The InteliRack project

Creating a performance rating model for industrialized server racks based on heat mitigating principles

Migjen Hakaj
Master’s Thesis Spring 2016
The InteliRack project

Migjen Hakaj

May 23, 2016
Abstract

Servers have substantially increased their workload capacity over the years as an effect of a more digitalized way of living. With server capacity growing, so does the corresponding heat accumulation in data centers and server racks. There is little clear consensus of how heat as an environmental threat to server racks is mitigated.

This study addresses how system administrators can address thermal threats to server racks through its own methodology and instrumentation; an approach based on heat mitigation standards and own experimentation. An own rack health indicator has been created. InteliRack facilitates its own calculation of a health metric for server racks. Based on principles of thermal segregation in terms server placement and temperature relationships in the rack.

Various experimentation of technical and physical nature was implemented to manipulate server performance and analyze effects of realistic and unrealistic scenarios.

Through experimentation and detailed data collection, the study has come to the conclusion that short term physical and technical manipulation of the rack does not interfere with heat behavior to a considerable extent. The project provides its own approach to create an health indicator for monitored racks based on heat convection principles and temperature data.
3 Approach

3.1 Objectives ........................................... 25
3.2 Implementation of Data collection ......................... 26
  3.2.1 Data collection design .............................. 26
  3.2.2 Data collection .................................. 27
  3.2.3 Temperature collection ............................ 27
3.3 Measurement Tool ..................................... 28
3.4 Storage .............................................. 29
  3.4.1 OpenTSDB ......................................... 29
  3.4.2 Script .............................................. 29
3.5 Visualization of data ................................... 30
  3.5.1 3D visualization .................................... 30
  3.5.2 2D visualization .................................... 30
3.6 Experiments ........................................... 30
  3.6.1 Leap detection test .................................. 30
  3.6.2 Average and differential Values ..................... 31
  3.6.3 Physical Experiments ................................. 32
  3.6.4 Technical Experiment ................................. 33
  3.6.5 Expected results .................................... 33
3.7 Plan overview ......................................... 33
3.8 Alternative Design ..................................... 34

4 Implementation of measurement tool ......................... 35

4.1 Model overview ...................................... 35
4.2 Electrical Management of the Arduino .................... 36
  4.2.1 Connecting the components ......................... 36
4.3 Programming of the IDE ................................ 40
  4.3.1 Dallas temperature ................................. 40
  4.3.2 Onewire and sensor resolution ....................... 41
  4.3.3 Web server ........................................ 42
4.4 OpenTSDB ............................................ 42
4.5 Rack Installation ...................................... 43
4.6 Parsing data for Opentsdb ................................ 44
  4.6.1 Creating Heamaps with R ......................... 46

5 Implementation of experiments .............................. 51

5.1 Extracting value of the differential values .................. 52
5.2 Setup of Leap detection test ................................ 52
5.3 Basic sensor measurements result .......................... 54
  5.3.1 Basic sensor measurement data ....................... 55
  5.3.2 Usage trend: day versus night ....................... 55
  5.3.3 No usage ............................................ 56
5.4 Complete obstruction .................................... 56
  5.4.1 Complete obstruction results ......................... 59
5.5 Loose obstruction ...................................... 60
  5.5.1 Loose obstruction results ......................... 61
5.6 Shield removal experiment ................................ 62
  5.6.1 Shield removal results ............................. 63
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.7</td>
<td>Bezel removal</td>
<td>64</td>
</tr>
<tr>
<td>5.7.1</td>
<td>Bezel results</td>
<td>65</td>
</tr>
<tr>
<td>5.8</td>
<td>Technical experiment</td>
<td>65</td>
</tr>
<tr>
<td>5.8.1</td>
<td>Experiment specifications</td>
<td>66</td>
</tr>
<tr>
<td>5.8.2</td>
<td>CPU Experiment results</td>
<td>66</td>
</tr>
<tr>
<td>5.8.3</td>
<td>Bottom Heavy results</td>
<td>67</td>
</tr>
<tr>
<td>5.8.4</td>
<td>Top heavy cpu</td>
<td>68</td>
</tr>
<tr>
<td>6</td>
<td>Analysis</td>
<td>71</td>
</tr>
<tr>
<td>6.1</td>
<td>Statistical methods</td>
<td>71</td>
</tr>
<tr>
<td>6.2</td>
<td>Experiment analysis</td>
<td>71</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Areas of effect</td>
<td>72</td>
</tr>
<tr>
<td>6.3</td>
<td>Obstruction cases</td>
<td>72</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Complete obstruction</td>
<td>72</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Loose obstruction</td>
<td>75</td>
</tr>
<tr>
<td>6.3.3</td>
<td>Shield and bezel removal</td>
<td>77</td>
</tr>
<tr>
<td>6.3.4</td>
<td>Comparing the cases</td>
<td>77</td>
</tr>
<tr>
<td>6.4</td>
<td>CPU Experiments</td>
<td>78</td>
</tr>
<tr>
<td>6.5</td>
<td>InteliRack, proposing an health metric. Using the golf rating approach</td>
<td>80</td>
</tr>
<tr>
<td>6.5.1</td>
<td>Cooling effectiveness</td>
<td>80</td>
</tr>
<tr>
<td>6.5.2</td>
<td>Consolidation of cooling efficiency and temperature</td>
<td>81</td>
</tr>
<tr>
<td>6.5.3</td>
<td>Challenges of the health metric approach</td>
<td>82</td>
</tr>
<tr>
<td>7</td>
<td>Discussion</td>
<td>83</td>
</tr>
<tr>
<td>7.1</td>
<td>Design</td>
<td>83</td>
</tr>
<tr>
<td>7.1.1</td>
<td>Background for design</td>
<td>83</td>
</tr>
<tr>
<td>7.1.2</td>
<td>Testing duration</td>
<td>84</td>
</tr>
<tr>
<td>7.2</td>
<td>Implementation</td>
<td>85</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Implementation of the measurement tool</td>
<td>85</td>
</tr>
<tr>
<td>7.3</td>
<td>Experiments</td>
<td>86</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Data validity</td>
<td>86</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Impact of experiments</td>
<td>86</td>
</tr>
<tr>
<td>7.3.3</td>
<td>Compute rack possibilities</td>
<td>87</td>
</tr>
<tr>
<td>7.3.4</td>
<td>Alternative experiment approach</td>
<td>87</td>
</tr>
<tr>
<td>7.4</td>
<td>Loss of data</td>
<td>88</td>
</tr>
<tr>
<td>7.5</td>
<td>LDT Analysis</td>
<td>88</td>
</tr>
<tr>
<td>7.6</td>
<td>Future work</td>
<td>89</td>
</tr>
<tr>
<td>7.7</td>
<td>Humidity measuring</td>
<td>89</td>
</tr>
<tr>
<td>7.8</td>
<td>Effects of an adjacent racks</td>
<td>89</td>
</tr>
<tr>
<td>7.8.1</td>
<td>Dynamic heat load balancer</td>
<td>90</td>
</tr>
<tr>
<td>7.8.2</td>
<td>Monitoring tool</td>
<td>90</td>
</tr>
<tr>
<td>7.8.3</td>
<td>InteliRack for groups</td>
<td>90</td>
</tr>
<tr>
<td>7.9</td>
<td>Impact</td>
<td>91</td>
</tr>
<tr>
<td>7.9.1</td>
<td>Project experiences</td>
<td>91</td>
</tr>
<tr>
<td>8</td>
<td>Conclusion</td>
<td>93</td>
</tr>
</tbody>
</table>
9 Appendix

9.1 Arduino, Mastertemp ........................................... 99
9.2 OpentTSDB Parser ............................................. 111
9.3 Leap Detection Test script ................................. 121
9.4 2D Heatmap .................................................... 124
9.5 3D Heatmap .................................................... 125
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Raised Airflow Design</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>CACS</td>
<td>10</td>
</tr>
<tr>
<td>2.3</td>
<td>HACS</td>
<td>11</td>
</tr>
<tr>
<td>2.4</td>
<td>Rack types</td>
<td>13</td>
</tr>
<tr>
<td>2.5</td>
<td>Passive heat management in server racks</td>
<td>14</td>
</tr>
<tr>
<td>2.6</td>
<td>Active heat management</td>
<td>15</td>
</tr>
<tr>
<td>2.7</td>
<td>Arduino overview</td>
<td>17</td>
</tr>
<tr>
<td>2.8</td>
<td>Opentsdb</td>
<td>20</td>
</tr>
<tr>
<td>3.1</td>
<td>Possible temperature collection model</td>
<td>28</td>
</tr>
<tr>
<td>3.2</td>
<td>Project plan</td>
<td>34</td>
</tr>
<tr>
<td>4.1</td>
<td>Measurement design</td>
<td>36</td>
</tr>
<tr>
<td>4.2</td>
<td>Shield on top of Arduino Mega</td>
<td>37</td>
</tr>
<tr>
<td>4.3</td>
<td>sensor Connection</td>
<td>39</td>
</tr>
<tr>
<td>4.4</td>
<td>Plot of temperatures in Opentsdb</td>
<td>43</td>
</tr>
<tr>
<td>4.5</td>
<td>Rack Installment</td>
<td>44</td>
</tr>
<tr>
<td>4.6</td>
<td>Script Flow</td>
<td>45</td>
</tr>
<tr>
<td>4.7</td>
<td>3D heatmap</td>
<td>49</td>
</tr>
<tr>
<td>5.1</td>
<td>Planes</td>
<td>51</td>
</tr>
<tr>
<td>5.2</td>
<td>Naming</td>
<td>54</td>
</tr>
<tr>
<td>5.3</td>
<td>All sensors trend</td>
<td>55</td>
</tr>
<tr>
<td>5.4</td>
<td>Sensor reference</td>
<td>56</td>
</tr>
<tr>
<td>5.5</td>
<td>No Usage</td>
<td>57</td>
</tr>
<tr>
<td>5.6</td>
<td>No usage comparison</td>
<td>57</td>
</tr>
<tr>
<td>5.7</td>
<td>Complete Obstruction</td>
<td>58</td>
</tr>
<tr>
<td>5.8</td>
<td>Complete obstruction results</td>
<td>59</td>
</tr>
<tr>
<td>5.9</td>
<td>Complete obstruction differentials</td>
<td>60</td>
</tr>
<tr>
<td>5.10</td>
<td>Loose obstruction</td>
<td>61</td>
</tr>
<tr>
<td>5.11</td>
<td>Loose obstruction result</td>
<td>61</td>
</tr>
<tr>
<td>5.12</td>
<td>Loose obstruction differentials</td>
<td>62</td>
</tr>
<tr>
<td>5.13</td>
<td>Shield removed</td>
<td>63</td>
</tr>
<tr>
<td>5.14</td>
<td>Shield removal result</td>
<td>64</td>
</tr>
<tr>
<td>5.15</td>
<td>Shield differential result</td>
<td>64</td>
</tr>
<tr>
<td>5.16</td>
<td>Bezel removal</td>
<td>65</td>
</tr>
<tr>
<td>5.17</td>
<td>Bezel removal result</td>
<td>66</td>
</tr>
<tr>
<td>5.18</td>
<td>Bottom heavy cpu</td>
<td>67</td>
</tr>
</tbody>
</table>
5.19 Bottom heavy cpu differentials .............................. 68
5.20 Top heavy cpu .................................................. 69
5.21 Top Heavy Differentials ....................................... 69

6.1 Normal versus complete obstruction ....................... 72
6.2 P2_leapdetections .............................................. 74
6.3 Difference front Top vs bot .................................... 75
6.4 LDT difference between front and back .................... 76
6.5 All Cases .......................................................... 78
6.6 Top heavy load versus bottom heavy load .................. 79

7.1 Data center Health metric ...................................... 84
7.2 LDT sensor flow ................................................. 89
## List of Tables

2.1 Measurement accuracy ............................................. 19
4.1 Wire management .................................................. 39
4.2 3D Coordinates ...................................................... 48
Acknowledgements

I would like to acknowledge Kyrre Begnum as my supervisor who supported and guided me throughout the process of developing this thesis. A great thanks to Evangelos Tasoulas. Without his expertise and guidance there would be no thesis. I sincerely express my gratitude to both Kyrre and Evangelous for their contributions.

Thanks to Oslo and Akershus University College and the University of Oslo for facilitating the opportunity to take part in the network and system administration course. A special thanks to all lecturers, professors, fellow students and friends for providing guidance and fruitful discussions.

Lastly I would like to give thanks to my family for supporting and encouraging me to complete this masters programme.

Sincerely, Migjen
Chapter 1

Introduction

1.1 Introduction

Almost every aspect of our daily life has a software counterpart that runs in a data center somewhere. Wherever we go we are affected by a data center directly or indirectly. The probability that a person is wearing or utilizing a device or service that connects to a data center frequently is high in today’s way of living. Everything from purchases one makes at the local grocery store to the swipes one makes on their cellular phone.

In essence, a data center is what facilitates an organization’s IT operations and services. A very general description could be an assembly of servers put together. At a large scale data centers could contain hundreds of servers which store petabytes of information and multiple hundreds gigabit of bandwidth which serves its respective users 24/7 [1]. Companies depend on services supported by data centers to various extent. They serve general services like public transportation, hospital services to traffic control. With the continuous trend of a lifestyle which surrounds us with technological features, data centers play an important role.

A data center is susceptible to multiple threats. Threats revolving misconfiguration and computer security often receives most of the attention. Environmental factors are a threat to electronic equipment. Poor management and control over environmental factors like heat could have everything from fatal to barely noticeable effects for an organization. An increase from 20 °C to 30 °C could reduce long term reliability by as much as 50 % [2].

Heat has a grave effect on a data centers performance which brings multiple challenges when administrating a data center. This is something that has been giving data center administrators mind boggling issues through the lifespan of data centers. In March 2010 one of Wikipedia’s European data centers overheated. While these servers were down the administrators had intentions to reroute the incoming traffic to a North American facility. Because there was an error with the DNS configuration, this faulted and led to the site being inaccessible [3]. Wikipedia received about 430 million
hits per day, as of May 2015 [4]. An example of a potential consequence of not being able to react to heat.

A better model is needed to provide system administrators the tools needed to deal with data center heat challenges. There are several ways of going around when dealing with heat. A solution is implementing more sensors which will provide more data. Untreated, this does not give more knowledge over the general health of the infrastructure. A model needs to combine the sources of data into a visual output that gives an overview of the data. By having a model that utilizes heat maps it is easier to see trends, hot-spots and draw analysis over cooling management. This will facilitate the possibility of creating an alarm system based on set thresholds. A model could also produce a health metric that is derived from all the sources of data. These are some features that could draw value from adding more sensors into a data center.

A methodology to provide counteractive measures to prevent and react early to overheating is of importance for the short and long effects. Today’s method mostly comprise of measuring heat from a single sensor which gives a metric for the general state of the room. This becomes inaccurate as server racks are designed differently which influences how air flows through them. There will be a fluctuation in temperatures from one zone to another. Taking into consideration that 40% of the total energy consumption in data centers is spent cooling the housed hardware. Makes it vital that the process of doing so is handled in an efficient manner [5].

1.1.1 Problem statement

I want to develop a model and instrumentation that achieves a precise overview of the temperature in a server rack that enables a system administrator to develop temperature aware placement strategies.

A model could be explained as a representation that shows the construction or appearance of something. A model can be anything from an image, video or a drawing.

Instrumentation is a term commonly used in the field of research to describe a measurement device. In this case a deliverable that helps develop methods of treating heat based on knowledge extracted from a dynamic visual presentation of data. The instrumentation in project will contain of both sensors and software.

A server rack is a place holder for servers. Multiple servers are stacked and linked together to serve different purposes or to increase a service’s capacity. These racks have different designs of how to ventilate the containing hardware.
A system administrator is someone who is responsible for the up-keep, administration and configuration of computer systems. In this thesis the system administrator scope is narrowed down to the individuals that are in charge of the data center or the server room.

The architecture of a server room is a key factor for managing heat. Temperature aware placement strategies are needed to implement a structured plan for optimization of energy utilization. An administrator will have to map out where the different components are to be placed within a server room in before hand. They need to be prepared to change the existing layout based on new knowledge derived from services monitoring environmental aspects.
Chapter 2

Background

This chapter explains technologies that are going to be used as well as concepts and methods which will be utilized to reach the goal of providing an answer to the problem statement. Before getting to the stage of making intelligent choices of how to manage heat we need to look at different factors and concepts.

2.1 Data center

Any room which is used to host a group of servers could be defined as a data center. The evolution of IT has forced the data centers to grow exponentially. The data centers facilitation of IT services has been a major key. One could say that they are a victim of their own success [6]. The usage of services that are supported by a data center have skyrocketed over the years. Banking, e-commerce and online transactions are all dependant on a data center and their corresponding server capacity. Every year there are approximately five million servers that are deployed into different data centers. This gives a sense of the growth that data centers are experiencing [7].
A data center has four main components [8]

- **Operations**
  The administrators of the data center itself. In general the operations group makes sure that the infrastructure is maintained, upgraded and repaired when necessary.

- **White space**
  This is the usable raised floor environment in square feet. The term white space may still be used for data center that does not have raised floors.

- **IT equipment**
  All the hardware used to operate the data center. Server racks, servers, cables, management systems and network gear.

- **Support infrastructure**
  Generators, computer room air conditioners (CRAC’s) figure: 2.1.3, chillers, air distributions systems etc. These types of inventory takes claim of up to 4-6 times as much space as white space and therefore is an essential part of planning the data center layout.

With the general trend being that more capacity is pushed into a less space, design and layout plays a essential role.

### 2.1.1 Data center infrastructure

Having a well thought out floor plan of your data center greatly affect power density capability and resource usage. Despite having such a critical role many utilizes an incremental style of deployment instead of following a plan [9].

When designing the layout for a data center there are three main principles [10].

- **Space savings**
  Is an increasingly prioritized factor especially with the growth and future demand of network connections, bandwidth and storage space. Data centers and server racks takes up a lot of real estate and space which is expensive. When designing the data center layout an evolving business needs to be accounted for. Therefore excess space needs to be left available to make room for growth and change. This creates problems for the cooling systems that bases itself on design principles of how air flow through the room.

- **Reliability**
  Downtime could be directly translated to loss of income. Therefore measurements to prevent downtime has to be implemented. A data center should be designed to be redundant, fail safe, reliable and available at all times. Virgin Blue’s ticket reservation system went down due to hardware issues in their data center in September 2011.
The outage lasted for 11 days where the company reported in missing out of 20 million dollars in profit [11].

- **Manageability**
  The data needs to be highly reliable and have the utilities to handle many different scenarios. Everything from disaster recovery, upgrades and modifications. This contains cable management, configuration of routing paths, a centralized point of patching and more.

### 2.1.2 Energy consumption

Data centers have become the largest and fastest growing consumer of electricity. Last year U.S data centers consumed an astonishing 91 billion kilo-watt hours of electricity. By 2020 its estimated that the usage will be over 120 kilowatt-hours [12]. With the shift in the IT paradigm moving over to a cloud based technologies, the need for bigger and more efficient data centers is increasingly growing. As more capacity is pushed into the data centers, more energy is spent. Especially when managing heat. Measures to maximize the data centers capabilities and performance are continuously being implemented. Blade servers which has a modular design where the goal is to minimize the physical space needed are an example of that. This allows organizations to feed more servers within less space, naturally creating more heat. The open compute project led by Facebook, focuses on redesigning how hardware technology is implemented to efficiently support the growing demands [13].

### 2.1.3 Heat management

Heat management is a general term which could be utilized in different fields. In this case heat management is referred to managing heat within a server rack/room. There are many ways of going about the implementation of measures to control heat. The most convenient solution is to use an incremental deployment strategy. A poor method as factors like cooling, energy consumption, space and airflow are not taken into consideration when setting up the layout of the data center. Once a poor layout plan has been deployed for the server room its difficult to impossible to recover from the consequences [9].

**Computer room air generator**

A CRAC unit could be situated in various setups. The most successful one being the process of cooling air and having it dispensed through an elevated floor where the floor is perforated, allowing it to flow up from the floor.

The CRAC system is what provides the cool energy source that alleviates the data center and server racks of heat. It monitors and maintains the air temperature, temperature and humidity in a data center.
[14]. It's likely that the usage of CRAC systems will endure for several years. According to [15] we might see a shift in a 10 year period from now where data center administrators will shift to more eco-friendly methods like free cooling, liquid cooling and chilled water cooling.

