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 Literature Review

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 down the robotics side and can be seen as the study of practical implementation of swarms, whether that be movement or communication within the swarm. This is on the rise in the 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 two areas often are highly integrated rather than being 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 will be focusing mainly within said grey area, but predominantly in intelligence dealing with a practical problem.

Focusing down on distributed and local memory of swarm-like agents, that will be covered by this project. Most research 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 this 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 how invaluable it would be to applications like mapping of a dangerous area [2]. By being able to handle the loss of agents and the collection of data on agents with limited memory, we could ensure efficient solutions that have high redundancy compared to other implementations.

An explanation for swarm based memory management solutions to be a less developed area of study is the existence of subjects like cloud-based and raid based storage systems. The argument for having these two subjects partially-separated is due to the nature of a swarm's locality and ever-changing network style, compared to a server farm. Storage of data on an ever-changing network of devices is a hard task to complete, handling loss of connection between servers, reliability to access of the data and loss of services, whether it be non-correlated or correlated failures [9].

Most elements of cloud storage policies at a high level of abstraction could work effectively within a swarm based environment. However as explained above, key adoptions would need to be changed to create said effective policy. A prime example of the type of algorithm is "SKUTE" from [10]. "SKUTE" will be the main inspiration for this project's solution, too storage on 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 the objective of this report 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 the proposed solution has been derived and why said solution might be a relevant stepping stone for future research. This will be covered in the two sections, Section 1.3 of which goes into detail about current cloud based storage technologies and Section 1.4 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 this report will cover is defined as follows. We need to create a storage policy for agents of a swarm, to be able store directional abstract data as a collective, without complete duplication.

This problem can be split into two separate sub-problems. One is the handling of data duplication throughout the swarm as to be able to control for failures of agents, and provide a mechanic for recovery from said failures.

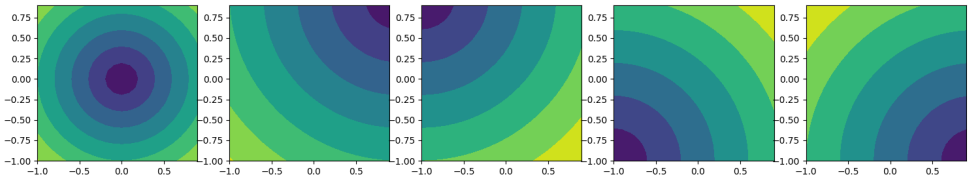

Figure 1.1: Data duplication density based off distance to datas desired location

The second is controlling where the duplicated data is within a world compared to where the data is needed in that world. A visualization of this can be seen in Figure 1.1, this shows the density of duplications within an area based on distance from the desired position of that data. As the reader can see the further we get from the desired point we should have less duplicates of data compared to agents without duplications of that data.

There are lots of examples where a problem like this could arise, a good example of where this could be applied is operating in a mine filed [2]. Near the start of the agents operation they will need to map out where potential mines might be and record their location. Obviously once an agent finds a mine we would want it to spread the word of the location of said mine. However due to memory restrictions the agents can only know about a minimal amount of mines, therefore agents closer to the mine should prioritize knowing its whereabouts compared to mines it may never come across. Now bringing this into a more of a practical solution if agents fail individually for example running out of battery, or as a group from a mine going off. We still need the swarm to not lose data, in the case of our example it would still be relevant to know where a possible crater could be.

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

1.3 Cloud/Backup storage policys/schemes

Like most things in computer science, this area of study used to be simple, with small data-warehouses and backups on to a medium like magnetic tape, following a grandfather, father, son backup policy. As the years have progressed, technology has become greatly complex requiring larger files to be stored and accessed frequently. Leading to the need for complex backup systems to provide availability and longevity of data stored, across a network or even locally. A component to the lead of complexity of these algorithms other than providing a service better than competitors is the Legal Services Act. 2007 [11]. This enforces that a company's cloud storage solution has to be reliable and fast in data collection for users.

Most algorithms used in production are called random replication policies [9], this is where data is partitioned and randomly distributed among other storage devices usually on different racks of the datacenter. This is an efficient design policy for handling non-correlated errors, however, lacks the robustness against correlated errors. These algorithms are substantial for long-term data storage with average popularity of collection/use of that data.

A non-correlated error is when devices go down randomly for example in a swarm an agent's battery might explode, destroying that singular agent, therefore it can be modeled by a random chance of happening on each member. A correlated error happens when multiple agents go down due to a reason unbenounced to us.

Following on from our swarm example, let us say a tsunami hits a certain section of the swarm destroying those agents therefore meaning that those failures were connected by some sort of event. In data-warehouses, this is usually a failure of power on a server or a bug, not as permanent damage as a tsunami.

Some of the problems arisen from random replication policy has spawned new replication policies, which can handle correlated and non-correlated errors, more effectively whilst also taking into account the demand of such items stored[9, 10]. Tackling these problems, two approaches were undertaken. The first approach is to come from a higher level of control where you have a manager which can choose items to replicate and where based on demand, knowledge of other replications and outside factors [9, 12]. This does not only apply to data but also schema changes of said servers or databases [13]. Working versions of these over a cloud service are tied in/together with a "Distributed key-value store" [14] where you have these key-value pairs on multiple devices on a network where duplication only leads to more fault tolerance of the data stored.

Another way to handle this which doesn't rely on a more privileged controller, and creates a distributed system is by having something like "SKUTE" as proposed in [10]. Each individual key-value replication has its own manager and can choose what it as a singular entity can do on that distributed system. The policy as described in "SKUTE" are as follows; Migration, Suicide, Replication, and Nothing. Migration is the moving of its data to lower costing and more redundant servers. Suicide is the removal of itself usually based on the number of duplicates and uses something like Paxos [15] to decide if it should suicide. Replication decides that it is being used enough to warrant the need to be duplicated.

