide at the same time, therefore leaving us with no replications. This is where a consensus algorithm comes into play. Paxos [15] is an example of how both nodes couldn't suicide at the same time, therefore leading to the ideal case in this example of only one duplicate.

Within the domain of server storage networks an approach like "SKUTE" is less commonly used because it adds complexity which is not needed within a global and consistently powered network. Most data warehouses would prefer to have one server running as a controller and other servers running at full capacity compared to all servers at a slightly lower capacity due to extra self computation. However this approach allows for greater redundancy because of having no single point of failure.

An approach like this is highly adaptable towards a homogeneous swarm. However with heterogeneous swarms the previous algorithm may work a lot better. Differences between the swarm types will be explained in Section 2.2.

Moving towards local redundancy and optimisation is the stagnated study of RAID. This is where we change orderings of multiple storage discs to gain redundancy and/or performance increases. This grouping of disks is

called a RAID array and can be structured in a multitude of ways. Common structures are labelled as RAID levels and give different attributes based on what functionality you are pursuing [**?**].

One of the key components of multiple RAID levels is the use of parities [8]. This is where a function (typically an XOR) is done on two or more sets of data to create one or more parities. The function has a property thus that if a tolerant amount of disks are lost the data that was lost can be reconstructed. In the case of data A, data B and A XOR B, if data B is lost then can be reconstructed using data A and A XOR B. With different levels and functions more than one disk can fail and still retain data however if failures go over that then all data is lost. RAID is predominantly used internally within a storage node to provide redundancy against disk failures and increase speed of writes that are typically on hard disks, because of cost and possible reads after failure of heads.

The methodology is that as long as the nodes individually are redundant enough and then there is control on a higher level with our replication schemes, that is a sufficient redundancy. RAID could be adapted to run over multiple different storage nodes, however the complexity compared to performance is heavily in the favour for the above methodology for storage networks. We will come back to the possible uses of RAID internally and externally in Section 5.1.

// Finish rewriting

2.2 Swarm robotics

As explained in the introduction, the study of swarms are split into two subsections, mechanics and intelligence. Both explained broadly in the Section 1.1.

Continuing on the discussion about swarm intelligence. Typically swarm intelligence focuses on solving abstract problems, like the traveling salesman problem, in a local distributed manner compared to a global manner. From the papers read by the author, it seems that predominantly the topics undertaken are testing distributed solvers compared to already researched solutions to see how they measure up. An example of this within the TSP domain is a genetic algorithm versus AS-TSP [5]. These algorithms provide benefits and drawbacks compared to their counterparts.

The concept of swarm intelligence is creating a solver to a problem using a distributed algorithm that can rely on natural parallelism, doesn't rely on global knowledge and is adaptable on the fly, compared to their counter-

parts. A good example of where these algorithms excel is the networking domain, because of the high parallelism and need for adaptability [5].

The other subsection can be broadly known as swarm mechanics. This encompasses all of which swarm intelligence doesn't cover. Swarm mechanics focuses on problems that are less abstract and are usually in the domain of physical implementation [2, 4]. Swarm robotics can be seen as the same as swarm mechanics, however, it doesn't have a broad enough scope/name to fit everything that can be researched in this author's opinion.

Two arguments to justify the naming of swarm mechanics are as follows; within this domain we focus on the emergent behaviour of a swarm compared to the solution it might give. The second is that swarm robotics doesn't encompass the study of swarm behaviour in nature [5, 22].

Delving deeper into swarm robotics we have three different methodologies, heterogeneous, homogeneous and hybrid [1]. These can be adopted in multiple ways, however, we will focus upon the adoption of these methodologies on decision making and physical attributes of agents. Firstly we will talk about the physical adoptions.

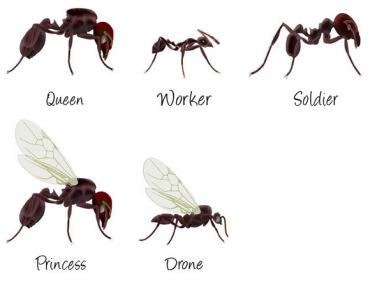

Figure 2.1: Example of a hetrogenus ant colony. https://www.pinterest.co.uk/pin/777363585651532845/

A heterogeneous swarm is defined by having differences between agents of the same swarm, as in Figure 2.1, whether physical or mental [1, 5]. These occur commonly in nature and are less studied [5], because differences in agents are a rarely needed property in research-based problems. For real-world solutions, heterogeneous swarms can be of great use, allowing other agents to pick up the slack of the swarm, or complete tasks that other swarm members cannot complete. A good example as described in [1] with a mother ship being a navy boat and a swarm of quadcopters. The boat picks up for the slack of the swarm by being able to transport them longer distances than the swarm could normally cope with.

The argument against heterogeneous swarms is that there is a tendency to over rely on the differences of agents. Running with the example from

[1], the agents rely on the naval boat to be able travel longer distances and have to have a recharge point. Without the boat the swarm fails after some time. The decision to have a heterogeneous swarm in this case is valid because if the naval ship is lost, then something has already gone horrendously wrong.

To have physical differences in agents, is to breed efficiency. However this can only hold true if the vulnerability of losing too many agents of the same type, that's work is key to the survival of the swarm, is mitigated. This vulnerability can be controlled for in multiple ways. Common rules are found in nature's swarms and can be extracted from them. These are; jobs need to be either interchangeable between all types of agents however some agents are more efficient at that job [22] or the jobs that are specific need to be non-essential for the colony's survival.

Ants typically fit into this category where ants have different types, as shown in Figure 2.1. Some ant species, like Leaf-Cutter Ants, even have subcategories within a category of type. Leaf-Cutters have workers that will specialise in certain tasks like fungus farming. The best example of showing the interchangeability of these roles is when major ants do worker jobs when there is a significant loss of workers [5]. This leads us more towards a hybrid approach which is explained later on in this section.

Because practical implementations of what humans can achieve currently in their robots, they don't have the adaptability that biology can provide, without making agents too complex.

Homogeneous swarms are defined by each agent being the same. This is found less often in nature, except for at the microbial level, and is commonly found in man-made agents. Because of biology's natural adaptability compared to current standards of robotics, semi-heterogeneous swarms are exploited better [1, 5]. Therefore we maximise for the floors of our current technology, however with a sufficiently complex agent homogeneous swarms are the most optimal, but that is getting into speculative futuristic technologies of self replication, advanced intelligence and nano size.

Homogeneous swarms benefit from maximum redundancy, this is because if any agent goes down there are still an entire swarm's worth of agents to take its place. With this benefit of redundancy we acquire some possible losses in efficiency which could have been exploited with agents with specific hardware. The design for homogeneous agents is a complex one, either the agent is too simplistic therefore loses efficiency in their tasks. Or they are too over engineered to the point that all agents have the ability for every specialism and may never need said hardware. There is a thin line between the two, where either we lose practical power of the swarm or we have to invest more into the swarm than it actually needs. An example to

show this dynamic is if we have agents that need to mine and farm, they all have hands so can do the task at a suboptimal speed. We then as an improvement give them picks and hoes instead of arms. This gives us an increase in mining and planting speed, but why would the miners need a hoe.

Within the practical implementation of swarms, everything gets a bit messy. Usually there is not a clear cut framework or design that a swarm is designed to be like. They are designed to be as efficient as we can make them to be in any type of problem faced, this is where research and engineering have a bit of a disconnect. This is where hybrid approaches come into their own.

A hybrid approach tries to exploit the benefits of both heterogeneous and homogeneous designs without the downsides. Carrying on from our example of farmer and miner swarms, a hybrid approach to the physicality of an agent would go something like as follows. Each agent would have exactly the same body and could wield either a pickaxe or a hoe, but the key point is that a miner could become a farmer if it was required. The reader might wonder why physical hybrid approaches aren't regarded as the best of all approaches. This is because when bringing a theoretical solution into the real-world we gain massive complexities. How can we guarantee job type distribution? What about the complexity of changing between job type hardware? These are all questions that are needed to be explored by the person creating the hybrid swarm. Usually it is easier to go for the simpler solution and deal with the possible loss of either efficiency or adaptability.

Moving on to the control/decision making of a swarm following these approaches. Homogeneous control follows the purest form of swarm robotics, where each agent has control of its own decision making based on what other agents in its locality are doing, sometimes labbelled distributed intelligence. This therefore creates an emergent/structured behaviour of the swarm even though each agent is acting of its own fruition and can usually only see locally. A simple example of this is [23].