A CRAC unit could be situated in various setups. The most successful one being the process of cooling air and having it dispensed through an elevated floor where the floor is perforated, allowing it to flow up from the floor.

**Data center design**

There are some concepts that need to be explained before taking on the challenge of heat management in data servers. A typical design of a data center would contain these items:

- CRAC system
- Elevated Floor
- Return Air plenum
- Hot and cold aisles

When these components are implemented, the actual placement of servers and how they are situated in terms of cooling needs to be set in place. Server racks are mounted so the servers own internal fans blow the hot air into the hot aisle of the data center and receive cold air from the cold aisles. These are all components and practices that are in place to control heat. See fig 2.1.
At figure 2.1 it visible how airflow is controlled to manage thermal cooling. Its important to know that hot and cold air does not mix [16]. Air current mixing is a important factor when looking at measures to control heat. When warm air faces cold air it forces the cold air down and pushes itself on top of the cold air. So its common that data centers are colder the closer one get to the floor. Physics has to be taken into consideration when designing heat management systems, like cooling. Air from the ceiling is pushed into the CRAC system which cools the air and pushes it back to the bottom to keep a flow of cool air to the server room.

**Hot and cold aisles**

We know that hot and cold air does not mix. A common implementation is to create aisles in the data center that separate these counterparts. Using this design principal is sought out to be the best practice principle [17].

**2.1.4 Hot and cold air containment**

Heat management techniques using air containment strategies has a prerequisite of having a hot and cold aisle setup. According to [18] having a server environment where implementing a hot aisle containment strategy could save 43 % in annual cooling cost.

Some benefiting factors using containment strategies are:

- Elimination of hot spots
  As the hot and cold aisles are physically separated, hot spots are eliminated.
• Economizer mode hours are increased
  The basic function of a chiller is to remove heat from a data center by compressing and expanding a refrigerant to keep chilled water at a set supply temperature which is about 7 Celsius. Having an outdoor temperature 11 Celsius below that point allows the CRAC system to bypass the chiller and shut it down which results in reduced energy consumption[18]. Not having to cool-down the chiller means that the system is running in economizer mode.

• Humidification/de-humidification costs are reduced
  By not mixing hot and cold air the air supply temperature can be increased and kept over dew point temperature. Resulting in not having to spend resources on humidifying the air.

**CACS**

Cold air containment system encloses the cold aisle. Meaning that the exterior room becomes a large hot air plenum. Figure 2.2 shows the fundamentals of cold air containment. Setting up CACS is achieved by enclosing the sides and top of the cold aisles in a data center.

![Figure 2.2: CACS](image-url)
HACS

Hot air containment systems isolates and collects the equipments hot exhaust air creating a cold air plenum in the exterior room. Figure 2.3 shows how the cold air is contained, giving a steady feed of cold air to the server inlets. HACS is deemed to be a more efficient method then CACS as it yields a higher hot aisle temperature which means that the trade off when spending energy to cool down the air is more efficient. Also HACS results in more economizer hours. A drawback for a CACS setup is that servicing the data center could be a troublesome affair as the room is very hot. This would result in mixing of hot and cold air which will disrupt the air flow [18].

Figure 2.3: HACS
2.2 Server racks

Server racks are the physical components which provide equipment placement within a data center.

“A data center rack is primarily designed to house servers in different form factors (such as rack-mounted or blade servers). Although they are mainly designed to hold servers, some are designed to hold other components”[19]

These components being

- Networking equipment
- Telecommunication equipment
- Cooling systems
- Uninterruptible power supply

Besides housing hardware components server racks also provide protection and infrastructure to cool and power the IT equipment [20]. Unlike data centers, server racks has an industry standard. According to the electronic industries association (EIA) the 19 inch rack has been a standard equipment rack for over 50 years [21]. When discussing racks units (U) as a metrics is used. (U) is a symbol of the server capacity load the rack has.

The are two types of the 19-inch rack which are commonly used:

- Fully enclosed rack
  Most commonly used as it provides protection due to lockable doors and side panels. Perforated panels let air flow freely. These racks are commonly sold from the major vendors withing a range of 42-47 (U) [21].

- Open frame rack cabinets
  Open frame racks are a bare solution. A significantly cheaper solution and has excellent cooling as there are no enclosing. They provide no physical protection or air containment. The open frame cabinets will not be subjected to further analysis in this project as it has no value in terms of researching heat management within server racks as air is not contained or controlled.

The EIA standard dates all the way back to 1934 and was not revised again until 1994. The electronic world has embraced the practical design of racks which allows system administrators to stack and compress multiple components. An important factor when leading companies which sees their data centers are there business edge spends about 3000 dollars per square feet when building their data centers [22].
2.2.1 Heat management

Server racks play a central role in heat management. There are many studies which revolve around thermal awareness, management control regarding data centers. Data centers are filled with racks, so it’s natural that the heat derived from these black towers are of interest. Closed server racks are the main focus as there is a likelihood that heat will build-up inside enclosed racks which has not in the open rack frame solutions.

There are two methods that are implemented as for heat management in server racks. Active and passive. Passive heat management relies solely on thermodynamics of conduction, radiation and convection to complete the heat transfer process [23]. Active heat management is where an external energy source is introduced i.e. a cooling fan. Before going into these management techniques the basic theory of cold versus hot air physics needs to be looked at.

2.2.2 Convection

principles of rack cooling are based on convection. Where one would want cold air to rise and push hot air out of the rack. It is based on the principal of density.

\[ \text{Density} = \frac{\text{Mass}}{\text{Volume}} \]

When air is heated the particles will vibrate causing them to move further apart. So if a fixed mass is set, the volume will increase, based on that a greater amplitude of vibrations takes up more space. If the mass is kept constant one could say that the density will be inversely proportional to the volume. Basically, if the volume increases for a fixed mass, then density will decrease. Forcing hot air to be lighter than cold air.
Passive heat management

Physics imply that cold air sinks while hot air rises. The goal with passive heat management is to create a "chimney" effect which transports the hot air out of the rack. There are several factors that can be identified before a good rack setup can be achieved. The hottest servers should be placed at the bottom because of the air flow. Servers often have their own intake and exhaust fans. This needs to be taken into consideration because placing these servers wrong could disrupt the airflow and cause re-circulation of hot air which will lead to temperature rises. Servers with intake and exhaust fans at the side should not be stacked. Figure 2.5 visualizes typical practice.

![Diagram](image)

(a) airflow

(b) faulty airflow

Figure 2.5: Passive heat management in server racks

Active heat management

In cases where natural convection is not enough for transporting heat it is necessary to force the heat out using an external force i.e. fans. When setting up fan speed we have to look at how many cubic feet per minute of airflow (CFM) a fan moves. The external fan that is set at the top of the rack which is used to draw air out needs to have more CFM capacity then all the servers internal intake fans have combined together to not disrupt the airflow. Instead of having a perforated backdoor a solid back plate should be implemented to better control the air. Another important factor is to not add a vented rack face at the top as this would disrupt the fan ability to circulate the air out of the rack. It will make the fan only suck out the cold air that it fetches from the outside room whilst the hot air would be stuck inside the rack [23].
2.2.3 Challenges

One of the biggest challenges is uneven cooling. The typical cause for this is that cold air from the bottom of the rack does not reach the top of the rack with the same cold temperature as it does at the bottom. Cold air is pushed from the CRAC system and therefore naturally the air is colder to the source. There is a difference when measuring the start of the air's trajectory versus at the end. This creates issues where servers usually have higher temperatures the further up in the rack they are placed. New methods are constantly under development. Other designs are viable options. Like Fully enclosed racks which contain their own CRAC system within the rack. These solutions have better cooling but the trade off with the high energy consumption is a challenge difficult to look past.

2.3 Incremental deployment

Following principles and a structured design when setting up the layout of a data center and server racks allows for good thermal management. The problem is that the opportunity to do so is often non-existent as the reality often follows incremental deployment of servers [24]. Having an environment where the thermal load is static is easier to control. When adding new hardware into either a rack or data center incrementally, the thermal load changes which could create a different need for the set up of optimized cooling. Often, its not an option to restructure the whole rack for each addition of hardware. The problem spectrum also revolves around upgrades, replacing of hardware after failure and sudden fluctuations in workload.

Figure 2.6: Active heat management
2.4 Hosted environments

The trend is moving over to renting server space rather than setting up an own data center with the corresponding server racks. Companies like Amazon, Rackspace and Google are offering server capacity as a "pay as you go" model [25]. Companies like Netflix, Airbnb, NASA and Yelp are all companies which are hosted at external sources like Amazon [26]. The effective reason for companies to follow this trend is the rising cost level of managing and tuning the servers. This has led to a business model where hosting companies have data centers which provision thousands of dense servers [27]. These server parks are often set up at small real estate in relation to the amount of server capacity pushed into that same area. This is why power management strategies have to be implemented. The practical meaning of this is that the user themselves seldom have physical access or control over their "own" servers as they are not owned by them selves. They are more likely to get rack space and servers assigned to them based on their customer needs. A customer comes and goes which makes the server parks dynamic and not static. This is a huge challenge to heat management as designing and planning the thermal balance of server racks and data center requires knowledge of where the thermal load will have its weight.

2.5 Temperature aware placement strategies

As terms and common practices of heat management within data centers and server racks have been explained it is important to look at methods and tools for analysis of different situations that might occur in data center and server rack environments. Tools that will help facilitate analysis that can create knowledge from measured data that will yield indicators of how the server rack is situated.

2.6 Tools

The following section will describe the tools used to fulfill the projects tasks and goals.

2.6.1 Arduino

Arduino has been used in over thousands of projects, from simplistic tasks to complex science projects. It was developed at Ivrea Interaction Design Institute. It is an open source platform which has gathered contributions from different types of groups like programmers, artists, professionals, students and hobbyists [28]. Arduino is based on easy to use hardware and software. Arduino boards are able to read inputs from sensors. Sensors measuring i.e. temperature, humidity and light while reporting the results back. The behaviour is controlled and processed through a micro controller on the board. It uses its own Arduino programming language which is based on C and C++.
Arduino has five traits that it is known for:

- Inexpensive
- Cross-platform
- Simple, clear programming environment
- Open source and extensible software
- Open source and extensible hardware

**Technical specifications of a Arduino Uno**

A quick overview over the board is necessary to explain the Arduino layout. The Arduino UNO which is shown at figure:2.7 is used as a reference board as this is the most commonly used board for projects and experiments.

From clockwise from the top center.

![Arduino overview](https://www.arduino.cc/en/Reference/Board)

**Figure 2.7: Arduino overview**

Source: https://www.arduino.cc/en/Reference/Board
As visible at figure:2.7 the Arduino mega board contains a set of 12 digital pins for digital transmission and six analog pins. The difference between digital pins and analog pins are that digital pins are either switched on or off, 0 or 1. There is nothing in between. Which works great for implementing light switches or sensors. While analog pins are everything in between. Say that digital is black and white while analog pins can be dark grey, light grey and so on. In terms of voltage the pins receive, digital pins has a output of either 0 or 5 volt and analog can produce voltage in between.

The light green and Orange are the pin for digital ground and analog reference. The dark green pins(TX and RX) are reserved for Arduino’s read and write functions. S1 marked is Arduino’s reset button, which sends the reset signal to the board. The power and ground are marked as orange and light orange where light orange is ground and orange is power. The mega board also has a external power supply input at X1(pink). The USB connection is used for uploading code and sketches to the Arduino board.

**Arduino megaboard**

Arduino mega board is a micro controller based on ATMEGA2560 [29]. It is equipped with 54 digital pins, 16 analog pins and four hardware serial ports. The mega board offers more memory than an Arduino Uno, which gives more freedom of how to program the micro controller. It can be powered by an external power supply or through a USB connection. The recommended volt operating level is 7-12 volts. It has three different types of storage. Flash memory, where code is stored. Sram and EEPROm which are used to store information that are in process and non volatile data.

**DS18B20 Temperature component**

The Arduino board supports multiple types of sensors and components, one of them being the DS18B20 sensor. The sensor provides a range of 9-12 bit temperature measurements. Both Fahrenheit and Celsius are viable. The set bit is linked to measurement accuracy. The higher number of bits the more accurate temperature reading is received, but it will take longer time to process 2.1. It has the capacity of running of ”parasite power” which means that it can power it self using the data pin. Making it sustainable without a dedicated power pin. Each DS18B20 has a unique 64 bit serial code which allows multiple sensors to run on the same one-wire bus. This
gives the opportunity to acquire and control multiple sensors connected to one pin through one micro controller.

Table 2.1: Measurements accuracy

<table>
<thead>
<tr>
<th>Mode</th>
<th>Resol</th>
<th>Conversion time</th>
</tr>
</thead>
<tbody>
<tr>
<td>9 bits</td>
<td>0.5°C</td>
<td>93.75 ms</td>
</tr>
<tr>
<td>10 bits</td>
<td>0.25°C</td>
<td>187.5 ms</td>
</tr>
<tr>
<td>11 bits</td>
<td>0.125°C</td>
<td>375 ms</td>
</tr>
<tr>
<td>12 bits</td>
<td>0.0625°C</td>
<td>750 ms</td>
</tr>
</tbody>
</table>

**PIC Microchip ENC28J60**

ENC28J60 contains all the necessary hardware to create a network interface on the Arduino. It supports a 3.3 and 5 volt power supply. The module utilizes ENC28J60 Stand-Alone Ethernet Controller to handle the network protocols [30]. It has its own library named Ethercard which supports its hardware. The library has the same functionality like drivers for software in programs.

**2.6.2 Web service**

The diversity of the Arduino serves possibilities limited to the scope of the developer. With the freedom of C++ programming language and having a network interface that can be latched on to the board, it is possible to create a simple web server that can do multiple tasks, i.e. store data, work as a monitoring tool and visualize data. Again, limited to the scope of the developer. Having a web server which holds the data as a "middleman" opens for other web tools and libraries. For example Httplib, Beautifulsoup in Python.

**2.6.3 OpenTSDB**

OpenTSDB is a time series database. It runs on top of Hadoop and Hbase. Hadoop is a framework that allows for distributed processing of large data sets over a cluster [31]. Hbase is Hadoop’s database which is a scalable and distributed big data store [32]. A time series database is a series of numeric data points of some numeric value over time i.e. Temperature readings. It also contains a plotting functionality. The basis of any given plot is the metric itself. It takes the metric, finds all time series for that specific metric and does arithmetic operations that aggregates values which then are plotted [33]. The plotting is highly adaptable, allowing for summarizing of data, retrieving excerpts of specific time windows, min, max values and much more.

Opentsdb utilizes a series of HTTP API’s for inserting and querying data from the database. For example, it has a restful put API which allows
users to push JSON formatted data through HTTP. It supports large chunks of data sent which makes it highly scalable. It also supports telnet for pushing data.

At figure 2.8 follows an illustration of how Opentsdb is works. It consists of a time series daemon (TSD) and a set of command line utilities. Interaction with OpenTSDB is achieved through running one or multiple TSD’s. There is no master and slave relationship between TSD’s. They all operate individually, and the amount of TSD’s can increase according to load, which makes OpenTSDB highly scalable to manage high throughput. Every TSD uses Hbase to store data which has a schema which is highly optimized for aggregation of time series.

2.6.4 Python

Python is a powerful programming language which allows for effective integrating of systems [34]. It is a high level programming language which was developed by Guido Van Rossum. The design philosophy behind python is that it should emphasize code reliability and have a syntax which demands fewer lines of code than lower level programming languages like C#, C++ and Java.

Python has a large built-in library as well as numerous addons that contains a diverse set of functionality. It supports object oriented programming as well as functional programming. Matplotlib is a diverse library used for various plotting techniques that produces publication worthy figures [35].

2.6.5 R

R is a statistical and graphics based programming language which was developed by Bell laboratories former AT&T, present Lucent Technologies. Mainly by John Chambers [36]. R is often considered to be a modified implementation of the "S" programming language. R can be used for a variety of analysis such as classic statistical testing, time series modeling, classification, clustering and more.
2.7 Related work

As pollution, energy conservation and data processing are in the spectre of attention in today’s picture, there is research being conducted in the field of heat management as this touches into all these three topics. Different approaches are explored to find new innovative methods of optimizing performance hence to the three factors mentioned.

2.7.1 Smart temperature monitoring

In the paper “Smart temperature monitoring for data center energy efficiency” [37] an algorithmic method for placement of sensors is implemented. The goal is to create an algorithm for finding the most suitable placement for sensors and the required amount of sensor to find hot spots which could be used for adaptation of the CRAC system. The purpose of this is to decrease power spillage deriving from the CRAC system. The result claims to give direct pin points of where to put sensors based on simulations which would optimize the CRAC efficiency no matter work load of the servers.

2.7.2 Weatherman

Weatherman uses a machine learning approach with the use of neural networks to manage heat. Weatherman uses the neural network to predict how heat is generated and the air floor within a data center. Using this method, weatherman is able to reduce energy cost of the CRAC system with 13-25 %. Weatherman greatly utilizes heuristic data to filter the large space of possible workload distribution in their energy efficient solutions. The result is the product itself which is an accurate, automated, on line, and cost-effective thermal topology prediction [38].

2.7.3 Thermostat, a CDF approach

Many projects has followed principles of computation fluid dynamics(CFD). CFD is the use of applied mathematics, physics and computational software to visualize how gas or liquid flows [39]. In the thermostat project a modeling tool based on CFD is used to obtain thermal profiles of rack-mounted servers. Based on these thermal profiles, intelligent choices of how to place servers, chillers, set up the architectural layout and room design can be optimized [40].

2.7.4 TSM: Efficient Thermal and Server Management for Greening Data Centers

This study presents a workload management strategy called thermal and server management (TSM) to cut energy usage in a data center [41]. The framework consists of a thermal control coordinator (TCC) which is based on total energy minimization problematization. It basically keeps track
of its corresponding application agents, which are the individual chillers placed in the room. Based on a primal decomposition algorithm it decides how to spend energy.

2.7.5 Mercury and Freon: Temperature Emulation and Management for Server Systems

[42] Presents a framework that is named Mercury which is a suite, comprised of four pieces of software. A solver, a set of monitoring daemons, sensors library calls and a tool for generating thermal emergencies explicitly. The simulator for generating scenarios of thermal emergencies are based on newton’s law of cooling.

\[ Q_{transfer}, 1 \rightarrow 2 = k \times (T_1 - T_2) \times time \]

Based on the mercury framework for simulation thermal environments, freon was developed which is a tool for managing thermal emergencies. Freon manages components temperatures in a cluster environment. It uses a rationale based on that periodic temperature monitoring and feedback control that can tackle complex and unpredictable conditions that derives from thermal emergencies. Its physically implemented as a dynamic load balancer. Measuring the computers internal values like CPU temperature as a metric for showing the health of a server. Freon dynamically shifts workloads to other servers in the cluster based on the internal computer temperature thresholds set by the administrators. Based on these principles the projects results claims to be a measure against thermal threats without receiving thermal degradation.

2.7.6 We Didn’t Start the Fire: Using an Agent-Directed Thermal Simulator to Keep Servers Cool

"We Didn’t Start the Fire" is another algorithm based model for cooling servers. Agent-Directed Thermal Modeler (ADTM) provides a low level learning curve model which provides an overview quickly [43]. The overview which provides a graphical pictorial output could be used to find out which algorithm is most suitable for the specific data center. The speed factor allows for measurement of different algorithms to check their performance. ADTM gives a 13% increase in energy reduction in an overloaded center. ADTM main feature is its ability to compare different load placement algorithms through hasty modeling which shows how performance is compared to each other. Most reviews used the popular XInt-GA algorithm which is a thermal-aware genetic algorithm. It uses genes from parent population and forms them to create a descendant population.
2.8 Industry Example, It Watchdogs

It watchdogs is a company which specializes itself in providing technical solutions for monitoring physical factors within the IT and telecom industry. They have developed a monitoring tool which is able to monitor several environmental factors like temperature, door movement, power supply, humidity and smoke levels. Their monitors also have GSM and SNMP implemented which allows their monitor to be integrated with existing monitoring tools and alarm system administrators over telephone when situations occur [44]. They do not provide any methodology of how to place the sensors itself or analysis based on data received. Which is the real pay off for a system administrator where the end goal is to keep energy efficiency and a low temperature level.

