UiT THE ARCTIC UNIVERSITY OF NORWAY

Faculty of Science and Technology
Department of Computer Science

OUA: Observation Unit Autonomy

—

Emil Jønsson

Master thesis in Computer Science . . . June 2018

This thesis document was typeset using the *UiT Thesis LaTeX Template*.
© 2018 – `http://github.com/egraff/uit-thesis`

“So long, and thanks for all the fish.”
–Douglas Adams, The Hitchhiker’s Guide to the Galaxy

Abstract

More and more scientific equipment are being placed outside to monitor the environment. Such observation units can benefit from increased autonomy to function during harsh conditions and network partitioning. Environmental conditions like low temperatures can damage equipment. Therefor it is important for equipment to be able to autonomously launch countermeasures.

This thesis describes an autonomous task scheduling system called OUA: observation Unit Autonomy, which is designed to schedule tasks autonomously on a observation unit based the units internal and external states. To schedule tasks the system comes with a standard model that tells the scheduler what it should expect will happen when it performs a set of actions.

Not all observation units have the same hardware, or perform the same tasks, so a fixed model will be inaccurate. That is why the system has the ability to improve on the model based on the data it gathers about its internal and external state.

Experiments where run to determine if an observation unit could increase its internal temperature to avoid damage from the environment, and what those factors would allows the system designed here to be able to increase the internal temperature in the observation unit.

We have will in this thesis describe the architecture, design and implementation of a scheduling system, that schedules tasks based on temperature and network availability in order to increase the internal temperature of an observation unit.

Acknowledgements

I would like to thank my main advisor professor Otto Anshus and co-advisor Professor John Markus Bjørndalen at the University of Tromsø for their guidance, feedback and help through the process of writing this thesis.

I would also like to thank technical staff and administration at Department of Computer Science here at the University of Tromsø.

Finally I would like to thank my Mom for her endless support, when it comes to me pursuing a higher education.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

Monitoring the arctic tundra and seeing how its ecosystem is affected by climate change is important as it is more affected than most other ecosystems on earth.[1] A project doing this is COAT(Climate Ecological Observatory for the Arctic Tundra)[1][2]. Coat aims to monitor the arctic, as well as model and predict impact of climate change.

A problem posed when monitoring in the arctic is the cold. The solder on the hardware in the monitoring equipment will become brittle when it reaches below a certain temperature[3][4]. Another problem these monitoring device might have is intermittent connectivity, making it use resources to try and send data when it can't.

One of the things that can be improved on these devices, is to increase the autonomy of these devices, and have them make some of the decisions normally made by humans. By making these devices more autonomous, we could have them regulate their temperature to avoid them being damaged by the cold temperatures. Or have it decided what is the best time to send data.

We propose a system called OUA, observation unit autonomy. A system for autonomously scheduling workloads based on internal and external state of the observation unit. The system monitors the internal and external states of the observation unit, and based on the states it collects and its model it makes decisions about what to do.

Since not every observation unit are affected the same way by things, a model that works for one observation unit might not be accurate for another. Therefor it has the ability to update the model it uses to decide what action to take(like increase the temperature inside the device by sending some data).

1.1 Idea

A platform for the observation unit to increase autonomous operations based on data about itself and its surrounding. This is done by collecting data that can be used to establish internal and external states. The purpose of this data is to make the observation unit more autonomous.

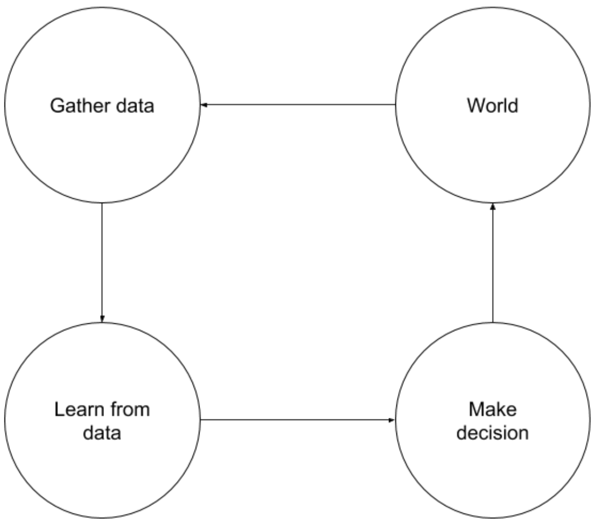

Figure 1.1: Idea for the system; data is gathered from the world, it learns from the data, and then makes an decision based on the data, and then that decision have an effect on the world

The technical requirements are:

- Collect data about internal and external states of the observation unit.
- Analyse the data concerning internal and external state, to create a model that tells it what the effect on an action will have
- Learn and improve based on the data
- Make autonomous decisions about what actions it should take.

1.2 Contribution

This thesis makes the following contributions;

- Lessons learned about how one can schedule workloads autonomously:
  - How to pick tasks
  - How to update model
  - How to starts tasks
  - How to gather states
  - How to analyse states
- Architecture, design, and implementation of a autonomous scheduling system for observation units.
- Insigth into if a process(workload) can be used to increase the temperature inside a device.

1.3 Outline

This report consists of 10 chapters including the introduction chapter.

Chapter 2 talks about the related work in the WSN and automation field. It describes the work and compares it to the OUA system presented in this thesis.
Chapter 3 shows a detailed overview of the architecture of the system.
Chapter 4 shows a detailed overview of the design of the system.
Chapter 5 goes into detail about how the system was implemented and what tools were used to build the system.
Chapter 6 explains how each simulation of a workload was implemented both for use in testing as well as when running the scheduler
Chapter 7 describes and discusses the experiments and the results that were done to see what if the heat generated from a process could increase the temperature inside a observation unit.
Chapter 8 discussion design and implementation decisions about the scheduler, workloads.
Chapter 9 conclustion concludes the thesis
Chapter 10 has sugestions for future work for ways to improve the system.

2 Related work

In this chapter we focus on previously related work. The related work we will be looking at is one where they use sensors to create an autonomous system. We will also look at how they differ from the OUA system we created.

2.1 Applying autonomous sensor systems in logistics—Combining sensor networks, RFIDs and software agents

The paper Applying autonomous sensor systems in logistics—Combining sensor networks[5], RFIDs and software agents, describes a system that uses sensors, RFIDs and software agents to autonomously monitor agricultural products.

The system monitors and reports on conditions that affects the quality and degree of ripeness in the agricultural products; temperature, humidity , illumination, acceleration/shock, and the gaseous hormone ethylene.

Instead of reporting detailed sensor information back to the end user the system reports statements about the freights conditions. To do this the system assessing unit configures itself to the specific agricultural product it is transporting. To do this it reads data about what it is transporting of a RFID labels.

Although both this system and the OUA system are autonomous system there are differences. One is that decisions about what to do under different conditions are made by the end user in this system, but in OUA are made by the system itself. Making OUA even more autonomous then this system.

2.2 Wireless intelligent sensor network for autonomous Structural Health Monitoring

The paper Wireless intelligent sensor network for autonomous structural health monitoring[6], describes a system that uses sensor nodes to autonomously monitor the structural health of civil infrastructure, such as bridges.

The sensors nodes in this structural health monitoring(SHM) system are organised into a two level architecture. The architecture has 3 types of nodes, sensor nodes, cluster heads and super nodes.

Sensor nodes are nodes that use little power and usually have a battery attached to them, cluster heads are more powerful nodes that are connected to a more powerfull energy source like solar panels or a power line. Supernodes, are like more powerful cluster heads with more computational power.

To reduce energy use, one thing the SHM system does is to have the sensor nodes send data to the cluster head/super nodes. This is so it can broadcast the data long distances but also so that the sensor nodes does not have to do any computational work, allowing it to save energy.

Unlike the OUA system, here they don't need to schedule work. This is because the sensor nodes all send their "work" to the cluster heads or supernodes.

2.3 Automated Irrigation System Using a Wireless Sensor Network and GPRS module

The paper Automated irrigation system using a wireless sensor network and GPRS module[7] presents an approach for irrigating agricultural crops using a distributed wireless sensor network. They describe the development of an automated irrigation system that uses microcontrollers and wireless communication. The aim of the system was to show that automated irrigation would reduce water use for agricultural crops.

The automated system consist of two components, a wireless information unit and a wireless sensor unit. The wireless sensor unit collects sensor data about moisture and temperature, packages the data, sends the data to the wireless information unit and then sleeps. While on the wireless information unit, the data is received, identified, recorded and analysed. The wireless information unit makes decision about when to run the irrigation system based on the data it has gathered from its wireless sensor units.

The main difference between this system and the OUA system is that here data is gathered by a set of nodes, and sent back to a single node to make a decision. While on the OUA system decision are made locally.

2.4 ResiDI: towards a smarter smart house system for decision-making using wireless sensor and actuators

The article "ResiDI: Towards a smarter smart home system for decision-making using wireless sensors and actuators"[8] describes an autonomous system for a smart house using wsn and actuators.

The system splits the nodes in the network into four types. Sensor nodes, a type of node that collects data. Decider nodes, a node that receives the data from the sensor nodes, processes it and makes a decision with the data. The decision is forwarded to the next type of a node, the actuator node. This node performance the action it is told to do by the decider node. The last type of node is the sink node, this node handles th external communications, performance updates and trains the neural network used for decision making.

The system has a trained an artificial neural network, that it uses to find the actions it should perform.

This system differs from the OUA system in 2 ways, it consist of multiple nodes, and each node does not make its own decisions, gather its own data, or execute its own actions,but rather they have different type of nods that have specialised in doing each type of job.

The second way it differs is that they use an artificial neural network to find the best action to perform.

3 Architecture

OUA is a platform that schedule workloads on a observation unit autonomously based on the internal and external states of the observation unit. The system architecture can be broken into three parts(see figure 3.1);

1. The states that affect the observation unit.
2. The analysis of the states that affects the observation unit.
3. Choosing a workload to run based on those states.

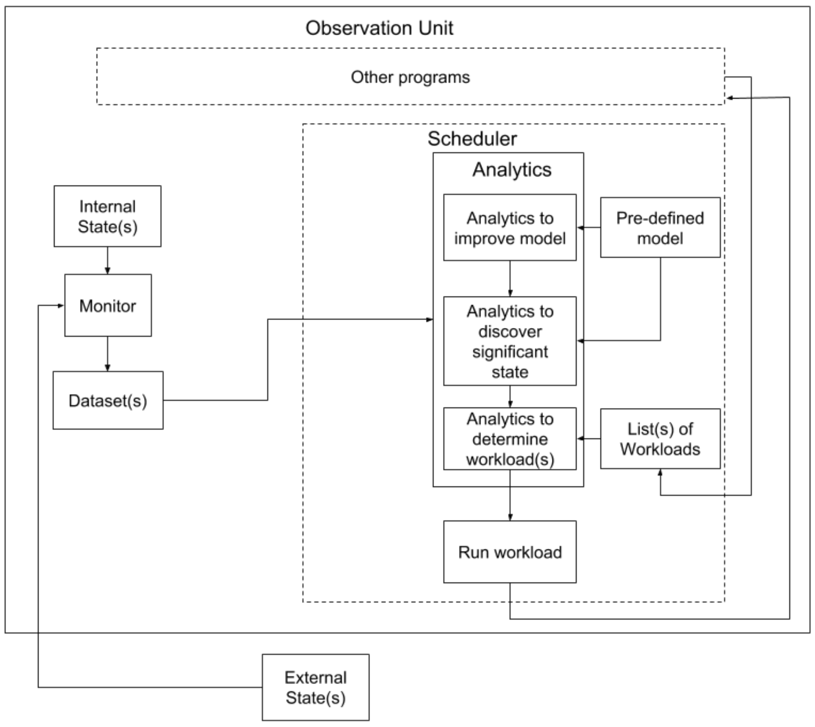

Figure 3.1: Architecture of system, arrows indicate data flow in system

3.1 State(s)

The states in the program are various variables(weather, temperature, internett access) the program has gathered and that the scheduler has to take into account when making a decision to do something. The scheduler has two types of states, external and internal.

3.1.1 Internal

Internal state is data about the state inside the observation unit, that the observation unit have gathered itself. The internal state can both be states the the observation unit have some degree of control over(battery, power usage) and states that it has no control over(temperature inside the observation unit).

3.1.2 External

External state is data about the state outside the observation unit, that both the observation unit have gathered, as well as other data from other sources. The external satte is a state that the observation unit have no control over(i.e temperature outside the observation unit)

3.1.3 Monitor

The OUA system has functionality called a monitor. Its job is to gather the internal and external states. Sources for it;

1. Sensors reading states outside and inside the observation unit.
2. Log of successful package send times

The monitor checks the sources, and retrieves a set of states from the sources. Once it has gathered the states it formats the states according to a set of specifications, turning it to a string of formatted data. After that the formatted data is written into a local dataset that contain the same type of formatted data.

3.1.4 Dataset(s)

The dataset is a collection of data about the states gathered by the monitor. These datasets that are created on the observation unit are stored locally on the observation unit. The datasets are formatted data organized into a data structure.

3.2 Analytics

The datasets are used by the scheduler for analytics. The scheduler has three types of analytics;

1. Analytics using the data to improve model.
2. Analytics using the data to find significant states.
3. Analytics to find the correct workload.

3.2.1 Analytics to improve model

To begin with the observation unit has a predefined model, this is a general model for an observation unit and not specific for the this observation unit. This model was created using analysis before the observation unit was deployed.

The first type of analytics the scheduler has, is looking at all the states we have gathered and analysing them. This is to get a more accurate model of what happens when we perform a set of workloads in a certain state.

3.2.2 Analytics to discover significant state

The second type of analytics analyses the most recent state and if it is discovered that this is a state where the scheduler wants to do something based on the model it has, it will move on to the analytics to select a workload for that state.(subsection 3.2.3).

3.2.3 Analytics to select workload

The last type of analytics try to find the appropriate workloads to do(from the lists of workloads) when a significant state has been found.

From the list of workloads a workload or a set of workloads are picked to be executed. The workload or workloads that are picked will be the one that best meets the requirements for the workload or workloads needed(requirements such as, do we have internett now) in that current state.

3.3 List(s) of workloads

The scheduler has a list of workloads that other programs wants the scheduler to run. These workloads can be both workloads that needs to be executed at a certain time or workloads that can be executed when the scheduler wants. The workloads that are going to be scheduled are added to the lists by the programs themself.

3.4 Run workload

All the workloads that are added to the lists of workloads are started by the scheduler. When the scheduler decides that it want to run a workload, it selects a set a workloads from the list of workloads, and then starts them.