Heterogeneous control is also very simple in nature, where certain agents control other agents' decision making. This can be handled in multiple ways, two promoniante ways are hivemind control [18] and royalty/hierarchical control. Hivemind is where one agent controls the entire swarms decision making e.g. The hivemind controller will say the swarm needs to move to the left and then the agents handle that the way it decides. In a hierarchical approach it follows the same style as hive mind however deals with scalability better. Where we somehow have a power structure of certain agents being sub leaders of leaders. This control structure leaves itself vulnerable in the same way as its physical implementation, however, also

has the fear of bad actors and power loss distributions over the swarm. These still can affect homogeneous swarms however is less of a threat than on a heterogeneous controlled swarm.

Hybrid approaches to control of a swarm are just heterogeneous control policies that can adapt to changes of leaders and are usually designed for homogenous/hybrid swarms. The choice of leader(s) is usually done through a consensus algorithm [15] rather than based on any form of physical or mental difference. This allows for homogenous style bots to act in heterogeneous fashion. This can create more deterministic behaviours compared to emergent behaviors of homogeneous control, and can also help with the power loss distribution problem by re-electing leaders in different locations to help distribute where messages are relayed.

Humans themselves are a great example of a fully hybrid based swarm both in design and communication. Though humans have variations in characteristics they can be seen as pretty homogeneous in terms of the tasks that they can perform, obviously removing edge case actions like child birth. Tools and knowledge can be spread between humans to make the swarm more efficient and an agent can specialize in a certain area. However, if some agents are lost other agents can replace them by using the same tools and knowledge from the remaining agents of that specialism. Also, the natural power-based structure of humans fits a hybrid model in terms of electorship of some kind, and not of genetics (Except with royalty, however, this is more of a label rather than a genetic difference). The leaders aren't needed for every single action so fit into a usually hierarchical power structure, compared to something of a hivemind model.

3 Design

3.1 Simulation Infomation

This section will describe the environment of which the agents will be simulated in. The simulation is a 2D representation of a flat surface where agents can move freely around. Agents are randomly located on the surface within the center 75% area. Points are then selected on the surface, and at the start of simulation the closest agent to that point learns the data, that data can never be learnt again other than from passing information from agent to agent. The data is directional so when stored in agents memory the coordinates of where it originated is saved as well. The directionality is needed later for policy calculations.

Agents are homogenous, with a small connection radius around them. The world is width/height of 2, agents connection radius is 0.25. We are assuming that connections between agents are perfect, and that each agent can simultaneously respond to incoming packets and send outbound packets.

Agents have two partitions of memory, public and private. This is mainly a symbolic partition rather than an actual necessity. Private memory is data that has been learnt directly by said agent, and public memory is information that the agent has learnt from another agent. Both public and private memory cannot have duplicate data in themselves or across partitions. This partition is symbolic because it is not required, however useful to have. It allows us to not require memory management of which having the memory as one partition might require. An example of this in a one partition model, if an agent's memory is full with lots of public information and tries to learn a private data point, we would have to construct a way of dealing with these collisions. This is because of the priority difference between holding duplicate data compared to the gathered data, this is because the gathered data can only be learned once. With this partition we don't have to deal with this problem, however we do lose efficiency in how much information an agent can store. For the purpose of our testing and implementation, this efficiency is not a priority and is overlooked.

The control structure of the simulation runs 10,000 iterations. Each iteration

all agents are put into a parallel for loop to do its iteration, they are selected randomly in order to achieve the most real life-like simulation of individual agents that is possible for the author to use. The reason for not having the agents run in all different threads is because the scheduling policy has a tendency to prioritize a few agents over all other agents therefore over a certain time period one agent might have completed 500 steps and another just 1, this is unlikely in real life implementations. As a future improvement it would be good to have hardware that can support the number of agents all at the same time, compared to having to restrict the frequency because of the scheduling algorithm.

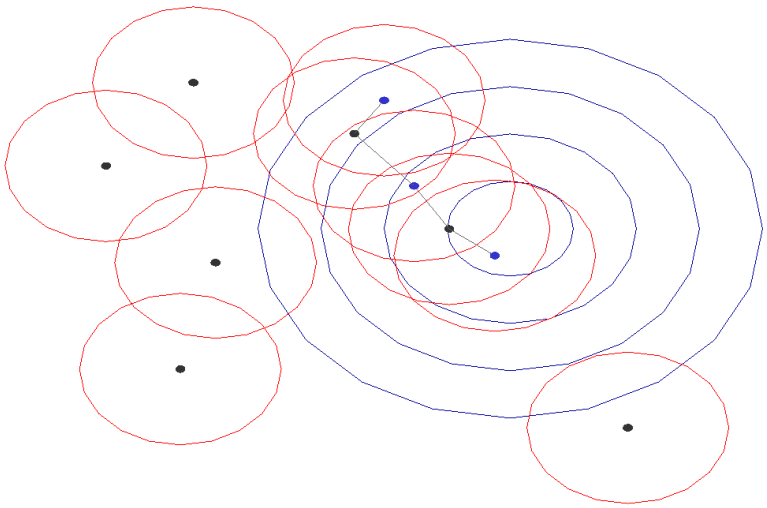

Figure 3.1: Example of simulation looks when running

In Figure 3.1 we can see an example of what the 2D simulation looks like. Dots are agents, and the red circle around them is their connection radius of which they can talk to other agents, these will be omitted from other images of the simulation to keep it cleaner and more understandable. The blue concentric circles are an example of the datas positional attribute, these will be omitted for a red circle in other images for the same reason as beforehand. The connection lines between the agents are just showing that data can pass between these two agents. These lines change colour based on what packets are being passed across them, this is for debug purposes and most likely won't be seen in any images. As a simple visualisation of how data is being passed between the agents, in this example the agent is blue signifying that it holds the duplicate of blue data, e.g. the data from the concentric circle data point.

3.2 Static Heursitic

This solution takes heavy inspiration from [10], we take the methodology of each agent controlling its own data and how it wants to distribute it. We do this using the actions of Replication, Suicide and Nothing. Migration is taken out due to the natural movements of the swarm, we don't want all data to be funneled back to the point, this is to deal with correlated errors.

We then use a heuristic to decide whether to do either three of these actions. In this case we have two heuristics based off these factors:

- Amount of agents in connection range that have a duplicated data compared to not having duplicated data
- Distance to datas target area
- Agents around average available public memory

We then used a weighted sum, to calculate the heuristic value. If the replication heuristic is above the replication threshold then we replicate and same for suicide, else do nothing. The weights of either action are set such that they sum to one, so they can be seen as what percentage of the output should this value account for. The reason for this algorithm being called static is that, mappings of where duplicated should be is deterministic and doesn't rely on any previous knowledge.

A pseudo code version of this can be seen in Algorithm 1. In this we can see how the agents work on an individual basis. Every step we check whether we have learnt any new private data, if so then duplicate that data onto all agents around. If this doesn't happen because of external factors like no other agents around, it will class that data has just been learnt in the next iteration. This is done to gain as much redundancy as quickly as possible, if a correlated failure happened early and we didn't do this, there is a high probability that all that data could be lost, so we transfer it to all agents and take the possible transfer energy cost losses.

We then move onto the factors mentioned in the bullet points above, in lines 8-10. For this iteration we focus on one data point in public memory, and in line 18 we switch to a new item in public memory. This means we are sequentially going through the public data and checking if we should do an action to it, over iterations equal to size of public memory stored. This is not good for scaling however for the size of data that we are using for testing it is addicaute, this will be talked about more in Section **??**. To gain the information needed for these three factors the agent broadcasts a packet to other agents in its vicinity. They will then respond by sharing whether they have a duplicated version of the data in their public memory, and how full their memory is. To check whether they have duplicated data

is a simple look if they have a data item with the same id in private or public memory. Once collated at the originating agent we can work out the specified values as shown in the code. There are improvements that can be made to the handling of broadcasting and replying to mean we aren't using as much power draw, however we will not go into this in the project and will be explained a bit more in the Section further improvements.

After getting the three factors we make sure they are on a scale from 0 to 1, and rearrange them to grow higher when we need to do a certain action. For example duplications go up when there is a higher density of duplications therefore on the replication heuristic we want it to be the other way around hence (1-dupes).