2.9 Project positioning

Many of the projects described have taken different approaches to implementing countermeasures to the rising issue of heat management in server racks. Managing these racks also has direct influence on the total health condition of the data center. There are many intelligent methods of simulation future heat development using CDF models. Also there are implementations of different tools which help facilitate the management of heat with focus on reducing energy consumption, monitoring and cost reduction. This projects implements its own tool for monitoring the heat in a standard 19 inch server rack, but the goal of the project itself is to create instrumentation and methodology based on the data received. Test out theories, different types of workloads distributions to see the short and long term effects of these experiments.
Chapter 3

Approach

The approach describes the tools, methods and experiments which will be used to address the problem statement set in the introduction. The problem statement describes key features like "develop a model and instrumentation that achieves a precise overview of the temperature in a server rack" and "developing temperature aware strategies". This chapter facilitates content for providing measures and design path to solve the problem statement. The following tasks will need to be accomplished.

3.1 Objectives

The objectives of this study structures the field of interests touched in the background. The purpose is to achieve a clear path of creating a model which will serve as a source of information for system administrators to make intelligent heat management decisions.

1. Implementation of data collection
   (a) Data collection design
   (b) Data collection
   (c) Temperature collection

2. Measurement Tool
   (a) Arduino
      i. Electrical management
      ii. Placement strategy
      iii. Programming

3. Storage
   (a) Opentsdb
   (b) Scripting

4. Experiment designs
3.2 Implementation of Data collection

This project will be based on data derived from a measurement tool developed for temperature collection. Hardware and testing environment will be provided by the university college of Oslo and Akershus. As the aim of the project is to create value of the data that is collected in a server rack environment, it is of importance that the processing of data through analysis can give better understanding of the raw data than the raw data itself. Through revision of trend and relational analysis of the gathered data it is expected to yield a more detailed health status of the rack. Performance will be measured through temperature measurements over time linked up to typical production cases.

3.2.1 Data collection design

There are no standardized tools for collecting temperature data from multiple locations within a server rack. As this study focuses on analyzing the data derived from sensors in a rack, there needs to be a possibility to measure temperatures on a high frequency and retrieve them in a joint manner. The tool and environment created in this study will work as a monitoring device where system administrators continuously can look at the latest and historical heat developments. To achieve this, several components will be integrated together. An OpenTSDB database will store and dynamically plot the data that is fed from the sensors in the rack controlled by an Arduino board.
3.2.2 Data collection

Data collection is one of the most crucial factors of making this a successful study. In order to give system administrators a model and tools to mitigate heat threats, knowledge is needed. Knowledge is achieved through information, which in this case is retrieved through data collection and analysis. As this study is undertaking the environmental threat that is heat under the scope of analysis, it is natural to incorporate temperature measurements as a data metric. All data will linked to a corresponding time stamp which will give the possibility of looking at trends over time that can derive new information.

3.2.3 Temperature collection

Common practice as mentioned in the introduction is to measure the ambient temperature in the data center itself to monitor the state of the whole data center in terms of heat condition. This study will implement a more detailed approach where 20 measuring points will be set in a server rack. The standard 19-inch racks are the elements which house the electrical equipment of the data centers which are the main source of heat. This project will collect temperature readings in Celsius due to the wide usage of this metric as a temperature reading. The temperature collection design is comprised of splitting the server rack into five planes, where a sensor will be put in each corner. This will require measurement and physical setup of over 100 meters of wire. A proposed model is at figure: 3.1.
3.3 Measurement Tool

A tool and environment that can provide the temperature data wanted from a server rack is needed. To create a measurement tool that can simultaneously measure 20 different location and report these results over ethernet is not publicly available and therefore needs to be developed. The tool needs to be dynamic in terms of placement in the rack and sensor reach. Other necessary factors are stability, heat resistant and prone to electrocution.

Arduino

Arduino is a highly adaptable piece of hardware that can be used for implementing sensor technologies i.e. temperature sensor. There are several steps that needs to be addressed before an Arduino can be utilized as a stand alone measurement device that measures multiple locations at the same time.

1. Electrical management
2. Placement strategy
3. Programming

When programming the IDE on the Arduino factors like external serving of temperature readings, availability, speed and data

All of the components will be in close contact with the rack that is made of metal. Metal leads electricity. Electrical management is a crucial factor as there is no second chances if the machine were to be short circuited. Which is a likely possibility when having low risk management in terms of treating electrical leading surfaces that will be on all of the sensors and board. Electrical tape together with double sided tape will be used to mount the components and remove risk of leading electricity. As for achieving a dynamic sensor reach for the different sensor locations. Cables will need to be soldered, shielded and marked to create the sufficient unit structure. This will be planned after a placement strategy based on space in the rack.

3.4 Storage

A centralized storage unit which can continuously take in data from an external source is needed as data will be fed to the storage unit every two seconds. Storage of data is essential when doing heuristic analysis of data. A database with high throughput which allows data to be read and pushed externally is well suited for this project. That is why Opentsdb which has these traits is chosen.

3.4.1 OpenTSDB

The initial plan is to measure temperatures every second from 20 sensors. That would mean 20*60*20*24 = 172 8000 measurements every day. OpenTSDB which is known for a database that handles a large throughput will be used for storing the data. It will be set up at an external server. Another reason for choosing OpenTSDB is that it has its own built-in graph functionality. This will provide an immediate visualization of the temperature readings.

3.4.2 Script

As multiple measurements will have to be injected into the database its necessary to script the process of doing so. OpenTSDB utilizes an API for injecting multiple data entries. A python script will fetch temperature readings of a plain web site at the Arduino web server using a web scraper. The output will then be subjected to different regular expressions so that the desired data can be fetched for different combinations, in terms of relationships linked to placement in the rack. The results will have to be formatted into JSON so that it can be sent to OpenTSDB through a simple TCP connection using HTTPLIB in python. When the data is inserted into the database the study of data and testing can begin.
3.5 Visualization of data

A form of visualization of the rack is needed to give a detailed overview over the heat situation. Therefore multiple tools and methods are implemented to achieve a complete picture over multiple scenarios.

3.5.1 3D visualization

Opentsdb gives a visual time line where trends easily can be identified. It lacks the ability to give an good overview of the heat situation in the rack. Therefore a 3D representation will be implemented to visualize how heat is divided throughout the rack. There are multiple tools that offers 3D functionality. The optimal solution is to have a script which fetches live data from the sensors so that it can be treated by another R script. R has libraries which supports 3D modelling together with heat maps. This gives the potential of having a real time picture of how heat is spread out in the rack and how how certain areas are based on color codes.

3.5.2 2D visualization

As a more detailed view, 2D heat maps will be visualized through a matrix will be implemented using R. Specific data measurements will be extracted to visualize heat relationships between the specific sensors and scenarios. A small python script will format the result into a csv format so that it easily can be treated in R studio with packages like rcolorbrewer, gplots and heatmap2.

3.6 Experiments

There is a great variety of testing scenarios that would be useful to look at. Both physical and technical aspects of the test design will be explained under this topic. Creating knowledge out of the information received is a major key. Compression of the data into meaningful value which will inform administrators of the state of the server rack is desired.

3.6.1 Leap detection test

A method for detecting local events is needed to detect sudden changes in the rack which can alert the system administrator of changes that occurs in the server rack. Therefore a statistical method for detecting these sudden changes is implemented. Leap detection test is a statistical test based on the $X^2$ test for detection of leaps proposed by Cochran [45]. The formula is expressed as:

$$X^2 = \frac{(x_1 + x_2 + ... + x_i - i \times x_{i+1})^2}{i \times (i + 1) \times \bar{X}}$$

where $x_1, x_2, ..., x_i$ represents previous observed values in the time series and $X_{i+1}$ is the most recent observed value. $i$ denotes the memory of the
size of the memory. The mean on the second line includes all \( i + 1 \) values. What this test is addressing is that the hypothesis that the most recent observation is different from the ones observed in its memory. The \( X^2 \) is used to compare against a threshold value that is set in accordance with a confidence policy level and uncertainly levels. How long the memory is set defines how sensitive or insensitive the algorithm is. A shorter memory span will increase the forgetfulness of the algorithm and thus increase its sensitivity as it quickly forgets its measurements. Whilst a long memory will decrease the sensitivity as there is a higher chance that the most recent observation has been observed in the current memory buffer. A Perl script will fetch temperature measurements during experiments which will be analysed for testing purposes to explore the possibility of using Leap detection test (LDT) as a sensor system that alert when anomalies occur. With the use of LDT one hopes to find leaps at a level where the naked eye wouldn’t reveal changes.

3.6.2 Average and differential Values

The collected temperature measurements needs to be compressed into a metric of value. The first step will be to develop a script that will calculate average values of different combinations in the rack so that trends for different aspects of the rack could be discovered. Not only trends of each specific sensor. I. E The average value of all the sensors in the front of the rack or the average of all the back sensors. The differential values will also be fetched by a script so that administrators can quickly see deviating values. For example how much warmer or colder one side of the server rack is compared to another over time.

Combinations that will be fetched are:

- Per-plane average: The average value per plane
- Front average: The average value of all the front (cold aisle) values
- Rear average: The average value of all the rear (hot aisle) values
- Left / right average: Average for the left and right side
- Front / back difference: back average - front average
- Left / right difference
- Per-level front/back difference
- Top/down difference front: The difference between the lowest plane average, and the highest plane average on front side.
- Top/down difference back
3.6.3 Physical Experiments

Multiple case scenarios will be conducted to force situations where meaningful data could be derived. i.e. it will be interesting to look at long term effects of a sudden heat evolution in the rack. How much time the rack spends to recover and return to normal operation temperature after an incident. These are important factors as they tell us the actual consequences of the incidents that occur in the rack. Another interesting point is also to see how fast changes take effect after an incident has occurred.

Complete obstruction

Complete obstruction of the racks means that a kind of physical blockage will interrupt the natural flows of the air to provoke situations. A simple piece of cardboard fitted to block the access path of the air will be used to simulate an environment where complete obstruction occurs. It is not likely that that a server rack could be choked up by a complete obstruction by coincidence, but it is still of interest to see the consequences of it. When experimenting with a case such as complete obstruction there are multiple factors that needs to be addressed. When setting blockades like set in figure:5.7 means that the air is lead into other directions. This would mean that air would be pushed further into the rack towards the hot aisle disrupting the best practice of having a hot and cold aisle figure:2.1.4. That would maybe imply that there could be a change not only at the temperature sensors from bottom to top but in the relationship between front and back side.

Loose obstruction

Counter to complete, loose obstruction is a scenario that frequently occurs in an operational environment. An example of loose obstructions are cables. In setups where a load of cables are needed there is an arising issue of air blockage. Poor cable management or just the shear amount of cables can lead to decreasing performance of cooling in a server rack. Therefore an extensive amount of cables will be put up in the rack to test the effect of loose obstruction. Loose obstruction is a realistic case scenario as racks often get filled up with cables and devices that all can obstruct rack cooling.

Shield and bezel removal

At the outer frames of the rack, protective shields are mounted. These shields help tp contain the air and leads it upwards. A testing scenario will be to see the results of having these shield and not having them. The same experiment will be conducted for the bezels that are mounted directly on the servers.
3.6.4 Technical Experiment

Technical cases that can be induced to create heat reactions in the server rack are in this case limited. As the Arduino sensor system is installed in a control rack and not a compute rack where virtual servers are hosted after need. Making it easier to increase workload on demand. Within a compute rack there would have been a possibility of shifting the load to different areas of the rack to see what kind of effect the thermal load would have in some areas counter to other. i.e. shifting the heavy load to the top of the rack and see if this would disrupt the natural air flow going out the top of the rack. Had the monitoring system been installed in a compute rack, algorithms that shifted server load based on the heat measurements received could have been implemented. Which would give the possibility of a self healing rack.

CPU manipulation

A possibility is to run CPU heavy programs on specific servers that in theory will rise the heat levels at certain areas in the rack. It will be done at strategic locations to see how it affects the total heat relationship. This is a realistic scenario as servers in production environments have CPU spikes over a variable duration of time due to various reasons.

3.6.5 Expected results

The expected results of this project is that the methodology provided through the framework design will aid system administrators to be aware over the state of their server rack in such away that they are able to create heat mitigating measures. When collecting measurements some expectancies are taken in relations to the results that will be derived from the produced measurement tool. Expectancies i.e. that it will be hotter at the top of the rack as hot air is pushed upwards. That it will be hotter at the aisle where the server spew out hot air through their exhaust vents. Information and understanding will be created together with a visualization, that will enable the administrator to make hardware and load placement strategies based on case experiments that will revolve around these assumptions.

3.7 Plan overview

Figure 3.2 shows a flow chart that visualizes the project plan of this thesis. It takes the points mentioned in the approach and shows how they are linked together. The goal of setting up an approach featured in this design will achieve the goal set in the problem statement of facilitating system administrators with methodology and information to create heat mitigating placement strategies in the server rack.
3.8 Alternative Design

There are alternative approaches of setting up a model and instrumentation of gaining knowledge about thermal conditions in a server rack. A very general approach could be a survey of today’s practices and standards. There are constraints linked up to heat management and a survey of these and the common practices to mitigate these constraints does often not follow a general line. The field is researched at at various levels from fine grained to a course overview of management standards. A meta survey is something that is often used in medical research where one creates a survey of existing research and creates value out of these. As this is more of an engineering problem a meta analysis is not to favourable. Hard data is wanted to discover and confirm assumptions made on the basis of heat management theory described in the background.
Chapter 4

Implementation of measurement tool

This chapter describes the process of implementing the design set in the approach for the measurement tool. A thorough explanation of the setup of the measurement tool will include electrical management and the programming of the interface that facilitates the setup of temperature measurement over ethernet. The section also provides insight of how tools and technologies presented in the background are implemented and utilized.

4.1 Model overview

Projects addressed in the background describe various methods of mitigating thermal threats in the server rack, through the use of sensor systems, CDF models and prediction models. The projects have different concepts of purpose beyond that. Being anything from alarm systems, counteractive models and information gathering. Through a set of different technologies this study provides it own measurement device to facilitate collection of temperature data that will be subjected to analysis in order to answer the problem statement.

A Model of the components which provide continuous measurement and trend visualization is presented at figure: 4.1
4.2 Electrical Management of the Arduino

The measurement tool consists of one ethernet module, 20 DS18B20 temperature components, 20 corresponding capacitors, an external power supply, an Arduino sensor shield expansion card that allows for multiple sensors and connector wiring that connects the expansion card with the coherent sensors to the Arduino board. The following points will elaborate how these parts are interconnected to achieve the model set in Figure 4.1.

4.2.1 Connecting the components

The first step was connecting the Arduino sensor shield to the Arduino board. The purpose of this shield is to make it easier to connect cables and devices to the right pins [46]. Without the shield one would only have one power and one ground pin available for all of the 20 sensors. The shield removes the need of soldering all the ground and power supply wires to the same pin, as it provides its own connection to the ground and volt pin on the Arduino which is distributed over several pins on the shield. Using the Sensor Shield also allowed for the usage of three pin female connector cables which eliminated the job of soldering loads of cables.
Wiring

When connecting the sensors to the board, simple female three pin connector cables were used. The Arduino was mounted at the underside of the roof in the rack. The need for 20 different measurement spots over five levels meant that cables of various lengths were needed for the different sensor locations. Regular cobber cable where measured up after specific needs of lengths following the dimensions of a 19 inch server rack. Dimensions being:

- Depth: 72 CM
- Width: 55 CM
- Length: 190 CM

The length of the cable had to be multiplied by three since there is a need for individual cables for ground, power and signal. With these specifications and the Arduino mounted at the roof of the rack, a total length of cable that were needed was approximately 11280 centimeters. Resulting in over 123 meters of cable for connecting sensors from the board to their designated locations. All of the wiring had to be soldered to the 3 pin female connector cable. The area that was soldered also needed to be encased with a plastic tubing that was melted to stick to the soldered area which then provided protection against short circuiting from outside environments.
Wire management

Wire management is often not prioritized enough. The importance of good wire management can save countless hours in terms of troubleshooting when something goes wrong. In this case where specific sensors that are assigned for specific areas at the rack, it was of importance that the cables having the specific length meant for one location went to that particular location. This was to prevent having issues with cables that were too short. Another factor was eliminating the possibility of getting mixed measurements where one would believe that one was measuring from plane three but is actually measuring from plane one. This is because each sensor needs to go to its pre-programmed pin on the Arduino board. Wire management was therefore of importance. The wires which were in triples needed to be twisted to control the wiring situation within the rack. They also needed to be labeled and color coded for the different planes. Using this system one would always know which position every cable was intended for. Layout for cables 4.1. A reference system for the planes were also created 4.2.1.

- FR = Front right
- FL = Front Left
- RR = Rear right
- RL = Rear Left
- P* = Plane level
Table 4.1: Wire management

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Length (CM)</th>
<th>Color code</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1_FR</td>
<td>256</td>
<td>Dark blue</td>
</tr>
<tr>
<td>P1_FL</td>
<td>311</td>
<td>Dark blue</td>
</tr>
<tr>
<td>P1_RR</td>
<td>256</td>
<td>Dark blue</td>
</tr>
<tr>
<td>P1_RL</td>
<td>311</td>
<td>Dark blue</td>
</tr>
<tr>
<td>P2_FR</td>
<td>208</td>
<td>Light blue</td>
</tr>
<tr>
<td>P2_FL</td>
<td>263</td>
<td>Light blue</td>
</tr>
<tr>
<td>P2_RL</td>
<td>263</td>
<td>Light blue</td>
</tr>
<tr>
<td>P2_RR</td>
<td>208</td>
<td>Light blue</td>
</tr>
<tr>
<td>P3_FR</td>
<td>160</td>
<td>Yellow</td>
</tr>
<tr>
<td>P3_FL</td>
<td>215</td>
<td>Yellow</td>
</tr>
<tr>
<td>P3_RR</td>
<td>160</td>
<td>Yellow</td>
</tr>
<tr>
<td>P3_RL</td>
<td>215</td>
<td>Yellow</td>
</tr>
<tr>
<td>P4_FR</td>
<td>112</td>
<td>Orange</td>
</tr>
<tr>
<td>P4_FL</td>
<td>167</td>
<td>Orange</td>
</tr>
<tr>
<td>P4_RR</td>
<td>112</td>
<td>Orange</td>
</tr>
<tr>
<td>P4_RL</td>
<td>167</td>
<td>Orange</td>
</tr>
<tr>
<td>P5_FR</td>
<td>66</td>
<td>Red</td>
</tr>
<tr>
<td>P5_FL</td>
<td>121</td>
<td>Red</td>
</tr>
<tr>
<td>P5_RR</td>
<td>66</td>
<td>Red</td>
</tr>
<tr>
<td>P5_RL</td>
<td>121</td>
<td>Red</td>
</tr>
</tbody>
</table>

Temperature sensors DS18B20

The temperature sensors that were used in this project were fairly straightforward to implement. It has a three pin connection that could be connected to a three pin female cable that was wired and soldered together with extension cable. The only thing that was done here was to implement and solder capacitors on each sensor. Capacitors were introduced as they reduce the potential noise from other sources. A capacitor is like a battery. Storing energy although they work differently. The sensors are connected to digital pins 16-35 illustrated in figure:4.3.

Figure 4.3: sensor Connection
ENC28J60

The ethernet module which is based on the ENC28J60 chipset requires a steady 5 volt (V) current to run smoothly. It operates around 3 V, but at times of having a high activity load it can peak at 5V. Drawing current directly from a USB powered Arduino board which provides 5 V was not optimal as the Arduino has multiple other components connected to it. It would result in an unstable power supply to the ethernet module. The ethernet module was soldered to a breadboard which had a MB102 Breadboard Module connected to it to support for external power given by a regular power adaptor. Using this method, stable power supply was ensured. There were some implications regarding random shutdowns due to bugs in the library used for the module. This was solved by connecting the ENC28J60’s reset pin to the Arduino mega board. A work-around a health check was implemented by pinging the gateway of the Arduino. If a ping back was not received within 10 seconds a reset signal was sent which restarted the ENC28J60 module.

4.3 Programming of the IDE

The programming of the Arduino IDE is done in its own programming language which is a set of the C/C++ programming language[28]. There are different libraries developed for temperature sensors for Arduino but this thesis is utilizing the most common temperature sensor DS18B20 which is dependant on the Dallas temperature library. Dallas utilizes the OneWire library for reaching out to its temperature sensors. Other libraries of relevance are the Ethercard library which is the driver for running the ENC28J60 chipset.