An algorithm like "SKUTE" e.g. scattered key-value store [10, 16] will be best suited for swarm like agents due to the distributed nature of a swarm. This is less relevant in heterogeneous swarms with something like hivemind or hierarchal control however still relevant. For homogeneous swarms this is ideal due to not wanting to have a static/temporary leader because of the issues like communication bottleneck, power loss near the leader due to flow of information to the leader, loss of a leader in a hybrid static swarm [1, 5]. Also, one of the fundamental philosophies of swarm robotics is the inherently parallel nature that they bring to the table. When creating leader-based algorithms they do not fit within the spirit of the design of homogeneous swarms, unlike a heterogeneous swarm might.

An area of study which is stagnated is the storage of data on local disks. How to keep either backup for disk failure and/or improve write performance onto disks, rather than just duplicating data like as described above. An example schema for this is RAID, which has different levels based on the type of attributes that you may need for your array of storage devices [**?**]. In terms of cloud-based storage RAID arrays are used commonly internally rather than externally to a different NAS or storage server. This is because we will have the guarantees of RAID for internal disk failures if they occur, for example, a server goes down we have the above replication schemes to be able to handle that loss of storage. Leading to RAID arrays across multiple nodes to be sort of redundant. However, by using a parity [8] based higher-level schema you can get space savings on the duplication of data. This is harder to implement though and most likely not needed due to servers that are lost due to power outages usually coming back online pretty soon. An example of this is might be a power issue causing a server to restart.

// TODO: Say what a correlated and non correlated failure is

1.4 Swarm robotics

Within the research of swarms, there is a split between practical solutions of agents or fictitious agents for an algorithm. This split can be seen as swarm robotics and swarm intelligence. Swarm intelligence is where we use distributed swarm-like behavior to solve a problem, for example traveling salesman problem [5], this means that we use an agent like code to compute the solution to a task. These tasks have usually been solved using a different algorithm beforehand, like TSP using a genetic algorithm to solve, and the swarm algorithms like AS-TSP [5] are alternatives to that algorithm. Said algorithms provide benefits and drawbacks compared to their counterparts. An area in which these algorithms could excel, and researched, is the networking space [5] due to the natural parallelism that can be exploited from the design of a distributed solver. The main concept of swarm intelligence is creating a solver to a problem using a distributed algorithm that doesn't rely on global knowledge and can is adaptable on the fly.

This is not the route of research that will be needed for this project, rather we will be looking at swarm robotics research. Swarm robotics has the same concept as swarm intelligence, it focuses on tasks that are designed to have agents complete, mainly for the practical space like moving objects or mapping an area [2, 4], whether simulated or not. Swarm robotics focuses on the behavior of the swarm more than the solution that it gives us. These swarms come in three types: heterogeneous, homogenous, and a subcategory of homogeneous, hybrid swarms [1]. These three methodologies are mapped onto both the decision making and the agent's body/abilities.

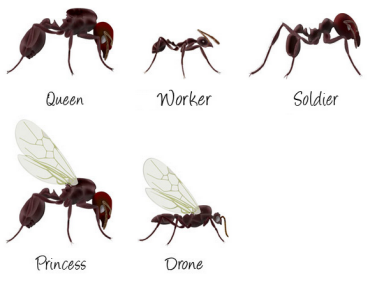

Figure 1.2: Example of a hetrogenus ant colony. https://www.pinterest.co.uk/pin/777363585651532845/

A heterogeneous swarm is where there are differences between the agents, as in Figure 1.2, whether physical or mental [1, 5]. These most commonly occur in nature and are not usually studied [5] due to the differences in agents being a rarely needed property in research-based problems. In

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. For example, as described in [1] with a mother ship being a navy boat and a swarm of quadrocopters, this examples shows how 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. Less research is done into this area is due to some key drawbacks of having a heterogeneous swarm. Usually, you will have a hivemind like a system if you have a heterogeneous swarm where you have leaders giving commands to subordinates or even one leader commanding the entire swarm. This is less desirable if there is a loss of those leaders you lose the ability to control the swarm, in our example, this doesn't matter so much due to if you have a loss of the mothership something has gone significantly wrong already. . Due to the differences in the swarm agents, it allows for greater efficiency of the swarm, having robots that can mine and that can farm exclusively. This leads to the vulnerability of major loss of one type of agent can lead to the loss of the colony.

With swarms like these to get the efficiency from a heterogenous swarm without leading to vulnerabilities, you need to have some jobs be interchangeable, e.g. agents are adaptable like in an ant colony [22]. Also, the jobs that can't be done by all agents need not be vital to the survivability of the swarm. Ants usually fit into this type of swarm where you have a queen, worker, and major ants, some ants like Leaf-Cutter Ants also have a subcategory of workers like a fungus farmer. If there is a significant loss of workers, majors start doing tasks that normally workers would do [5]. With the farmer subcategory, if a significant loss happens other workers/majors can take over that job and learn how to do it. This leads us more towards homogeneous agents within a heterogeneous swarm. This is one of the main reasons for not wanting to use heterogeneous swarms because human-made machines are less adaptable in this job switching method, without redundant hardware or software.