4 Design

The previous section described a high level view, the architecture, of the OUA platform. This section gives a more detailed design overview(figure 4.1 and 4.2) of how the scheduler works by looking at how the workloads are found and started and how the states of the observation unit is gathered and analysed

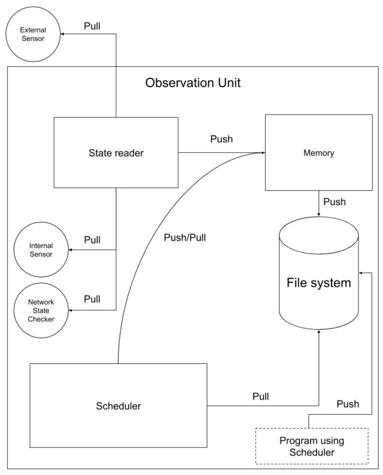

Figure 4.1: Figure shows design of system, arrows indicate who talks to who

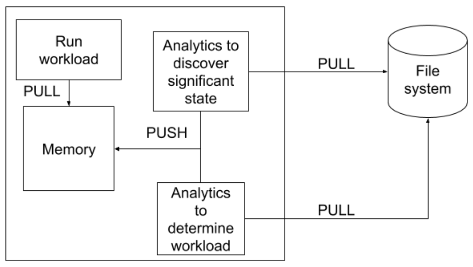
Run workload
PULL
Memory
Analytics to discover significant state
PUSH
Analytics to determine workload
PULL
PULL
File system

Figure 4.2: Figure shows design of the scheduler in the system, arrows indicate who talks to who

4.1 Workloads

The workloads that other programs need to have scheduled are programs that the scheduler have the power to start.

4.1.1 List of workloads

The lists of upcoming and available workloads that the scheduler has access to are a set of files that contain lists of workloads. These files contains the name of the workload, the duration the workload will need to run, and in some cases a timestamp for when the workload should run(in the case when the workload needs to run at a set time). The workload lists are written to by the programs using the scheduler and not the scheduler.

4.1.2 Scheduling workloads

Workloads that are to be executed on the Observation Unit are grouped in either one of two ways;

1. A workload that needs to be scheduled at a specified time.
2. A workload that can be scheduled whenever it is optimal for the observation unit.

Scheduling workloads at a set time

Some workloads need to be run a certain intervals or times. When the scheduler wants to run a workload, and start to look in the list of workloads. It will first check if any workloads are already assigned to run during the time period it wants to schedule work. If it finds that a workload is already going to be run during that period, it will stop looking for work to schedule during that time. And it checks when the workload will finish, and checks the state again after the workload is done running to see if it should run any more workloads.

Workloads that are assigned to run at certain times, will be started by the programs that use the scheduler(the scheduler has no control over this program, it is only notified through the workloads list that a workloads is going to run at a set time). This is so that the scheduler won't miss a scheduling that should have happened when it sleeps between schedulings.

Scheduling workloads at arbitrary times

Some workloads don't need to be run at certain times, and can be started when it best suits the observation unit. These workloads are under the control of the scheduler. When the schedulers model says to run a workload, it will find a set of workloads from the workloads lists that will run for t amount of time. These workloads will run until completed, and these is no limit to how long a workload can run for(in section 8.1 the effect of this will be discussed in more detail).

Since these workloads are under the control of the scheduler they are also started by the scheduler and not by the programs that add them to the workloads lists.

Making sure no workload starves

When picking workloads to run, it is important that we pick a workloads so that all workloads get to run at some point in time. To avoid the problem of making sure all workloads getting to run, the algorithm for picking a workloads starts by picking the workload that have been in the lists the longest. It then needs to check if the run time of the selected workload takes t amount of time to run or if its needs to find more workloads to run.

If the runtime is less than t amount of time. It will then check the workload that has been in the list the second longest. If it's run time + the run time of the workload already to the block of workloads that he scheduler are going is less then or equal to the max runtime that is sent for a block of workloads($t1 + t2 <= t$). Then we add it to the block of workloads the scheduler are going to run. If if is more then it will move on down the list of available workloads starting from the back, and continuing until it fills up the entire block of workloads the scheduler is going to run(figure 4.3 shows exampel of how it works).

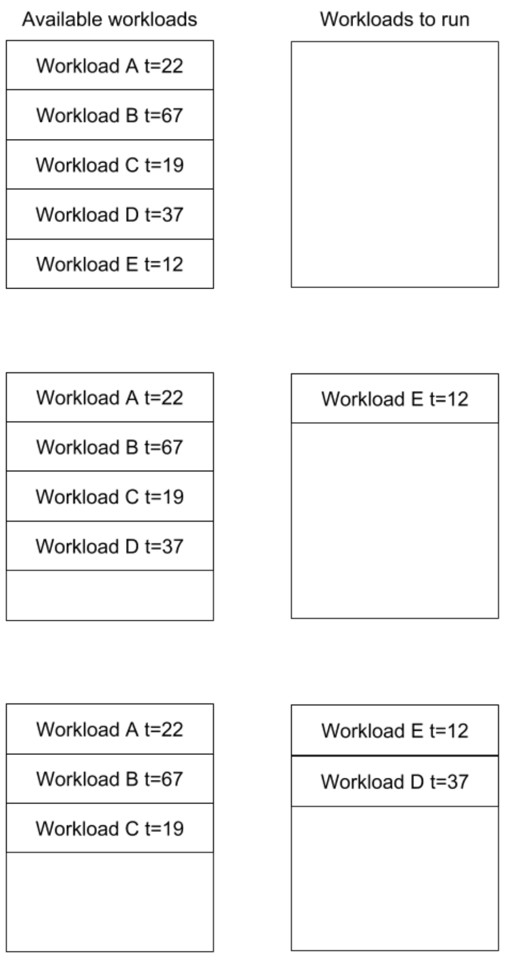

Figure 4.3: Shows how workloads are picked to fill an block of T time of 50, where it first picks workload E the oldest workload and then picks workload D the next in the list that was th best fit

Starting workloads

When the list of workloads that the scheduler wants to run is selected, it is time to run it. The scheduler starts by starting the first program in the list it has gathered as its own process. It will then wait for the workload it started to finish and then run the next in the list. It will continue to do this until the list of workloads are complete. Sometimes the scheduler has created multiple lists of workloads it wants to run concurrently. When this happens the scheduler will the start the first workload in each of the lists concurrently with each other, and run all workloads lists concurrently.

4.2 Model

The model of an observation unit is used to describe how the temperature changes based on the behavior of a action that the scheduler might do or a future state as a result of a action/non-action. When the observation unit reaches the threshold temperature set, it will schedule workloads based on the model(how long to run workloads for and how long to wait between each time it runs workloads).

The scheduler has two models; a pre-made model that it starts with, and a more accurate custom model that it builds as it runs workloads over time.

4.2.1 Premade model

A premade model is a model that was created to be used initially for scheduling until the scheduler itself can create a better and more accurate custom model specifically for that observation unit. The premade mode can be created by either a different scheduler, or by the scheduler it self(with the help of humans).

4.2.2 Custom model

Once enough data has been collected, the scheduler will create a new model based on the data it collected. And after it has created the model it will update itself with the model and use it when scheduling workloads. The new custom model is created the same way as the old model, the only difference is that it uses data that is specific for that observation unit, rather than data that is specific for another observation unit.

4.3 Collecting state(s)

The observation unit has datasets on four states;

1. The network state, which has data showing at what times the network connection is up.
2. The temperature inside the case of the observation unit.

3. The temperature of the CPU on observation unit.
4. The temperature outside the case of the observation unit.

4.3.1 Temperature states

The observation unit collects temperatures inside and outside the observation unit;

There are two temperatures inside the observation unit that is gathered. The first one is the temperature inside the casing containing the OU hardware. This temperature is gathered by having a temperature sensor read the temperature, and logging it to the file system of the observation unit. The second temperature gathered is the cpu temperature, this temperature is gathered by reading the cpu temperature and logging it to the file system of the observation unit.

The temperature outside the observation unit is gathered by a sensor that is outside the observation unit. The sensor data from the outside sensors are logged to the file system on the observation unit.

5 Implementation

5.1 Equipment and sensors

The system is implemented using Python 2.7[9]. The system runs on a observation unit that is a Raspberry Pi 3 model B[10] running Raspbian version 9("stretch"). To get the temperatures that was needed, two temperature sensors were connected to the Raspberry Pi(figure 5.1). One inside the protective case, and one outside the case.

For the sensors to find the outside and inside temperature of the observation unit, the DHT22[11] temperature and humidity sensor was used. To get the DHT22 sensors to work, the python adafruit DHT library[12] is used. To gather data on the CPU temperature the monitor function uses the gpiozero python library[13] to read the temperature.

To power the observation unit, it is connected to a power outlet on a wall. For the observation unit to have Internet it uses a wired ethernet connection because the observation unit is placed in a freezer for some of the experiments, and therefore cannot use WIFI to communicate as the freezer blocks the signals.

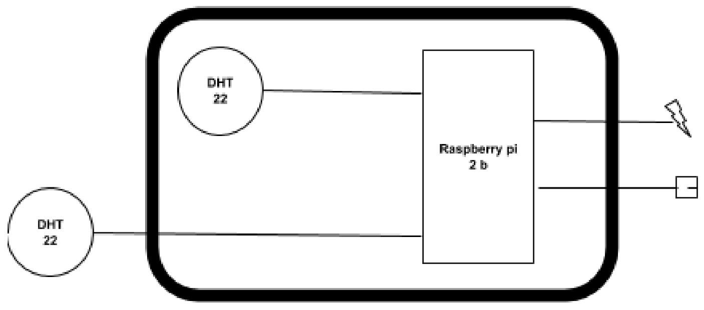

Figure 5.1: Figure shows how everything is connected

5.2 Model

To create any of the models, it requires that we find out two things. Number one is how long does it take to regenerate n amount of heat inside the case of the observation unit, and number two is how long does it take for n amount of heat to dissipate inside the case of the observation unit.

5.2.1 How the premade model is created

To generate the data needed for the premade model, the four different types of workloads are run like this for each workload:

Algorithm 1 Create data for premade model

```
1: for $EachWorkloadType$ do
2:     for $i < 25$ do
3:         run workload for 2 minutes
4:         sleep 10 minutes
5:     end for
6:     sleep for 1 hour
7: end for
```

Once the data has been gathered we first try to find how much heat is generated by a workload. For each dataset we have on the workloads, we compare the the temperature before a workload starts, and after a workload starts. We do this for the entire dataset for that workload. After that we find the average for those values. We now have the average temperature increase for a 2 minute workload. We repeat the same steps for each of the different workload types.

Next we take all average temperature increase values from the different workloads, and we average them. We can now with this find how long it will take to increase the temperature n amount. We use this for the premade model to define how long a workload should run for so it increases the temperature with n amount.

Next we need to find out how long it will take for the temperature to drop inside a observation unit after a workload has run. Here we go over each dataset for each workload. We compare the temperature after a workload has run to one of the temperatures that comes after. And we see how much the temperature has dropped. We then do that for the entire dataset. After that we take all the values of the drop in temperature and find the average amount of temperature the that fell after a workload from those values. We repeat this process for each type of workload. After that we take those average temperatures and average them to find the average of any workload. We can now use this for the premade model to define how long it should wait before checking if it should run another set of workloads.

Lastly we manually update the code with the model we created.

5.2.2 Updating the model

To update the model we need to gather data first. So first we use the premade model while we gather n amount of data. After the we gather the required amount of data we can start the analyses of the data to update the model.

Next step is analyzing how heat is generated by a workload. This is done by looking at the temperature before a workload and after a workload. We do this for the entire dataset. And then we calculate the average of those values. With this we can calculate how much how long we need to run a workload for to increase the temperature n amount. Then we can use this information to update the model with the new time it needs to run workloads for.

To update the mode for the time it should wait before it checks if it needs to run a workload. We analyse the data, by comparing the temperature after a workload, and the next temperature reading that is taken. We then do this for our entire dataset, and we average the results. To find the average temperature drop in a given time span. Using the average temperature drop over n time, we calculate how long it would take for temperature to drop n amount. Lastly we update the model with this information, so it can know how long to sleep between each time it checks if it needs to run a workload.

5.3 Scheduling algorithm

The scheduling of workloads is done by a scheduling algorithm(Algorithm 2). It first checks if there is a conditions that warrants a workload to run. The next step is to check whether a workload is already set to run now(a workload that has to run at a specific time), if such a workload is found. The algorithm stops and starts again when a new condition is found. If it does not find such a workload it checks if it should run network workloads, by checking the network model. If it finds out it should run network workloads, it will select the network workloads from the list of workloads. If it finds it does not need to run network workloads, it will pick any workload to run.

The scheduling algorithm model also eventually needs to be updated to better match the observation unit it is running on. But to do this we first needs to gather data, so the scheduling algorithm will first use the premade model, and when it determines it has gathered enough data. To update how long a workload block should be, it first waits until it has scheduled workloads N times. It then calculates the new values for how long a workload block should be(as described in section 5.2.2). To update how long it should sleep for it needs to gather N entries of workloads running and then waiting at least T time before starting a new workload. This is done by having timer keeping track of how long it was since the last workload ran. Once it has all this data it will update the model with new values that better match this observation unit.

Algorithm 2 Scheduling algorithm

```
1: while $True$ do
2:     if data collected >= N then
3:         Update Model
4:     end if
5:     if $Temperature <= MinimumTemperature$ then
6:         if $NoScheduledwork$ then
7:             if $NetworkAvailable$ then
8:                 Do workload that requiers network
9:             else
10:                Do any workload that dont requiere network
11:            end if
12:        end if
13:    else
14:        Sleep
15:    end if
16: end while
```

5.4 Network state

The network state in the program is just a model of the uptime of a network. This simulation is implemented as a list of times, with a start time and an end time for when the network is up. The network checker will look at this list of times and compare it to the current time. If the current time falls in the range of the one of times in the list, it will report the network sa up. Else the network is down.

5.5 Reading temperature

The reading of the temperature is done during two different times, one is at a set interval and the other is after the scheduler has run a block of workloads. To read the temperature at set intervals, we have a seperate thread that reads the temperature every 60 seconds.

5.6 workloads

For the workloads, the we have implemented simulations of 4 different types of workloads, a disk workload, a cpu workload, a memory workload, and a network workload(more about each workload in chapter 6). These workloads are implemented as python processes.

5.6.1 Starting a workload

All workloads are run as their own programs(processes). They are started by using the python subprocesses library.

5.6.2 Stoping workloads

All the workloads will either run for a very long time or forever. This was implemented so that we could more precisely control how long a workload runs for, by killing it. The way we kill a workload is a by searching with the process name and getting the PID. with the process PID we can send a kill command to terminate the process.