4.3.1 Dallas temperature

Dallas temperature is the library that is utilized for collecting temperature sensors from the DS18B20 temperature sensor. The library is made with focus on intuitive use and low complexity. The fundamental commands that were needed to initiate temperature readings are listed below.

```c
//Start up the library on all defined pins
DeviceAddress deviceAddress;
for (byte i = 0; i < oneWirePinsCount; i++) {
  // initialize one wire object to the specific sensor pin [i]
  temp_sensor_oneWire[i].setPin(oneWirePins[i]);

  //initialize the dallas dallas temperature object.
  //Tells the dallas object where the onewire object is located.
  temp_sensor[i].setOneWire(&temp_sensor_oneWire[i]);

  // initialized the process.
  // Tells the dallas object where the onewire object is located.
  temp_sensor[i].begin();
}
```

---

40
An array of all the sensors with their corresponding pin number on the Arduino shield as been mapped out in an array [oneWirePins]. The device Address as stated in the referenced code above starts up the library on the pin the code is at in the present loop. It continues to initialize the OneWire object which is the sensor located at a specific pin on the Arduino. Pins used are mapped at figure 4.3. The code continues to initialize the Dallas library on the one wire pin that has been established in the segment before. After assuring that the pin is set and that the Dallas protocol is initialized the process of collecting the temperature data is activated. A specific resolution for temperature accuracy is then set. See 2.1 for effects of the different bit sizes. The last but an important command is the setWaitForConversion(false) command which makes the conversion process asynchronous. Which allows the code to run efficiently and not wait for all the sensors to complete its conversion before reading the result from the scratchpad. If this command was not implemented, a situation resembling a blocking/unblocking state would appear.

### 4.3.2 OneWire and sensor resolution

The OneWire library is essential as it allows the Arduino(master) to communicate with multiple slaves (sensors) over one data line. The process of reading the temperature sensors goes over two steps. Conversion and reading the scratchpad which is an internal register for temporary storage. When a command to read the temperature sensors is called, an internal conversion operation is started. When the conversion process is done it writes the results to the scratchpad where the result can be read. This is where the resolution comes in. The higher resolution chosen for temperature readings the longer it takes for the conversion process to take place. This is why this project has a resolution of 10 bits which makes it possible to poll the data every second. Conversion using 9 bit takes 94 ms while 12 bits uses 750 ms. Naturally one would think that having 20 sensors will make the Arduino utilize more than one second to poll the temperatures. One wire is able to start the conversion process of multiple sensors and come back and read the scratchpad when its ready instead of waiting for the operation to complete.

### Connection tracker

A problem encountered was that modern web browser has a functionality of firing multiple syn packets when requesting content from a web server. This is a result of inpatient browsers that will send a new syn packet toward the web server until it receives an acknowledgement. The problem that occurred was that for every request that was sent to the web server

```c
// Sets temperature resolution
temp_sensor[i].setResolution(deviceAddress, TEMP_RESOLUTION_BITS);
// makes the conversion process asynchronous
temp_sensor[i].setWaitForConversion(false);
```
prompted a new request for temperature collection at the Arduino. This caused unnecessary workload. The problem was solved by implementing a function that would only serve a request of a TCP sequence number it has not seen before. This fix manages to exclude the other requests as the syn stream of packets sent to the web server will have the same sequence number until it gets its response.

### 4.3.3 Web server

The Arduino has its own web server programmed on the IDE which is serving for the purpose of data accessibility at an external source. It serves a simple web page which contains the current temperature status within the rack and is continuously refreshed every time the site is polled. This provides the opportunity of polling the data every second. This is due to internal operations of fetching the temperatures takes just under one second.

### IP configuration for adaptability

A bit of code that was added for adaptability. A function to alter the IP configuration remotely. Information regarding IP configuration is stored in EEPROM memory which called for the need to include the EEPROM library that allows for altering of this memory area on the board. This allows a system administrator to remotely put the Arduino on different networks. Making it more robust, hence to security.

### 4.4 OpenTSDB

The OpenTSDB server is running in a virtual environment at the university college in Oslo and Akershus’s Alto cloud. Where its dedicated to running the database service alone. The setup of this time series database was handled through a simple installation script made by Kyrre Begnum [47]. The git repository contains a simple bash script which installs the components needed for running a simple Opentsdb database as well as basic configuration of elements like DNS naming, creating a table for Hbase and starting the OpenTSDB server. The components that are installed are Hbase, zookeeper, TSDB and Java. Java is needed for the plotting functionality of the data. The server itself is the main storage area of all the data measured over time.

OpenTSDB provides several API’s for querying, plotting and pushing data to the web server through HTTP. This project utilizes an HTTP PUT API for pushing data. Other than following the standard setup of OpenTSDB some modifications had to be done to adapt the server to receive temperature data from Arduino. By default Opentsdb does not support chunked requests. Meaning that when the amount of data being sent to server is at a size where it does not fit into a single tcp packet, the whole transmission is
rejected. Allowing for chunked requests are enabled by adding following line in the configuration file.

tsd.http.request.enable_chunked

Another factor that needs to be addressed is frequent querying of data. Every time a visualization of the data is queried, it is stored in a temporary file which has a max capacity of 500 megabytes. When this folder is full, the plotting functionality will seize to function. Therefore a CRON job deleting the contents of this file has to be implemented to ensure a fail-safe plotting service. Opentsdb’s plotting functionality which can be specified through its own GUI is exemplified through figure: 4.4

![Figure 4.4: Plot of temperatures in Opentsdb](image)

### 4.5 Rack Installation

The whole board had to be taken apart in terms of wiring before mounting it on the rack as preparations had to be done. As the rack is made of metal which conducts electricity, all pins had to be covered up on the Arduino to not short-circuit the board and temperature sensors if was to accidentally come into contact with the metal side. Therefore multiple layers of tape had to be put on all exposed pins on the Arduino and ethernet module. With the measurement device being installed at the roof of the rack, double-sided tape was used to glue the device to the top of the rack. By using this position, gave the possibility of drawing the cables from the top, a method that puts less strain on the cable attachments attached to the rack. A real image of the measurement tool as it sits now is at figure: 4.5
4.6 Parsing data for Opentsdb

Visible in figure: 4.1 a script at an external client is continuously polling data from the Arduino mounted in the rack. The data is presented at a simple web site running on the web server created on the Arduino. The script which is written in python, facilitates the fetching of temperature measurements, making calculations needed based on the experimental setup and parsing of data to a json formatted setup so it could be send to Opentsdb using its HTTP PUT API.

A basic pseudo model for the script design figure: 4.6
It utilizes HTTPLIB for accessing the web site and RE for implementing filters with regular expressions. As described in the approach, there are multiple variations of data that is of interest. These are generated from the script by filtering the temperature readings with regular expression and applying basic arithmetic functions for extracting values such as average and differential values. For every match these operations are done before they are appended to the results list. A typical case would look like:

```
for key, value in mydict.iteritems():
    #print key, value
```

Figure 4.6: Script Flow
stringvar = "%s,%s " %(key,value)
g = re.findall('(P5_F[A-Z]{1}),(.*\s', stringvar)
if (g):
    P5FSUM.append(float(value))
g1 = re.findall('(P5_R[A-Z]{1}),(.*\s', stringvar)
if (g1):
    P5RSUM.append(float(value))
sensor_description = 'Plane5_DIFF'
value = abs(P5diff)
data = {
    'metric': 'rack.differantials.temp',
    'timestamp': ts,
    'value': value,
    'tags': {}
}
data['tags']['sensor_description'] = sensor_description
opentsdb_data.append(data)

As the results are appended into the results list they are shipped together
to OpenTSDB through an URLlib connection. The outcome of the process
is then logged as successful, error with the corresponding error type or a
notification that the site is inaccessible. Regardless of the outcome the script
will repeat the process at the speed set in the script. Measurements are
polled every to seconds to ensure success of polling temperature from the
Arduino. Polling of data could be issued all the way down to one second.

4.6.1 Creating Heatmaps with R

When the point of data storage was reached, a better way to visualize
the thermal relationship of the measurements needed to be implemented.
The heatmaps two dimensional maps was implemented using R with
libraries such as Gplot and rcolorbrewer, whilst 3d maps was created with
Matplotlib.

Two dimensional heatmap

The script basically takes in a file that has comma separated values and cre-
ates a matrix of specific set of that matrix.

> data <- read.csv("data_c", comment.char="#")
# assign labels in column 1 to "rnames"
> rnames <- data[,1]
# transform column 2-5 into a matrix
> mat_data <- data.matrix(data[,2:ncol(data)])
# assign row names
> rownames(mat_data) <- rnames
After the data is read into R’s buffer its ready to be treated. To create a heatmap a color palette was implemented with a color range of blue to red. Where green represents cold and an increasing grade of red shows how hot the measurement is. This is where Rcolorbrewer comes in.

```r
> my_palette<-colorRampPalette(c("green","yellow","red"))(n=149)
> col_breaks = c(seq(0,15,length=50), # for green
+                seq(16,25,length=50), # for yellow
+                seq(26,100,length=50))  # for red
```

This bit of code sets a color grade going from green to red. Where the range of temperatures from 0-15 is shown with different shades of green corresponding to lower or higher in the range of 0-15. From 16-25 shades of yellow is used, whilst from 26-50 red is implemented to show the hottest zones. This type of heat map excels at showing an enhanced picture of changes to specific data sets over time.

When the color palette is set the last remaining function is to plot the picture using a standard png function that sets the pixels, labels and type of diagram. Example result can be viewed at figure: 5.6.

**Three dimensional heat map**

3D heat-maps was supposed to be implemented using R. After facing several difficulties using this approach an alternative Python approach was used. Through Matplotlib and Numpy, a 3D visualisation came to life. Where coordinates of the specific sensors are plotted with a color corresponding to its temperature value. Showing how hot each measurement location is through the use of colors.

To plot the sensor on the graph, the specific sensors had to be plotted with coordinates. Three dimensions of X,Y and Z was used. Where X and Y show width and depth. Z was used to set plane level. After the sensors where positioned on the map the values had to be linked to the specific sensor placement. A fourth dimension was added, containing the temperature values used as a basis for the color map. Giving a coordinate table of figure:4.2.
Table 4.2: 3D Coordinates

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>18.25</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>17.75</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>18.00</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>17.50</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>19.25</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>18.25</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>20.50</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>19.00</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>20.75</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>19.75</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>23.50</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>20.00</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>3</td>
<td>30.25</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>22.00</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>3</td>
<td>28.75</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>21.75</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>4</td>
<td>30.50</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
<td>26.00</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>4</td>
<td>28.75</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>26.50</td>
</tr>
</tbody>
</table>

After setting the coordinates, it could be fed to Matplotlib for heat map modeling.

The script takes the csv formatted file as input and plots the four dimensions. The color map is then based on the temperature values as mentioned above. Then its just a matter of plotting the map using a scatter plot function together with a color map function.

```python
with open('3d_data.csv', 'rb') as csvfile:
    reader = csv.reader(csvfile)
    for row in reader:
        xs.append(int(row[0]))
        ys.append(int(row[1]))
        zs.append(int(row[2]))
        the_fourth_dimension.append(float(row[3]))
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
    n = 100
    colors = the_fourth_dimension/max(the_fourth_dimension)
    colors = [the_fourth_dimension*2**n for n in range(len(xs))]
colmap = cm.ScalarMappable(cmap=cm.hsv)
colmap.set_array(the_fourth_dimension)
yg = ax.scatter(xs, ys, zs, c=colors, marker='o')
```
cb = fig.colorbar(colmap)
ax.set_xlabel('Cords')
ax.set_ylabel('Cords')
ax.set_zlabel('Planes')
plt.savefig("heatmap19")

This results into a model which gives a overview of the temperature situation in rack in a more organized way so its easier to locate hot zones. Example figure: 4.7

![3D heatmap](image)

**Figure 4.7: 3D heatmap**

Having all the functionality in place to properly store, analyse and represent data. Tweaks and manipulation of the environment could then be initialized to experiment with different situations while monitoring the effects of these. Typical scenarios that is likely to occur and not so likely is both of interest to discover information of how heat reacts to different types of change.
Chapter 5

Implementation of experiments

This chapter will provide a description of the process of making physical and technical changes to a rack environment to facilitate the experiments referenced in the approach. Four physical and two technical experiments will be briefly explained and presented in terms of how they were implemented in practice, as well as presentation of results. A common principal of the physical experiments revolves around tampering with the cold aisle to manipulate behaviour to see the effects of disrupting normal air trajectory, whilst technical experiments consists of work load tampering. Figure 5.1 shows the different planes that are experienced with in this chapter.

Figure 5.1: Planes
5.1 Extracting value of the differential values

By looking at the differential metrics, one can create value for a system administrator. The relationships of these values linked up to heat management practice will inform a system administrator of the cooling performance in the server rack. Relationships that will be extracted are:

- **Total difference between front and back.**
  The total difference between these two metrics will yield information regarding cooling optimization in the server rack. As server racks and data centers utilize the principles of hot and cold aisles. Meaning that difference between temperatures in the front and back will indicate how well the hot and cold aisle is kept. In terms of cooling, the larger difference is positive whilst the lesser is negative.

- **Difference between front and back, per level.**
  The same principles as the previous point applies here. A per level view gives a more detailed view. A per plane view could for example identify how behaviour for specific servers relate to the ambient heat situation on the plane. Deviating differences could tell the administrators that there could be something of physical nature interfering with the natural airflow.

- **Total difference between top and bottom.**
  In the case of differentials between top and bottom, the case is different from front and back. Regarding that environment it is wanted that the temperature difference is as small as possible. Cold air flows from the CRAC system into the elevated floor and through the perforated tiling into the server rack. From the bottom of the server rack the cold air is pushed upwards in an attempt to push the hot air out of the top of the rack. Of physical nature, hot air is lighter than cold air. Therefore an external force is needed to push the cold air upwards. The effect of this is that as cold air flows up it will deteriorate as the hot air takes it effect on it. Meaning that the difference between the top and bottom sensors will indicate how well the cooling system performs, where a smaller difference is positive.

- **Difference between front and back per level.**
  A more detailed view over the same principles as the total difference.

5.2 Setup of Leap detection test

To utilize the leap detection test one used a perl script which calculates if a leap has occurred in a given data set based on the algorithm provided in 3.6.1. The script performs the analysis with a different set of memory lengths to find the best suitable memory length. Where one computes the analysis trying a range of 5 to 25 data entries and sequentially increase the sensitivity of the algorithm to find the best match. After the memory length which dictates the sensitivity is set, the script continues to classify and store
the potential leaps.

The LDT script is ran on a set of experiment data files for following combinations

- P1 Average
- P2 Average
- P3 Average
- P4 Average
- P5 Average
- Difference between left and right side
- Difference between front and back side
- Difference between Front side, Top and bottom level
- Difference between Back side, top and bottom level

A collection of files which contain various combinations of data. Running the LDT test with an iterating memory increase, creates over 203 leap detection files for each experiment. Analyzing each of the data files with a different memory span. Using this method and sorting based on the level of how many leaps detected one is able to optimize the memory sensitivity of the algorithm. The script itself is referenced in the appendix.

To look at the results of 200 Leap detection results using eyesight could be somewhat inaccurate. Therefore two bash commands were developed for properly organize the output of the analysis.

```bash
### All code presented is supposed to be in one line.####
for mem in $(seq 5 25); do for fil in $(find Experiment_data/complete_obstruction_secondgo -iname "*leapdetection.txt"); do echo "$mem fil: $fil"; cat $fil | ./ldta.pl -v -D -a -m $mem -e 150 -l 300 > ${fil}.$mem.ldt; done; done
```

The bash command locates all leap detection files, and runs them from a specific line point which is correlated with timezones in the experiment which is of interest for analysis. It then prints an output of how many leaps that were detected in data set.
Get statistics over false and true positives:

```bash
### All code presented is suposed to be in one line.###
for fil in $(grep -rl "DETECTED" Experiment_data/complete_obstruction_secondgo/*);
    do
falsecount=$(grep "FALSE LEAP DETECTED" $fil | wc -l);
fullcount=$(grep DETECTED $fil | wc -l);
truecount=$(grep "TRUE LEAP DETECTED" $fil | wc -l);
dif=$(echo "$truecount - $falsecount" | bc); echo "$dif,$truecount,$falsecount,$fullcount,$fil";
done | sort -n
```

The second bash command presents statistics over how many false positives and true positives. The classification is based on if the leap was detected in the start-up phase of the experiment, cool-down or during the experiment phase. Where a true positive occurs only during the experiment phase. This is linked up to how experiments are set up which will be explained later.

### 5.3 Basic sensor measurements result

The basic value of looking at raw data is not to be overlooked as itself gives value. Temperature data over a N period of time is valuable to see if there is correlation between peak hours in terms of use and temperature effects. Measurements made every 2 seconds have resulted in over 3220900 temperature readings from the different sensors. A full week of measurements with no down time in the measurement tool would result in over 6048000 temperature readings.

At figure 5.2 its visible how the sensors are color coded and named for future reference.

![Figure 5.2: Naming](image)
5.3.1 Basic sensor measurement data

Looking at figure 5.3 a data set with a duration of five days is presented. With an X axis that has time stamp every 24 hours. Looking at the data a widespread of 20°C temperature at the bottom of the rack measuring all the way up to 35°C. That would mean that there is a 15°C differential average from bottom of the rack to the top side.

It should be noted that the top sensor marked with the color "yellow" corresponding the Plane 5 front right is invalid result. After measuring for a while it was noticed that this sensor had fallen down from its original placement causing it to be located in front of a exhaust vent. Making the result of this value invalid. The hottest area in the rack is as expected at plane 5 front left side which has the adjacent rack next to it. Marked with the color "red" and not not P5_FR = plane 5. Front right like the figure indicates.

5.3.2 Usage trend: day versus night

There is a fluctuation of somewhere between 1-2 Celsius degrees in the active time zones. It's apparent that that its hotter during day time when students are utilizing the servers versus night time. One could assume that the user base is sleeping as the main reason for that. This is expected but assuring to see as this is an indicator of when the cooling system should be at lower working capacity, which is during nighttime to conserve energy. This seems to be the case for all areas of the rack which is interesting as in later cases it will be visible that lower planes are not effected by forced case scenarios.

Figure 5.3: All sensors trend
5.3.3  No usage

How much workload the hardware it exposed to over time is an essential factor for how much thermal energy is produced within the rack. During the testing phase, there was an outage rendering the services provided by rack inaccessible. Causing a ripple effect of no usage from the users over a longer period. This allowed for the project to measure the effects of usage versus no usage. Meaning that internal operations running on the hardware itself to keep the system sustained was the only operations generating thermal activity.

Naturally this lead to lower temperatures in the rack. The thermal relations and conditions were not the same. Looking at figure:5.5 compared to figure:5.3 there is a clear difference in terms of temperature and trends. During regular usage there is a wavy trend showing that there is a difference in temperature during nighttime and daytime which has a correlation to amount of usage. Looking at the data set figure: 5.5 which has no usage the highest measured sensors are yielding temperatures at 31 Celsius at most, which is a 4 Celsius downfall from figure: 5.3. The lowest measuring sensors are showing the same result. Where 20 is the lowest measured result at figure:5.3, one is receiving measurements all the way down at 16-17 Celsius at figure:5.5

Looking at figure:5.6 a heat map shows the thermal relationship from an excerpt of four measurements where Measurement one (M1) is from a data set where the rack is in use

5.4  Complete obstruction

The purpose of complete obstruction was to obstruct the cold aisle’s air that flows from the bottom side of the rack going upwards to see what effects this would have in a rack environment. The case was implemented by cutting out cardboard that was used to block the air flow at three separate
levels. See figure: 5.7. As the scenario was implemented in a production environment it was not possible to run the experiment over a long period of time in risk of disrupting or taking down the functionality of the servers.
hosted in the rack. The complete obstruction case was deployed over a time period over 20 minutes. Containing a five minute start up period, ten minutes of measurement and five minutes to cool-down. The obstruction was done at level two, three and four as these were sought out to be the levels that was most interesting to block as level 1 is at the very bottom floor level and level 5 is at the ceiling level. To implement a blockage at these levels would have little effect at the ceiling level whilst blocking the floor level would completely choke the air supply and would ruin the air supply totally thus ruining the testing environment.

Figure 5.7: Complete Obstruction
5.4.1 Complete obstruction results

By adding elements that obstruct the natural airflow, one expects to see a different heat behavior as the air is led into another “unwanted” direction. Looking at the general sensor measurements, it is possible to see that the obstruction has made behavioural changes to the heat relationship in the rack figure: 5.8.