A homogeneous swarm is where each agent is the same, found less often in nature, and is more towards man-made agents. This is due to nature taking to a more efficient approach, having variety in a semi-homogenous swarm allows for greater efficiency, and being of a natural organism has more adaptability than a man-made robot agent [1, 5]. In homogeneous swarms we get significant redundancy, if an agent goes down we have swarms worth of replacements for that agent. With this redundancy, we gain possible losses of efficiency, either an agent is too simple therefore losing specialism or each agent has parts for specialism but may never need said part. An example of this homogenous flaw is we have a humanoid agent that has arms, and we want the swarm of agents to mine and farm. If the humanoid has arms only then it will take them longer to do these tasks, compared to having specific equipment for the job. Flipping this the other way around, if

each agent has equipment for mining and farming, some of these agents won't need both items and means the swarm would use more resources to create. In a homogeneous style of swarm, we have homogeneous control, also known a distributed intelligence, where all agents decide what they want to do based on what other agents are doing around them, an easy to understand example of this is [23]. This decision making involves internal parameters and can be equated to an emergent/structured swarm. Homogeneous control is different from heterogeneous control due to following a distributed problem solving/communication design compared to leader based design like hivemind or structured/hierarchical controlled swarms.

Within swarm robotics everything gets a bit messy, usually, there is no clear-cut name or design that can be assigned to swarm models and behaviors. This is down to wanting the agents to act in a specific emergent pattern rather than being stuck to a strict certain arbitrary rule, this is where hybrid approaches come into play. A hybrid approach is a mixture of both heterogeneous and homogeneous natures in both communications and agent design [1, 5]. In terms of communication when taking a hybrid approach to a swarm, you will have a swarm leader/leaders designated by the swarm, and handled with a consensus algorithm, e.g. Paxos [15]. In a hybrid model for agent design, if we want to gain the efficiency of the heterogeneous model and the adaptability/reliability of a homogeneous swarm we can use something like part-time tools. This allows our homogeneous bots to act in heterogeneous fashions.

Humans, themselves are a great example of a 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 that humans do. 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/humans can replace them by using the same tools and learning from the remaining agents of that task. 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 hierarchal power structure, compared to something of a hivemind model.

1.5 Motivation

As described above in Chapter 1 it is quite clear that swarm robotics is becoming and already is a soloution to many problems currently faced and going on into the future. With indurvidual speeds of computers stagnating [7], the world becoming more data oreintated and the drive for humans to explore, distributed technologys need to rise to the challenge. The next big step in computer science history being quatum computers, not solving the sequentail issues of our current problem solving algorithms. This is why swarm robotics/intellegince needs to brought to the for front of research.

In our day and age there doesn't seem to be much use of these innovative works, possibly due to the complexity or to the newness of said subject. The main applications nowadays are surevailence [21, 18] and delievery, however looking into the future, which we need to do otherwise we delay technological breakthroughs further, we can see that their are so many other uses for this type of distributed thinking. Whether it be pyshically with robots, like in space exploration [20], nano-robot medicene or military based applications. Or even conceptually like algortihms that can rely more on parrellel computation compared to their global sequentail counterparts. A good example of this distributed thinking changing and revolotionising our subject is block chain [19].

It is for this reseon that I have decided to contribute my part to this extensive and breakthrough field.

2 Main

2.1 Simulation Infomation

The base simulation that the policys will be running on is a specified. There will be an area of which agents can move around, each point within that area can have a peice of data learned in that area. When a peice of data is learned in that area , the data stores where is was created and the policys will take that starting place into account. This is how we have defined our directional data aspect to the simulation, in real life this could be something like a learnt behavoiur that is needed for that specifc enviroment, however, for test purposes it is just a string.

Each agent has two bounded partitions of memory, these are called private and public memory. Private memory is for knowledge that is learnt by that agent, and public memory is infomation that that agent has learned from other agents. Both public and private memory cannot have duplicate data inside including across partitions. This split was made arbitraraly and isn't needed for a practical soloution however was created due to agents being more likely to pick up learned data by not having to delete infomation in memory to make space for said leanred infomation. In general public memory should be larger in size than private memory due to there being magnitudes more public infomation compared to private infomation for an agent.

Agents are homogenus, and have the same connection radius of which they can talk to other agents within. Agents are assumed to be able to recieve and send packets whilst also doing actions like sending and recieveing itself, in real world soloutions this would likely be that each agent would need two recivers and senders or a control algortihm to handle this type of behavouir. In this simulation we assume that data transfer is perfect.

Each agent moves randomly in circles varying in shapes and sizes to gain a swarm like movement of connections being broken and reconnected. Agents our running in diffrent threads so run at diffrent speeds and acts more like a real world swarm. Data is introduced at the start based of an amount wanted, and during the simulation data can be learned based an amount wanted.

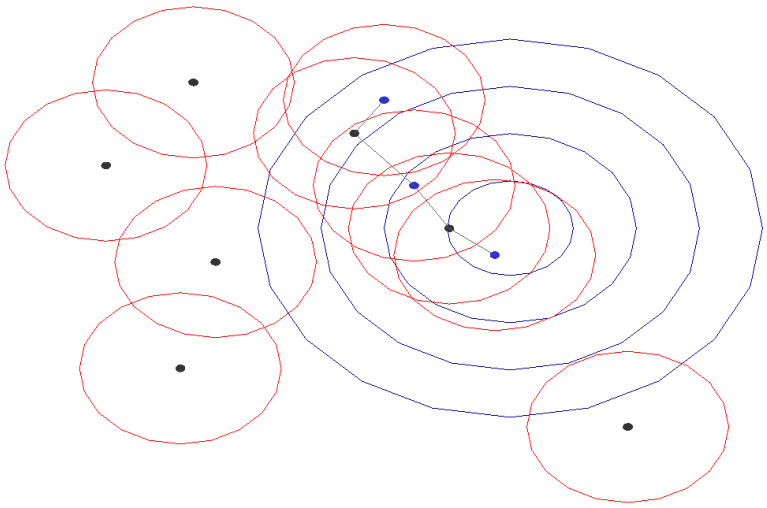

Figure 2.1: Example of simulation looks when running

In Figure 2.1 we can see an example of what the 2D simulation looks like. Dots are agents, and the red circle around them are their connection radius of which they can talk to other agents, these are usally omitted from being draw due to making the images messyer and harder to understand. The blue consentic circles are a positional datas stages of allowed duplications, this is explained more in Section 2.2. The connection lines between the agents are just showing that data can pass between these two agents, these lines change colour based of what frames are being passed across them, this will be touched upon more in specific examples due to colours meaning diffrent things. As a simple visuallisation of how data is being passed between the agents, in this example the agent is blue signifiying that it holds the duplicate of blue data, e.g. the consetric circle.