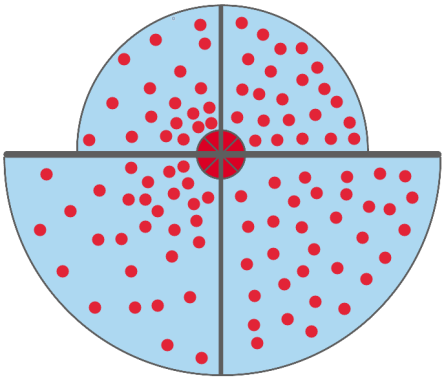

Figure 3.2: Example of changing of replication and suicide threshold on a uniform agent density

With changes to different weightings and thresholds, we get significantly different behaviours. An example of this is the increase of the replication threshold means that duplications will spread out less, avoid higher duplication density areas and not spread when agents are saturated in other data. The higher the suicide factor means there is less of a duplication density diffrence across the network.

For an easier understanding of the heuristics behaviour when changing the threshold values we can look at Figure 3.2. This is not a full representation of how the heuristic will work, this diagram shows what the duplication density would be like over a uniformly spaced swarm with exactly the same number of data in public memory. The smaller red dots are data duplicates originating from the datas area which is signified by the large red circle. When the replication threshold is increased in size, this can be seen as changing the range that agents can have duplicated data, as visualized with the blue circle getting smaller. Suicide threshold increasing, increases the diffrence in density across distances from the data area, as visuallised from left to right being lower to higher.

Weightings can also be changed to fit the behaviour style of the heuristic, for example if you wanted to favour just distance information you could change values accordingly. Having this weighted sum and threshold lends the algorithm to being optimised with a genetic algorithm, with a fitness function dependent on the results that a user might want. This would have to be done for all different applications types because the difference in weights changes the behaviour. For our runs we use, b = 0.45, 0.45, 0.1 and p = 0.3, 0.7, b was picked arbitrary based off preliminary guesses of what would be best, and p was picked to make the average values duplications ratio and distance to data be roughly equally weighted.

Originally “suicide data” in Line 16 of Algorithm 1, was done using Paxos [15]. This was used to ensure that in a scenario where all of one data duplications are in a locality and could all be deleted at once, therefore having no duplications left of that data, wouldn’t occur. However from preliminary testing this didn’t make much of a difference in our scenario, but should be added into any practical applications of this algorithm for more guaranteed redundancy.

// Further improvements refrence

3.3 Dynamic Heuristic

To deal with instability of the static heurisitic, Algorithm 1, described in Section 4.1. We need to do something to control when to replicate and suicide based of local infomation. The overall goal is to be able to reduce instability of the algorithm, whilst also taking into account practical limitiations of a real implementation. We could learn a lot about instability of the system if we polled agents around the selected agent, however doing that poll will be an increase in either amount of times we need to message and recieve infomation or the size of said infomation. Currently we only contact other agents around once, asking for their poisition, whehter they have a certain id in memory and how much free space in memory they have. To implement an accurate representation of global instability locally would also require all agents to have a table holding all data id’s that has been held by the agent and how many times it has been changed over n iterations. We would then have to pass this infomation between agents which would then lead to loss of performance in practical soloutions.

We therefore use a diffrent completely design to keep track of instability. First of all we want to restrict suicides for a small time period after a suicide has been completed, this is to slow down the rate at which we might lose data. This is not nesicarliry because of instability however helps to solve

the rate of which instability happens, e.g. means less agents change per iteration and smooths changes to help with predictrability of the second part of the algorithm. This is where we restrict replication tending with instabilty. Instability occurs between three scenrios, either threshold parameters have been set incorrectly, an agent has failed or natural agent movement has broken its connection. We therefore restrict a agent from replicating its data if instability happens to try smooth out said instability.

We do this using sigmoid functions with the above factors as inputs, this gives the threashold a dynamic ability to change based of the behavouir we give the sigmoid function. For replication it is:

$$\lambda_{rep} = \left(\frac{1}{1 + e^{-\frac{\theta - \alpha}{\beta}}}\right) \tag{3.1}$$

We then use a classical weighted sum as before:

$$\mathbf{W}_{rep} = \begin{bmatrix} 0.45 & 0.45 & 0.1 & -0.6 \end{bmatrix} \mathbf{X}_{rep} = \begin{bmatrix} 1 - r & \frac{\sqrt{8} - d}{\sqrt{8}} & s & \lambda_{rep} \end{bmatrix} \tag{3.2}$$

For suicide we use similily the same version of a weigthed function however it has less parameters and the sigmoid is flipped:

$$\lambda_{sui} = -\left(\frac{1}{1 + e^{-\frac{\psi - \alpha}{\beta}}}\right) + 1 \tag{3.3}$$

$$\mathbf{W}_{sui} = \begin{bmatrix} 0.3 & 0.7 & -0.6 \end{bmatrix} \mathbf{X}_{sui} = \begin{bmatrix} r & \frac{d}{\sqrt{8}} & \lambda_{sui} \end{bmatrix} \tag{3.4}$$

We then workout h_{rep} and h_{sui} using multiplication as below:

$$h = \mathbf{W}\mathbf{X}^T \tag{3.5}$$

Where r is duplication ratio, d is distance to the datas target point, s is average space in agents in the locallity, this is the same as in Section 3.2. θ in Equation 3.1 is iterations since last suicide, once a suicide occurs on this agent $\theta = 0$. ψ in Equation 3.3 is a value based off how many agents asked the current agent to store a bit of data, this naturally increases as instability is increased. ψ is reduced every iteration until 0.

Sigmoid functions where used because we can control when the heuristic should be unimpedded or impredded with a grey area inbetween so if a data needs to duplicate quickly it can still over power the block. α and β can be chnaged on both heuristics to give diffrent behavouirs.

3.4 Dynamic Heursitic with Migration

With the success of the dynamic heuristic in Section 3.3, we need to think about even more further improvements. One problem mentioned before is that because we are using a swarm we have inherent problems of connections between agents. As desribed about before if we have an agent that learns something but all agents around it are full up in memory then we are essentially locked out of the swarm. We could handle this using internal control of memory by assigning prioritys, and having a mangement system quite easily. However this could lead to loss of duplicates that might later be needed. Therefore we come up with a system for agents with higher data loads to pass of duplicates to agents with lower data loads. This is also known as Migration from "SKUTE" [10].

This affect is likely to happen if there are massive disparitys in agents avialable memory. We can currently see in most runs we have a wide spread in our memory, this is not optimal for best performance of the swarm. The most optimal soloution is where every agent has roughly the same amount of data as every other agent, to teduce the affects possible disconnect from the wider swarm because of memory. As a biproduct of this proposed behavouir we get correlation with gloabl infomation as a single local agent.

We implement this similarily to suicides where we don't want them to happen to often because of stability, however there isn't a complex heuritic for this. If the agent with the the most space in memory has two more than the current agent the migrate the data. Migrating is just when we transfer the data to the other agent then delete on our own agent. We don't use a complex heuristic like increasing the chance over time after a migration due to wanting this to be a definite behavoiur. We get a bit of a natural stability controller with the amount of leway we give agents, for example we picked two however if you really didn't want stability problems you would pick a larger leeway. You could also use parameters as provided by Section 3.3.

4 Analysis

4.1 Static Heursitic

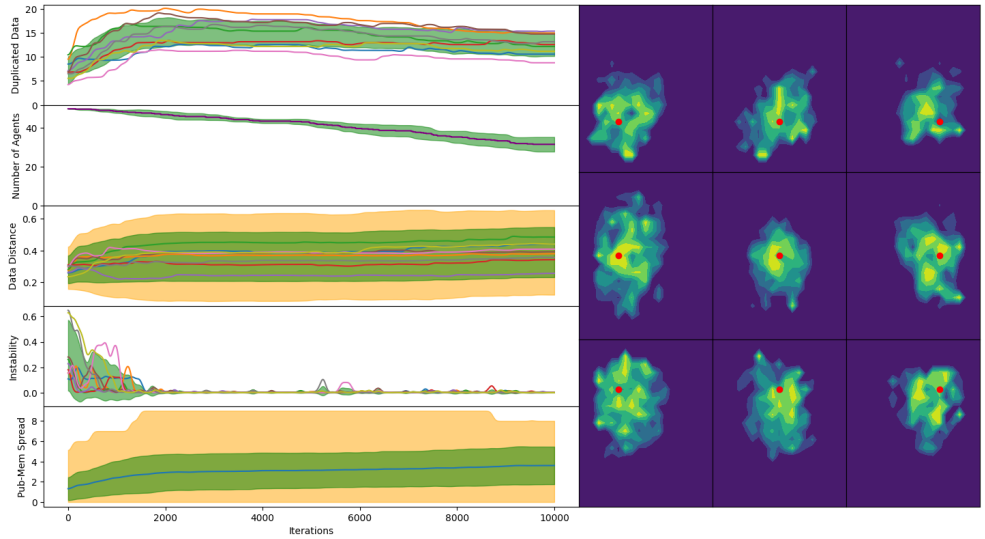

Figure 4.1: Static Heuristic on semistatic movement swarm, with non_correlated failures

To understand and evaluate our solotuin we look at how the solotuion reacts to diffrent scenrios. These are as follows:

- Semi-Static moving swarm with non-corrilated failures, Figure 4.1
- Semi-Static moving swarm with a corrilated failure, Figure B.2
- Circlular moving swarm with non-corrilated failures, Figure B.3
- Circular moving swarm with a corrilated failure, Figure B.4
- Changing of replication and suicide thresholds, Figure B.1

However first of all we will explain what each diagram means inside of Figure 4.1.