![Figure 5.8: Complete obstruction results](image)

The start-up of the project started at 14.20 and at 14.25 the rack door was opened for inserting the obstruction. The obstruction was in place at 14.27 and the measurement lasted until 14.37 following a cool-down period of 5 minutes. Looking at the chart figure: 5.8, it is visible that there is a slight temperature decrease in the middle level planes, whilst the plane higher planes remains unchanged. The general trend of the lower planes also shows a trend of temperature decrease whilst measuring.

Looking at the relationships most of the changes occur at the the middle planes figure: 5.9. Plane 2, 3, 4 all have visible changes. They fluctuate between 1-1.5 degree difference from their front side and back side during the start up phase while reaching as much as a 3 degree difference during the experiment.
5.5 Loose obstruction

Loose obstruction was implemented through pushing a mesh of cables into the rack to exemplify a rack where cables obstruct the air flow. This was a case that is realistic in a hosting environment where the cable management is poorly implemented without taking heat management into consideration. The case was implemented to reveal if there are any effects of poor cable management. When setting up the loose obstruction experiment the rack cabinet had to be opened for quite a while to insert the "poor" cable management. This subjected the rack to a free air flow from the surrounding environment while the environment was being set up. A start up phase of the experiment is present at all experiments but not in the "loose obstruction" experiment. Because of the time it takes to set up the test.
5.5.1 Loose obstruction results

The effects of loose obstruction were not substantial but changes were present. The Average values per plane increased by 0.5 degrees per plane. Which means that poor cable management led to an overall increase of rack temperature by 2.5 degrees.
With the experiment starting at 15:48 and ending at 15:59 in the time series at figure:5.11, it shows effects that are not visibly substantial. There are little variations in terms of relational change in the different planes. The general temperature per plane is stable, but an interesting discovery is that some of the planes have a decreasing difference in temperatures between the intake and exhaust side figure: 5.12. Especially plane 3 receives a substantial drop. From a 3 degree difference to 1.5.

![Figure 5.12: Loose obstruction differentials](image.png)

### 5.6 Shield removal experiment

By implementing an experiment which consist of removing the plastic shields used in server rack to close the U frame slots that are not in use, one expects that air will be led in to the rack, disrupting the hot and cold aisles. The shields help air flow upwards and not in to the rack. The shield removal experiment emulates a scenario of poor setup. Where best practice follows having no excessive room between the units as it would lead to a disruption of air flow going from bottom to top of the rack. Then again, one could argue that when most of the cold air is left at the bottom, it could be favourable for the whole rack if a principal of putting the most heat contributing servers at the bottom of the rack.
5.6.1 Shield removal results

The experiment had a five minute start up period, similar to the start up phase in the complete obstruction experiment. Where normal activity is measured for five minutes, following an insertion of the case experiment. The duration was 10 minutes, followed by a five minute cool-down period. Looking at the ten minute testing window, there is no significant difference in temperatures by removing the shields. Neither the differential relationship took mentionable effect as one would have thought.

- Start-up 14:50
- Experiment 14:56-15:06
- cool-down 15:06-15:11
5.7 Bezel removal

All servers mounted in the server rack has a protective bezels that are in place to hinder accidental use, i.e. touching the power switch or hard-drives. The last physical experiment consists of measuring normal behaviour over a five minute interval, then open the rack and quickly remove the bezels to measure if these have direct effect on heat accumulation in the rack.
5.7.1 Bezel results

The results shows little effect from removing the bezels. The only fluctuation that deviates from normal behaviour is at 15:16 and 15:26, meaning the times when the rack was opened to remove and insert the bezels. During operation the bezels has seemingly little effect. The case was identical for the differential relationship between front and back side on the different planes.

The timers for the different phases was:

- start-up 15:11
- Experiment 15:16-15:26
- cool-down 15:26-15:31

5.8 Technical experiment

The technical experiment conducted had an advantage that the physical experiments does not have. Being that it could be issued remotely and without having to open the server rack door. Opening the door results in disturbing the air flow of the rack, and letting ambient air outside the rack affect the temperature within the enclosure. The technical experiments revolved around manipulation of workload on the machines hosted in the rack to see the effects of a high work load on server at specific locations.
5.8.1 Experiment specifications

The experiment consisted of three sub-experiments. Where different planes were manipulated to analyze the different effects of heaviest loaded server placement. being:

- Plane One Heavy
- Plane three Heavy
- Plane five Heavy

The background of using these specific planes is to see how hardware placement affects the ambient temperature in the rack. Following the general principal of convection 2.2.2 that heat rises while cold air weighs down, sinking downwards. One would assume that server rack hardware placement design would benefit from a bottom-up build. This experiment was then implemented to confirm or disprove that a bottom up-build is an optimal design.

5.8.2 CPU Experiment results

The experiments used CPU power as a method of pushing server capacity to an extent where heat accumulation increased. The operation that was used to increase the CPU load was a method of issuing multiple mathematical operations which pushed the CPU cores to 100 %. A bash command initiated the experiment with:

```bash
for tall in $(seq 10); do echo $tall; echo "1234567^1234567" | time bc & done
```
The experiment ran over three sub experiments where each one lasted 20 minutes each, including a five minute start-up phase and a five minute cool-down phase to measure the accumulation and recovery phase of the experiment.

5.8.3 Bottom Heavy results

When boosting the CPU load on there is not much change in temperature. There are small spikes on the middle planes at the start of the experiment as well as some temporary temperature rises in the cool-down period of the experiment. The bottom servers which are closest to the air intake seems unaffected by load manipulation in terms of temperature.

Figure 5.18: Bottom heavy cpu
Differentials

Boosting the workload of the lowest server resulted in a more stable differential heat relationship between the hot aisle at the front and cold aisle in the back side. Looking at figure 5.19, there are more fluctuations in the differential results for most of the sensors, especially at plane 2 in the start-up and cool-down period compared to the actual experiment window. There are spikes early in the experiment window but they seem to stabilize further out in the test before spiking again in the cool-down period when the cpu utilization goes back to a normal state.

5.8.4 Top heavy cpu

The effects of a top heavy loaded server did not make a great impact when experimenting with CPU manipulation. One was able to increase the internal server heat from approximately 53°C to 64°C without having a noticeable effect to the ambient temperature in the 10 minute experiment period.
Top Heavy Differentials

When having the heaviest loaded server located at the top, one sees the opposite effect of having the heaviest loaded server at the bottom. Looking at the experiment window lasted from 12:58 to 13:58, it’s visible that the heat relationships between the front side and the back side are increasingly becoming unstable by having the heaviest loaded server at the top. It should be noted that the fluctuations are at a detailed level. There are no significant changes to the heat relationships in the vertical planes.
Chapter 6

Analysis

This chapter describes how data acquired from the developed measurement model will be processed in order to provide knowledge and understanding of the thermal situation in a server rack derived through analytical methods. Methods and case experiments conducted in this thesis with their corresponding results will be analyzed under this chapter. Through analysis of experiments, a goal of achieving direct value in terms of knowledge which could assist system administrators mitigate heat uprising and identifying flaws and bottlenecks in the rack setup.

6.1 Statistical methods

The study is measuring temperature data which is of low variance. The gathered data has predictable measurements. When variances occur they are within reason. There are little to no implications that can cause systematic errors which would pollute the results. Therefore statistical analysis like standard deviation, P-value are not implemented to confirm the measurements validity in terms of statistical significance. It gives little value to the paper.

6.2 Experiment analysis

When taking a step back and looking at the different results yielded from the different case experiments it was visible that some of the experiments had more effect than others. One wished that the span of impact would have been larger as the effects from the shield and bezel removal were close to non existent while the obstruction cases had some effect of value. Even though they did not have as much effect as one would hope it does not mean that the findings are of no value. As shielding and bezels are implemented for physical protection, it is a reassuring factor that these common elements does not disturb the heat balance in an enclosed rack.
6.2.1 Areas of effect

When looking at all the different results from the case experiments it’s visible that the lower sensors planes take little to no effect from the different use cases. The changes take more effect at the mid to high level planes. It seems that the heating process the air goes through when flowing from bottom to top is increasingly taking effect when subjected to the case experiments.

In situations where one would only implement four sensors it would be natural that one would want to measure the effects that the rack is exposed to at the mid to higher level planes as the air has more time to be subjected to the experiment.

6.3 Obstruction cases

There were some interesting findings when experimenting with obstruction of natural airflow. The two different obstruction cases had different reactions which is natural.

6.3.1 Complete obstruction

Looking at figure:5.8 it’s visible that the middle planes had their temperatures slightly lowered which is not coherent with the hypothesis that was set before the project. The expected result of this experiment was that one would see an increased cooling effect of the lower planes, especially P1 and P2. At the same time seeing an uprising in temperature at the higher planes as air was not guided to the top levels. The blockage had different results, unexpectedly. The Obstruction seemed to have a more cooling impact at the middle planes planes. A natural explanation of that was that the blockage stopped the air from flowing upwards thereby cooling the middle planes more extensively. The lowest P1 plane took little effect. The differences between normal operation and the experiment phase is visualized at figure:6.1.

![Figure 6.1: Normal versus complete obstruction](image)

(a) Start up phase  (b) Experiment phase
The fluctuations in the results figure:5.8 at 14:27 and 14.37 occurs due to the rack door being opened and closed for inserting and removal of the obstruction. It's interesting to see how differently the rack recovers from the rack door being opened. With the obstruction in place it has a more controlled decrease in temperatures in the middle planes while when removing the obstruction and closing the rack door, huge temporary decreases are measured. As a result of regaining free air passage.

Still the experiment yielded information. Regardless of obstruction or not, it's naturally colder at the bottom as the air intake is placed there. In situations where one is carefully placing their most heat generating server at the bottom unit frames, using obstructions will seemingly aid the cooling of the lower racks.

LDT

When the LDT testing of the complete obstruction commenced, there were results showing that having a long memory slot of 20-25 caused results that triple the amount of false positives over true positives. While the middle planes had more promising results. A test using a data set containing plane 2 average values. It gave as much as 70 true positives in terms of detected leaps in the data set and only 9 false positives. The threshold for defining what a true positive and a false positive is explained under 5.2

Figure:6.2 shows an excerpt from the result of a leap detection experiment using a memory window of 24 data points.
Figure 6.2: P2_leapdetections

Figure 6.2 shows the best result. Where a high memory slot was promising when measuring differences between top and bot side. All over, the results shows a trend of the model showing more false positives over true positives. The red markers show triggers of the LDT in the start-up phase of the experiment while the yellow mark the true positives. As the chart visualizes there is a clear overweight in false positives. figure: 6.3
6.3.2 Loose obstruction

The loose experiment is a great example of how only looking at the 20 different sensors is unfavorable and how it does not necessarily yield enough information of how heat is situated in a server rack. Looking at figure 5.11 there are little fluctuations in temperature during the experiment which could indicate little change. However, if one were to look at figure 5.12 there is approximately a 1.5 degree drop in differential values from the front and back side of the middle planes. This shows a reduced functionality of the heat principles of maintaining a hot and cold aisle where a bigger difference would mean the better cooling efficiency.

Having these kind of drops or anomalies in the data stream is what could trigger the LDT algorithm which then again could alert a system administrator of temperature changes in the rack. Which possibly powers the model to function as an alarm system for system administrators.

One could argue that as the general temperature remains the same it does not have any actual effect how the heat is dispersed through the rack, but following the principles of air conversion its is known that mixing cold and hot air currents will disrupt the controlled air flow one is trying to achieve. Which is pushing warm air upwards and out of the rack.
LDT Loose obstruction

Due to the circumstances of setting up the test environment there where no start up period or cool-down period to classify between a false positive or a true poistive of the lead detection. Only the actual experiment data was tested. Figure 6.4 shows an LDT analysis that was utziling a 6 data point memory window which yield over 58 counts of leaps in the data set. The problem was that this was a very detailed level. One can see that the level of leaps are at at fluctuations between 0.25 and 0 degrees. A temperatue meausrment that is so detailed that it would not have significant effect in the rack. Triggering leaps based on such small temperature differences also indicates that the sensitivity is to high.

![Figure 6.4: LDT difference between front and back](image)

Figure 6.4: LDT difference between front and back
6.3.3 Shield and bezel removal

Shield and bezels are two components that are often used in server racks for physical protection purposes, as well as air flow guidance. It was therefore of interest to see what effects these components has in a rack. The results showed little effect of shield and bezel removal experiments.

Shield removal

When experimenting with removal of shields there was an hypothesis that air would flow between the servers as the shields was not there to guide it upwards. Looking at the different metrics, that was not the case. The effects of removing the shields were minimal when looking at results at figure 5.14. A more telling figure would be to look at the differential chart. Looking at both the differential values and the chart for average values its clear that the shield removal did not have the expected effect. The effects of the experiment were close to non existent ref figure:5.15.

The fact that the effects were non-existent does give a system administrator information of value. That is that the usage of these shields are not essential for heat management using the basis of this experiment as a conclusion on that term.

Bezel removal

The removal of the protective bezels mounted on the faces of the server did hardly have any effect on any of the measured metrics. As this data were consistent over the three different stages of the experiment analysis of the effects of the bezel removal was not prioritized. This provided more time to study the experiments which yielded more promising results in terms of information value.

6.3.4 Comparing the cases

As mentioned earlier there were not substantial effects deriving from the physical experiments. Neither were there any huge differences between them looking at sensor measurements alone. Some differences were discovered when monitoring the trends of the differential values over time which is a important discovery. Looking at figure 6.5 where a data-set from the experiment is fetched from every experiment and visualised and together for comparison, one can see the actual differences.

<table>
<thead>
<tr>
<th>Complete obstruction</th>
<th>M1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loose obstruction</td>
<td>M2</td>
</tr>
<tr>
<td>Shield removal</td>
<td>M3</td>
</tr>
<tr>
<td>Bezel removal</td>
<td>M4</td>
</tr>
</tbody>
</table>
The ambient rack temperature around plane 1 seems to be close to unaffected besides a fluctuation from half degree more or less. The same goes for the highest planes. Where the temperatures fluctuate with 1 degree at most. In the middle planes there are some changes as mentioned under the different case analysis. This result is confirmed through a comparison of the cases. The effects of the cases are mostly taking effect in the middle planes. Where there is up to 1,5 degree difference from the complete obstruction case versus shield and bezel removal. This strengthens the theory that by obstruction the air flow will have a consequence of stopping the air of flowign to the top, thus making it cooler at the middle planes. [2]

6.4 CPU Experiments

When experimenting with CPU manipulation of the servers within the rack, one expected that the increased temperature of the physical server to directly influence the ambient temperature within the rack. Looking at the results from the tests ran on a top heavy loaded rack and a Bottom heavy loaded rack, its visible that the experiments did only effect the ambient temperature to a fine grained extent.

As best practice suggests that a server rack is built bottom-up ref:5.8 where one puts the server which has the heaviest expected load at the bottom. Meaning that the hottest server will not have the chance to accumulate heat as it is very close to the source of the cold air plenum. Therefore it was expected that there was no significant heat increase at that experiment. At
the top heavy experiment, the hypothesis was that the highest plane would receive an temperature increase as the air supply is not as effective at the top as it is at the bottom. Looking at the results yielded from figure 5.20, it shows that this was not the case. The ambient temperature in the rack had close to no visual effect.

As the CPU usage of the servers only where boosted for a 10 minute interval one could suspect that it was not sufficient time for the heat to build up. Then again, if that where the case one should have seen some tendencies of heat accumulation at the later stages of the testing period, which was not visible either. A healthy sign is that the rack is not sensitive to huge but short boosts of server utilization, which is a very common occurency. Such boosts in CPU usage can be linked to short time operations, cronjobs and scripts that run on various servers in the rack.

![Figure 6.6: Top heavy load versus bottom heavy load](image)

Figure 6.6 compares the temperatures in both experiments over the sensors location after 4200 measurements has been collected for each experiment. Which corresponds to 7 minutes in to the experiment which lasts of a total of 20 minutes. The figure better compares the results presented under the results chapter. It confirms, what one could see out of the results. That the temperature differences are not significantly different. There is a slight difference at Plane 5 when having the heavy load at the top server.

After interpreting and analyzing the results in terms of what they mean for system administrators, its important to consolidate this information into something of direct value for a system administrator. An approach that does not require major background knowledge about heat management but still gives insight of rack health and performance.
6.5 InteliRack, proposing an health metric. Using the golf rating approach

The goal for the project was to facilitate a method and instrumentation for a system administrator so that administrators could make temperature aware decisions concerning the server rack. For both hardware placement and cooling strategies. Through this mindset, a suggestion of creating a value consolidated of information derived from measurements and best practice surfaced. A principle which is inspired from the health sector. A health metric showing the heat status of a server rack. InteliRack uses a golf like rating would give users a sense of performance on a scale. Golf uses a system where the lower score one gets, the better. Where having a score of 0 or negative value is classified as a great results where as normal values above is worse.

An easy approach would be to look at all the sensors and the measurements they are recording. Then consolidate these into a average value. This could then be linked to standards of recommended ambient temperatures in a server rack. A problem is that there is no clear consensus of what the recommended temperature should be. The recommended value according to Openxtra being 20-21°C [48] while Google recommends 26°C [49]. Finding a temperature value one is comfortable with, and using it as a reference value is a prerequisite for the approach to function. By using this recommended value as a pin point for a score in terms of how far or close the rack is to recommended temperature. The server rack would then have a better deemed health measured in how much deviation one has fluctuated from the recommended value. An excerpt from a measurement shows that the average ambient temperature in the server rack utilized in this project was about 24°C. Stating that the recommended temperature is 20, would give a score of +4. Where as the closer to zero or minus value, the better.

6.5.1 Cooling effectiveness

A weakness using this methodology is that it does not take the cooling efficiency into consideration. This study has chosen to look at differential values as a metric of how effective the cooling is. Where as two parameters are used. Difference between bottom and top as well as front and back. Looking at temperatures at the bottom (intake) and top (exhaust) is an example of how one could look at cooling effectiveness. Where difference in ambient temperature at the top side compared to the bot side can inform how well air is being carried from the lower level to the top. Meaning that less difference is good in terms of cold temperature. Whilst one would want a bigger difference when looking at the difference between front and back as there is a designated hot and cold aisle. A greater difference will then indicate that the aisles are separated which is in line with heat management.
practice. By including these metrics one can identify indicators for cooling performance.

6.5.2 Consolidation of cooling efficiency and temperature

Consolidation of the gathered data is tricky as there are different factors involved. One has to take account for potential situations which could pollute the result. An explanation of the variables in the formula is:

- Google recommended Temperature (GRT)
- Ambient Rack Temperature (ART)
- Top plane temperature (TBT)
- Bottom plane Temperature (BPT)
- Front rack temperature (FRT)
- Back rack temperature (BRT)

A proposed approach to calculating the health metric is listed below:

\[
\text{proposed calculation} = (\text{ART} - \text{GRT}) + (\text{TPT} - \text{BPT}) - (\text{BRT} - \text{FRT})
\]

example:

- Recommended ambient temperature = 26
- Average ambient temp in the rack = 28
- Difference between bot and top = 10
- Difference between front and back = 3

Using the formula presented above the translated calculated would give:

\[
(28 - 26) + (20 - 10) - (6 - 3) = 9
\]

A health score of 9 where the closer to 0 one gets is better. Say that the system administrator was unhappy with the result and decides to increase the cooling at the cold aisle, thus lowering the difference between the top and bot side. Reducing the difference between top and bot by 50%. Giving a new score of 4, which is substantially better. Making the new equation:

\[
(28 - 26) + (15 - 10) - (6 - 3) = 4
\]
6.5.3 Challenges of the health metric approach

Using this approach there are some factors that needs to be looked at. For example utilizing the assumption that the lower differential value between the bottom and top plane is good does not work if the ambient temperature has risen beyond accepted levels due to various reasons like an outage in the cooling system. If the lower plane has the same high temperature as the top plane then the difference will not be in a positive favour. An example could be where both the lowest and highest place has temperatures over 30 Celsius. This could be solved through usage of if statements that sets threshold values. This should also be indicated through the first part of the formula, as such a case would impact the general ambient temperature which is also a part of the formula.

There is also the factor of prioritizing the different metrics so that the algorithm can be optimized. One usually operates with a larger differential value going from the bottom plane to the top versus the difference between front and back side. One could then consider to weight one of the values more or less to create a more balanced equation between the values.