2.2 Simple Scattered Memory Policy

Taking on from the design of algorithm presented in [10], I applied a simple distributed storage policy onto a swarm that takes into account positional data. This policy has two actions that it can perform, suicide and replication, suicide is when a piece of data decides using a heuristic that it is not worth being stored in an agents memory so deletes itself, replication is where based on a heuristic the data believes it is worth spreading the infomation so will replicate to one member of the swarm.

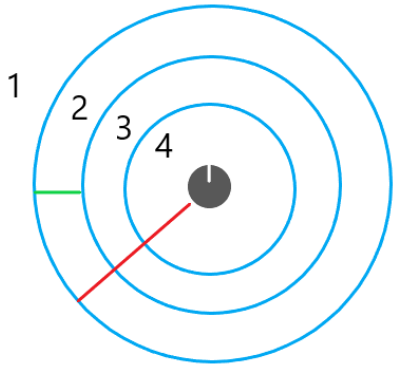

Figure 2.2: Poistional data heuristic for agents

The heuristic mentioned above follows these rules, and Figure 2.2 helps visuallise the conecept. At the grey agent in the middle is an example of when the data is learned, when learned this data will have a poistion that it was learned a radius at which you want the allowed duplications (including own duplicate) of that data to be one. The red line signifys that distance and the value in this example four is the maximum duplications of that data. The green line is just an easy way to compute which band an agent might be in.

As an agent moves closer through the gradient of allowed duplicates it will want to have more duplicates in its area, therefore allowing for a higher concentration of said duplicated data in that area. We can then use this simple heuristic per item of data to be able to get the distribution that we want, one implementation of this is described below in Algorithm 1. The reason for having a random replication when the allowed duplications equal the actual duplications is to try and breakthrough the boudarys at the edge where allowed dupes equals 1.

Without this replication we would have data sit inside the data area to much, this can be seen on some prilimany test results Figure B.1 and Figure B.2. Blue line is amount of agents with that duplicated data. Purple line is the mean distance of agents with duplicated data from the data point, Green area is the standard deviation and yellow/orange is the data range. A guassian filter has been passed over the results due to the mild fluctions every iteration creating a harder to see results.

From the results shown in Figure B.1 we can see that the distance from the area point likes to stay within side the data_radius bounds, this was tested on diffrent sizes of data radius and max duplications allowed and still found to hold true. We then tested the algorithm which has a random replication value to be able to try push through the borders of the data radius. We ran these on two diffrent values of replication value for five

times on each, Figure B.3 is with a random replication value of 10% when allowed duplicates = actual duplicates. Figure B.4 is the same values as before however with a replication value of 50% to see how roughly this will effect the results.

We can see that having the replication value higher keeps data at a more consistent distance away from the data point rather than dealing with the fluctuations of the circluar movement of the agents. From personal observations of the algorithm running it was observed that the data transfers became more volatile with the higher replication number. Volatility meaning that data was replicated therefore making actual duplications to high then suiciding to get back down to the correct value, this would repeat often. In a real world soloution this would not be ideal due to data transfers costing energy and possible data loss, this means that we need to design an algorthim that can take this into account more.

When comapring Figure B.1, Figure B.2, Figure B.3 and Figure B.4 we can impliy that the algorithm change doesn't affect the max distances like our desired outcomes want. With this algorithm we do get the desired affect when dealing with non-correlated errors as shown in B.5. In this graph we can see as the number of agents is decreasing the number of replications trys to fight that change, however some improvements might want to be made when a more thourgh investigation is done. Correlated errors were not tested currently due to the failures in distance from the data point, we want a larger min max range of values before this is tested.

2.3 Improved Heuristic

To improve from the last design was to create a new heuristic that can take more into account and have scalable values, that can be modifyied to get diffrent behavoiur from the swarm network. This was done by identifying diffrent factors that we want to play a role in the choose to replicate and the chose to commit suicide. These were:

- 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

The updated code can be seen in Algorithm 2, with this we can see that there are diffrent heurisitics for both suicide and replication. With the above factors we made sure that they were bounded [0, 1] by doing ratios of the

max value. Then we had parameter values that sum to one as to show the percentage of what the parameter effects the heursitic, this can be seen in lines 12 and 15. After these have been summed together the value is bounded between [0, 1] which we then set a threshold value for at what point we want the code benith to activate, e.g. Replication or Suicide. Changing of these parameters creates diffrent charatersistics as an example can be shown with just replication and no suicide, Figure B.6 and Figure B.7.

With this we can see that making the replication threshold higher decreses the likelhood of replication and gives a semi bound in a uniform density swarm and within non-uniform density swarm. In figures shown you will see there is usally a black agent close to the red cricle (Data target area), this is because the agent that learned the data has it stored in private memory so isn't counted as changing colour to show it is a duplicated piece of data in public memory. With both Figures we can see that suicide is definetly needed due to movement of agents as in Figure B.7 and within Figure B.6 we can see its needed due to all agents in an area around the point have the data. This goes against what this storage policy is meant to provide, we need a way to vary duplicate density within that semi-bound around the data target area.