"Number of agents with duplicated data", as the name explains it shows how many agents had that data at that iteration. Each of the lines show a

diffrent datas mean duplicates over five simulations. The green coloured area is the standard deviation from the mean over all duplicated data. The reseaon for having all the indurvidual lines is incase one of our data duplicates had significantly lower results we could see. This could happen if in the runs the data was completly lost all the other data points would scew the mean to look like nothing bad had happend.

"Number of agents", also as the name implies is the number of active agents over all five runs. The purple line is mean over all runs and the green area is the standard deviation from the mean. This is displayed to see the effects of agent lose.

"Distance from desired point", this is what is the average distance that a data duplicate is from its desired point. The lines and green area are the same as before, and the yellow area shows the mean maximum and mean minmum distances from the desired point. This is shown to see how much of a fault tolerance we could have in a correlated failure. If the distance maxs out at 0.5 and we have a correlated failure that destorys all agents in a 0.6 raduis then we will lose that data from the swarm.

"Duplicate to no duplicate or vice-versa", this shows the stability of the system, indicating when an agent has changed from either having a peice of data to not or from not having a piece of data to having said data. Everytime this happens we count it as one point and this is averaged over five runs.

All data has been guassian filtered of a range of 50 iterations. This is to keep data readable, if we had a system that wasn't stable the mean would change drastically over each iteration, 10,000 times. We are only really intreseted in the trend and stability will show us the behavouir that we lose by using the filter.

And lastly we have duplication density visualliser on the right. We take a 20 by 20 grid and at each iteration get how many duplicates of data there where of a specific data point in that grid section. We also take note of the complete number of agents in that grid on that iteration. We then can cumulate this and average out over five runs to see how the data duplication density changes from distance around the point of datas origin. The reseaon for doing the ratio rather than just a summation is because of this example, if a gridpoint has 100 agents in it and 10 duplicates of data the summation would say that there is more data there, howerver a gridpoint with 20 agents and 10 duplicates has a much higher density of duplicates. This therefore gives us a better depiction of what is going on.

Firstly we need to see whether the algorithm runs as expected from the design. In Figure 3.2, we can see how the algorithms duplication density

is meant to change across the network, not accounting agents density or space in memory. Running the tests on a semi-static movement swarm changing the suicide and replication threshold values as in the Firgure 3.2 we get roughly the correct behavouir, as can be shown in Figure B.1. At very extreme values of replication threshold we can see because we have such little reduis of agents that can have the data the suicide threshold doesn't affect density changes as when we have a lower replication threshold. When replication threshold is low enough we get a larger spread so the density of the data can be seen much clearer. We can also deduce that suicide threshold is inversily corrilated with the stability of the network.

In the bttom left corner of Figure B.1, we can see it has a nice spread however its is very unstable with on average 13 agents changing from either having a duplicate to no or vice-versa. This could be solved using a new factor in the heuristic that takes into account stability. Within the threshold domain we can see that it performed as expected however will need to be improved, this will be covered in Section 3.3.

Now moving onto the most pressing topic of the algorithm is how it performs with correlated and non-correlated failures in two completly diffrent swarm enviroments. Starting with non-correlated failures, in Figure 4.1, we can see in a semi-stable movement swarm that even as agents fail indurvivdually the number of duplicates of data fights the downwards trend. Even though over the swarm we lost on average 2/5 of our agents the data our duplicated data has a diffrent gradient of loss, this shows good redundancy against non-correlated failures. We can also observe that distances and the density spreads are correctly working. In the stability graph we can see that it is very unstable at the start and becomes stable over time however this is indicative of a semi-stable movement swarm rather than of the algorithm as can be seen in Figure B.3

In a circular movement network we get a lot of connections broken at each iteration and in a repeating fashion, this can be seen in the stability graph with an osicilating pattern. Something obeserved is that in circular moving swarms we were getting larger spreads over distance of duplicated data, as can be seen in the diffrenses in the duplication densitys of both figures. This can be explained by the behavouir of the swarms movement. Agents that have the data will move with it so therefore will change distances from the data area, and contribute to the data looking more spread out on a average over time desnity graph. I beilieve it to be this becuase in the regions comparitivly semi-static has an effect of a hill, whereas the circular movement is like a pancake. Another observed behavouir is the duplications spreading across the network is faster in the semi-static movement model, I beileve this to be inhernet diffrences in the swarms networking rather than the algorithms effectiveness. The behavouir of a circular moving network is indicative of having gaps in connections from one side of the plane to the

other comapred to the semi-static swarm because they are usally linked well, so infomation is spread unimpeeded.

Moving onto correlated failures testing. The failures happened at the 3000th iteration and happened at the centre with a raduis of 0.25, so the same range as an agent can talk to. In Figure B.2, we can see that number of agents drops at the 3000 mark, this is smoothed out, however in the real simulation it happened instantanoulsy.

Puple/liliac is the centre data on this graph as can be implied from the corrilated failure affecting the average distance the most. We can see that other data points were affected by the correlated failure but was an inverse affect of moving the average close because data was lost further away from them. Data duplicates handled very well and recorverd strongly and stablity was only changed slightly except for on lime. I beileve the stability of lime is an outlier, however not a really bad one because the scale of stability is very insuggnificant. Lime was kept in the graph for transparancy of results.

We can see that the diffrence between Figure B.2 and Figure B.4 are nearly exactly the same as diffrences between Figure 4.1 and Figure B.3. With Figure B.4 we can see that there it is less affected by the correlated failure, mainly because of the amount of agents that were actually lost. In the duplication density graphs we can see the lasting affects of the correlated failure on the distrubutions of the corner data points.

My beilef is that this algorithm works more effectivly on a swarm that is higher natural instability and irregular movements, compared to a regularly organised swarm that doens't change much iteration to iteration. This is my beleif because of the distance spreads of data, and the handling of both corrilated and non-corrilated failures. Because of the identified instability that was found from the changing of the static thresholds we will focus now on creating a more stable enviroment to the swarm, using a dynamic models comapre to static, in Section 3.3.

/// Talk about static movement because it isn't talked about

4.2 Dynamic Heuristic

We go about the same approach as in Section 4.1, however we are focusing on how the new approach has affected the stability of the algorithm. We also need to llok at how the new heurisitc might have been affteced with the chnage to the heurisitc, even though it was designed to make a minimal impact when the prevouis heursitic is running smoothly.

For all tests ran we use α_{rep} = 15, β_{rep} = 2, α_{sui} = 150, β_{rep} = 3, θ is 50*(number of times successfully replicated other data onto self). These are in accordance to the equations in Section 3.3.

First we want to test the distribution like we did last time with the Static Heuristic, Section 4.1. Within this test we saw how the thresholds being incorrect can lead to massive instability. With the results in Figure B.5 we can see that instability has been massivley reduced compared to with the same threasholds as shown in Figure B.1. Although this is indicitive of showing that we have more stability, the graph has the same shape as in Figure B.1. This means we could just be slowing down the rate of stability rather than solving it. There is the possibility that we have more or less duplications creating the stability comparitivly so we will need to look further into the data. This could also be because of the suicide rate being at a reduced rate, if this is the case then we should see that there will be a suggnificant increase in duplications comparitivly to the static heuristic.

Comparing Figure B.4 and Figure B.9 we can see that we have managed to stabilize for the random flucuations of the circular movement, this is also true for Figure B.8. However the fears of having more duplications is true in these two results where the number of duplicates doesn't seem to level off as much as in the static heuristic, this could be a problem and will have to be examined in the other results. Overall in this test it performed like it should have.