There are many situations to account for when creating a algorithm that properly addresses all variances in types of occurrences that can occur in the rack. For example where the top server is solely creating a substantial amount of heat on its own compared to the rest of the servers. That would directly affect the ambient temperature at the top which is a metric used in the algorithm. Affecting the score so that one could receive a score that is not entirely representable. The algorithm needs to be adapted to all of these situations through further experimentation with trial and error before one finds the best solution.
Chapter 7

Discussion

This study chose its own approach in a field of study where there is not much common consensus of guidelines mitigate heat threats to the server rack. A critical retrospective look of how the experiments were executed and how information was parsed hence to answering the problem statement. A discussion of experiences made from creating the measurement device, approach and implementation will appear in this chapter.

7.1 Design

As mentioned earlier there are several complex approaches to attempt to mitigate heat threats in both data centers and server racks. Some methods inspired from the aviation industry use cdf models and complex algorithmic prediction models for anticipating how air flows within confined spaces.

7.1.1 Background for design

Through the design of this project, one tried to attack the problem statement using an easy to understand approach which could easily be replicated and understood by the common system administrator. The design of measuring 20 different areas within the rack created multiple opportunities to look at relationships in the rack at a fine grained level. Where most projects measure the general ambient temperatures, this project had a precise overview of where the bottlenecks where accumulating and the general effects of these. With the design one had a goal of creating something of value for system administrators. Being a model, product or information which would facilitate for temperature aware decision making. Through the data collection design used it was up to its own imagination of how one could analyze data to create a result of value. This is the approach that was wanted when starting this study as there are limited research and industry standards of heat management in server racks.

The design of collecting data over five planes with four sensors on each
The design as it sits now could have been implemented in multiple racks to see the performance of a group. A simplified version containing sensors at the bottom and top could be facilitated to a group of racks to measure the health metric of a group of server racks and not only one rack. Creating a consolidated health metric over a group of racks. Visualized in 7.1. It would have been of interest and value of the project to see how different racks react to similar kinds of experiments conducted for a singular server rack.

Figure 7.1: Data center Health metric

Experiments which had a realistic and unrealistic chance of occurring where implemented in the design to manipulate the rack behaviour so one could see how heat reacted to changes down to the plane level. Not just the general ambient temperature. Challenges were faced when setting the design for data collection and experiments. Both expected and unexpected.

7.1.2 Testing duration

The test cases were conducted in a production environment at the university college of Oslo and Akershus. The server rack was a so-called multifunctional rack containing everything from storage servers, networking nodes, domain controllers and so on. Due to a restriction policy kept by the college, there was limited physical access to the server room. Frequent ac-
cess to the room was not an option, unfortunately dividing the experiments into multiple segments with longer duration was not an option. Having access to a server room with an elevated floor for cooling with corresponding heat and cool aisles are not commonly accessible as they are used only for larger server rooms. Availability forced the project to go through all of the cases at one day, where each experiment had a 30 minute given duration. Naturally one would like to have to test on a larger data set to be more confident of the results yielded.

Due to time constraints, testing had a cool down period of 10 minutes between the case tests. A longer cool-down period would be optimal as it would be interesting to see the effects each test had over a longer duration of time and how much time the rack spent reaching normal operating temperatures. Also in cases where there were a significant temperature change, it would be interesting to see if the hardware took direct effect in terms of no service or shutdown as an effects of the test at a longer perspective as there is a possibility for it.

7.2 Implementation

Measuring temperatures it self is not something new. Creating own tools to measure temperatures has many different approaches as well. To measure temperature at such a detailed level as this project has done with limited capacity have had a steep learning curve.

Conducting experiments which tries to manipulate heat which is affected by multiple factors such as server usage, air flow, adjacent racks and ambient temperature in the data center creates difficulties to replicate an identical situation for each experiment so that one could have a precise comparison of the experiments.

7.2.1 Implementation of the measurement tool

Using Arduino has an advantage of being openly available for all. Even though the component and technology is openly accessible does not mean that the competence is. The usage of it requires electrical and software programming knowledge which is harder to come by. One estimates that the replication process of replicating the tool created in this study is estimated to take around 1-2 weeks. Meaning that if one were to have the financial capacity of having a team develop this solution in their own data center, the design set in this study will give means of value for system administrators.

Physical challenges of implementing the Arduino

The project faced difficulties when implementing the measurement tool in the server rack. Another approach of how to mount the Arduino and
the sensors should in retrospect been implemented in another way. The usage of double sided tape seemed like a decent method, but the due to the heat situation in the rack the glue deteriorated to a level where the sensors sometimes fell down from their position. This caused wrong measurements in some cases as it was hard to discover without having continuous physical access to the rack. One could suspect that something had happened due to irregular values plotted in OpenTSDB, but in a dynamic heat environment, a change in temperature could have several reasons.

A proposed solution would be to mount the Arduino on top of the rack instead of inside it. Then one could string the sensors through venting holes at the top of the rack. This would eliminate the risk of the Arduino falling down from the roof and potentially ruining the whole device. As for the sensor which had a rigid surface, taping the sensors became troublesome as it was hard to make them stick to the flat surface of the rack. This was later on solved using string wire to keep them in place. A more optimal solution that should have been implemented from the start.

7.3 Experiments

Multiple experiments were conducted, which created experiences. The implementation of them was deemed to be successful in the essence of being able to conduct all of the physical experiments one wanted to.

7.3.1 Data validity

The results that were obtained are reliable as the sensors never measured deviating values that could indicate kinds of glitching in the hardware or software on the temperature sensors. Instability in the measurements were a problem as the ethernet module had stability issues. These shutdowns required the module to reboot which had a flaky success rate, creating some instability in measurement frequency. This was expected when polling 20 different sensor temperatures every two seconds. Technically it was supposed to be feasible to poll data at such a high frequency but it seemed to strain the ethernet module, which became somewhat of a bottleneck.

7.3.2 Impact of experiments

Looking at the effects of the experiments retrospectively one was surprised over the lack of effect the rack took from being subjected to the experiments. the results still yielded information of value. Meaning that if the cooling is at level where one is of liberty to have an extensive cooling energy output. Then design, air streams and hardware placement plays a less dominant role. Though it would be energy wasteful. The equation of calculating the cost of pushing extensive amount of air to the rack over a long period versus spending money on setting up a good infrastructure that takes heat management into consideration would most likely pay off over time, but
that equation is not a topic under this thesis. As it would take advanced statistics and creation of prediction models to do so.

**Increased cooling**

When the infrastructure housing the OpenTSDB server crashed which led to a major set back in terms of data loss, the project took a major set back. Explained under 7.4. As the environment had changed when the experiments was re-ran. The cooling had been significantly boosted. Pre-crash, the highest planes were measuring 35°C, whilst the lowest one had ambient temperatures of 20°C. Making it a 15 degree difference. With these high temperatures, the rack was more susceptible to the experiments compared to the reality in today’s situation. The reason for that was that the administrators were not comfortable with the temperature situation in the rack, thus substantially increasing the power of the CRAC system and cooling of the rack. Naturally this led to lower ambient temperatures and lesser effects of experiments. One gets to a certain point of when enough cold air is pushed into the rack, the smaller effect the rack is going to take from manipulation.

7.3.3 **Compute rack possibilities**

All of the experiments had to be conducted in the multi-usage case as the Arduino was mounted in this rack. The multi-usage rack was utilized as it was more convenient hence to physical space, making it easier to mount the tool. There was more space in that rack in contrary to the adjacent rack which was available. That rack being a compute rack it contained more cables that were tough to wiggle around. This eliminated the possibility of dynamically distributing workload in the rack based on heat. This would have been of significant interest to see how the rack could “self regenerate” based on logic derived from heat management practices and real time data analysis. A case could have been where a sudden heat increase in the top level compute rack would lead to virtual machines or workload being migrated to server located further down the rack where it is supposedly cooler. This would in theory lower the general temperature of the rack as the lower planes are more susceptible to handling more load as they are closer to the air intake. Making them able to operate at higher workloads whilst keeping a lower temperature. There are projects that have implemented a preemptive model based on computational fluid dynamics where analytics on gas behaviour is made and conclusions of future behaviour are made and bases on that preemptive load balancing is performed [39]. Implementing this project into a compute rack would allow to experiment with a reactive model that was based on real time data.

7.3.4 **Alternative experiment approach**

Instead of conducting many experiments on one type of rack one would test multiple racks. To see if there are different reactions using a narrowed
scope of experiments on different racks that are differently situated in terms of cooling and heat management. Using this approach would yield results of how a poorly managed rack reacts to typical heat manipulation scenarios compared to a proper setup.

7.4 Loss of data

At a late stage in the project period the virtual environment hosting the OpenTSDB database with the collected temperature data went down due to technical difficulties. Causing a massive loss of collected temperature data. At this stage, valued data that was showing promise had been collected. Case experiments which had yielded interesting results in terms of analysis was also lost. Some of the data was salvaged, but most dispersed. Causing a major set-back for the project. After the database environment was restored, there had been made crucial change to the environmental factors of the server room and rack.

7.5 LDT Analysis

The LDT analysis was conducted on two experiments to monitor its effectiveness and to unveil if it had any potential in utilization as a sensor system for heat management. Multiple data files containing information regarding different relational data was gathered under each experiment. The LDT tests were ran on each of them to see the effects and performance on each of them. The different data sets reacted differently to the variety of memory length or "sensitivity" of the algorithm. There were no clear line of a general best performing sensitivity setting.

When reading the results of the different LDT tests there were some positive results containing a moderate amount of true positives, but at levels of extreme detail. Where registered leaps was in temperature fluctuations at the decimal level, which is too detailed. In most cases the results yielded a much higher false positive leap detection rate. Making the algorithm not viable to use as a sensor system.

Before the algorithm can be used it needs to be further developed to exclude the huge overweight of false positives. The question is if they can be excluded or another approach has to be tested. Standing on its own LDT would have trouble working as a sensor system, especially with the amount of false positive and the frequency of alarms. A solution could be using a frequency filter and optimizing the memory settings.

Most of the false positives occur when the sever rack door is opened, which creates a influx of air. This influx creates huge leaps in the data set. One would need to have some kind of a sensor system which is aware of when the door is opened and not.
7.6 Future work

The work that has been done in this study has areas which could be further developed, both in the technical aspect as well as physical. Additional functionality to further utilize the information derived from this study is also of interest.

7.7 Humidity measuring

Humidity measurement is often measured in data centers as it has direct influence on the electric environment. Where a range between 40 - 60 percent relative humidity (rh) is recommend [50]. Where a too low rh would cause static electricity on the systems which could cause an electrical outage. A too high rh would cause corrosion on the hardware. Therefore keeping an acceptable humidity level is important. Humidity sensors are at the same price range as temperature sensors. One could argue that humidity sensors could have been implemented in retrospect to give a even more detailed overview over the most common physical threats to a server rack.

7.8 Effects of an adjacent racks

Through detailed measurements it has come fourth that adjacent racks has an impact on surrounding racks in terms of heat. When looking at the results presented under the analysis section 6 there is a significant heat impact on the temperatures on the left side of the rack. The left side sensors all have an adjacent rack producing heat next to them. The right side is placed next to an aisle which is neither a hot or cold aisle. It is just used for maneuvering between the rack aisles. The operating temperatures between the sides are substantially cooler on the side that has no adjacent rack. It would have been of interest to see the effect of having an adjacent rack on both sides to see the general temperature effect in the whole rack. A testing
case would be to lower or increase the temperatures in the adjacent racks to see the impact it would have on the measured rack.

7.8.1 Dynamic heat load balancer

As mentioned under section 7.3.3 through an implementation in a compute rack a, possibility of creating a dynamic load balancer using heat as a vector for decision making came to life. As the project has provided a model measuring heat temperatures from 20 different locations over five planes, there is possibilities that can be derived from such a detailed overview of the heat balance in the server rack. This enables the possibility of creating a tool and algorithm that shifts work load in a compute rack based on heat. There are several models which shift work load based on different prediction-utilization algorithm as well as reactive algorithms based on overall system usage. Using typical metrics like CPU and memory usage. All these methods are in place to hinder performance loss and potential overheating. By creating a reactive model based on heat, one would automatically mitigate and cope with heat threats without user interaction.

7.8.2 Monitoring tool

Throughout the project different types of visualisations have been implemented. A time series plot from OpenTSDB data, a two dimensional heat map as well as a three dimensional plot. As the two and three dimensional plots are implemented with R and python it is possible to use iterative methods to make the visualisations refresh themselves continuously. Connecting the plot to a web server would make image accessible over Internet. This would enable the monitoring tool to work as a monitoring device like Nagios or Sensu.

The initial thought was to link this with the LDT algorithm to implement alarm functionality. With the results derived from the LDT tests that is not an option with the current status. Alternative approaches needs to be investigated. There already multiple approaches being tried out, like CDF and machine learning techniques.

7.8.3 InteliRack for groups

Figure 7.1 visualises that the InteliRack approach has potential of being utilized for groups or zones of server rack. Having a pre-programmed Arduino with the coherent sensors, fixed cable lengths for measurement positions, functional database and pin placement labels one would reckon that that it will take approximately a full day to install the measurement tool. Utilizing InteliRack as an approach for measuring rack health for a group of racks, one would have to simplify the tool. An approach of doing so would be to remove the sensors of the middle plane which would eradicate the setup of 12 sensors. The Algorithm would also need to modified to take factors as where the server rack is placed. If its placed in
a hot or cold aisle and how much energy that is spend cooling the specific zone. When measuring zones multiple factors come into play which the calculation of the metric needs to take into consideration. Through these modifications InteliRack could be adapted to work for group of server racks, which gives the model a new dimension.

7.9 Impact

The project was able to utilize its own instrumentation and modelling to create information that system administrators directly could benefit from. Through results gathered one was able to look at ways to implement this information into a product which could aid system administrators in keeping an overview over cooling performance and health status in the rack. A solutions based on best practices in the production environment linked up to own experiments, results and discoveries.

The project’s design and approach has created a model for measuring health rack. Possibly groups of racks as well. If the model was to be further developed and optimized it could have a significant impact on the product side of server farm monitoring. Through experiments conducted the project has concluded that the experiments used in this study had little effect which informs successors experimenting the same topic could inspect other kinds of experiments for manipulation of air trajectories within server racks.

The results of this thesis does not revolutionize the way of managing heat in server racks but has both confirmed theories and created a suggested approach which could be viewed as a stepping stone to facilitate system administrators with the tools for better heat management.

7.9.1 Project experiences

The project is deemed to be successful in the perspective that all of the planned data collection and experiments where implemented. One would wish that the experiments had a larger influence on the rack behavior as one had expected on beforehand. Still, one was able to create value of the experiments made combined with results of these through analysis and modeling.

Creating good presentations of the thermal environment during experiments and comparisons of these environments was a significant challenge. Through a significant amount of trial and error the study was able to create different kinds of presentations being time line series, 2D and 3D visualizations which became sufficient to present the data gathered.

The loss of data was a major set back for the study. Loss of valued data with testing conducted in an environment which provided less cool air to
the rack. An environment which was more susceptible to the conducted experiments. One has answered the problem statement of creating a model and instrumentation that aids development of temperature aware strategies. There is aspects and elements which can be improved. Given more time, one would have liked to further experiments with elements under the future works section.
Chapter 8

Conclusion

The goal of this project was to create a method and instrumentation that facilitated temperature aware decisions in terms of heat mitigation in server racks.

The problem statement was addressed by creating an own measurement tool which was used to conduct various physical and technical experiments. With both realistic and synthetic scenarios, while monitoring the derived effects.

Results show that common server utilization has more effect on ambient temperature compared to physical obstructions and workload manipulation in the server rack. One has also learned that the rack is resilient to change in shorter time periods. Where boosts in utilization does not affect ambient temperature in the rack.

The study investigated the possibility of using a Leap Detection Test as an approach to create an alarm system, which keeps track and informs administrators of changes that occurs in the rack. The algorithm showed promise, but was deemed to be unfit on its own for environment where change accumulates over a larger time space.

Through analysis and modeling linked to best practice of heat mitigation, the study created its own suggested approach for measuring rack health. "InteliRack" measures rack health based on ambient temperature and thermal relationships in to symbolize rack health through one single metric.

The study has brought its own suggestion of creating a tool for system administrators based on heat mitigating principles like hot and cold aisles, rack setup and air convection. The study has reached its goal of providing measures to inform system administrators of rack health and facilitate instrumentation that mitigates heat in server racks.
Bibliography


Chapter 9

Appendix

9.1 Arduino, Mastertemp

Arduino code format:

```c
#include "OneWire.h"
#include "DallasTemperature.h"
#include "EtherCard.h"
#include "net.h"
#include "EEPROM.h"
#include "NetEEPROM.h"
#define DEBUG 1
#define ETH_SPI_CHIP_SELECT_PIN 53
#define HOSTNAME_MAX_SIZE 50
#define TEMP_RESOLUTION_BITS 10  // 9, 10, 11 or 12 bits resolution with
                                // 93.75ms, 187.5ms, 375ms and 750ms temperature
                                // reading time respectively.
#define TEMP_P1_FL_PIN 16  // Temperature sensor Plane 1 (P1), Front-Left (FL)
#define TEMP_P1_FR_PIN 17  // Temperature sensor Plane 1 (P1), Front-Right (FR)
#define TEMP_P1_RL_PIN 18  // Temperature sensor Plane 1 (P1), Rear-Left (RL)
#define TEMP_P1_RR_PIN 19  // Temperature sensor Plane 1 (P1), Rear-Right (RR)
#define TEMP_P2_FL_PIN 20
#define TEMP_P2_FR_PIN 21
#define TEMP_P2_RL_PIN 22
#define TEMP_P2_RR_PIN 23
#define TEMP_P3_FL_PIN 24
#define TEMP_P3_FR_PIN 25
#define TEMP_P3_RL_PIN 26
#define TEMP_P3_RR_PIN 27
#define TEMP_P4_FL_PIN 28
#define TEMP_P4_FR_PIN 29
#define TEMP_P4_RL_PIN 30
#define TEMP_P4_RR_PIN 31
#define TEMP_P5_FL_PIN 32
#define TEMP_P5_FR_PIN 33
#define TEMP_P5_RL_PIN 34
#define TEMP_P5_RR_PIN 35
#define TEMPSENSORNAME_STR_LENGTH 6

byte oneWirePins[] = {
  TEMP_P1_FL_PIN, TEMP_P1_FR_PIN, TEMP_P1_RL_PIN, TEMP_P1_RR_PIN,
  TEMP_P2_FL_PIN, TEMP_P2_FR_PIN, TEMP_P2_RL_PIN, TEMP_P2_RR_PIN,
  TOTAL_TEMPS
};
```
TEMP_P3_FL_PIN, TEMP_P3_FR_PIN, TEMP_P3_RL_PIN, TEMP_P3_RR_PIN,
TEMP_P4_FL_PIN, TEMP_P4_FR_PIN, TEMP_P4_RL_PIN, TEMP_P4_RR_PIN,
TEMP_P5_FL_PIN, TEMP_P5_FR_PIN, TEMP_P5_RL_PIN, TEMP_P5_RR_PIN
};

String tempSensorName[] = {
  "P1_FL", "P1_FR", "P1_RL", "P1_RR",
  "P2_FL", "P2_FR", "P2_RL", "P2_RR",
  "P3_FL", "P3_FR", "P3_RL", "P3_RR",
  "P5_FL", "P5_FR", "P5_RL", "P5_RR"
};

const byte oneWirePinsCount = sizeof(oneWirePins) / sizeof(byte);
float *temperature = (float*)malloc(sizeof(float) * oneWirePinsCount);
unsigned int PREV_TCP_SEQ_NUM = 0;

OneWire temp_sensor_oneWire[oneWirePinsCount];
DallasTemperature temp_sensor[oneWirePinsCount];

// Array to store ethernet interface ip address
// Will be read by NetEEPROM
static byte myip[4];
// Array to store gateway ip address
// Will be read by NetEEPROM
static byte gwip[4];
// Array to store DNS IP Address
// Will be read by NetEEPROM
static byte dnsip[4];
// Array to store netmask
// Will be read by NetEEPROM
static byte netmask[4];
// Array to store ethernet interface mac address
// Will be read by NetEEPROM
static byte mymac[6];

// Used as cursor while filling the buffer
static BufferFiller bfill;