To this we add in the suicide option which only takes into account the first to features meantion in the bullet points above. This will change the actions of the swarm to create a better policy as described in Section 1.2.

As we can see in Figure B.8 adding in the suicide ability and changing the threshold value creates diffrent duplication densitys within the replication bounds. In simple terms it changes the desnity of duplications closer to the data target area if the density of agents is uniform in desnity. When talking about non-uniform densitys it wil also take into account the desnity of agent duplications, however basically applies the same affect of having less duplicates where that is wanted, an example of this can be seen in Figure B.9.

With this current design it leads itself to diffrent observations based of parameter values. The right values need to be picked for the algorithm to work effectivaly, for example sometimes if suicide threshold goes to low it leads to massive instability with repeating duplicates and suicides which is wasted bandwidth and energy. This problem could be taken into account by also taking into account when the data was last replicated and allowing a sort of grace period within the heuristic to be able to allow natural movements to take those unstable agents apart. The heurisitc and threshold values lends itself towards a learning algorithm for example a gentic algorithm to pick the best value for the certain task or wanted behavoiur.

As a prelimanry test run on an average of five runs, Figure B.10, we can see that we have managed to create an algorithm that is more stable to the periods of the swarms movement, in our case circles. This was designed into the simulation to be able to test an algortihms abilitys to be able to handle fluctions in a repeating manner that can be visible. We can also see that the range from maximum and minimum distances has now been increased which was something we wanted in for improvement from Algortim 1. However we can see that the mean value is roughly within the middle of the maximum and minimum value which leads me to believe that the algorithm isn't performing as expected. We prefurably want the mean to tend towards being closer to the minimum rather than in the middle, this is to show the desnsity diffrence of duplicated data. This will have to be investigated further as to whether it is the nature of the heuristic or whether it was the values of the parameters which were picked manually.

There is also an example of this algorithm running on a non-correlated errors in Figure B.11. With this we can see roughly that the duplicates stay at about 40% of the total number of agents even with non-correlated errors happening at a very extreme rate. An observation within the simulations is some failures of agents can affect the total value suggnificantly until it gets into a stable state. Due to the nature of the heuristic it is deterministic and likes to hang in an optimum as much as possible, therefore some loss of agents doesn't change much. However sometimes an important agent is removed the duplicate layout changes quite quickly changing the layout across the swarm in a ripple like effect, until it gets back to an optimium (Stable state).

3 Conclusion

A Code apendix

Algorithm 1 Agent's control loop

```
 1: procedure STEP
 2:     move()
 3:
 4:     if item in private mem not in other agents public mem then
 5:         Replicate item
 6:         return true
 7:
 8:     item ← random public mem item
 9:     alloweddupes ← heuristic(item)
10:     currentdupes ← (local) number of agents with item in public mem
11:
12:     if allowed dupes > current dupes then
13:         Replicate item once
14:     else if  allowed dupes < current dupes then
15:         Suicide item using paxos
16:     else
17:         Randomly Replicate
18:
19:     return true
20:
21: procedure HEURISTIC(ITEM)
22:     dist ← agent distance to items target point
23:     alloweddupes ← items max dupes − (dist/items step)
24:
25:     if allowed dupes <= 0 then
26:         return 0
27:     else
28:         return allowed dupes
```

Algorithm 2 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

B Results apendix

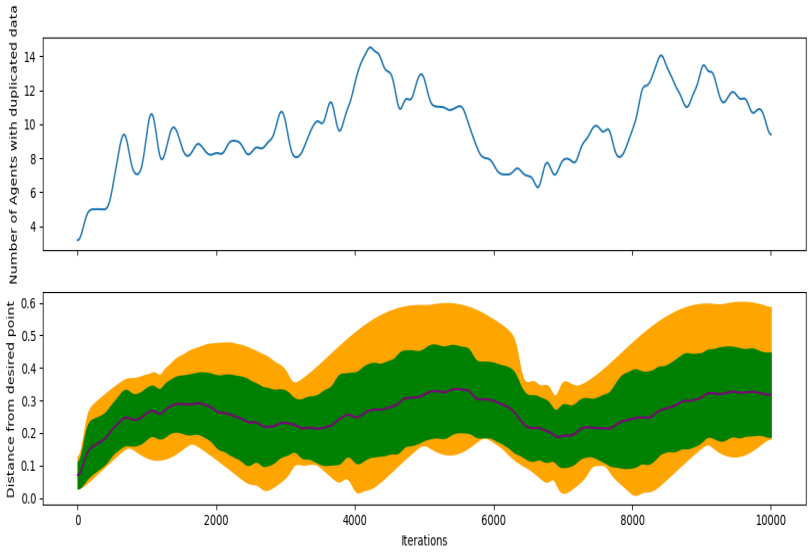

Figure B.1: Distance and number of duplicates on a run without a random replication factor

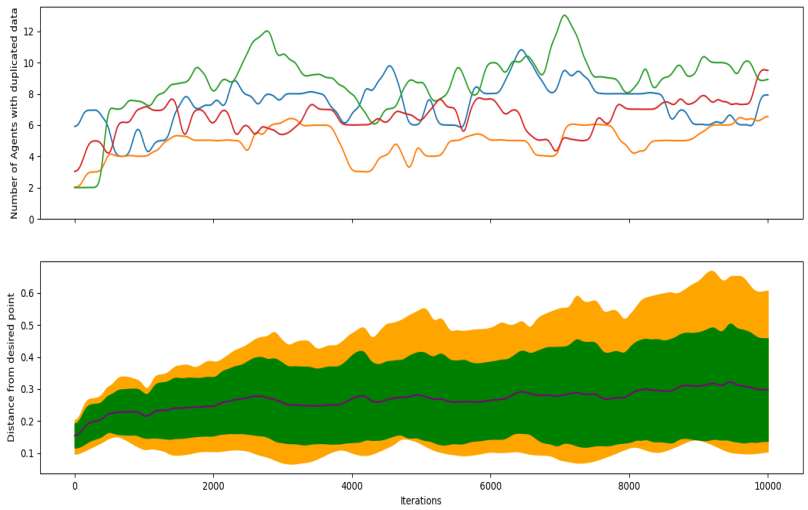

Figure B.2: Distance and number of duplicates on a run without a random replication factor