When comparing Figure B.3 and Figure B.8, we can also see that the stability has been made more consistent, however the deviation is reduced as number of duplications reduces. This is to be expected, however wasn't visible in the past circluar movement tests, this could be due to randomness or the algorithm.

With both circular movements the duplicates density graphs show data having a tendancy to hug the sides of the plane. This can be counted as partially onmouless because within viewing of the simulation some agents with the duplications of the data were hanging on the outer ages without connections therefore never suiciding making the duplication denisity in that area seem high.

There is not much to say about Figure B.7, it looks pretty much exactly the same as Figure B.2, except for duplications being lower and that stability converges nicely. These are all plossibly down to randomness in the simulation.

Compairing Figure B.3 and Figure B.8 we can see that stability has been increased, however it doesn't look like it due to the y-axis bounds being larger in Figure B.3.

Overall all test points the new dynamic algorithm has performed the same if not better in all of them. This shows that the dynamic thresholding works as inteneded to not affect the algorithm when things are running smoothly, and then step on the breaks when we are put in a bit less of a stable state.

4.3 Dynamic Heursitic with Migration

Within this section we review the results found of the algorithm described in Section 3.4. We did not compute threshold changes graph, this is because nothing would have changed since the last computation unless there was more than one public data to be learned. This is because the algorithm in our case needs at least three data points to be even activated.

The aim of this algorithm was to decrease the spread of duplications per agent. We can see in Figure **??**, that this is the case, however at the cost of instability in the system. We can also see that it also affects the spread in Firgure **??**, however by less of a degree but with roughlt the same increase to instability.

In the duplication ratios sections on all dyanmic heurisitic with migration we can see the spread has been increased in size, espically in the semi-staic movement data. This could be an affective way of creating an artifically larger area with also keeping the duplications down.

When comparing this algorithm to the prevouis algorithm not much has changed, the affect of memory spread to the instability is a cost which will need to be investiagted for your persoanl needs of the project.

// Need to redo test data and ensure that we aren't losing more duplicates because of migrating to a agent already with said migrated data

5 Conclusion

5.1 Futher Improvements

There are some major improvements that could eb amde to this algorithm to make it much more effective, in performance and also computationally. First lets focus on computationally. Currently at the start of each agent cycle it will message each other agent for the infomation its needs to gain perspective of its local scope. This is inefficent because for every agent in the vicinity each agent will have to send a packet to it every time it cycles. If we flipped the packet sending the other way around we would take a hit on the memory required to run the agent however would be computationally faster. Each agent at the start of its cycle or end will tell other agents its infomation which will be updated in tables on each indurvdual agent. We can then use the internal tables to calculate what needs to be done. This would offer overall less transmissions over the network, and would speed up the frequnecy of which we can run the agents control loop.

For performance of the algorithm there are many possible changes that could be improved, ranging from better parameter adjustments either from machine learning techniques or hardcoded adjustments. This would give diffrent desirable affects based off what is currently happening in the network, an example of this is for stability. Currently we have it restrict replication however there might be more effective ways to control stability changing multiple parameters at varying scales.

With our ability to solve for corrilated failures, we lack range of which a corrilated failure can happen. To be able to handle corrilated failures that could be much larger, larger than the average distance to the point. Soloutions to this problem could be after a certain range we make sure that duplications have no other duplications in their vissinity, and lock the data from suiciding. This wold mean that data further away would survive with minimising density. To get the data out their currently relys on movement as can be seen in the diffrence in dulication density when comapring static movement to circular movement. To solve for this we could have a directionaliy component to migration, or after a certain distance we push data away from the data point.

// Possibly talk about the effectivness of how much data the network can store compared to the amount of data storage.

5.2 Conclusion

The algorithms proposed, can be seen to atleast solve the problem provided in Section 1.2. How effective they are compared to other theoretical solouions is a diffrent topic. With the first algorithm proposed in Section 3.2, we saw that it worked effectively for a simulation based of this problem. We had good results when coiming to spread, less so in a semi-static swarm, and also duplications were handled correctly. However when abstracting this algorithm to a real world soloution, floors became apprant in the algorithm which relyed to much on perfect communication, well picked threshold values and smaller sized corrilated failures.

Due to the thought expriment of abstracting this idea to real world soloution, some things within the heurstic needed to be changed. This was the instability of the system in certain scenrios, for example a more contrived situation was when the suicide threshold is to low. This would mean that agents would suicide to often and then other agents would replicate again, theerefore leading to useless transistions of data which was not needed.

To cmbat this we create some dynamic threshold changes so that we can slow down the instability using diffrent parameters. We directly put a time based slow on suicides and for replication we slowed it by the amount of local instability. This gave us the best of both worlds with fast growth of duplication and a slower release of the data. Overall this significantly improved the algortihms perfomance, espically when coming to a circular movement swarm. We can see that even though we slowed down the stability issues we didn't end up actually solving the issues, this could be seen with the fact that stability graphs had roughly the same shape but at lower magnitudes. This was touched upon in Section 5.1, on ways that this could be improved.

The third algorithm proposed in Section 3.4 is the most controversial of the bunch. This is because it sets out to do what it was propsed to do however we gaina natural boost to the instability due to movement of the data duplications from, agent to agent. We could see that this algorithm was affective for larger distance corrilated failures, however amount of duplications seemed to be affected more than prevouis algorithms, and in a circular movement swarm the affects on the memory spread weren't even that pronounced as in a semi static movement swarm.

Overall Section 3.3's and Section 3.4's algorithm are the best and should be the only to be used, unless there are extreme memory and computation constraints on your agents. Section 3.4's algorithm should only be used in cases were large corrilated failres could happen and instability is less of a deal. However this would still not be as affective as a soloution propsed in Section 5.1.

The algorithms propsed assume that data is highly important and that all agents in that area need to know that data as much as possible. This therefore makes them not as relevent if you were trying to adopt them to a cloud based soloution. When coming to a swarm soloution this is definitly no the most affect soloution that could be possibly created. If we ruled out hybrid based models, which would be much more affect as a storage soloution, due to knwoledge of where all duplications are and we could specifically give data priorityes on how many duplications to spread out and how many redundancys you want further away in the swarm. We talk about possible more effective ways to store the data in Section 5.1.

This project achieved what it set out to do, and has highlighted downsides and possible improvements that could be researched into further to improve this algorithms performance and adaptability.

A Code apendix

Algorithm 1 Agent's control loop

procedure STEP
    move()

    if Learned new private memory data **then**
        Replicate item to all agents in connection radius
        return True

    $dist \leftarrow$ euclidean distance to data
    $dupes \leftarrow$ (local) duplicates on agents / number of agents
    $pubspace \leftarrow$ (local) average space available / max public memory

    if $(1 - dupes) * b1 + ((\sqrt{8} - dist)/\sqrt{8}) * b2 + pubspace * b3 > repthreshold$ **then**
        Replicate item to all agents in connection radius

    if $dupes * p1 + (dist/\sqrt{8}) * p2 > suithreshold$ **then**
        Suicide data

    Iterate to next public memory data

    return True

Algorithm 2 Semi-Static movement

```
1:  procedure MOVE
2:      dirforce ← define vectors from other agents to self
3:      forces ← dirforce with magnitudes (0.24 − current magnitude)
4:
5:      Apply small force to center of map
6:
7:      face ← angle of resultant force on forces
8:      point at angle face and move forward by 0.0002
9:
10:     return True
```

B Results apendix

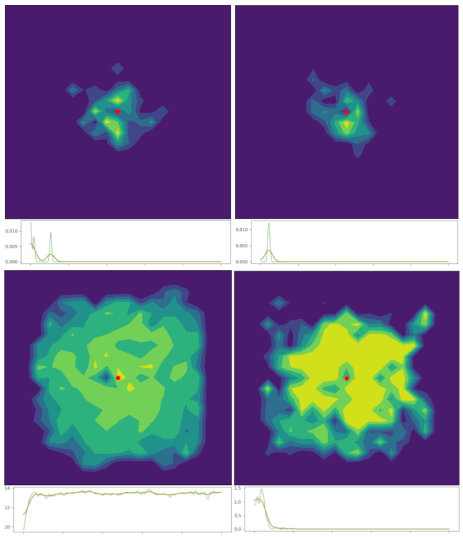

Figure B.1: Static Heuristic on semistatic movement swarm, with threshold changes