5.6.3 List(s) of workloads

Workloads are stored in lists(textfiles) on the local file system. When a program wants to add a workload it appends it to the lists. Adding it to the bottom of the file. When a program appends the workload to the file, it adds the name of the workload(the name of the python program) and the duration of time the workload runs for. For the workloads that need to run at a set time, it also adds a timestamp for when the workload will run.

6 Workloads

The system has four different types of work that uses a lot of a resource on a computer;

6.1 Disk workload

The disk workload is a workload consisting of a series of writes to a file on on disk. The workload starts by creating a file. After that it writes random 10240000 bytes(arbitrary large number chosen) to the file on disk using the pythons os.urandom method. The next step is to delete the file and then it starts over again creating the file and filling it up. It will repeat this process 1000 times(arbitrary large number chosen) or until killed by the scheduler program.

6.2 CPU workload

The CPU workload is a workload that try and use as much of the CPU as possible. To do this the workload loops over a range of numbers from 2 to 1000000 trying to find every primes. To find these primes it loops over the numbers and then for each number it compares it to all the numbers it has already looped over. The workload will continue to search for prime numbers until it reaches the max number(1000000) or until interrupted.

6.3 Memory workload

The memory workload is a workload that tries to put as much pressure as possible on the memory usage. To do this the workload reads a 3367 byte file into memory. Then the workload starts at the beginning of the file searching word by word for a specific word(this word is placed at the end of the file), once the word is found the word resets to the beginning of the position of the file in memory and starts again. It will keep doing this forever or until interrupted.

6.4 Network workload

The network workload is a workload that generates a lot of network traffic. To do this the workload is client that talks to a server. The client first creates a file if random 10240000 bytes generated by the os.urandom method. After that the clients opens a connection to a remote server and then reads a chunk of 65536 bytes from the disk, it then sends the chunk to the remote server. It will continue to send chunks until it reaches the end of the file we created. After that it sleeps for 1 second and then starts sending the file again using the same sending technique as described above.

7 Experiments

The experiments chapters consist of how insulation affects workloads ability to increase the temperature inside an observation unit, an analysis of how different types of workloads affect temperatures, and what parameters when running the workloads affect the temperature.

The system is configured with a Raspberry Pi 3 model B running the OUA system. A macbook pro(3,3 Ghz Intel Core i7, 16GB memory) runs the remote server for the network workloads.

All the measurements are performed on the raspberry pi.

7.1 Testing equipment

7.1.1 Insulation

To keep the observation unit warm, we need to insulate it. For insulation we placed the observation unit in thermal bags. To increase the amount of insulation(from 1 layer to 2 layers) we put the observation unit into a thermal bag and then placed that thermal bag into another bag.

7.1.2 Freezer

To run experiments in low temperatures a freezer was used. The freezer used for the experiments is a Matsui 98L freezer chest(M98CFW15E). The freezer has 7 temperature settings. The temperature in the freezer will go as low as about -20°C according the documentation(measurements during experiments showed it can go lower) when the settings is set to 7. For the experiments where the freezer was used, the setting was set to 6(setting 6 was chosen over setting 7 as setting 7 is used for freezing items. And it should be dial down from seting 7 after 24 hours according to the manual) and the temperature here is about -18°C(but can dip lower). The freezer temperature fluctuates up and down. When the freezer reaches a set temperature it will stop cooling(this can be seen on figure 7.6), and then the temperature starts going up again. When it reaches a set point it will start to cool down again.

7.2 Different levels on insulation

To measure how much of an impact thermal insulation has on the observation unit, we test the observation unit with increasing number of layers of insulation. We split the measurements into 3 experiments. Experiment 1 has no insulation just the protective casing. Experiment 2 has one layer of thermal insulation, and Experiment 3 has 2 layers of thermal insulation.

For all the Experiments here, we placed the observation unit running the OUA system in a freezer. While in the freezer it had a wired internet connection and power from the power grid.

We measure the workloads with different levels of insulation by running them as follows for each workload:

- Step 1: run a workload for 2 minutes.
- Step 2: sleep 10 minutes.
- Step 3: repeat step 1 and 2, 25 times for the selected workload
- Step 4: sleep 60 minutes.
- Step 5 select a new type of workload and start start at step 1 again.

The temperatures are sampled once every 1 minute for the CPU, inside the observation unit and outside the observation unit. The same temperatures are also sampled once before a workloads starts and after a workload is done running.

7.2.1 Experiment 1: no thermal insulation

On figure 7.1 we see the temperature inside the observation unit, outside the observation unit, and for the CPU on the observation unit, while the observation unit is idle with no insulation.

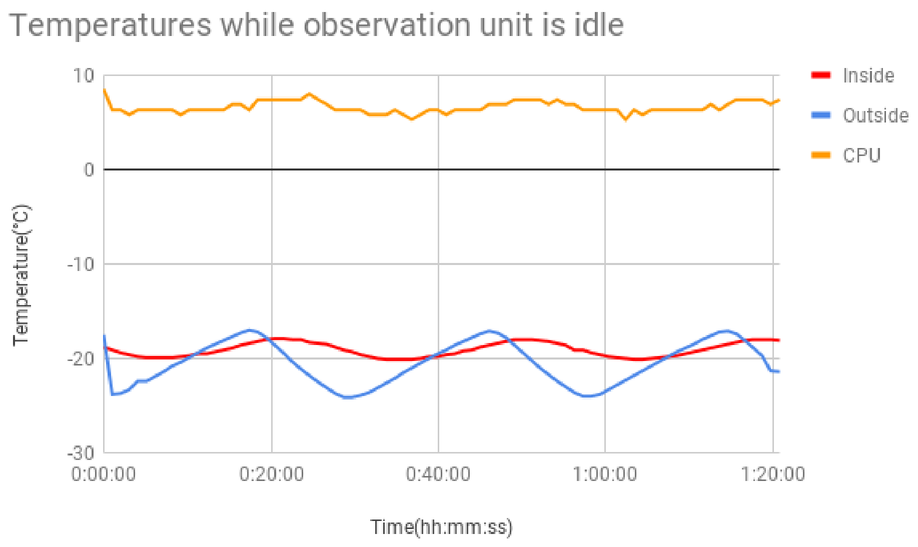

Figure 7.1: Figure shows inside, outside, and CPU temperature of the observation while idle with no insulation over an 1 hour and 20 minute period.

On figures 7.2, 7.3, 7.4, 7.5 we see the inside and outside temperature when running a CPU workload, a disk workload, a memory workload, and a network workload. We see on these graphs that the inside temperature decreases as the outside temperature decreases, and increases as the outside temperature increase.

The outside temperature fluctuates up and down, as the freezer starts the heat pump(to remove the hot air) when the temperature gets to high and stops when it reaches a certain threshold.

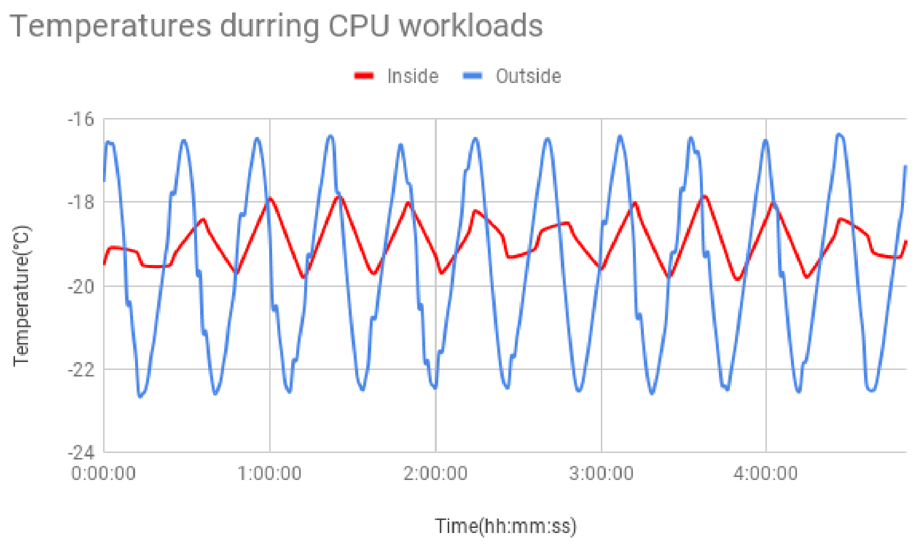

Figure 7.2: Shows temperature when cpu workloads are run with 0 layers of insulation. Measurements are over an 4 hour and 50 minute period.

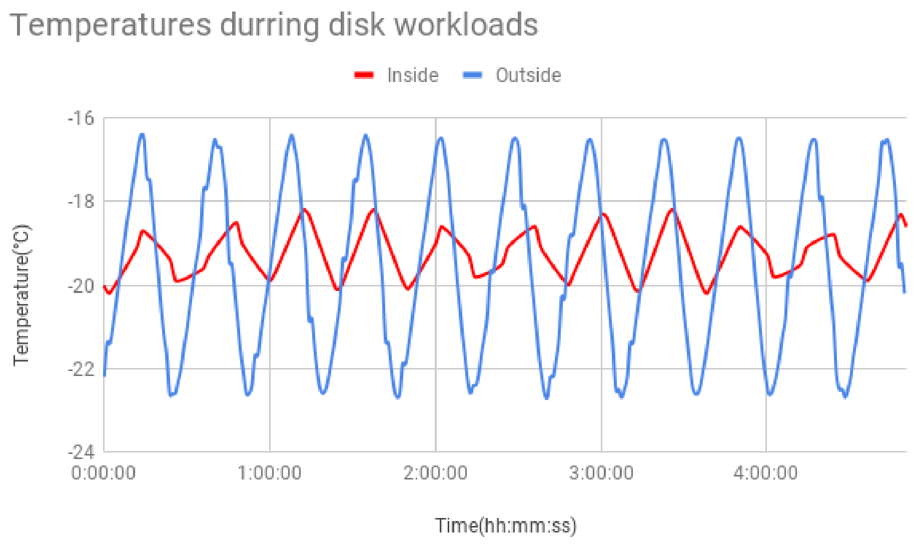

Figure 7.3: Shows temperature when disk workloads are run with 0 layers of insulation. Measurements are over an 4 hour and 50 minute period.

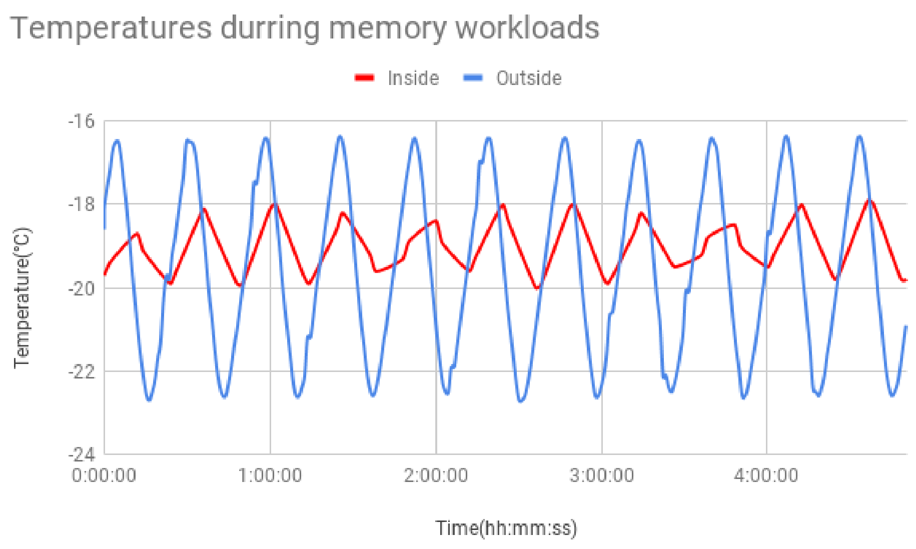

Figure 7.4: Shows temperature when memory workloads are run with 0 layers of insulation. Measurements are over an 4 hour and 50 minute period.

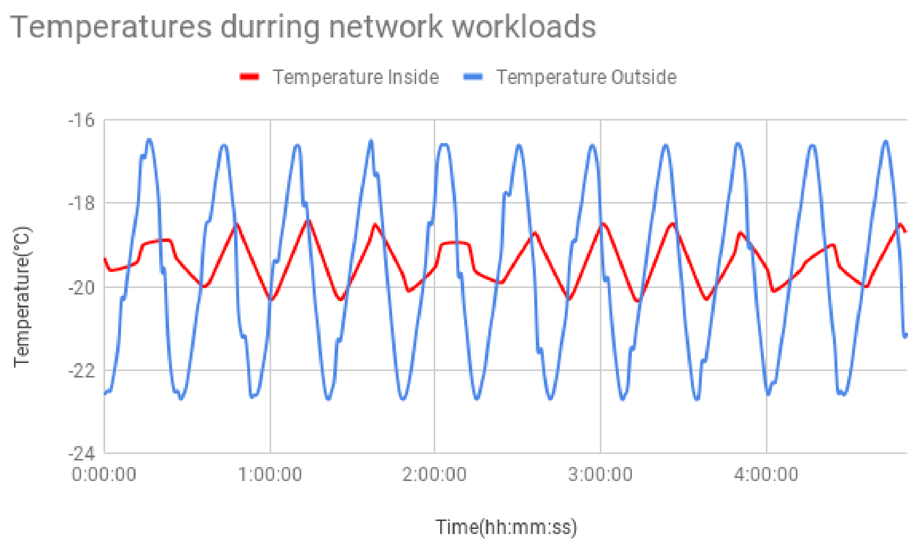

Figure 7.5: Shows temperature when network workloads are run with 0 layers of insulation. Measurements are over an 4 hour and 50 minute period.

We see from figure 7.6 that the temperature inside the observation unit stays within a range of about -18 celsius and -20 celsius, while the outside temperature fluctuates between -16.5 celsius and -22.5 celsius.