// TCP/IP send and receive buffer
byte Ethernet::buffer[2000];

void software_Reset() // Restarts program from beginning
{
  asm volatile(" jmp 0");
}

//get the sequence number of packets
unsigned int get_seq(byte *ethBuf) {
  unsigned int seq = 0;
  seq = (unsigned int)ethBuf[TCP_SEQ_H_P] << 24 |  
       (unsigned int)ethBuf[TCP_SEQ_H_P+1] << 16 |  
       (unsigned int)ethBuf[TCP_SEQ_H_P+2] << 8 |  
       (unsigned int)ethBuf[TCP_SEQ_H_P+3];
  return seq;
}

// A function to print the MAC address
const void print_macAddress() {
  for (byte i = 0; i < 6; ++i) {
    Serial.print(mymac[i], HEX);
    if (i < 5)
      Serial.print(':');
  }
A subnet mask must be composed of a sequence of one's (1) starting from the MSB, followed by a sequence of zeros (0). It can be something like 255.255.255.0, but not 255.254.255.0. In the latter case, 254 equals to the binary number 1111 1110 which is followed by 1111 1111 (255).

```cpp
bool subnet_mask_valid(byte subnet_mask[]) {
    byte i, j, test_mask;
    bool found_zero = false;

    for (i = 0; i < 4; i++)
        { 
            test_mask = 0x80; //0b10000000

            for (j = 0; j < 8; j++)
                { 
                    //Serial.print("test_mask = ");
                    //Serial.println(test_mask, HEX);
                    /* If the bit is zero */
                    if ((subnet_mask[i] & test_mask) == 0)
                        { 
                            //Serial.print("0");
                            /* If a zero hasn’t been found yet */
                            if (!found_zero)
                                /* Once we found a 0, the found_zero flag is set to true. From now on, the of the bits should be zero. */
                                found_zero = true;
                            else
                                { 
                                    //Serial.print("1");
                                    /* if we run in this "else" clause it means that the current bit is "1" So if a zero has already been found, return false (not a valid subnet mask) */
                                    if (found_zero)
                                        return false;
                                }
                                test_mask = (test_mask >> 1);
                            }
                        }
                    else
                        { 
                            //Serial.print("0");
                            /* If a zero hasn’t been found yet */
                            if (!found_zero)
                                /* Once we found a 0, the found_zero flag is set to true. From now on, the of the bits should be zero. */
                                found_zero = true;
                            else
                                { 
                                    //Serial.print("1");
                                    /* if we run in this "else" clause it means that the current bit is "1" So if a zero has already been found, return false (not a valid subnet mask) */
                                    if (found_zero)
                                        return false;
                                    test_mask = (test_mask >> 1);
                                    } 
                                }
                        }
            //Serial.println(" ");
        //Serial.println(" ");
    /* If the execution reached the end, it is a valid subnet mask*/
    return true;
}
```

```cpp
data setup() { 
#if DEBUG
    Serial.begin(115200);
    Serial.println("Dallas Temperature IC Control Library Demo");
    Serial.println("============ Ready with ");
    Serial.println(oneWirePinsCount);
    Serial.println(" Sensors ===============");
#endif
```
// Start up the library on all defined pins
DeviceAddress deviceAddress;
for (byte i = 0; i < oneWirePinsCount; i++) {
    temp_sensor_oneWire[i].setPin(oneWirePins[i]);
    temp_sensor[i].setOneWire(&temp_sensor_oneWire[i]);
    temp_sensor[i].begin();
    #if DEBUG
    if (!temp_sensor[i].getAddress(deviceAddress, 0))
        Serial.println("Unable to find address for Device 0");
    #endif
    temp_sensor[i].setResolution(deviceAddress, TEMP_RESOLUTION_BITS);
    temp_sensor[i].setWaitForConversion(false);
    #if DEBUG
    Serial.print("Device Resolution on Pin ");
    Serial.print(oneWirePins[i]);
    Serial.print(" ");
    Serial.print(temp_sensor[i].getResolution(deviceAddress), DEC);
    Serial.println();
    #endif
}

// To reset the ip configuration, write the value 0 to NET_EEPROM_OFFSET.
// 0 is the offset
//EEPROM.write(NET_EEPROM_OFFSET, 0);
NetEeprom.init(mymac);
Serial.print("MAC: ");
print_macAddress();
Serial.println();

while (ether.begin(sizeof Ethernet::buffer, mymac, ETH_SPI_CHIP_SELECT_PIN) == 0)
{
    Serial.println("Failed to access Ethernet controller");
    delay(5000);
}

if (NetEeprom.isDhcp())
{
    Serial.println("Try DHCP");
    if (!ether.dhcpSetup())
        Serial.println("DHCP failed");
}
else
{
    NetEeprom.readIp(myip);
    NetEeprom.readGateway(gwip);
    NetEeprom.readDns(dnsip);
    NetEeprom.readSubnet(netmask);
    Serial.println("Static IP");
    ether.staticSetup(myip, gwip, dnsip, netmask);
}
ether.printIp("My IP: ", ether.myip);
ether.printIp("Netmask: ", ether.netmask);
ether.printIp("GW IP: ", ether.gwip);
ether.printIp("DNS IP: ", ether.dnsip);

// while (ether.clientWaitingGw())
// ether.packetLoop(ether.packetReceive());
// Serial.println("Gateway found");
// Serial.println("\n");

// function to print the temperature for a device
void printTemperature(float temperature, String sensor_name, byte pinConnectedTo)
{
    #if DEBUG
    Serial.print("Temperature for the sensor ");
    Serial.print(sensor_name);
    Serial.print(" (Pin ");
    Serial.print(pinConnectedTo);
    Serial.print(" is ");
    Serial.println(temperature);
    #endif
}

const char http_OK_200 PROGMEM =
"HTTP/1.0 200 OK\r\n"
"Content-Type: text/html\r\n"
"Pragma: no-cache\r\n"
;

const char http_unauthorized_401 PROGMEM =
"HTTP/1.0 401 Unauthorized\r\n"
"Content-Type: text/html\r\n"
;

const char http_not_found_404 PROGMEM =
"HTTP/1.0 404 Not Found\r\n"
"Content-Type: text/html\r\n"
"Pragma: no-cache\r\n"
;

const char webpage_unauthorized PROGMEM =
"<!DOCTYPE HTML>\r\n"
"<h1>401 Unauthorized</h1>"
;

const char webpage_not_found PROGMEM =
"<!DOCTYPE HTML>\r\n"
"<html><head>\r\n"
"<title>404 Not Found</title>\r\n"
"<style type="text/css">\r\n"
"a {\r\n"
"color: #003399;\r\n"
"background-color: transparent;\r\n"
"font-weight: normal;\r\n"
"text-decoration: none;\r\n"
"}\\r\\n"
"</style>\r\n"
"<body>\r\n"
"<p>The requested URL /$S was not found on this server.</p>\r\n"
"<hr>\r\n"
"<address>Vaguino Web Server at $D.$D.$D.$D</address>\r\n"
"</body></html>"
```html
"<script src="http://code.jquery.com/jquery-latest.min.js"></script>"
"<script type="text/javascript">
"$$\(\ function\(\)\{\"
"enable_cb();\}
"$$\(\"#g\).click(enable_cb);
"$$\(\"input.g\\).prop(\"disabled\", $$\(\"g\\).prop(\'checked\'));\}
"$$\(\"input.g\\).prop(\"disabled\", this.checked);
"}\)
"</script>"
"<form method="post">
"<table>
"<tr><td colspan="2">
"<a href="http://$S">Home</a></td></tr>
"<tr><td colspan="2" align="right"><input type="checkbox
"name="dhcp" value="1" id="g" value=""></td></tr>
"<tr><td align="right">IP Address:</td><td><input type="text
"name="ip" class="g" value="$D.$D.$D.$D"></td></tr>
"<tr><td align="right">Subnet mask:
"</td><td><input type="text
"name="subnet" class="g" value="$D.$D.$D.$D"></td></tr>
"<tr><td align="right">Gateway:
"</td><td><input type="text
"name="gw" class="g" value="$D.$D.$D.$D"></td></tr>
"<tr><td align="right">DNS Server:
"</td><td><input type="text
"name="dns" class="g" value="$D.$D.$D.$D"></td></tr>
"<tr><td colspan="2" align="right">
"<input type="submit
"value="Submit and save"></td></tr>
"</table>
"</form>
"</body></html>"
const char webpage_please_connect_manually[] PROGMEM =
  "Please connect manually to the newly configured IP address."
;
void get_hostname_from_http_request(char data[], char hostname[], int hostname_size)
{
  int i;
  /* Whenever a new line is found
     end_of_line becomes true;
     The hostname should start right after
     a \r\n in the beginning of a new line.
     */
  bool end_of_line = false;

  for (i = 0; i < strlen(data); i++)
  {
    /* If we are in the middle of a line,
       keep on searching until a \r\n is found.
    */
    if (!end_of_line)
    {
      if (data[i] == '\r')
        {
          if (data[++i] == '\n')
            {
              /* if found, set end_of_line = true*/
              end_of_line = true;
            }
        }
    }
    else
    {
      /* The code runs in this else case whenever a new line is starting */
      end_of_line = false;
      /* If the line starts with 'Host: ', the it should be our line! */
      if (data[i] == 'H' && data[i + 1] == 'o' && data[i + 2] == 's' &&

data[i + 3] == 't' && data[i + 4] == ':' && data[i + 5] == ' ')
{
    i += 6;
    int j;
    for (j = 0; j < hostname_size; j++)
    {
        /* So read the hostname and store it in the hostname char array
         * until the next line comes,
         * /
        if (data[i] == '\r' && data[i + 1] == '\n')
            hostname[j] = '\0';
        return;
    }

    hostname[j] = data[i+1];
    }
}

void loop() {
    // Copy received packets to data buffer Ethernet::buffer
    // and return the uint16_t Size of
    received data (which is needed by ether.packetLoop).
    word len = ether.packetReceive();
    // Parse received data and return the uint16_t Offset of TCP payload data
    // in data buffer Ethernet::buffer, or zero if packet processed
    word pos = ether.packetLoop(len);
    if (pos) {
        // Store the received request data in the *data pointer
        unsigned int CURRENT_TCP_SEQ_NUM = get_seq(Ethernet::buffer);
// Store the IP of the connect client in the *clientIP pointer
byte *clientIP = (byte *) Ethernet::buffer + IP_SRC_P;

        // bfill stores a Pointer to the start of TCP payload.
        bfill = ether.tcpOffset();
        if (PREV_TCP_SEQ_NUM != CURRENT_TCP_SEQ_NUM) {
            PREV_TCP_SEQ_NUM = CURRENT_TCP_SEQ_NUM;
            /* Use the 'hostname_client_connected' array to store the hostname that the
             * client is using
             * and use this hostname for the links.
             * Initially I was using the IP of the ENC28J60 module,
             * but I found out that if I connect
             * from the Internet using a dyndns,
             * this doesn't work because the
             * IP is usually in a private
             * 192.168.x.x range, which is not publicly routable.
             * /
            char hostname_client_connected[HOSTNAME_MAX_SIZE];
            memset(hostname_client_connected, '\0', sizeof(char) * HOSTNAME_MAX_SIZE);
            ether.printIp("Got connection from ", clientIP);
            //Serial.println(data);
            /* Temporary string to store the values read from the http request */
            char str_temp[20];
if (strncmp("GET /", data, 5) == 0) {
    get_hostname_from_http_request(data, hostname_client_connected, HOSTNAME_MAX_SIZE);
    //Serial.println(hostname_client_connected);
    data += 5;
    if (data[0] == ' '){
        bfill.emit_p(http_OK_200);
        bfill.emit_p(webpage_main, hostname_client_connected, hostname_client_connected);
    } else if (strncmp("temp ", data, 5) == 0) {
        Serial.println("Requesting Temperatures...");
        //for (int i = 0; i < 1; i++)
        for (int i = 0; i < oneWirePinsCount; i++)
            temp_sensor[i].requestTemperatures();
        switch (TEMP_RESOLUTION_BITS){
            case 9:
                delay(94);
                break;
            case 10:
                delay(188);
                break;
            case 11:
                delay(375);
                break;
            default:
                Serial.println("Unknown resolution...");
                case 12:
                delay(750);
                break;
        }
        //Serial.println("Requesting Temperatures Done...");
        bfill.emit_p(http_OK_200);
        for (int i = 0; i < oneWirePinsCount; i++) {
            //for (int i = 0; i < 1; i++) {
            temperature[i] = temp_sensor[i].getTempCByIndex(0);
            tempSensorName[i].toCharArray(str_temp, TEMPSENSORNAME_STR_LENGTH);
            bfill.emit_p(webpage_temperature, str_temp, dtostrf(temperature[i], 4, 2, str_temp + TEMPSENSORNAME_STR_LENGTH));
            //printTemperature(temperature[i], tempSensorName[i], oneWirePins[i]);
        }
        //Serial.println("Added to network buffer done...");
        }
    else if (strncmp("ipconfig ", data, 9) == 0) {
        NetEeprom.readIp(myip);
        NetEeprom.readGateway(gwip);
        NetEeprom.readDns(dnsip);
        NetEeprom.readSubnet(netmask);
        bfill.emit_p(http_OK_200);
        char checked_dhcp[8] = "";
        if (NetEeprom.isDhcp())
            strcat(checked_dhcp, "checked\0");
        bfill.emit_p(webpage_ipconfig, hostname_client_connected, checked_dhcp, myip[0], myip[1], myip[2], myip[3], netmask[0], netmask[1], netmask[2], netmask[3], gwip[0], gwip[1], gwip[2], gwip[3],...
else {
    int i;
    for (i = 0; i < sizeof(str_temp) - 1; i++) {
        if (data[i] == ' ') {
            str_temp[i] = '\0';
            break;
        }
        str_temp[i] = data[i];
    }
    bfill.emit_p(http_not_found_404);
    bfill.emit_p(webpage_not_found, str_temp, ether.myip[0], ether.myip[1], ether.myip[2], ether.myip[3]);
}
}
else if (strncmp("POST /ipconfig ", data, 14) == 0) {
    //Serial.println(data);
    get_hostname_from_http_request(data, hostname_client_connected, HOSTNAME_MAX_SIZE);
    //Serial.print("Client connected to: ");
    //Serial.println(hostname_client_connected);
    int datalen = strlen(data);
    //Serial.println("Data length: ");
    //Serial.println(datalen);
    /* Find the offset that the POSTed data start */
    while (data[datalen--] != '\n')
        continue;
    data += (datalen += 2);
    //Serial.println(data);
    //Serial.println(strlen(data));
    int i, j = 0;
    /* if read_var_name == true, 
    then we parse the name of the variable we want to read 
    i.e. ip, dns, subnet or gw 
    if read_var_name == false, 
    then we parse the value for the previously read var_name */
    bool read_var_name = true;
    /* If the configuration parameters are ok, 
    set the corresponding IP addresses */
    bool static_conf_ok = true;
    char value[20];
    if (strlen(data) == 0) {
        len = ether.packetReceive();
        pos = ether.packetLoop(len);
        data = (char *) Ethernet::buffer + pos;
        //Serial.print("Second read: ");
        //Serial.println(data);
    }
    for (i = 0; i <= strlen(data); i++) {
        /* if we read '=', then the value should follow */
        if (data[i] == '=') {
            /* if we read ' = ', then the value should follow */
            if (data[i] == ' = ')
                {
/* so add a string termination character in the str_temp */
str_temp[j] = '\0';
j = 0;
/* set the read_var_name=false */
read_var_name = false;
continue;
}

/* if the current character is '&', or the string termination character '\0'
which marks the end of the available data, then the value reading has finished
and we can start reading the next variable (if more variables exist) and process
the current one */
if (data[i] == '&' || data[i] == '\0')
{
    /* so terminate the value string */
    value[j] = '\0';
    j = 0;
    /* enable the flag read_var_name so that we can start reading a
    variable name in the next loop */
    read_var_name = true;
    if (strncmp(str_temp, "dhcp", 4) == 0)
    {
        // Serial.print("dhcp: ");
        // Serial.println(value);
        /* If it is not set already to DHCP, do it now.
        and set the static_conf_ok = false since we
        configure the ip address setting by using DHCP
        */
        static_conf_ok = false;
        if (!NetEeprom.isDhcp())
        {
            NetEeprom.writeDhcpConfig(mymac);
            bfill.emit_p(http_OK_200);
            bfill.emit_p(webpage_please_connect_manually);
            ether.httpServerReply(bfill.position());
            software_Reset();
        }
        else if (strncmp(str_temp, "ip", 2) == 0)
        {
            //Serial.print("ip: ");
            //Serial.println(value);
            if (ether.parseIp(myip, value) != 0)
            {
                static_conf_ok = false;
                break;
            }
        }
        else if (strncmp(str_temp, "gw", 2) == 0)
        {
            //Serial.print("gw: ");
            //Serial.println(value);
            if (ether.parseIp(gwip, value) != 0)
            {
                static_conf_ok = false;
                break;
            }
        }
        else if (strncmp(str_temp, "dns", 3) == 0)
        {
            //Serial.print("dns: ");
        }
    }
}
//Serial.println(value);
if (ether.parseIp(dnsip, value) != 0)
{
    static_conf_ok = false;
    break;
}
else if (strcmp(str_temp, "subnet", 6) == 0)
{
    //Serial.print("subnet: ");
    //Serial.println(value);
    if (ether.parseIp(netmask, value) != 0)
    {
        static_conf_ok = false;
        break;
    }
    else
    {
        //Serial.println("subnet is a valid IP");
        /* It is not enough for the subnet mask
to be a valid IP address.
It needs to follows some additional
rules, so call subnet_mask_valid()
function to make sure it is a valid
subnet mask.
*/
        if (!subnet_mask_valid(netmask))
        {
            Serial.println("subnet is not a valid mask!");
            static_conf_ok = false;
            break;
        }
    }
}
if (data[i] == ' ')
    break;
else
    continue;
}
if (data[i] != '=')
{
    if (read_var_name)
        str_temp[j++] = data[i];
    else
        value[j++] = data[i];
}
if (static_conf_ok)
{
    NetEeprom.writeManualConfig(mymac, myip, gwip, netmask, dnsip);
    bfill.emit_p(http_OK_200);
    bfill.emit_p(webpage_please_connect_manually);
    ether.httpServerReply(bfill.position());
    software_Reset();
}
NetEeprom.readIp(myip);
NetEeprom.readGateway(gwip);
NetEeprom.readDns(dnsip);
NetEeprom.readSubnet(netmask);
bfill.emit_p(http_OK_200);
```c
char checked_dhcp[8] = "";
if (NetEeprom.isDhcp())
    strcat(checked_dhcp, "checked\0");
bfill.emit_p(webpage_ipconfig,
    hostname_client_connected,
    checked_dhcp,
    myip[0], myip[1], myip[2], myip[3],
    netmask[0], netmask[1], netmask[2], netmask[3],
    gwip[0], gwip[1], gwip[2], gwip[3],
    dnsip[0], dnsip[1], dnsip[2], dnsip[3]);
```

9.2 OpenTSDB Parser

```python
#!/usr/bin/env python
import urllib2
import re
import time
import json
import httplib

# Lists for differential
leftright = []
sumright = []
sumleft = []
sumrear = []
sumfront = []
P1FSUM = []
P1RSUM = []
P2FSUM = []
P2RSUM = []
P3FSUM = []
P3RSUM = []
P4FSUM = []
P4RSUM = []
P5FSUM = []
P5RSUM = []