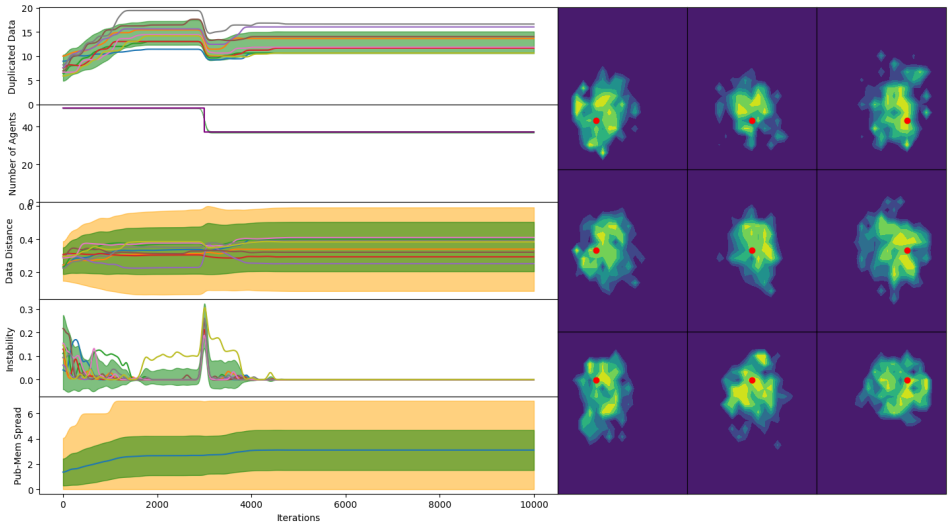

Figure B.2: Static Heuristic on semistatic movement swarm, with correlated failures

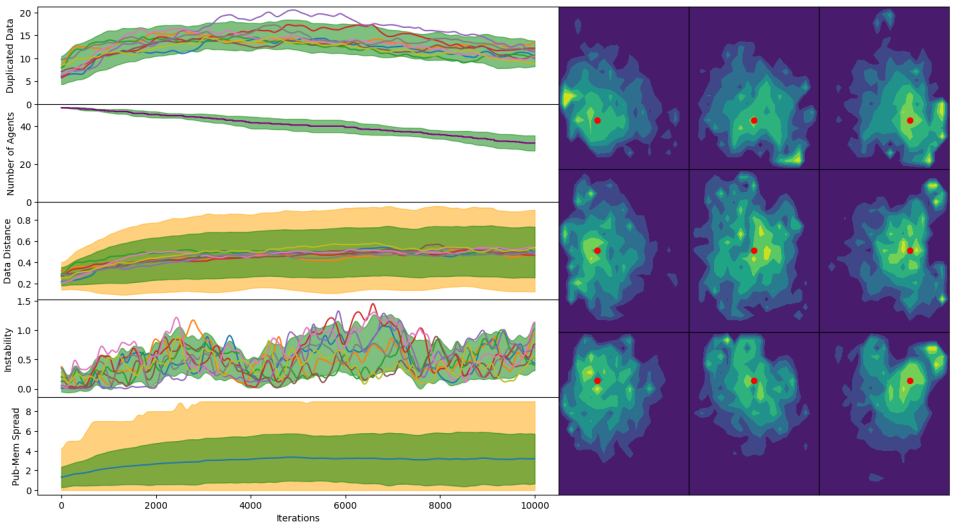

Figure B.3: Static Heuristic on circular movement swarm, with non_correlated failures

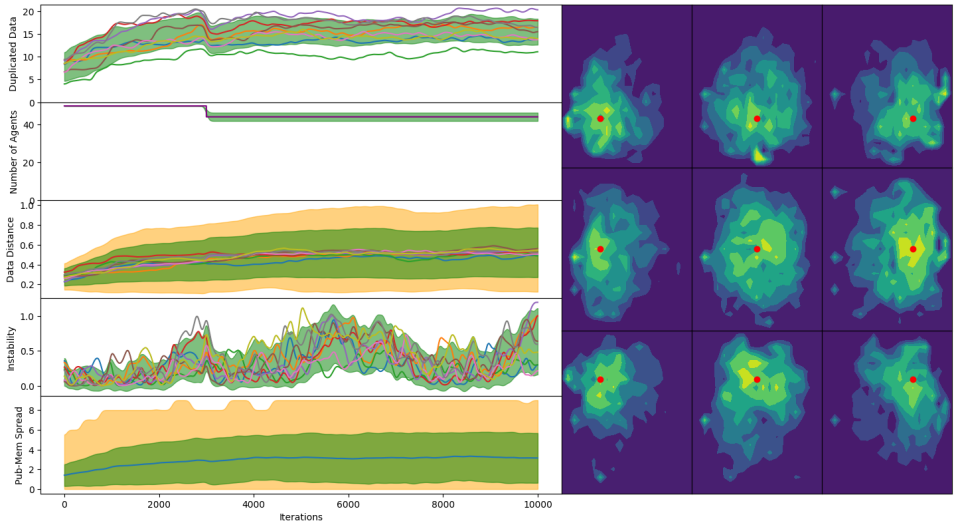

Figure B.4: Static Heuristic on circular movement swarm, with correlated failures

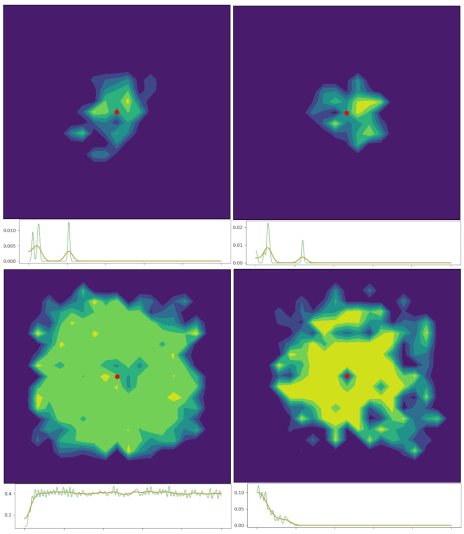

Figure B.5: Dynamic Heuristic on semistatic movement swarm, with threshold changes

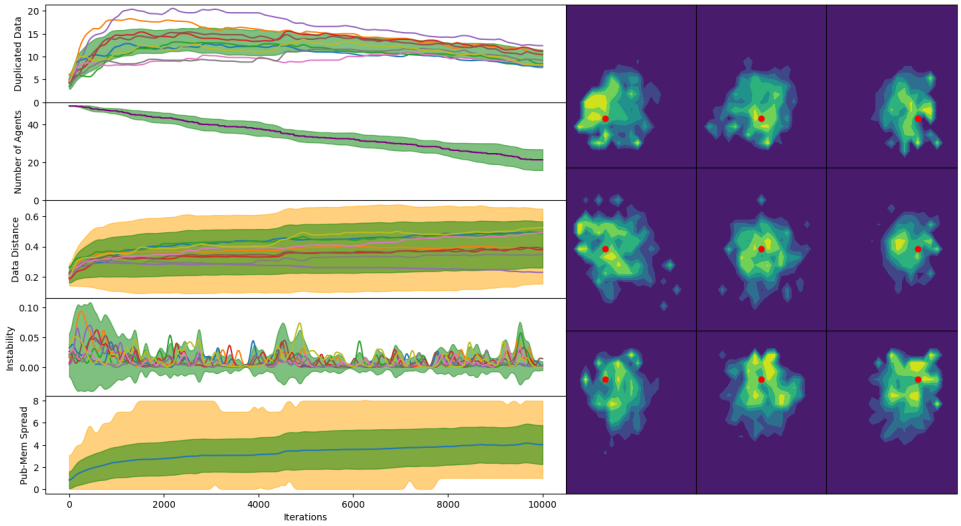

Figure B.6: Dynamic Heuristic on semistatic movement swarm, with non_correlated failures