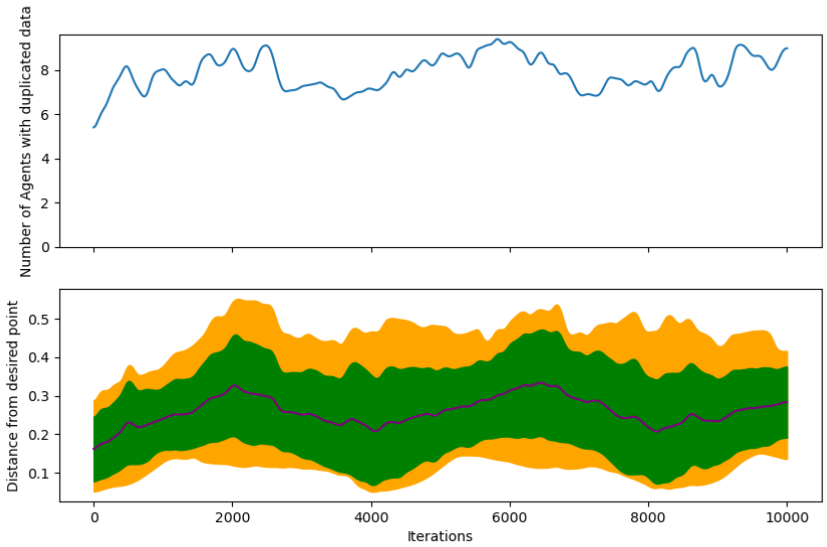

Figure B.3: Distance and number of duplicates on average of 5 runs with a random replication factor=0.1

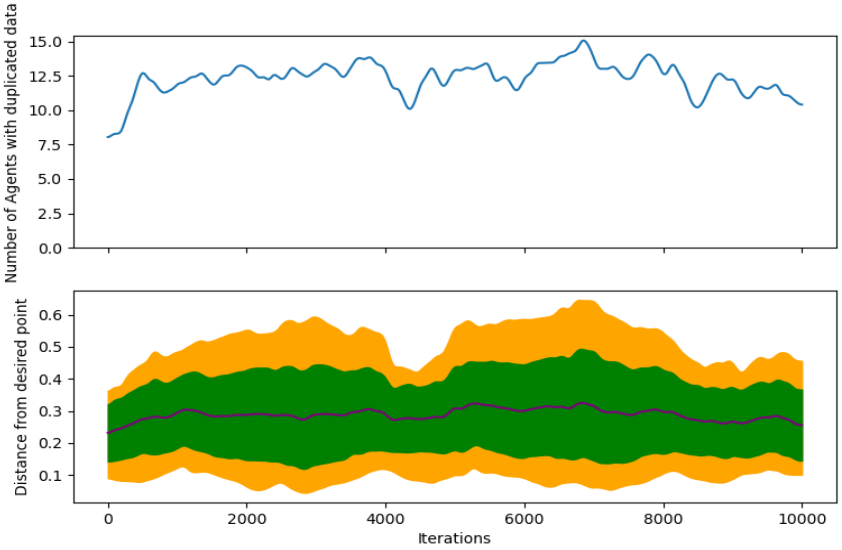

Figure B.4: Distance and number of duplicates on average of 5 runs with a random replication factor=0.5

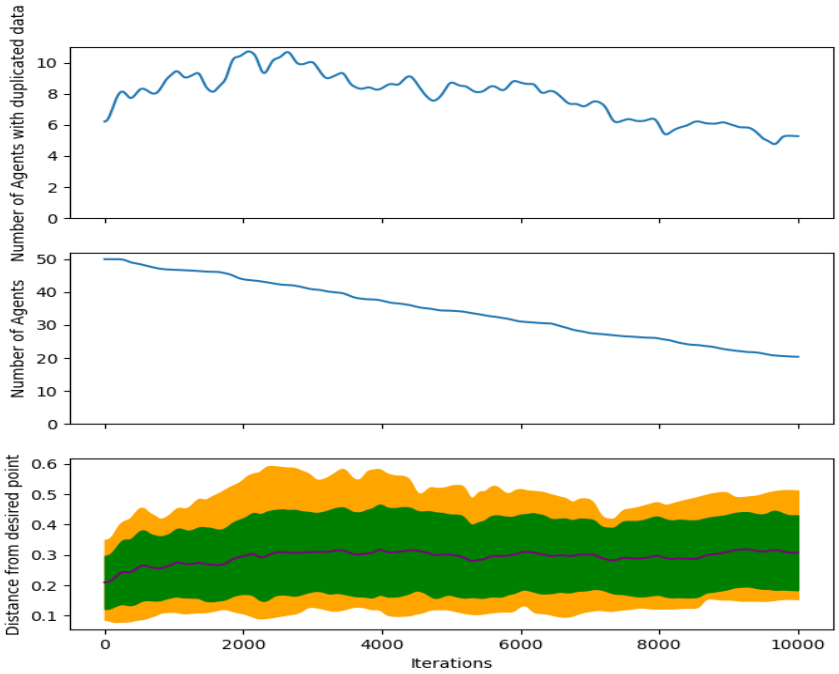

Figure B.5: Example of non-correlated errors on agents with a failure rate=0.003 per iteration