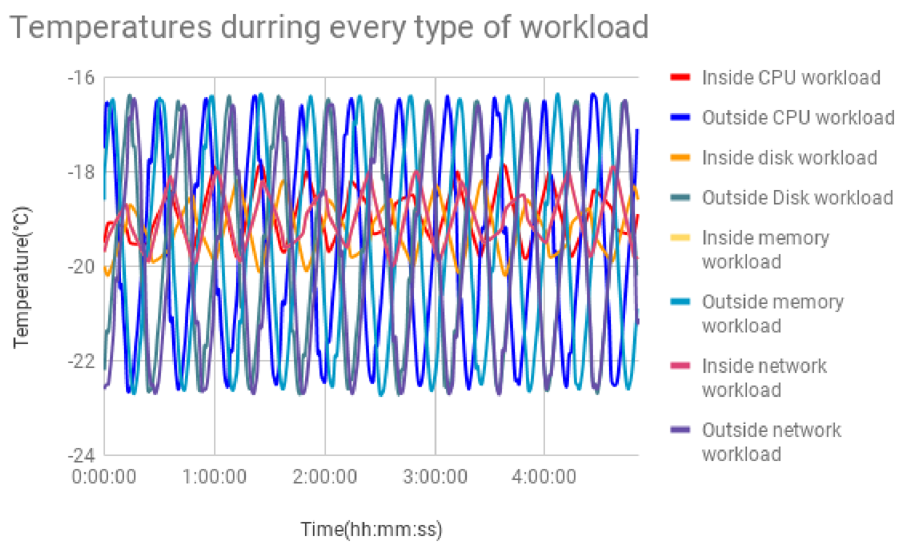

Figure 7.6: Figure shows inside and outside temperature of the observation unit as all the workloads are being run on it with 0 layers of insulation. Measurements are over an 4 hour and 50 minute period.

These readings(figure 7.6) indicate that the inside of the observation unit have a few degrees of difference between the outside temperature of the observation unit. We also see that the temperature outside thave a strong effect on the temperature inside, as we see the temperature inside rises when the temperature outside goes up and temperature inside goes down as it does down outside.

We also see that the outside temperature is higher than the inside temperature sometimes. This might be because the observation unit has trapped cold air inside the casing, and the air inside the casing will go up and down with the outside temperature but it will not change as fast, causing lower fluxuations when the temperature outside fluxuates.

From this we gather that what affects the inside temperature of the observation unit the most, is the outside temperature and not any workloads being run.

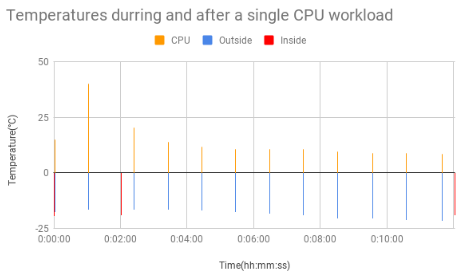

Figure 7.7: Figure shows inside, outside and cpu temperature as a cpu workload is run with 0 layer of insulation.

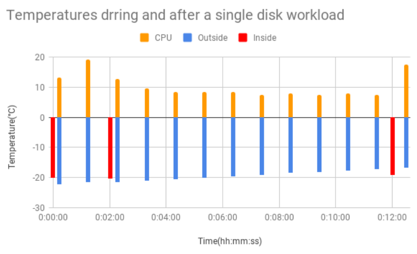

Figure 7.8: Figure shows inside, outside and cpu temperature as a disk workload is run with 0 layer of insulation.

On figures 7.7, 7.8, 7.9 and 7.10 we see how the inside temperature, outside temperature and the CPU temperature for the observation unit changes when we run one workload for 2 minutes and then wait for 10 minutes. For the CPU workload we see that the CPU temperature rises right away when the workload starts, and peaks at 40°C. When the workloads stops, the CPU temperature drops straight down after the workload has finished(it is almost back down to the original temperature a few second after the workload is done) and does not increase until the next workload starts. Both the inside and outside temperature stays about the same when the workload runs, as well as when the workload is done running.

For the disk workload we see that the cpu temperature goes up when the workload starts and reaches 20°C and when the workload stops drops down again. Both the inside temperature and the outside temperature of the observation unit is unaffected by this both while the workload is running and after it is done running.

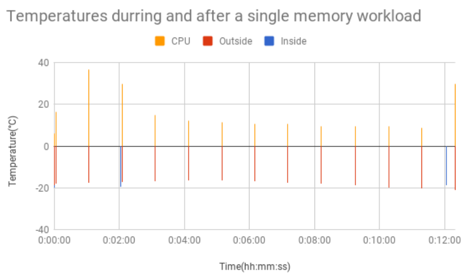

Figure 7.9: Figure shows inside, outside and cpu temperature as a memory workload is run with 0 layer of insulation.

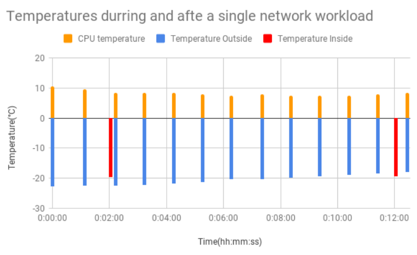

Figure 7.10: Figure shows inside, outside and cpu temperature as a network workload is run with 0 layer of insulation.

On figure 7.9 we see the temperatures inside the observation unit for a single workload, we see that the cpu temperature raises right after the workload begins, and peaks at about 37°C, and when the workload is done the temperature of the cpu drops down again. Both the inside temperature and outside temperature of the observation unit is unaffected by his workload.

On the last of these figures(figure 7.10) we see the network workload, here we see that neither the cpu, inside or outside temperature is affected by the workload.

Table 7.1: Average change in temperature inside the observation unit(with 0 layers of insulation) right after a workload and 10 minutes after a workload has run

Workload	Average change right after	Average change 10 mins after
CPU	-0.032	0.008333333333
Disk	-0.04	0.07083333333
Memory	-0.008	-0.004166666667
Network	-0.03333333333	0.225

In table 7.1 we see that that the average change in temperature inside the observation unit after running any of the workloads is about 0 degrees celsius. We also see that the average temperature change inside the observation unit 10 minutes after any workload is completed is 0 degrees celsius.

From this we can assume the the workloads have a very small effect on the temperature inside the observation unit, and the temperature inside the observation unit is mostly controlled by the temperature outside the observation unit. We can also conclude that the workload type has very little effect here on the ability to increase the temperature here.

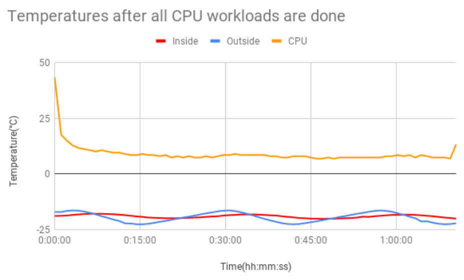

Figure 7.11: Shows temperature for inside OU, outside OU and for the OU cpu for an hour after the last cpu workloads have run

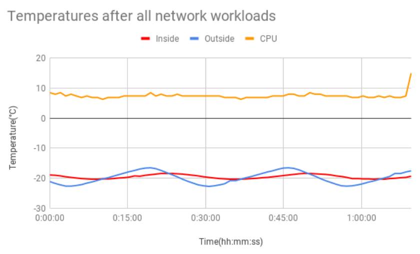

Figure 7.12: Shows temperature for inside OU, outside OU and for the OU cpu for an hour after the last network workloads have run

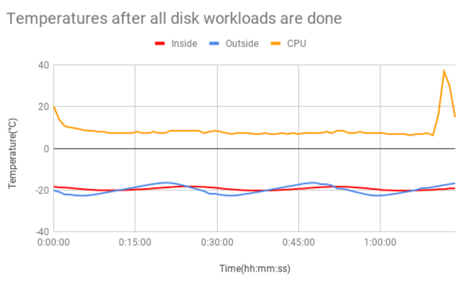

Figure 7.13: Shows temperature for inside OU, outside OU and for the OU cpu for an hour after the last disk workloads have run

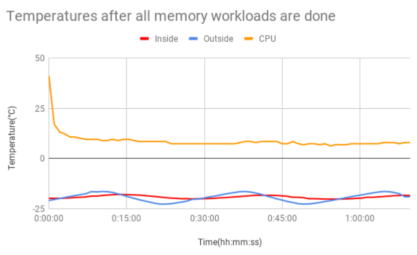

Figure 7.14: Shows temperature for inside OU, outside OU and for the OU cpu for an hour after the last memory workloads have run

On figures 7.11, 7.12, 7.13 and 7.14 we see the inside temperature, outside temperature, and cpu temperature for the observation for 1 hour after it is done running the workloads. Here we see that the temperatures don't show any change from when it was running any of the workloads. Based on this we can conclude that the workloads were not able to affect the temperature inside the observation unit.

7.2.2 Experiment 2: 1 layer of thermal insulation

On figure 7.15 we see the temperature for inside the observation unit, outside the observation unit, and the CPU on the observation unit, while the observation unit is idle with 1 layers of insulation. The observation unit is placed inside the thermal bag that is used for insulation. The thermal bag is much larger in size then the observation unit. This means that the insulation wont be wrapped tightly around the observation unit.

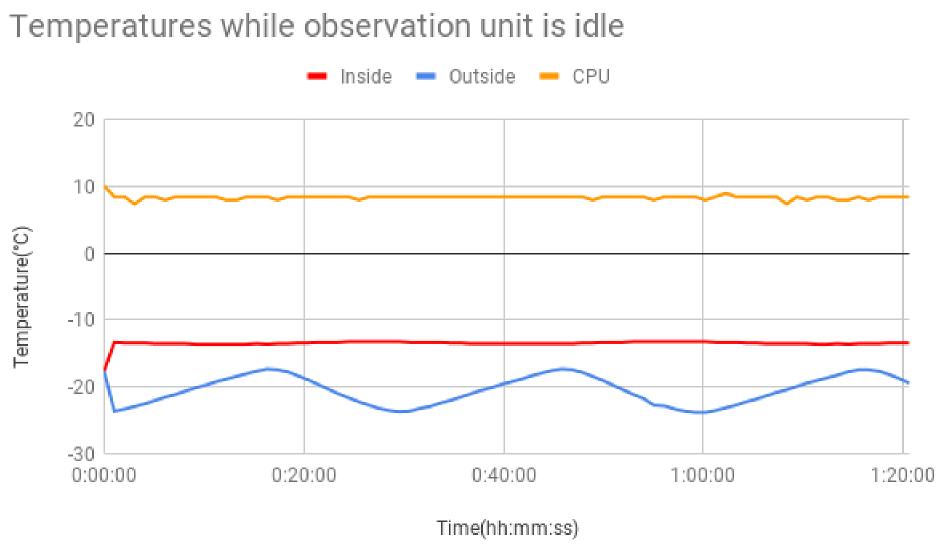

Figure 7.15: Figure shows inside ,outside, and CPU temperature of the observation while idle with 1 layer of insulation

On figure 7.16, 7.17, 7.18 and 7.19 we see the inside and outside temperature when running a cpu, a disk, a memory, and a network workload with one layer of insulation. We see that when running the cpu and memory workload, we are able to increase the temperature(it also looks like we are affecting the temperature inside the freezer and increasing the temperature inside it). We also see that the temperature inside the observation unit stays about the same as the outside temperature(we are probably leaking too much hot air, and therefore using energy trying to increase the temperature inside the freezer).

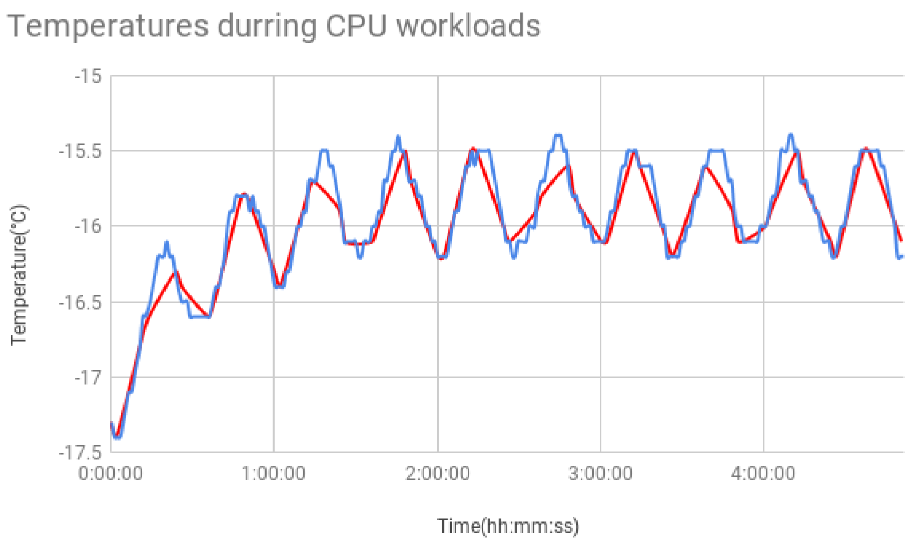

Figure 7.16: Shows temperature when cpu workloads are run with 1 layers of insulation. Measurements are over an 4 hour and 50 minute period.

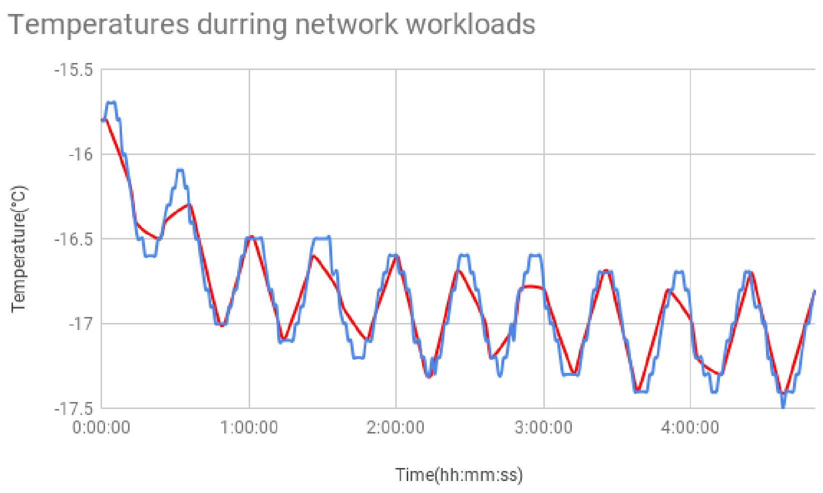

Figure 7.17: Shows temperature when network workloads are run with 1 layers of insulation. Measurements are over an 4 hour and 50 minute period.

We see from figure 7.16 and figure 7.18 that during the first hour of the experiment the temperature increases and after a while the temperature stops increasing. This probably shows the max temperature increase we can get with these parameters(workload length, time between workload and how many workload the scheduler runs at once) are 1.25°C(+- 0.25°C).