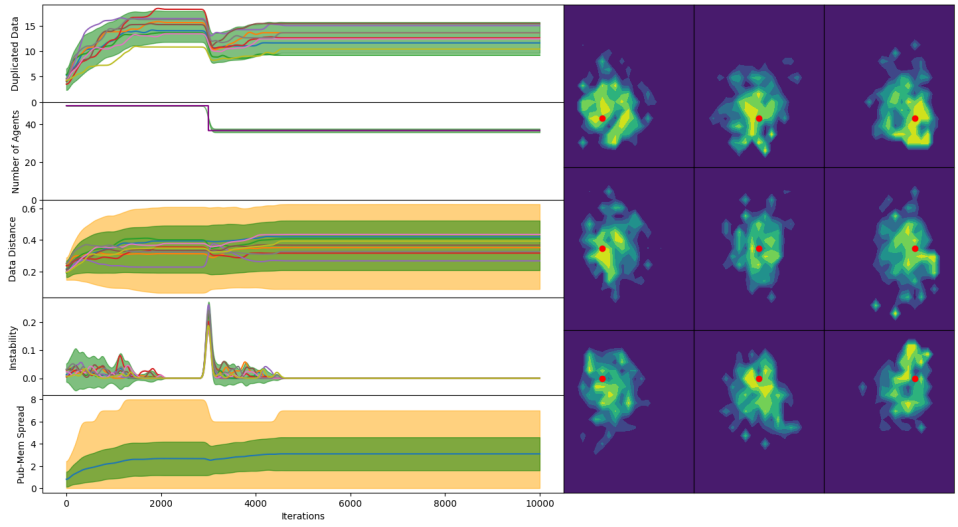

Figure B.7: Dynamic Heuristic on semistatic movement swarm, with correlated failures

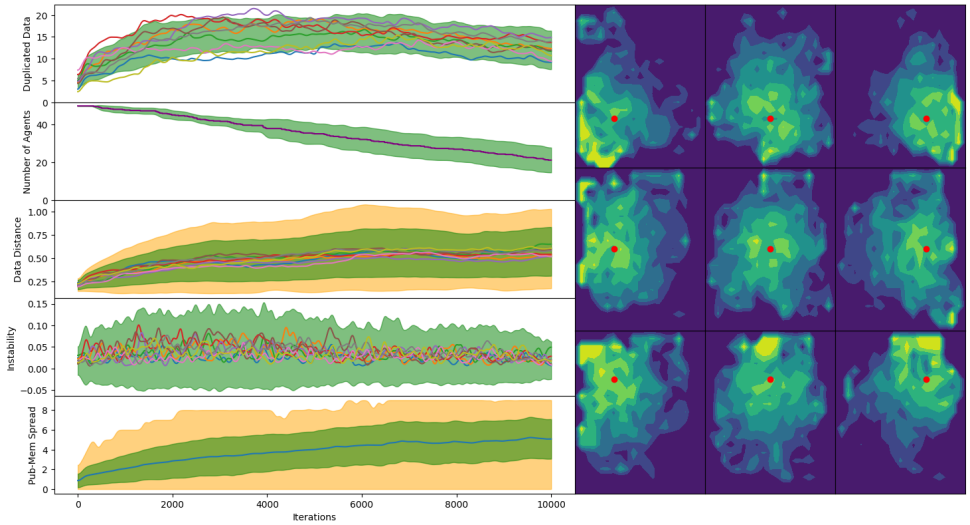

Figure B.8: Dynamic Heuristic on circular movement swarm, with non_correlated failures

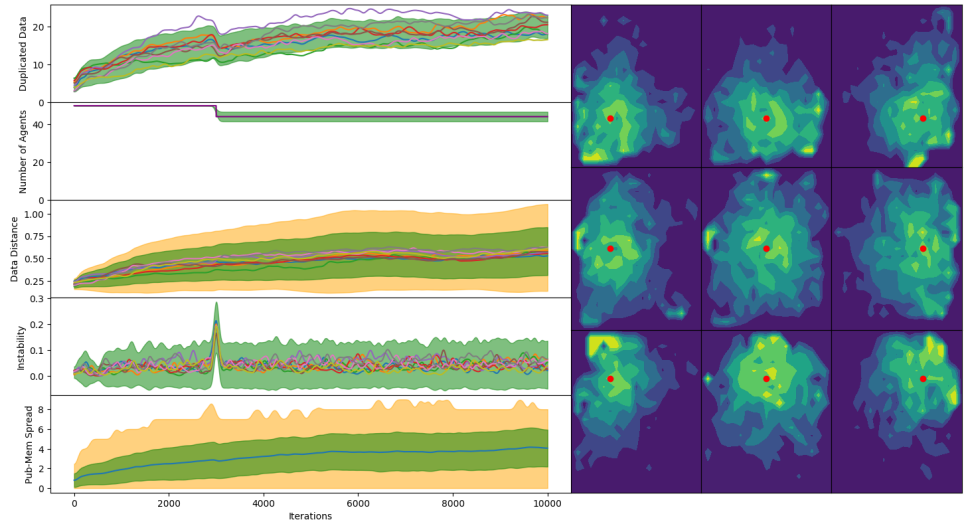

Figure B.9: Dynamic Heuristic on circular movement swarm, with correlated failures

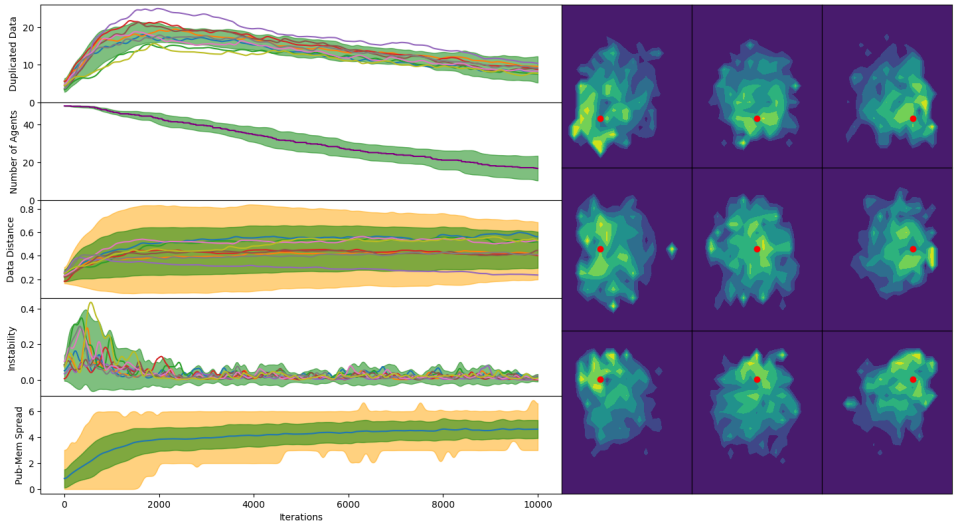

Figure B.10: Dynamic Heuristic with Migration on semistatic movement swarm, with non_correlated failures

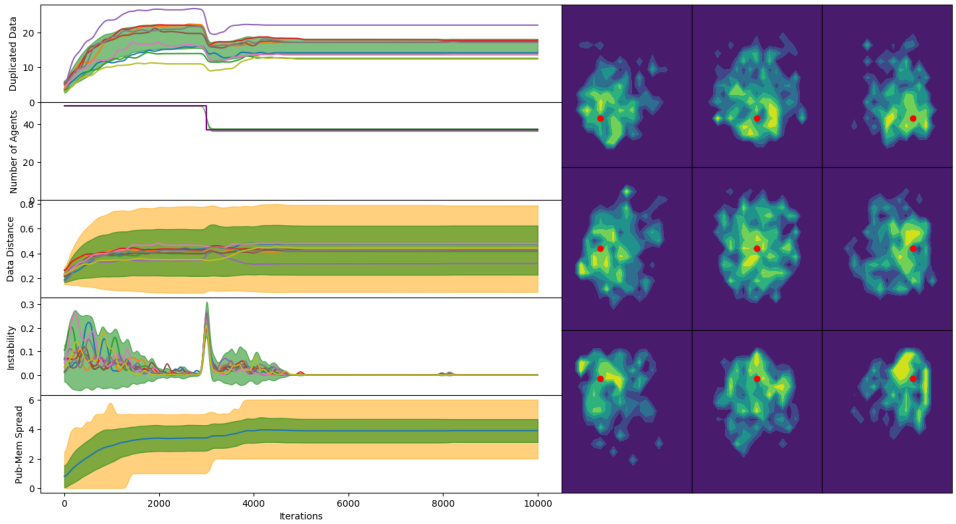

Figure B.11: Dynamic Heuristic with Migration on semistatic movement swarm, with correlated failures