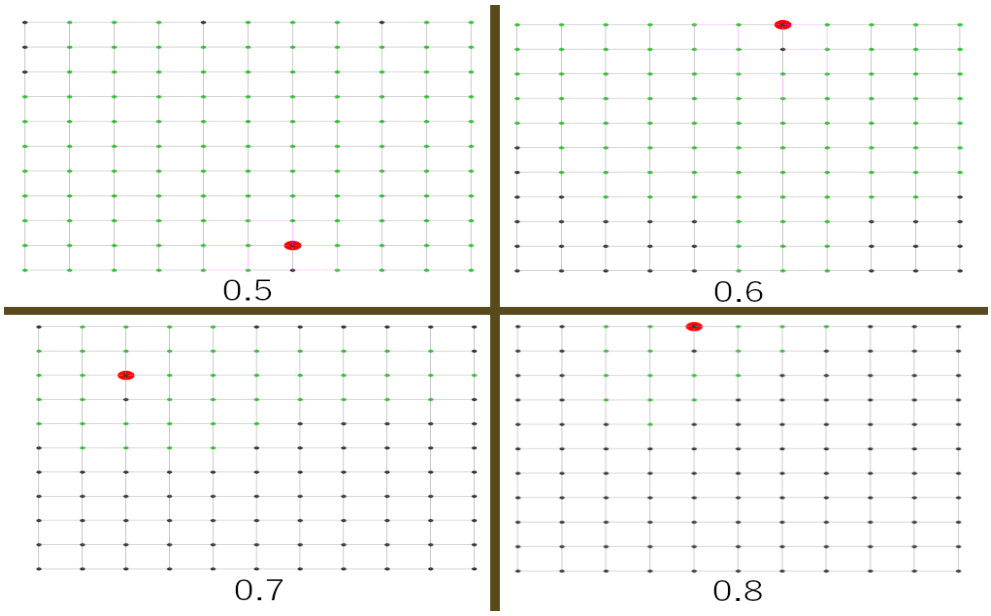

Figure B.6: Example of diffrent values of replication threshold on homogenus density swarm

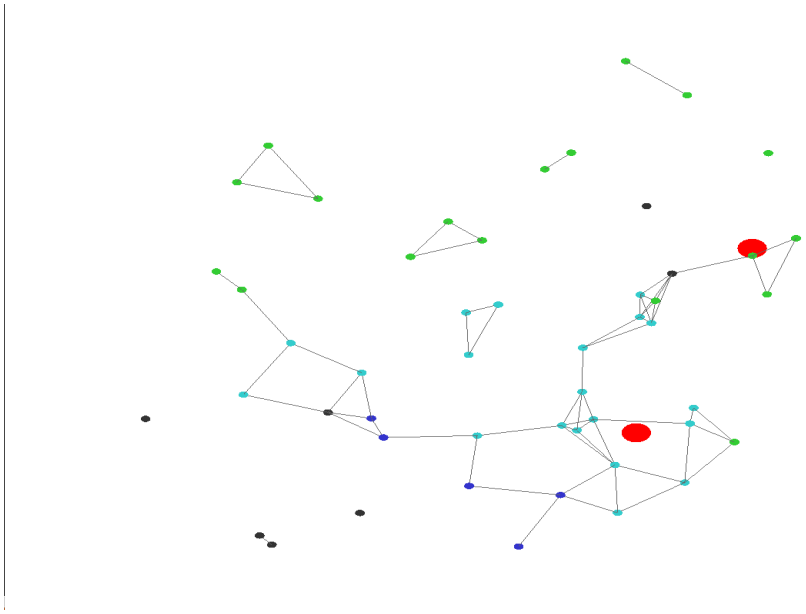

Figure B.7: Example of two datas on a non-uniform density agent network

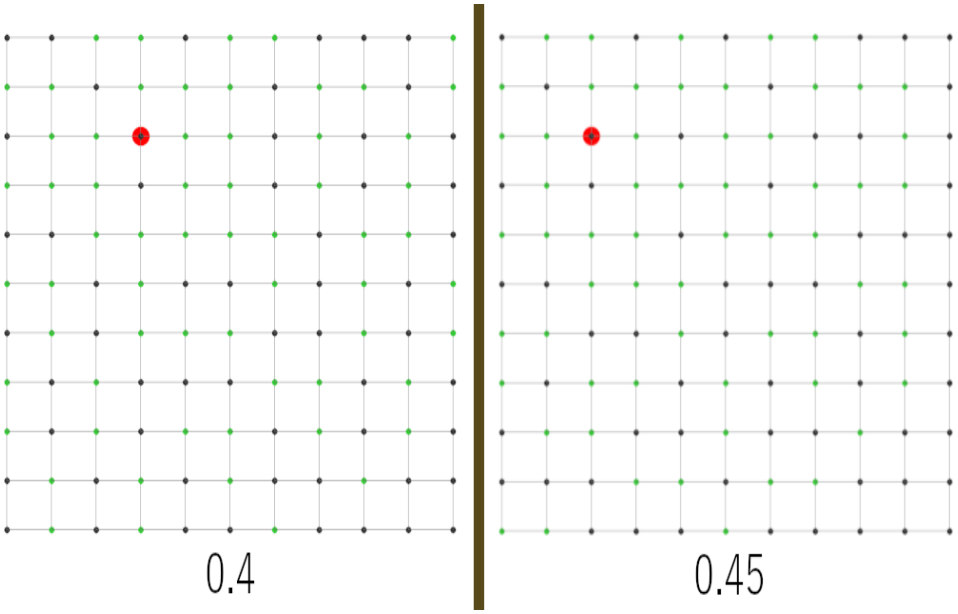

Figure B.8: Example of suicide threshold changes on data policy, on a uniform agent density