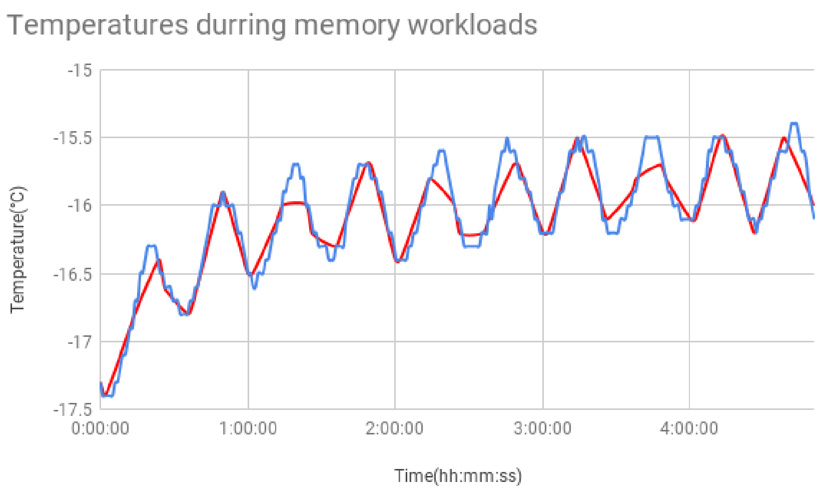

Figure 7.18: Shows temperature when memory workloads are run with 1 layers of insulation. Measurements are over an 4 hour and 50 minute period.

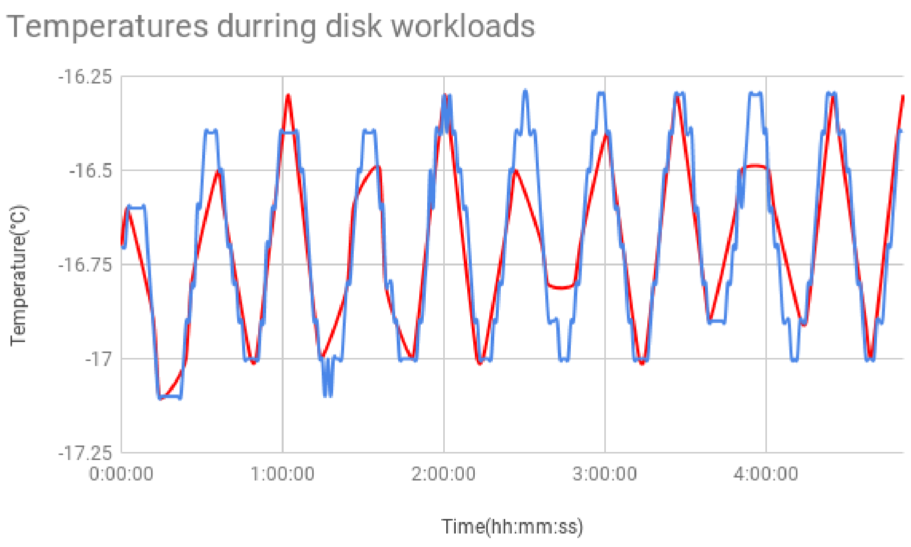

Figure 7.19: Shows temperature when cpu workloads are run with 1 layers of insulation. Measurements are over an 4 hour and 50 minute period.

We also see that for the disk and network workloads(figure 7.17 and figure 7.19), we are not able to increase the temperature inside the observation unit.

The inside temperature of the observation unit follows the outside temperature of the observation unit.

However, temperature fluctuations low(compared to when it has no insulation), where it increases and decreases about 0.375°C. We also see that these fluctuations are not as high as in the experiments with no insulations(figure 7.6).We also see that these fluctuations are higher than the fluctuations for the CPU and memory workloads that where ran with 1 layer of insulation.

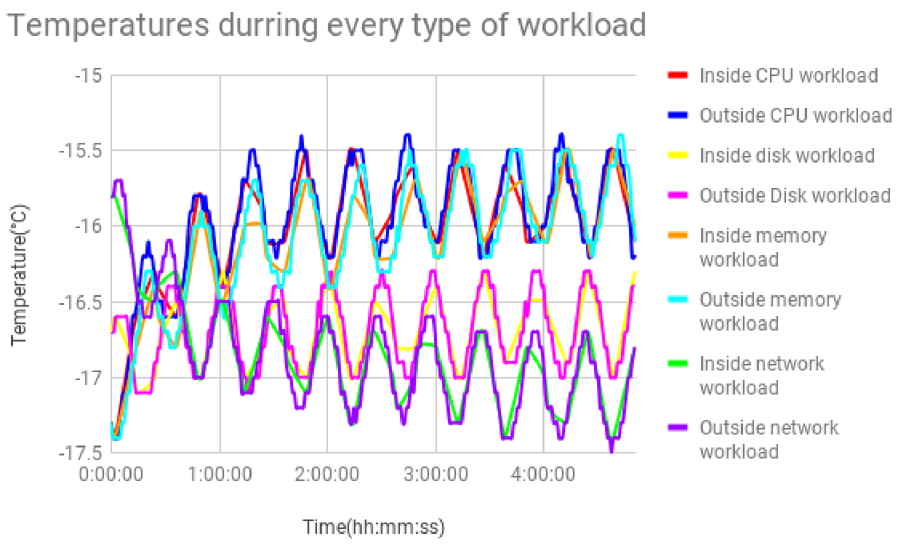

Figure 7.20: Figure shows inside and outside temperature of the observation unit as all the workloads are being run on it with 1 layer of insulation.

We see on figure 7.20 that the CPU workload and the memory workload are the ones that are able to raise the temperature the most out of the four workloads(an increase of allomost 2°C in under an hour while the memory workload was able to increase almost 1.5°C in the same time. The other two workloads(disk and network) are the worst for raising the temperature with the disk workload being slightly better then the network workload to raise the temperature in the observation unit with 1 layer of insulation. We also see that the figure(figure 7.19) with the network workloads starts at a higher temperature and then drops, this might be because it was the first type of workload to be run, meaning that the freezer that the observation unit has been placed in had to be open to place the observation unit in it, and it has not had time to cool down all the way like the others.

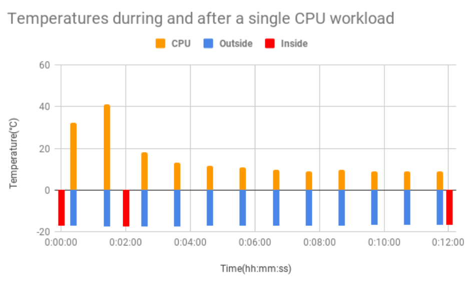

Figure 7.21: Figure shows inside, outside and cpu temperature as a CPU workload is run with 1 layer of insulation.

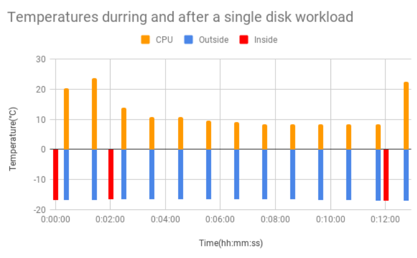

Figure 7.22: Figure shows inside, outside and cpu temperature as a disk workload is run with 1 layer of insulation.

On figure 7.21, 7.22 , 7.24and 7.23 we see the cpu temperature on the observation unit when a single workload(for all four types) is started, when the workloads stops and all the way until the next workload is started.

From these we see that the CPU workload and the memory workload are the only ones that result in a significant increase in the temperature for the CPU(up to 40°C). The disk workload increases the CPU temperature a also (up to 20°C), but not as much as the CPU and memory workloads.

The network workload is not able to increase the temperature of the CPU at all(it stays at around 10°C). The low effect on the CPU temperature from both the disk and the network workload is probably why those workloads have a hard time increasing the temperature inside the observation unit, while the other two workloads are able to increase it.

We also see that the CPU temperature increases right away when a workload is started, and drops fast down again right after a workload is done.

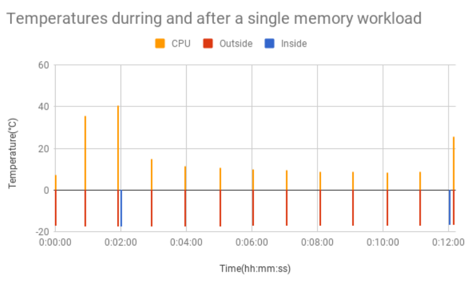

Figure 7.23: Figure shows inside, outside and cpu temperature as a memory workload is run with 1 layer of insulation.

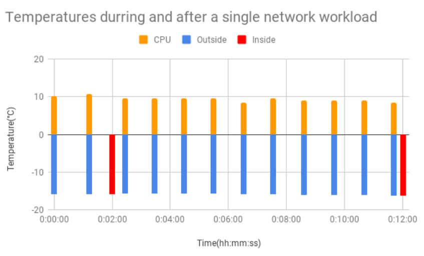

Figure 7.24: Figure shows inside, outside and cpu temperature as a network workload is run with 1 layer of insulation.

The reason for the table 7.2 showing that the disk workload has the highest positive change(after 10 minutes) in temperature is that it fluctuates more compared to the memory and disk workloads(since the disk has such large fluctuations it has higher numbers for when the outside temperature increases the inside temperature and therefore shows the largest average temperature increase).

And the reason for the network workload not having the same positive temperature as the disk workload, when it too has such fluctuations is that its temperature is still decreasing while the disk workloads temperature is staying at the same range.

Table 7.2: Tabel shows average change in temperature right after a workload and 10 minutes after a workload

Workload	Average change right after	Average change 10 mins after
CPU	-0.044	0.692
Disk	-0.01363636364	0.75
Memory	0	0.7041666667
Network	-0.02083333333	-0.04583333333

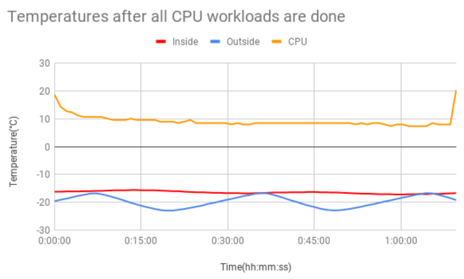

Figure 7.25: Shows temperature for inside OU, outside OU and for the OU cpu for an hour after the last cpu workloads have run

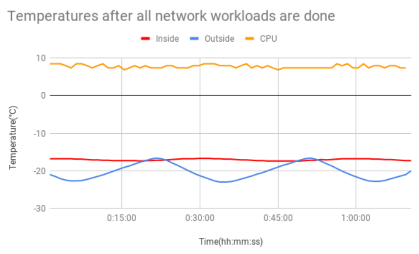

Figure 7.26: Shows temperature for inside OU, outside OU and for the OU cpu for an hour after the last network workloads have run

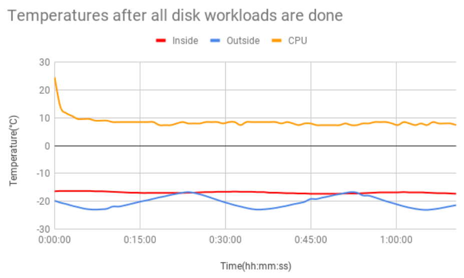

Figure 7.27: Shows temperature for inside OU, outside OU and for the OU cpu for an hour after the last disk workloads have run

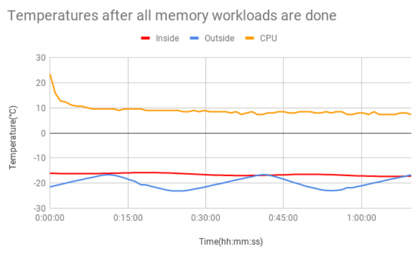

Figure 7.28: Shows temperature for inside OU, outside OU and for the OU cpu for an hour after the last memory workloads have run

We see from figures 7.25, 7.26 7.27 and 7.28 that after the workloads have finished running the observation units inside temperature stays about the same, while the outside temperature drops and starts fluctuating between -16°C and -23°C. But if we look on figures 7.16, 7.17, 7.18, and 7.19, we see that the outside temperature only fluctuates between -17.5°C and -15.5°C. And looking at table 7.4 we see that the average temperature inside when any workload experiments were running stays about the same, both during the experiments and when we don't run experiments any more. If we look at the average temperature outside(table 7.3) the observation unit during any type of workload and after we dont run workload any more, we see that the outside temperature decreases.

From this we can conclude that we are also heating up the freezer when we are running workloads. The lowest temperature increases by about 4.5°C and the highest temperature by 0.5°C.

Table 7.3: Shows differences in temperature outside observation unit durring workloads and once all workloads are done running

Workload	Average during workloads	Average after all workloads done
CPU	-15.9557554°C	-19.65362319°C
Disk	-16.695053°C	-20.39428571°C
Memory	-16.07333333°C	-19.87681159°C
Network	-16.85827338°C	-20.30142857°C

Table 7.4: Shows differences in temperature inside observation unit durring workloads and once all workloads are done running

Workload	Average during workloads	Average after all workloads done
CPU	-16.018°C	-16.44057971°C
Disk	-16.696°C	-16.85142857°C
Memory	-16.114°C	-16.53913043°C
Network	-16.84°C	-17.05714286°C

7.2.3 Experiment 3: 2 layers of thermal insulation

On figure 7.29 we see the temperature for inside the observation unit, outside the observation unit, and the CPU on the observation unit, while the observation unit is idle with 2 layers of insulation. We see that the inside temperatures are at almost -16°C and that the temperature is stable and does not fluctuate much. We also see that the outside temperature is always lower than the inside temperature. The outside temperature also fluctuates between almost -16°C at its highest and almost -23°C at its lowest. The CPU temperature stays at about 13°C. Once the Observation unit is placed inside 2 thermal bags for insulation, the insulation becomes tightly wrapped around the observation unit.

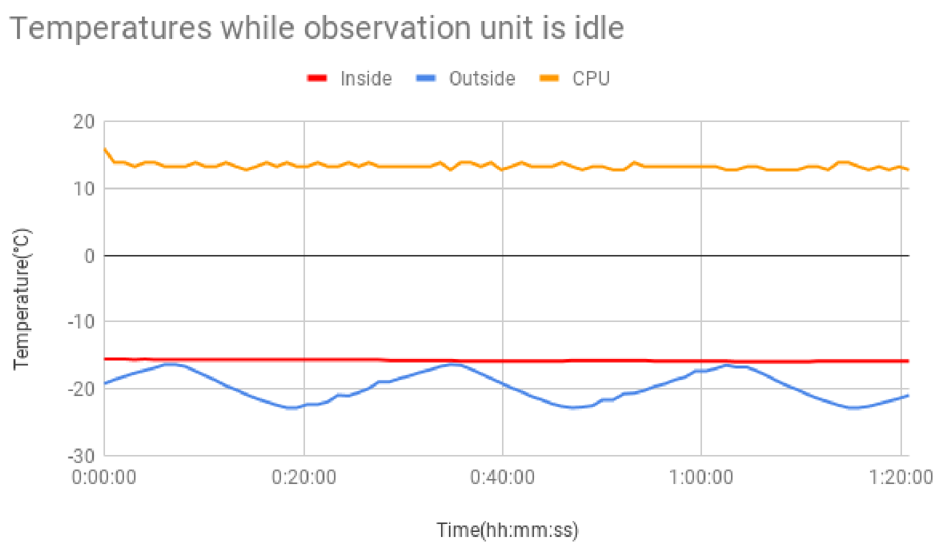

Figure 7.29: Figure shows inside ,outside, and CPU temperature of the observation while idle with 2 layers of insulation

On figure 7.34 we see the inside and outside temperature of the observation unit as different workloads are run. Here we see(for the CPU and memory workload) that initially the temperature inside the observation unit increases by a few degrees, and after 1 - 2 hours of running the workloads the amount that the temperature increases by goes down, and eventually levels out.