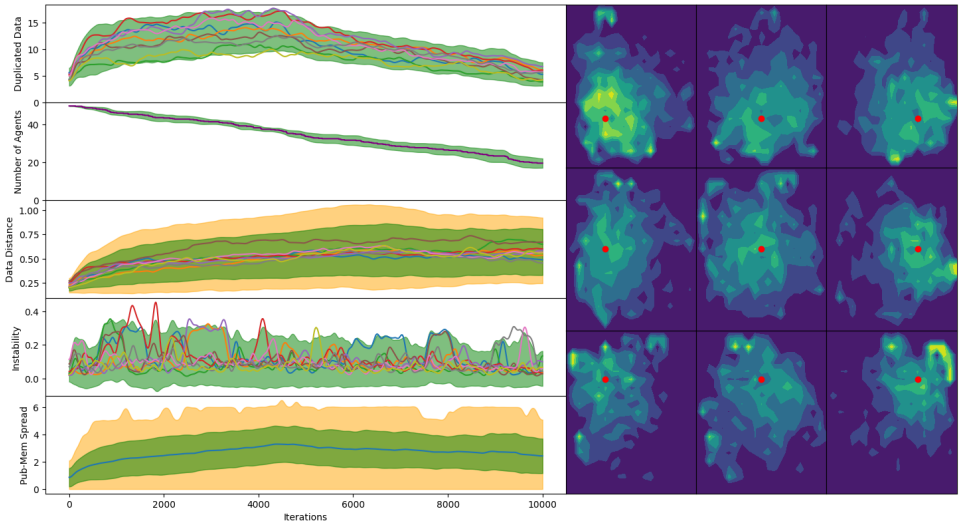

Figure B.12: Dynamic Heuristic with Migration on circular movement swarm, with non_correlated failures

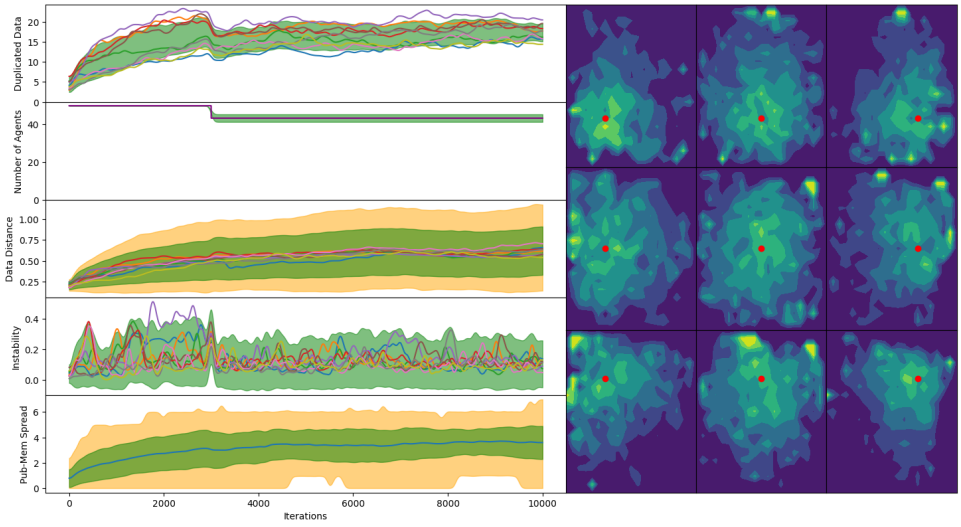

Figure B.13: Dynamic Heuristic with Migration on circular movement swarm, with correlated failures

Bibliography

[1] J. C. Barca and Y. A. Sekercioglu, “Swarm robotics reviewed,” Robotica, vol. 31, no. 3, pp. 345–359, 2013.

[2] V. Kumar and F. Sahin, "Cognitive maps in swarm robots for the mine detection application," SMC'03 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme - System Security and Assurance (Cat. No.03CH37483), Washington, DC, 2003, pp. 3364-3369 vol.4, doi: 10.1109/ICSMC.2003.1244409.

[3] H. Wang, D. Wang and S. Yang, “Triggered Memory-Based Swarm Optimization in Dynamic Environments,” in Applications of Evolutionary Computing, M. Giacobini, Ed. Berlin, Germany: Springer-Verlag Berlin and Heidelberg GmbH & Co. K, 2007, pp. 637–646.

[4] D. A. Lima and G. M. B. Oliveira, "A probabilistic cellular automata ant memory model for a swarm of foraging robots," 2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV), Phuket, 2016, pp. 1-6, doi: 10.1109/ICARCV.2016.7838615.

[5] E. Bonabeau, M. Dorigo, and G. Theraulaz, Swarm Intelligence: From Natural to Artificial Systems. Cary, NC, USA: Oxford University Press, 1999.

[6] L. Xiang, Y. Xu, J. Lui, Q. Chang, Y. Pan, and R. Li, ‘A Hybrid Approach to Failed Disk Recovery Using RAID-6 Codes: Algorithms and Performance Evaluation’, Association for Computing Machinery, vol. 7, p. 11, 2011

[7] C. Mims, ‘Why CPUs Aren’t Getting Any Faster’, MIT Technology Review, 2010. [Online]. Available: https://www.technologyreview.com/2010/10/12/199966/why-cpus-arent-getting-any-faster/. [Accessed: 01-Dec-2020].

[8] U. Troppens, W. Müller-Friedt, R. Wolafka, R. Erkens, and N. Haustein, ‘Appendix A: Proof of Calculation of the Parity Block of RAID 4 and 5’, in Storage Networks Explained: Basics and Applic-

ation of Fibre Channel SAN, NAS, ISCSI, InfiniBand and FCoE, U. Troppens, Ed. Chichester: Wiley United Kingdom, 2009, pp. 535–536.

[9] J. Liu and H. Shen, "A Low-Cost Multi-failure Resilient Replication Scheme for High Data Availability in Cloud Storage," 2016 IEEE 23rd International Conference on High Performance Computing (HiPC), Hyderabad, 2016, pp. 242-251, doi: 10.1109/HiPC.2016.036.

[10] N. Bonvin, T. G. Papaioannou, and K. Aberer, A Self-Organized, Fault-Tolerant and Scalable Replication Scheme for Cloud Storage. New York, NY, USA: Association of Computing Machinery, 2010.

[11] Legal Services Act. 2007.

[12] A. Prahlad, M. S. Muller, R. Kottomtharayil, S. Kavuri, P. Gokhale, and M. Vijayan, 'Cloud gateway system for managing data storage to cloud storage sites', 20100333116A1, 2010.

[13] B. Czejdo, K. Messa, T. Morzy, M. Morzy, and J. Czejdo, 'Data Warehouses with Dynamically Changing Schemas and Data Sources', in Proceedings of the 3rd International Economic Congress, Opportunieties of Change, Sopot, Poland, 2003, p. 10.

[14] 'Key-Value Scores Explained', HazelCast. [Online]. Available: https://hazelcast.com/glossary/key-value-store/. [Accessed: 02-Dec-2020].

[15] L. Lamport, 'The Part-Time Parliament', in Concurrency: The Works of Leslie Lamport, New York, NY, USA: Association of Computing Machinery, 2019, pp. 277–317.

[16] D. Agrawal and A. E. Abbadi. The tree quorum protocol: An efficient approach for managing replicated data. In VLDB'90: Proc. of the 16th International Conference on Very Large Data Bases, pages 243–254, Brisbane, Queensland,Australia, 1990.

[17] S. Lynn, 'RAID Levels Explained', PC Mag, 2014. [Online]. Available: https://uk.pcmag.com/storage/7917/raid-levels-explained. [Accessed: 06-Dec-2020].

[18] J. Hu et al., Eds., HiveMind: A Scalable and Serverless Coordination Control Platform for UAV Swarms. ArXiv, 2020.

[19] D. Calvaresi, A. Dubovitskaya, J. P. Calbimonte, K. Taveter, and M. Schumacher, Multi-Agent Systems and Blockchain: Results from a Systematic Literature Review. Cham, Switzerland: Springer Interna-

tional Publishing, 2018.

[20] L. A. Nguyen, T. L. Harman and C. Fairchild, "Swarmathon: A Swarm Robotics Experiment For Future Space Exploration," 2019 IEEE International Symposium on Measurement and Control in Robotics (ISMCR), Houston, TX, USA, 2019, pp. B1-3-1-B1-3-4, doi: 10.1109/ISMCR47492.2019.8955661.

[21] M. Y. Arafat and S. Moh, "Localization and Clustering Based on Swarm Intelligence in UAV Networks for Emergency Communications," in IEEE Internet of Things Journal, vol. 6, no. 5, pp. 8958-8976, Oct. 2019, doi: 10.1109/JIOT.2019.2925567.

[22] D. Jackson and F. Ratnieks, 'Communication in ants,'Current Biology,vol. 16, pp. 570–574, 2006.

[23] C. W. Reynolds, Flocks, Herds, and Schools: A Distributed Behavioral Model. ACM, 1987.