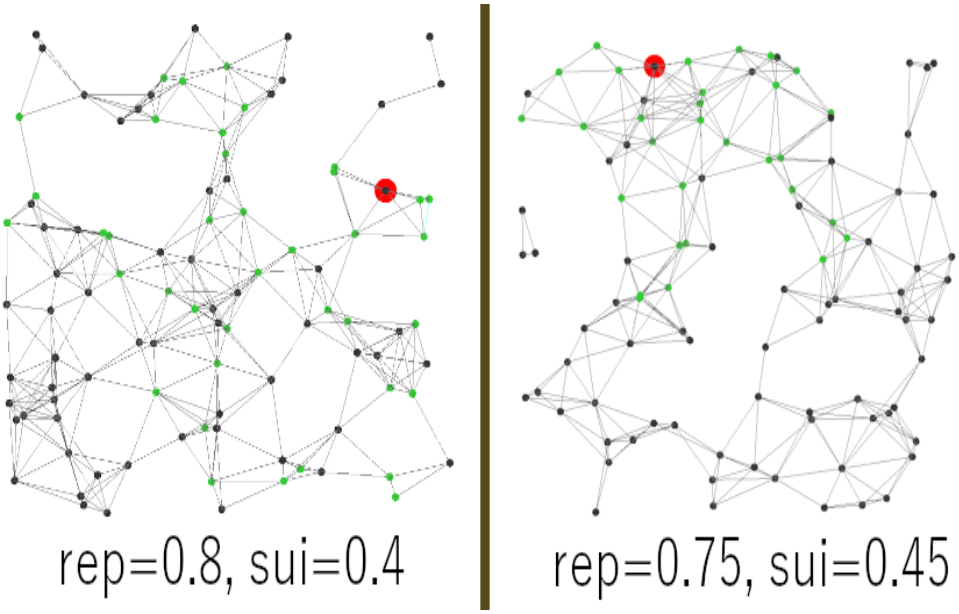

Figure B.9: Example of suicide threshold changes on data policy, on a non-uniform density

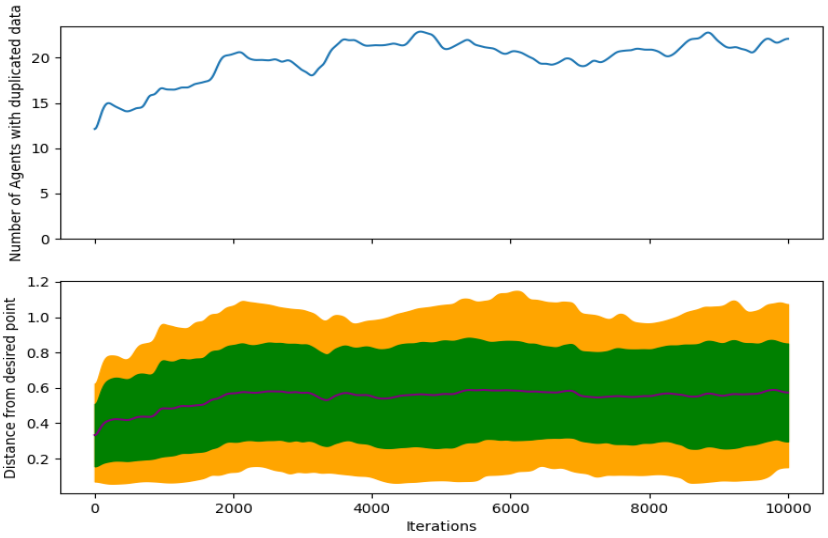

Figure B.10: Example of Algorithm 2, with no failures

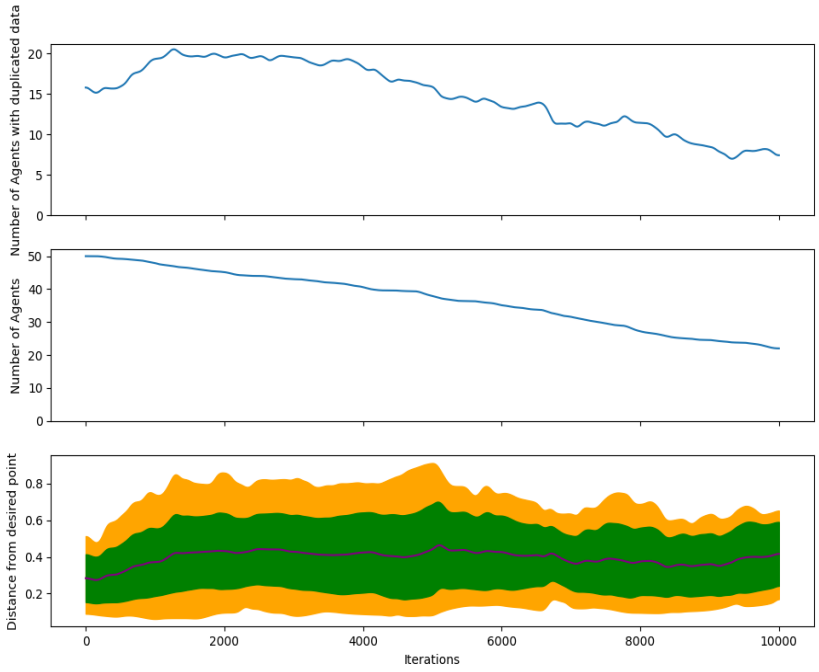

Figure B.11: Example of Algorithm 2, with failures at 0.003

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.