We also see that for the disk workload, there is no increase in the inside temperature as we run disk workloads.This might be because the disk workload consists of a long write, rather then lots of small writes, and a long write might not generate any noticeable amount of heat. We also see that the outside also is flat does not increase, unlike with the CPU and memory workloads. There we see that the outside temperature increases when the inside temperature increase, meaning that the some of the heat generated inside the observation unit might leaking out and warming up the freezer also.

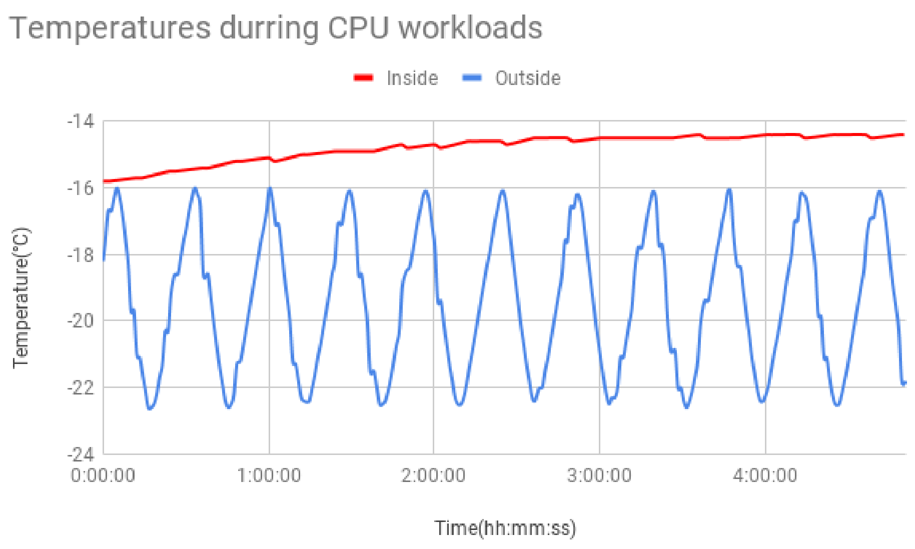

Figure 7.30: Shows temperature when cpu workloads are run with 2 layers of insulation. Measurements are over an 4 hour and 50 minute period.

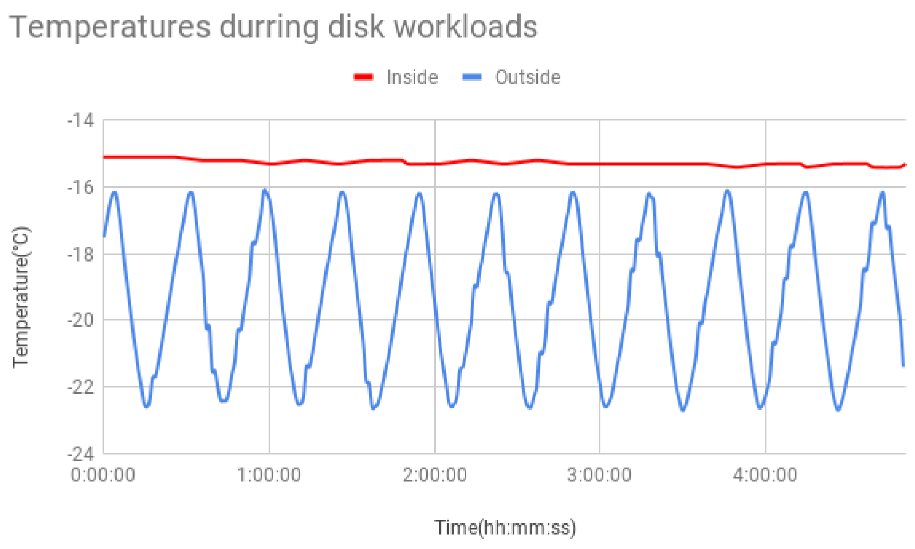

Figure 7.31: Shows temperature when disk workloads are run with 2 layers of insulation. Measurements are over an 4 hour and 50 minute period.

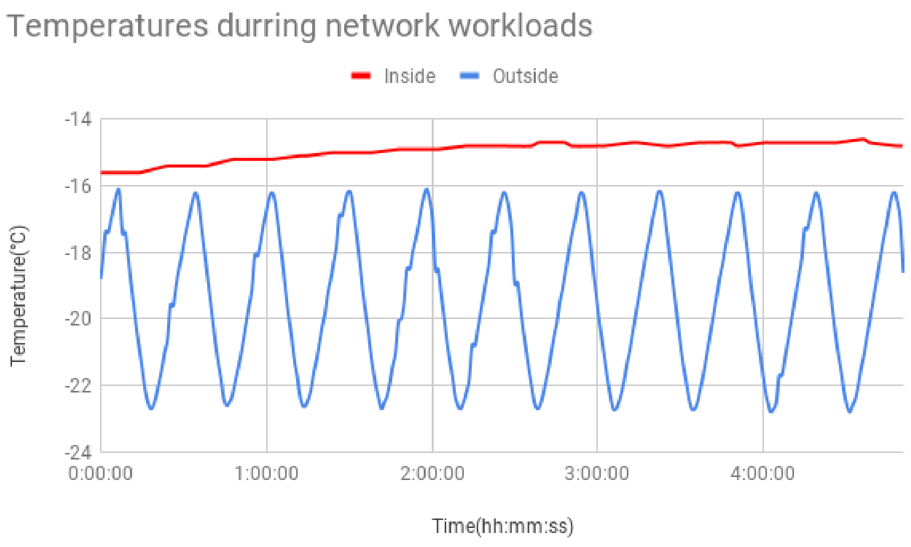

Figure 7.32: Shows temperature when memory workloads are run with 2 layers of insulation. Measurements are over an 4 hour and 50 minute period.

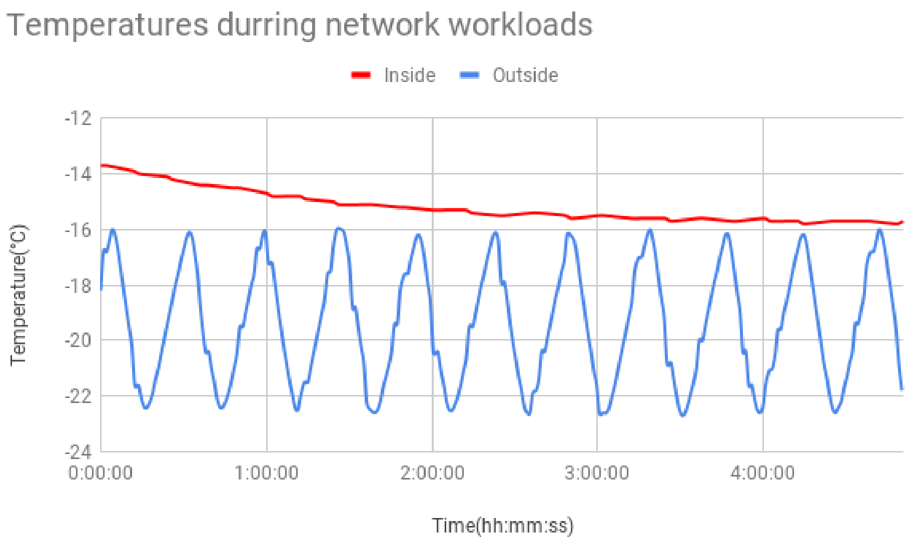

Figure 7.33: Shows temperature when network workloads are run with 2 layers of insulation. Measurements are over an 4 hour and 50 minute period.

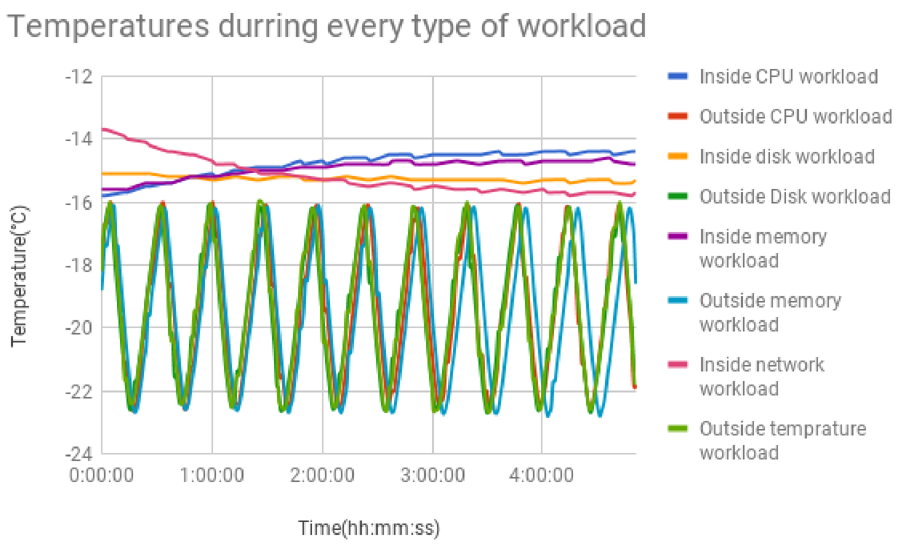

Figure 7.34: Figure shows inside and outside temperature of the observation unit as all the workloads are being run on it.

We see that for all the workloads the outside temperature stays in the same range, -16 too -22.5. We also see that for all the workloads the temperature inside is always higher than the outside temperature. The inside temperature is also stable and does not fluctuate like the outside temperature(like it did for 0 and 1 layer of insulation).

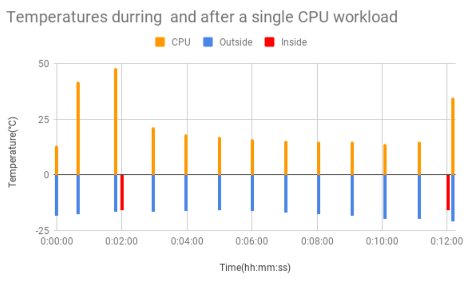

Figure 7.35: Figure shows inside, outside and cpu temperature as a cpu workload is run with 0 layer of insulation.

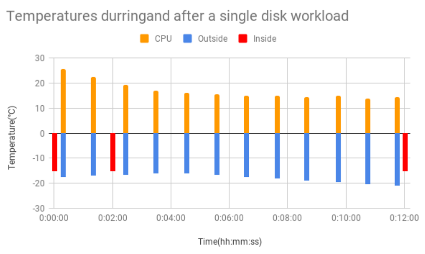

Figure 7.36: Figure shows inside, outside and cpu temperature as a disk workload is run with 0 layer of insulation.

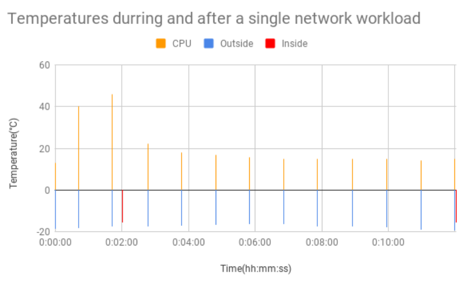

Figure 7.37: Figure shows inside, outside and cpu temperature as a memory workload is run with 2 layer of insulation.

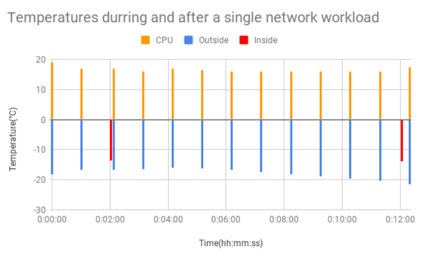

Figure 7.38: Figure shows inside, outside and cpu temperature as a network workload is run with 2 layer of insulation.

Another thing we see(table 7.5) is that the temperature inside the observation unit does not increase right after a workload has run(for 2 minutes), but rather a little while later(after 10 minutes).

On figure 7.35, 7.37 , 7.36 and 7.38 we see the cpu temperature on the observation unit when a single workload(for all four types) is started, when the workloads stops and all the way until the next workload is started. From these we see that the CPU workload and the memory workload are the only ones that get a significant increase in temperature(up to 46-48°C). While the disk workload can increase the CPU temperature a little bit(up to 25°C). And the network workload is not able to increase the temperature of the CPU at all(it stays at around 17-19°C). The low effect on the CPU temperature from both the disk and the network workload is probably why those workloads have a hard time increasing the temperature inside the observation unit, while the other two workloads are able to increase it. We see that the cpu and memory workload is able to increase or hold the temperature at the current level with one workload.

Table 7.5: Tabel shows average change in temperature right after a workload and 10 minutes after a workload

Workload	Average change right after	Average change 10 mins after
CPU	-0.032	0.0.056
Disk	-0.008	-0.012
Memory	-0.008	0.032
Network	-0.036	-0.084

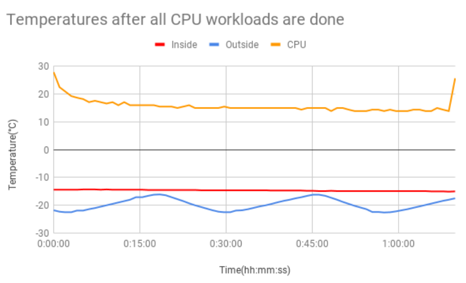

Figure 7.39: Shows temperature for inside OU, outside OU and for the OU cpu for an hour after the last cpu workloads have run

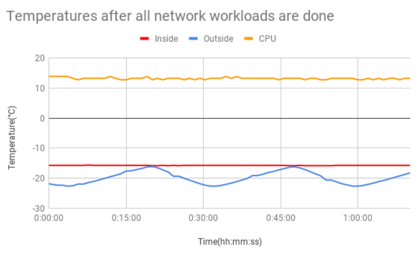

Figure 7.40: Shows temperature for inside OU, outside OU and for the OU cpu for an hour after the last network workloads have run

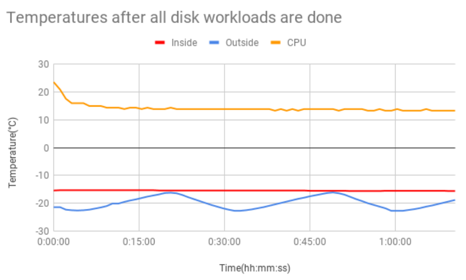

Figure 7.41: Shows temperature for inside OU, outside OU and for the OU cpu for an hour after the last disk workloads have run

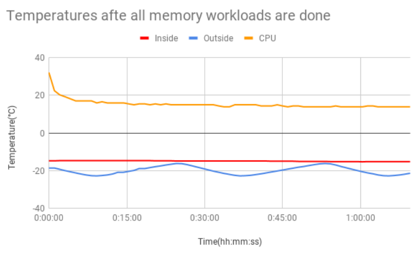

Figure 7.42: Shows temperature for inside OU, outside OU and for the OU cpu for an hour after the last memory workloads have run

On figures 7.39, 7.40, 7.41 and7.42, we see the inside temperature, outside temperature, and cpu temperature for the observation for 1 hour after it is done running the workloads with 2 layers on insulation. We see that for the disk and network workloads there is no drop in in temperature after the workload are done as they did not change the temperature inside. As for the workloads that managed the increase the temperature(cpu and memory workload), we see that over about an hour the temperature has dropped down to where it started before the workloads ran. Which is about -0.8°C to -1°C drop in temperature over an hour.

7.2.4 Experiment 1 vs Experiment 2 vs Experiment 3

From the results we see that what has the biggest impact on the inside temperature of the observation unit is not any workload, how long they are run for, how many processes are run at the same time, etc. but the amount of insulation the observation unit has.

We also see that with the CPU workload we can increase the temperature inside the observation unit by almost 1°C over an hour. To achieve this we had to run only 6 workloads for 2 minutes each, at 10 minutes intervals.

7.3 How long between each workload run

In the experiments were different layers of insulation was measured on the observation unit(figure 7.6, 7.20 and 7.34), we saw that the we needed to use at least 2 layers of insulation or else we would not manage to get the inside temperature of the observation unit above the outside temperature of the observation unit. In this experiment we will therefore use 2 layers of insulation when we run experiments to see how long a of a break we need between each time we run a set of workloads. To do this we will see how much the temperature drops after a set time when we are done running a set of workloads. As the temperature drop inside the observation unit does not have anything to do with what workload is run, we will run CPU workloads as they generate the highest temperature. These workloads will be run for 2 mins with different amount of time between each set of workloads. We will look at how much the temperature dropped after each workload after each set time.

On graph 7.43 we see the temperatures when running the workloads with 1 minute intervals. Here we so no increase in temperature. This might be because it takes some time after the workload is done running for the workload to running for the observation units inside temperature to increase.

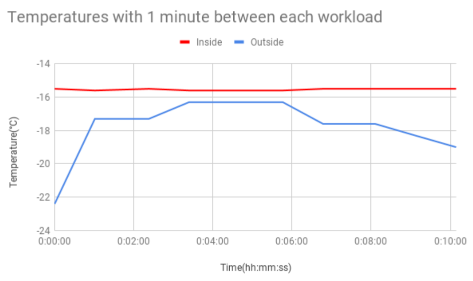

Figure 7.43: Shows temperature for inside OU, outside OU when there is 1 minute between each workload

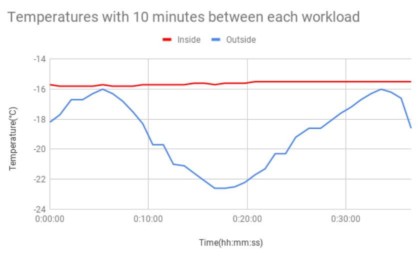

Figure 7.44: Shows temperature for inside OU, outside OU when there is 10 minutes between each workload

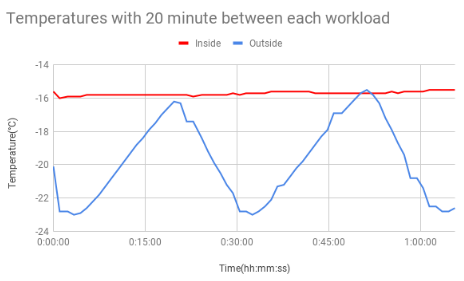

Figure 7.45: Shows temperature for inside OU, outside OU when there is 20 minutes between each workload

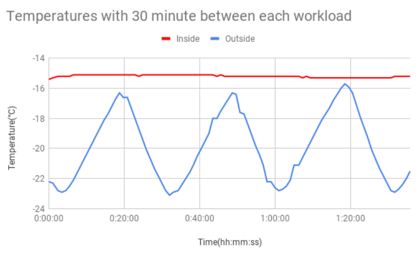

Figure 7.46: Shows temperature for inside OU, outside OU when there is 30 minutes between each workload

On graph 7.44 we see the temperatures when running the workloads with 10 minutes intervals. Here we see a increase in temperature of 0.2°C over a 36 minute period. For the experiment(graph 7.45) that is running workloads with 20 minute intervals on the workloads, we see a 0.5°C increase of the inside temperature over a period of 1 hour and 5 minutes.

We see on graph 7.46 that is running a workload at 30 minute intervals that the temperature it starts at and the temperature it stops at is the same. We also see that the temperature increases initially and then decreases.

7.4 Different amount of processes running at the same time

We have so far been running 4 sets of workload in parallel in previous experiments. Here we are going to run experiments to see the effects of running different amount of workloads sets in parallel, to see how many we should run in parallel. For these experiments we are only using the CPU workload(as it was the one who has had the greatest effect, when it comes to increasing the temperature inside the observation unit) and use 2 layers of insulation(since according to the previous results, we need at least that much insulation to get the temperature to increase)

For this experiment we will be running sets of CPU workloads 3 times. And for each experiment we will change the amount of processes we run in parallel. Starts with no workloads running in parallel, and working up to 4 workloads running in parallel.

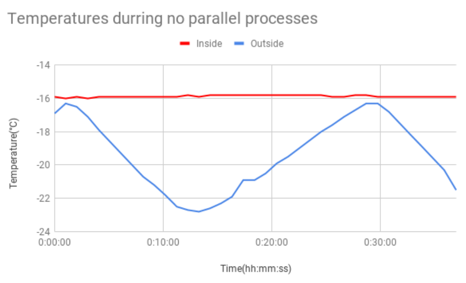

Figure 7.47: Shows temperature for inside OU, outside OU when there is 1 only one process running at a time

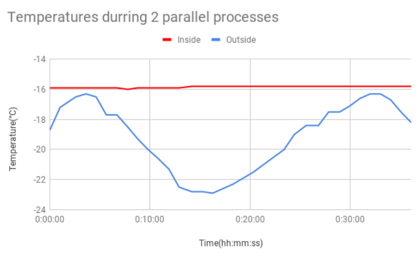

Figure 7.48: Shows temperature for inside OU, outside OU when there is 2 processes running in parallel

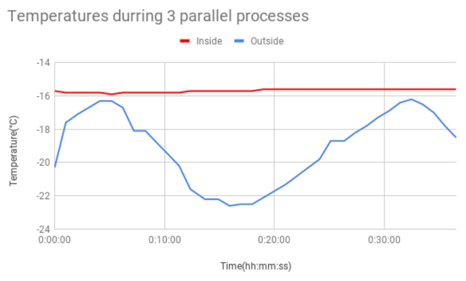

Figure 7.49: Shows temperature for inside OU, outside OU when there is 3 processes running in parallel

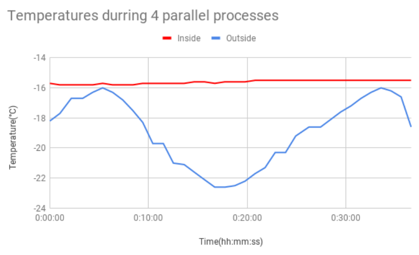

Figure 7.50: Shows temperature for inside OU, outside OU when there is 4 processes running in parallel

7.5 Amount of time to run the processes for

In the graphs 7.51, 7.52, 7.53 and 7.54, we look at the effects of running workloads of different lengths. We run 3 workloads after one another with a set time between each workload. This is done for 4 different workload lengths, 1 ,2 ,3 and 4 minutes workload length.

We see that on all the graphs, the inside temperature is able to increase, but the longer you run a workload for the more it increases. Another thing we see is that the longer with run a workload for the faster the inside temperature increases.

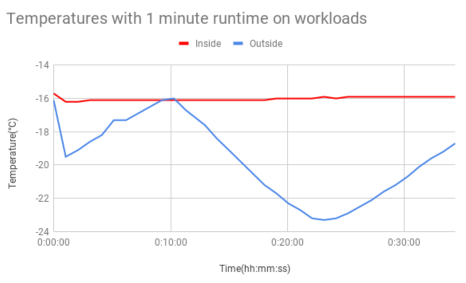

Figure 7.51: Shows temperature for inside OU, outside OU when workloads run for 1 minute

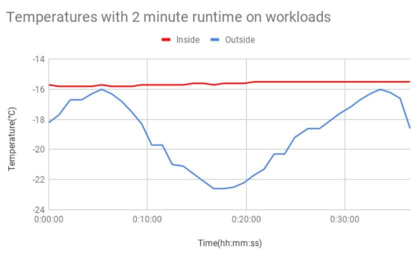

Figure 7.52: Shows temperature for inside OU, outside OU when workloads run for 2 minutes

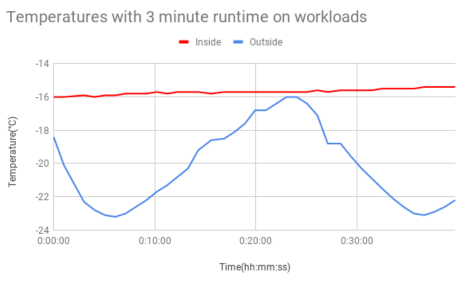

Figure 7.53: Shows temperature for inside OU, outside OU when workloads run for 3 minutes

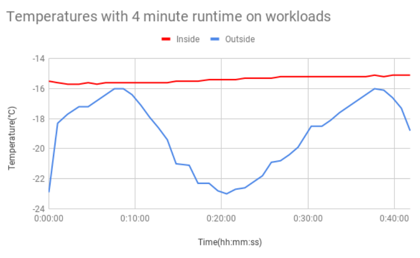

Figure 7.54: Shows temperature for inside OU, outside OU when workloads run for 4 minutes

7.6 Results used in the premade model

In table .. one can see the values chosen for the premade model for the scheduler. The 2 layers of insulation was chosen, since we needed a minimum of that in order to increase the temperature. We choose to run 4 parallel workload since it would increase the temperature the most.

The amount of time to run a workload was calculated by taking one run of a workload for each workload and calculating the temperature increase 10 mins after each one of those had run. And then averaging these results for all types of workloads. Then we divide that by 120 seconds and take that result and divide that by 2 to find how much it takes to increase the temperature by half a degree.

To find the amount each workload should wait look at table 7.6, and look at the temperature at the beginning of the graph, and then find the first temperature that is half a degree lower than the initial value. Then we take the difference in time between the first value found and the second found and use that as how long it takes for the observation unit to drop half a degree.

Table 7.6: tabel shows values for premade model

Layers of insulation	2
Amount of time to run a workload	600s
Time between each workload	46m
Number of parallel process to run	4

If we follow this model, we should run a workload for 600s, and then the temperature inside the observation unit should increase 0.5°C, and after 46 minutes it should drop down again.

8 Discussion

8.1 Stop(pause) and resume workloads

Right now the scheduler does not support pausing and resuming workloads. The problem with not supporting these features is that some workloads could end up starving(not fully, since no process can in this setup run forever) other workloads by taking to long to complete. A pause and resume system could be built by using the “kill -STOP <PID>” for pausing the workload, then adding it list of workloads as “kill -CONT <PID>”. By adding it as that command in the back of the workloads list, not many more changes would have to be done.

8.2 In memory cache

When a state is read and formatted into data, it is passed to memory and then straight to disk. And when the scheduler needs the data it reads it from disk. This method of doing it was chosen for it simplicity. A better method of doing this might be to have an in memory cache, that keeps a few of the latest pieces of data that has been gathered by the state reader.

One way this could be designed, would be to have a cache of n size in memory with a first in first out(FIFO) policy for the data. And only write the data to the file system when an item gets pushed out of the cache. Using a cache like this would mean that the scheduler could read data from the cache rather then reading it from the file system.

The effects of using a memory cache would be a faster access time on the data that the scheduler needs. This faster speed might a have a small effect on how much energy the devices use, as it would be able to be done with its tasks faster and therefore get back to sleep faster.

One problem with this setup would be that you could lose data in the memory chace if something were to happen that would cause the unit to lose power. This could be addressed by having the state reader in addition to adding the data to the chace, it would write it to disk. The good thing about doing it this way would be that it won't add that many more writes, as the number of disk writes would be m disk writes - n number of piece of data that can fit in memory.

8.3 Alternative way to run workloads

For a program to be able to schedule its workload using the scheduler, it needs to have those workloads written in python. This happens because it is the scheduler that runs the workloads for the programs, so it needs to be able to start the workloads.

This approach was chosen because of its simplicity when implementing it. But because of this, programs are forced into using python when they might not want to do so. Or for each new type language a workload is written in, the code that starts the workload would need to be updated.

An alternative to this would be to have another a system where the scheduler tells the programs when they can run their workload, rather than having the scheduler do it. A very basic system for this is presented in figure 8.1

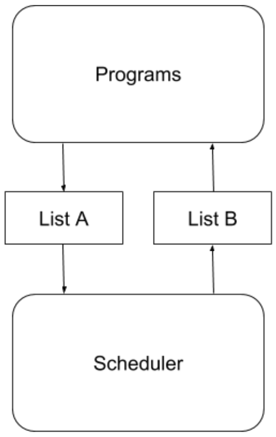

Figure 8.1: Alternative system for notifying the scheduler that a task needs scheduling

The scheduler has 2 lists, a list(now called list A) with workloads programs wants to run, and a list(now called list B) with workloads that programs can run now.

The programs push workloads they want to run to list A, and when the scheduler decides to run a workload, it pushes a workload from list A to list B. the programs subscribe to list B, and when one of their workloads are found in list B, they start it.

A second and more simple alternative would be to have everything that would have to be written into the terminal to start the workload within the file containing the list of workloads. This way the scheduler could just run that command.

8.4 Thermal bags

As mentioned, to test the effect of different levels of insulation on the system we use insulated thermal food bags. They were chosen because they were an easy way to provide some insulation for the observation unit. The main problem with them is since we have wires coming out from them, they don't close all the way. This means they leak some hot air, and are not able to keep the heat trapped as well as they should. This is ok though as we just wanted to see the general effect adding some insulation to the observation unit would have.

Another issue with the thermally insulated bags is that we don't have data on the properties of the materials used when insulating the observation unit.

Another thing with the freezing bags that affect the observation unit is how tight the freezing bags wrapped around the observation unit. When there is just one freezing bag used. The freezing bags fits loosely around the observation unit and there is a lot of air inside the freezing bag with the observation unit. Meaning there is a lot of room inside the freezing bag that the observation unit needs to heat up, and that can be filled with cold air.

8.5 Heat generated from sensor reading

When creating the model(the one that it starts with) for the OUA system you want as many data points as possible to get a more accurate model. The problem with taking many readings is that each reading generated some heat from reading the state from the sensor, processing the state into data, and storing the data on the local file system.

When deciding how often we were going to read the sensor we tried to strike a balance between getting as many data points as possible to use to create the model and generating as little heat as possible when collecting the data.

8.6 Finding out heat is generated over n time rather then finding heat generated by individual tasks

When trying to find the workloads that would create the right amount of heat, to heat up the observation unit, there were two approaches considered.

The first option was that we find out how much heat each workload creates, and then try and find a combination of workloads that would generate enough heat to reach the desired temperature.

The second option was not look at each workload, but rather combine the heat generated from all types of workloads and take the average from that. And use that to see how long any workload would need to run for to generate n amount of heat.

We chosen to go with option 2, as it simplifies the way we pick workloads. The second reason we chose to go with it, is that the first option requires the observation unit to gather more data, and it is more complex(and therefore uses more energy) both when it comes to calculating heat generated by each type of workloads, but also when picking which workloads it should run.

8.7 Problem with having the scheduler starting the workload that need to run at a set time

When a workload needs to run at a set time, it is started by the program that owns that workload. The reason we can't have the workload started by the scheduler(like the other workloads), is that the observation unit needs to conserve power usage. To do this the scheduler sleeps at intervals. So if a workload would need to run at a certain time, it could end up having that time when the scheduler sleeps and therefor missing the time it should have run. Alternatively we could have the workload constantly checking the list for workloads that needs to run at a set time, but the scheduler would then need to be on all the time.

8.8 Might never get to update model

For the model to get updated it needs data. To get this data its needs workloads to run at a set interval. If they are scheduled to fast, we can calculate how fast the temperature is dropping so we need to have a minimum of time to pass before each workload is scheduled.

If the observation unit is placed in a area that's so cold that the temperature outside is causing the temperature inside to constantly be below the threshold. The scheduler will keep starting workloads to get the termature inside the workload up above the threshold. This will make it so that he scheduler cannot update the model it uses.

Another way this might happen is if the premade model is to inaccurate causing the scheduler not to schedule the right amount of workloads, and thus not bringing the temperature up enough to get it above the threshold. Causing it to keep scheduling workloads that have a low effect on the temperature inside the observation unit.

The only way to fix this problem would be to stop scheduling workloads for a set amount of time if the scheduler have not updated the model after a set time. But this would cause major problems, since the temperature inside the box would drop below the threshold during this time. So this solution is not acceptable. This is of course because the observation unit might get damaged from having its internal temperature to low.

An alternative might be to allow for a partial update of the model. Istead updating both how long it would need to wait between each set of workloads and how long it should run workloads for. It just updates how long it should run workloads for. This can be done since the gathering of this data is not affected by having workloads constantly running.

8.9 Alternative way to gather data to update the model

A way to to improve the way the costume model for the observation unit is made would be to add so that instead of using the data it has on hand now, which is the data it gets from executing workloads based on the premade model.

It would have a function that generates data. This function would at some set times, put together a set of workloads of a set length, and test how that affects the observation unit and gather data on that.

Then it would change that parameter for the length it would run a set of workloads and gather data again at some later time. It would repeat this process n times, until it would have enough data to make a better model. (Algorithm 3)

Algorithm 3 Alternative way to gather data to update the model

```
1: if $SetTime$ then
2:     if i < n then
3:         Run workload for i mins
4:         Calculate and store change in temperature
5:         i++
6:     else
7:         Use stored results to update model
8:     end if
9: end if
```

8.10 Updating the model more than once

The observation unit starts with a premade model that it uses to make decisions about what workloads it should run. Once it gathers enough data it updates the model, to be more accurate for the observation unit it runs on. This updating of the model is done only once. The problem with this is that it can lead to an inaccurate model, because the temperature outside is constantly change, and might vary a lot depending on how long a observation unit is placed outside. The mode might say it only needs to run workloads for 1 minute to raise the temperature 1 degree, but if it has gotten a lot colder outside it might need to run it for 2 minutes. And this can end up causing the observation units internal temperature to drop to low, since its not running the correct amount of workloads. The opposite might also be true. It might be a lot varmer outside, and it can end up running workloads a lot longer than it needs too. And end up using more power then it needed to.

This can be fixed by having the updated model update again at a set time after it has updated. The problem with this is that you want to update the model as often as needed to keep it accurate, but not so often that you are using to much power. One would need to find a balance power use and accuracy of the model.

To find how often you update the model one could look at the climate in the area where you place the observation unit. And try and look at how much the temperature increases and decreases over time(months and weeks), and try find the average temperature for those times. And use those average temperatures to determine how often you should update the model.

8.11 Updating the model with the number of parallel processess it should run

The premade model the observation unit has consists of how long it should run workloads for, how long it should sleep before scheduler checks the temperature again after running a workload, and how many workloads it should run in parallel. But when the scheduler updates the model, it only updates the amount of time it should sleep, and the amount of time it should run a set workloads for. This is because the scheduler always runs 4 sets of workloads in parallel, and therefore never gets any data on any different amount of sets of workloads run in parallel.

A way it could have gathered this data would be to have a data creating workload. The purpose of this workload is to create data that is needed to update the model. This workload would run different amount of sets of workloads in parallel. But it could also be used to gather data on the other values that gets update in the model, and additional values that might get added to the model at a later date.

The main problem with this solution is that it would need to run workloads these workloads when the temperature was above the threshold temperature(you can run it when it is below in case it would increase the temperature to little), and therefore the scheduler would end up spending energy just to gather data that would be used to update the model.

And since one of the goals of the observation unit is to stay alive and it needs power to do so, one would need to weight the potential gain of an even more accurate model vs the energy cost of getting a more accurate model.

8.12 Memory workload

Except for the CPU workload, the memory workload is the only workload abel to increase the temperature inside the observation unit. It is able to do this since the memory workload makes the CPU increase in temperature. The reason it might be doing this is because the file loaded into memory is to small, and the file is being cached in the L2 cache on the CPU. this means that instead of searching through the memory, it is actually searching through the L2 cache on the CPU causing the CPU to increase in temperature.

8.13 Alternative workload selection algorithm

For selection of workloads right now we use a type of first-fit algorithm that first picks the oldest workload, and then tries to add more workloads, by looking through all the workloads to see if any of them fit the workload block(starting looking at the oldest workload). This algorithm was chosen to avoid to starve the workloads. The main problem with this approach is that a workload could end up having to run for hours(or indefinitely), and since we don't pause and resume workloads we could end up starving the rest of the workloads.

If we were to look at other scheduling algorithms to select workloads, one of the most important parts is that the workloads don't starve. Because of this algorithms like shortest workload first, or one where we select the newest workload first would end up causing starvation. Round robin is an alternative that could be used, and it would fix the problem caused by some workload running for hours. For this to work support for stopping/pausing workloads and resuming workloads like explained in chapter 8.1 would have to be added.

8.14 Only using the workloads that increase the temperature the most

When observation unit drops below a set temperature, it tries to increase the temperature inside it self by running sets of workloads. When it choses a workload, it will use any workload, but not all workloads increase the temperature the same amount, some increase the temperature more.

An alternative solution to the one that is implemented would be when the scheduler picks a set of workloads priorities the ones that increase the temperature the most, rather than the ones that increase the temperature the least. The good thing about this solution is that one might need to run a set of workloads for a shorter period of time to increase the temperature the set amount then if you were to use a mixture of workloads that increase the temperature a little and those that increase the temperature a little.

The bad thing about this is that you might end up starving some workloads as the scheduler would priorities the workloads that increase the temperature the most. And at worst you could have some workload that don't get to run at all.

9 Conclusion

In this paper we have done the architecture, design and implementation of a system for autonomously scheduling of tasks based on the internal and external states of the observation unit. Focus has been on the temperature of the environment, of the observation unit and of the observations units CPU.

The system job is to schedule tasks in order to keep the temperature inside the observation unit above a set temperature in order to keep it from taking damage from the cold. To be able to schedule tasks to keep the temperature up the system needs to know the effect running the tasks has on the temperature inside the observation unit. The scheduler comes with a premade model that tells it the effects of the tasks it can schedule. The observation unit is able to update this model based on data it gathers while scheduling tasks in order to make the model more accurate for the specific observation unit it runs on.

In order to make a model for the scheduler and see the effect of different layers of insulation on the observation unit, we have analyzed the effect of different layers of insulations with different tasks running on the observation unit. From the results of our experiments we see that the observation unit needs insulation(we found in our experiments that it needs 2 layers) in order to increase the temperature inside the observation unit by a few degrees(1°C to 2°C), and if we have enough insulation the temperature inside the observation unit is able to be higher than the temperature outside. We also see that the more insulation we add to the observation unit the more stable the temperature gets.

10 Future work

10.1 Machine learning

Finding the best possible model for the observation unit is difficult. There are many factors that needs to be taken into consideration(outside temperature,inside temperature, what workload is running, layers of insulation, how long a workload runs for, how long it weight between each workload). Right now we only have a very basic not very accurate model. A way to get a better model could be to try and use linear regression on the data in order to find a correlation between the different factors, and use that to make a better model. Another approach could be to use unsupervised learning with a neural network to create a model and based on the model it input the factors and it outputs a action.

To reduce energy usage by the observation unit using unsupervised learning, one option could be to transfer the data it gathers locally to a remote server and have the server create the model using unsupervised learning. Of course here one would need to compare the energy cost of transferring the data vs the energy cost of training the model locally.

Another way to use machine learning to increase the observation units autonomy would be to use neural networks with reinforcement learning and have the observation unit it self learn what each action it can perform does and when it is best to use them to best reach the desired goal of the observation unit. Likey they did in the paper “Playing Atari with Deep Reinforcement Learning”[14] and “Playing Tetris with Deep Reinforcement Learning”[15] where they take in some states(the screen pixels) and output some actions(what buttons to push).

To implement reinforcement learning like this, one would first need to also create a accurate model a simulated observation unit, in order have something to train the system on. This is because you would need to be able to run many 1000 of instances of training for the reinforcement learning. And doing it on a real physical observation unit would take to long.

10.2 Reduce energy consumption

Observation units are often placed far away from places where there are humans, and often have just batteries as a power source and don’t know when they will be recharged or replaced. Reducing the amount of power that the observation unit uses is critical for making the observation unit laster longer on a battery charge. Therefor finding different ways to reduce the OUA systems power use is vital to increase the survival time of the observation unit. A way to reduce the energy use would be to have the unit turn it self of. So if the hardware supports some kind of low power mode(where parts of the observation unit or all off it is turned of), this should be integrated into the scheduler and one could have the scheduler power down the observation unit instead of just having the scheduler program sleep to reduce power consumption.

10.3 Add more sources to gather states from

Observation units will probably have some sort of way to generate power. Therefore we should also be collecting the battery state, as well as the future weather state(weather forecast). This way the OUA can take into account these states when scheduling workloads. And scheduled tasks if they see the battery will be recharged to more than 100%(using equipment like solar panels), or to many to schedule tasks, if they see there will be a few days before the battery will be able to start charging.

10.4 Find a balance between how often we gather states, and power and resource use

For the observation unit to make a decision about what action it should take, it needs to know the state of the observation unit. This means checks various things like temperature and network state. But not only does it cost resources and energy to check these states, but checking the states might impact the state(reading the temperature might increase the temperature). Therefore it would be useful to find a balance between the amount of data that is needed to make a decision about an action and how often that data is gathered. How often the state is gathered might also be different depending on what you are gathering data on(network state might need to be monitored less than the temperature outside the observation unit).

Bibliography

[1] Åshild Ø. Pedersen, A. Stien, E. Soinien, and R. A. Ims, "Climate-ecological observatory for artic tundra - status 2016," *Fram Forum 2016*, pp. 36 – 43, 2016.

[2] Coat, "Coat website." `http://www.coat.no/`. Accessed on 2018-05-23.

[3] P. Limaye, W. Maurissen, K. Lambrinou, F. Duflos, B. Vandevelde, B. Allaert, J. Hillaert, D. Vandepitte, and B. Verlinden, "Low-temperature embrittlement of lead-free solders in joint level impact testing," in *2007 9th Electronics Packaging Technology Conference*, pp. 140–151, Dec 2007.

[4] Q. An, C. Wang, X. Zhao, and H. Wang, "The mechanism study of low-temperature brittle fracture of bulk sn-based solder," in *2017 18th International Conference on Electronic Packaging Technology (ICEPT)*, pp. 1233–1237, Aug 2017.

[5] R. Jedermann, C. Behrens, D. Westphal, and W. Lang, "Applying autonomous sensor systems in logistics—combining sensor networks, rfids and software agents," *Sensors and Actuators A: Physical*, vol. 132, no. 1, pp. 370 – 375, 2006. The 19th European Conference on Solid-State Transducers.

[6] R. J. Edward Sazonov, Kerop Janoyan, "Wireless intelligent sensor network for autonomous structural health monitoring," 2004.

[7] J. Gutiérrez, J. F. Villa-Medina, A. Nieto-Garibay, and M. Porta-Gándara, "Automated irrigation system using a wireless sensor network and gprs module," *IEEE Transactions on Instrumentation and Measurement*, vol. 63, pp. 166–176, Jan 2014.

[8] G. P. R. Filho, L. A. Villas, H. Freitas, A. Valejo, D. L. Guidoni, and J. Ueyama, "Residi: Towards a smarter smart home system for decision-making using wireless sensors and actuators," *Computer Networks*, vol. 135,

pp. 54 – 69, 2018.

[9] python, "Python website." https://www.python.org/download/releases/2.7/. Accessed on 2018-04-24.

[10] R. P. Foundation, "Raspberry pi website." https://www.raspberrypi.org/products/raspberry-pi-2-model-b/. Accessed on 2018-04-24.

[11] adafruit, "Dht22 adafruit website." https://www.adafruit.com/product/385. Accessed on 2018-04-24.

[12] A. Industries, "Dht-sensor-library." https://github.com/adafruit/Adafruit_Python_DHT, 2018.

[13] B. Nuttall and D. Jones, "python-gpiozero." https://github.com/RPi-Distro/python-gpiozero, 2018.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013. cite arxiv:1312.5602Comment: NIPS Deep Learning Workshop 2013.

[15] M. Stevens and S. Pradhan, "Playing tetris with deep reinforcement learning," 2016.