# Lists for average values
FLlist = []
FRlist = []
RRlist = []
RLlist = []
P1list = []
P2list = []
P3list = []
P4list = []
P5list = []
P1_F_avglist = []
P2_F_avglist = []
P3_F_avglist = []
P4_F_avglist = []
P5_F_avglist = []
P1_R_avglist = []
P2_R_avglist = []
P3_R_avglist = []
P4_R_avglist = []
P5_R_avglist = []
```
try:
    while(1):
        ts = time.time()
        if (ts - prev_ts < measure_every):
            time.sleep(measure_every - (ts - prev_ts))
        ts = time.time()
        opentsdb_data = []
        with open(logfile, 'a') as logger:
            try:
                arduino_response = urllib2.urlopen(vaguino_temperatures_url, timeout = 10)
                r = re.findall('(.*)=(.*)\s+', arduino_response.read().split('<br>'))
                if (r):
                    container.append(r)
                z = re.findall('(.*)=(.*)\s+', arduino_response.read().split('<br>'))
                if (z):
                    sensorlist.append(z)
            except urllib2.URLError:
                # If the URL cannot be reached, ignore this error and retry.
                log_message = '{0} : The URL {1}'.format(time.ctime(ts), requ}
            for item in sensorlist:
                senso
for key, value in mydict.items():  # loop through the dictionary.
    if (key, value)
        stringvar = "%s,%s "%(key, value).
    if (a):
        FLlist.append(float(value))
    sensor_description = 'FL_Average'
    FL = sum(FLlist) / float(len(FLlist))
    value = FL
    data = {}
    data['metric'] = 'average.values.sensors'
    data['timestamp'] = ts
    data['value'] = value
    data['tags'] = {}
    data['tags']['sensor_description'] = sensor_description
    opentsdb_data.append(data)
for key, value in mydict.items():  # loop through the dictionary.
    if (c):
        RLlist.append(float(value))
        sensor_description = 'RL_Average'
        RL = sum(RLlist) / float(len(RLlist))
        value = RL
        data = {}
        data['metric'] = 'average.values.sensors'
        data['timestamp'] = ts
        data['value'] = value
        data['tags'] = {}
        data['tags']['sensor_description'] = sensor_description
        opentsdb_data.append(data)
for key, value in mydict.items():  # loop through the dictionary.
    if (d):
        RLlist.append(float(value))
        sensor_description = 'RL_Average'
        RL = sum(RLlist) / float(len(RLlist))
        value = RL
        data = {}
        data['metric'] = 'average.values.sensors'
        data['timestamp'] = ts
        data['value'] = value
        data['tags'] = {}
        data['tags']['sensor_description'] = sensor_description
        opentsdb_data.append(data)
stringvar = "%s,%s " %(key,value)
e = re.findall('(P1_[A-Z]{2}),(.*)\s+', stringvar)
if (e):
P1list.append(float(value))
sensor_description = 'P1_avg'
P1 = sum(P1list) / float(len(P1list))
P1_leap = open('P1.AVG_leapdetection.txt', 'a')
leapstring = '"%s" %P1
P1_leap.write(leapstring)
value = P1
data = {} 
data['metric'] = 'average.values.sensors'
data['timestamp'] = ts
data['value'] = value
data['tags'] = {}
data['tags']['sensor_description'] = sensor_description
opentsdb_data.append(data)

for key, value in mydict.iteritems(): ## loop through they dictionary.
    iteritems is needed to loop through values and not only key.
    #print key, value
    stringvar = "%s,%s " %(key,value)
f = re.findall('([P][0-4][A-Z]{2}),(.*)\s+', stringvar)
if (f):
P2list.append(float(value))
sensor_description = 'P2_avg'
P2 = sum(P2list) / float(len(P2list))
P2_leap = open('P2.AVG_leapdetection.txt', 'a')
leapstring = '"%s" %P2
P2_leap.write(leapstring)
value = P2
data = {} 
data['metric'] = 'average.values.sensors'
data['timestamp'] = ts
data['value'] = value
data['tags'] = {}
data['tags']['sensor_description'] = sensor_description
opentsdb_data.append(data)

for key, value in mydict.iteritems(): ## loop through they dictionary.
    iteritems is needed to loop through values and not only key.
    #print key, value
    stringvar = "%s,%s " %(key,value)
g = re.findall('(P3_[A-Z]{2}),(.*)\s+', stringvar)
if (g):
P3list.append(float(value))
sensor_description = 'P3_avg'
P3 = sum(P3list) / float(len(P3list))
P3_leap = open('P3.AVG_leapdetection.txt', 'a')
leapstring = '"%s" %P3
P3_leap.write(leapstring)
value = P3
data = {} 
data['metric'] = 'average.values.sensors'
data['timestamp'] = ts
data['value'] = value
data['tags'] = {}
data['tags']['sensor_description'] = sensor_description
opentsdb_data.append(data)

for key, value in mydict.iteritems(): ## loop through they dictionary.
    iteritems is needed to loop through values and not only key.
    #print key, value
    stringvar = "%s,%s " %(key,value)
h = re.findall('(P4_[A-Z]{2}),(.*)\s+', stringvar)
if (h):
P4list.append(float(value))
sensor_description = 'P4_avg'
P4 = sum(P4list) / float(len(P4list))
P4_leap = open('P4.AVG_leapdetection.txt', 'a')
leapstring = '"%s" %P4
P4_leap.write(leapstring)
value = P4
data = {} 
data['metric'] = 'average.values.sensors'
data['timestamp'] = ts
data['value'] = value
data['tags'] = {}
data['tags']['sensor_description'] = sensor_description
opentsdb_data.append(data)
319 opentsdb_data.append(data)
320
321
322 for key, value in mydict.items(): # loop through they dictionary.
323     #print key, value
324     stringvar = "%s,%s " % (key, value)
325     i = re.findall("(P5_[A-Z]{2}),(.*)\s+", stringvar)
326     if (i):
327         P5list.append(float(value))
328         sensor_description = "P5_avg"
329         P5 = sum(P5list) / float(len(P5list))
330         P5_leap = open("P5.AVG_leapdetection.txt", "a")
331         leapstring = "%s\n" % P5
332         P5_leap.write(leapstring)
333         value = P5
334         data = {}
335         data["metric"] = "average.values.sensors"
336         data["timestamp"] = ts
337         data["value"] = value
338         data["tags"] = {}
339         data["tags"]['sensor_description'] = sensor_description
340         opentsdb_data.append(data)
341
342     for key, value in mydict.items(): # loop through they dictionary.
343     #print key, value
344     stringvar = "%s,%s " % (key, value)
345     j = re.findall("(P1_F[A-Z{R|L}]),(.*)\s+", stringvar)
346     if (j):
347         P1_F_avglist.append(float(value))
348         sensor_description = "P1_FRONT_AVG"
349         P1_FRONT_AVG = sum(P1_F_avglist) / float(len(P1_F_avglist))
350         data = {}
351         data["metric"] = "average.values.sensors"
352         data["timestamp"] = ts
353         data["value"] = value
354         data["tags"] = {}
355         data["tags"]['sensor_description'] = sensor_description
356         opentsdb_data.append(data)
357
358     for key, value in mydict.items(): # loop through they dictionary.
359     #print key, value
360     stringvar = "%s,%s " % (key, value)
361     j2 = re.findall("(P2_F[A-Z{R|L}]),(.*)\s+", stringvar)
362     if (j2):
363         P2_F_avglist.append(float(value))
364         sensor_description = "P2_FRONT_AVG"
365         P2_FRONT_AVG = sum(P2_F_avglist) / float(len(P2_F_avglist))
366         data = {}
367         data["metric"] = "average.values.sensors"
368         data["timestamp"] = ts
369         data["value"] = value
370         data["tags"] = {}
371         data["tags"]['sensor_description'] = sensor_description
372         opentsdb_data.append(data)
373
374     for key, value in mydict.items(): # loop through they dictionary.
375     #print key, value
376     stringvar = "%s,%s " % (key, value)
377     j3 = re.findall("(P3_F[A-Z{R|L}]),(.*)\s+", stringvar)
378     if (j3):
379         P3_F_avglist.append(float(value))
380         sensor_description = "P3_FRONT_AVG"
381         P3_FRONT_AVG = sum(P3_F_avglist) / float(len(P3_F_avglist))
382         data = {}
383         data["metric"] = "average.values.sensors"
384         data["timestamp"] = ts
385         data["value"] = value
386         data["tags"] = {}
387         data["tags"]['sensor_description'] = sensor_description
388         opentsdb_data.append(data)
389
390     for key, value in mydict.items(): # loop through they dictionary.
391     #print key, value
392     stringvar = "%s,%s " % (key, value)
393     j4 = re.findall("(P4_F[A-Z{R|L}]),(.*)\s+", stringvar)
394     if (j4):
395         P4_F_avglist.append(float(value))
396         sensor_description = "P4_FRONT_AVG"
397         P4_FRONT_AVG = sum(P4_F_avglist) / float(len(P4_F_avglist))
398         data = {}
399         data["metric"] = "average.values.sensors"
400         data["timestamp"] = ts
401         data["value"] = value
402         data["tags"] = {}
403         data["tags"]['sensor_description'] = sensor_description
404         opentsdb_data.append(data)
P4_F_avglist.append(float(value))
sensor_description = 'P4_FRONT_AVG'
P4_FRONT_AVG = sum(P4_F_avglist) / float(len(P4_F_avglist))

for key, value in mydict.iteritems():  # loop through dictionary.
    #print key, value
    stringvar = "%s,%s " %(key, value)
    js = re.findall('(P5_F[A-Z]{R|L}),(.*)\s', stringvar)
    if (js):
        P5_F_avglist.append(float(value))
        sensor_description = 'P5_FRONT_AVG'
P5_FRONT_AVG = sum(P5_F_avglist) / float(len(P5_F_avglist))
        value = P5_FRONT_AVG

        data = {}
        data['metric'] = 'average.values.sensors'
        data['timestamp'] = ts
        data['value'] = value
        data['tags'] = {}
        data['tags']['sensor_description'] = sensor_description
        opentsdb_data.append(data)

        httpServ = httplib.HTTPConnection("128.39.120.212", 4242)
        httpServ.connect()
        http_headers = {'Content-Type': 'application/json; charset=UTF-8'}
        json_data = json.dumps(opentsdb_data)
        #print json.dumps(opentsdb_data, indent=4, sort_keys=True)
        httpServ.request('POST', '/api/put', json_data, http_headers)
        response = httpServ.getresponse()
        if response.status == 204:
            log_message = '{0}: Data has been stored successfully'.format(time.ctime(ts))
        else:
            log_message = '{0}: An error occurred. Response code {1}'.format(time.ctime(ts), response.status)

        httpServ.close()
        #except urllib2.URLError:
        #    If the URL cannot be reached, ignore this error and retry.
        log_message = '{0}: The URL '{1}' could not be reached.'.format(time.ctime(ts), vaguino_temperatures_url)

        P1_R_avglist = []
        for key, value in mydict.iteritems():  # loop through dictionary.
            #print key, value
            stringvar = "%s,%s " %(key, value)
            a = re.findall('(P1_R[A-Z]{R|L}),(.*)\s', stringvar)
            if (a):
                P1_R_avglist.append(float(value))
                sensor_description = 'P1_REAR_AVG'
P1_REAR_AVG = sum(P1_R_avglist) / float(len(P1_R_avglist))
                value = P1_REAR_AVG

                data = {}
                data['metric'] = 'average.values.sensors'
                data['timestamp'] = ts
                data['value'] = value
                data['tags'] = {}
                data['tags']['sensor_description'] = sensor_description
                opentsdb_data.append(data)

                httpServ = httplib.HTTPConnection("128.39.120.212", 4242)
                httpServ.connect()
                http_headers = {'Content-Type': 'application/json; charset=UTF-8'}
                json_data = json.dumps(opentsdb_data)
                #print json.dumps(opentsdb_data, indent=4, sort_keys=True)
                httpServ.request('POST', '/api/put', json_data, http_headers)
                response = httpServ.getresponse()
                if response.status == 204:
                    log_message = '{0}: Data has been stored successfully'.format(time.ctime(ts))
                else:
                    log_message = '{0}: An error occurred. Response code {1}'.format(time.ctime(ts), response.status)

                httpServ.close()

        log_message = '{0}: An error occurred. Response code {1}'.format(time.ctime(ts), response.status)
data['value'] = value
data['tags'] = {}
data['tags']['sensor_description'] = sensor_description
open_data.append(data)

for key, value in mydict.iteritems(): # loop through dictionary.
    #print key, value
    stringvar = "%s,%s " %(key,value)
    a3 = re.findall('([P3_R[A-Z]{2,}]),(.*)\s', stringvar)
    if (a3):
        P3_R_avglist.append(float(value))
sensor_description = 'P3_REAR_AVG'
P3_REAR_AVG = sum(P3_R_avglist) / float(len(P3_R_avglist))
value = P3_REAR_AVG
data = {}
data['metric'] = 'average.values.sensors'
data['timestamp'] = ts
data['value'] = value
data['tags'] = {}
data['tags']['sensor_description'] = sensor_description
open_data.append(data)

for key, value in mydict.iteritems(): # loop through dictionary.
    #print key, value
    stringvar = "%s,%s " %(key,value)
    a4 = re.findall('([P4_R[A-Z]{2,}]),(.*)\s', stringvar)
    if (a4):
        P4_R_avglist.append(float(value))
sensor_description = 'P4_REAR_AVG'
P4_REAR_AVG = sum(P4_R_avglist) / float(len(P4_R_avglist))
value = P4_REAR_AVG
data = {}
data['metric'] = 'average.values.sensors'
data['timestamp'] = ts
data['value'] = value
data['tags'] = {}
data['tags']['sensor_description'] = sensor_description
open_data.append(data)

for key, value in mydict.iteritems(): # loop through dictionary.
    #print key, value
    stringvar = "%s,%s " %(key,value)
    a5 = re.findall('([P5_R[A-Z]{2,}]),(.*)\s', stringvar)
    if (a5):
        P5_R_avglist.append(float(value))
sensor_description = 'P5_REAR_AVG'
P5_REAR_AVG = sum(P5_R_avglist) / float(len(P5_R_avglist))
value = P5_REAR_AVG
data = {}
data['metric'] = 'average.values.sensors'
data['timestamp'] = ts
data['value'] = value
data['tags'] = {}
data['tags']['sensor_description'] = sensor_description
open_data.append(data)

httpServ = httplib.HTTPConnection("128.39.120.212", 4242)
httpServ.connect()

http_headers = {'Content-Type': 'application/json; charset=UTF-8'}
json_data = json.dumps(open_data)
#print json.dumps(open_data, indent=4, sort_keys=True)
httpServ.request('POST', '/api/put', json_data, http_headers)
response = httpServ.getresponse()
if response.status == 204:
    log_message = "(0): Data has been stored successfully.
    (time.ctime(ts))"
else:
    log_message = "(0): An error occurred. Response code
    1).format(time.ctime(ts), response.status)
httpServ.close()
a = re.findall('([A-Z_{4}]L),(.*)
', stringvar)
if (a):
    sumleft.append(float(value))
a1 = re.findall('([A-Z_{4}]R),(.*)
', stringvar)
if (a1):
    sumright.append(float(value))
leftsum = sum(sumleft) / float(len(sumleft))
rightsum = sum(sumright) / float(len(sumright))
diffLR = rightsum - leftsum
abs(diffLR)

DIFFLR_leap = open('DIFFLR_leapdetection.txt', 'a')
leapstring = '%s 
' % (diffLR)
DIFFLR_leap.write(leapstring)

for key, value in mydict.iteritems(): ## loop through dictionary.
    #print key, value
    stringvar = '%s,%s ' %(key,value)
b = re.findall('([A-Z_{4}]F[RL]),(.*)
', stringvar)
if (b):
    sumfront.append(float(value))
b1 = re.findall('([A-Z_{4}]R[RL]),(.*)
', stringvar)
if (b1):
    sumrear.append(float(value))
frontsum = sum(sumfront) / float(len(sumfront))
rearsum = sum(sumrear) / float(len(sumrear))
frontbackdiff = rearsum - frontsum
abs(frontbackdiff)

DIFF_Front_back_leap = open('DIFF_Front_back_leapdetection.txt', 'a')
leapstring = '%s 
' % (frontbackdiff)
DIFF_Front_back_leap.write(leapstring)

for key, value in mydict.iteritems(): ## loop through dictionary.
    #print key, value
    stringvar = '%s,%s ' %(key, value)
c = re.findall('(P1_F[A-Z]{1}),(.*)
', stringvar)
if (c):
    P1FSUM.append(float(value))
c1 = re.findall('(P1_R[A-Z]{1}),(.*)
', stringvar)
if (c1):
    P1RSUM.append(float(value))
P1diff = sum(P1RSUM) - sum(P1FSUM)
sensor_description = 'Plane1_DIFF'
value = abs(P1diff)
data = {}
data['metric'] = 'rack.differantials.temp'
data['timestamp'] = ts
data['value'] = value
data['tags'] = {}
data['tags']['sensor_description'] = sensor_description
opentsdb_data.append(data)

for key, value in mydict.iteritems(): ## loop through dictionary.
    #print key, value
    stringvar = '%s,%s ' %(key, value)
d = re.findall('(P2_F[A-Z]{1}),(.*)
', stringvar)
if (d):
    P2FSUM.append(float(value))
d1 = re.findall('(P2_R[A-Z]{1}),(.*)
', stringvar)
if (d1):
    P2RSUM.append(float(value))
P2diff = sum(P2RSUM) - sum(P2FSUM)
sensor_description = 'Plane2_DIFF'
value = abs(P2diff)
data = {}
data['metric'] = 'rack.differantials.temp'
data['timestamp'] = ts
data['value'] = value
data['tags'] = {}
data['tags']['sensor_description'] = sensor_description
opentsdb_data.append(data)

for key, value in mydict.iteritems(): ## loop through they dicitnary.
    #print key, value
    stringvar = "%.2f" % (key, value)
e = re.findall('P2_F[A-Z]{1},(.*)\s', stringvar)
if (e):
P2FSUM.append(float(value))
e1 = re.findall('P2_R[A-Z]{1},(.*)\s', stringvar)
if (e1):
P2RSUM.append(float(value))
P2diff = sum(P2RSUM) - sum(P2FSUM)
sensor_description = 'Plane3_DIFF'
value = abs(P3diff)
data = {}
data['metric'] = 'rack.differantials.temp'
data['timestamp'] = ts
data['value'] = value
data['tags'] = {}
data['tags']['sensor_description'] = sensor_description
opentsdb_data.append(data)

for key, value in mydict.iteritems(): ## loop through they dicitnary.
    #print key, value
    stringvar = "%.2f" % (key, value)
f = re.findall('P3_F[A-Z]{1},(.*)\s', stringvar)
if (f):
P3FSUM.append(float(value))
f1 = re.findall('P3_R[A-Z]{1},(.*)\s', stringvar)
if (f1):
P3RSUM.append(float(value))
P3diff = sum(P3RSUM) - sum(P3FSUM)
sensor_description = 'Plane4_DIFF'
value = abs(P4diff)
data = {}
data['metric'] = 'rack.differantials.temp'
data['timestamp'] = ts
data['value'] = value
data['tags'] = {}
data['tags']['sensor_description'] = sensor_description
opentsdb_data.append(data)

for key, value in mydict.iteritems(): ## loop through they dicitnary.
    #print key, value
    stringvar = "%.2f" % (key, value)
g = re.findall('P4_F[A-Z]{1},(.*)\s', stringvar)
if (g):
P4FSUM.append(float(value))
g1 = re.findall('P4_R[A-Z]{1},(.*)\s', stringvar)
if (g1):
P4RSUM.append(float(value))
P4diff = sum(P4RSUM) - sum(P4FSUM)
sensor_description = 'Plane5_DIFF'
value = abs(P5diff)
data = {}
data['metric'] = 'rack.differantials.temp'
data['timestamp'] = ts
data['value'] = value
data['tags'] = {}
data['tags']['sensor_description'] = sensor_description
opentsdb_data.append(data)

for key, value in mydict.iteritems(): ## loop through they dicitnary.
    #print key, value
    stringvar = "%.2f" % (key, value)
f1 = re.findall('P5_F[A-Z]{1},(.*)\s', stringvar)
if (f1):
P5FSUM.append(float(value))
f1 = re.findall('P5_R[A-Z]{1},(.*)\s', stringvar)
if (f1):
P5RSUM.append(float(value))
P5diff = sum(P5RSUM) - sum(P5FSUM)
sensor_description = 'FRONT_TOP/BOT_DIFF'
value = abs(Ftopbotdiff)
data = {}
data['metric'] = 'rack.differantials.temp'
data['timestamp'] = ts
data['value'] = value
data['tags'] = {}
data['tags']['sensor_description'] = sensor_description
opentsdb_data.append(data)

for key, value in mydict.iteritems(): ## loop through they dicitnary.
    #print key, value
    stringvar = "%.2f" % (key, value)
g = re.findall('P1_F[A-Z]{1},(.*)\s', stringvar)
if (g):
P1FSUM.append(float(value))
g1 = re.findall('P1_R[A-Z]{1},(.*)\s', stringvar)
if (g1):
P1RSUM.append(float(value))
P1diff = sum(P1RSUM) - sum(P1FSUM)
sensor_description = 'FRONT_TOP/BOT_DIFF'
value = abs(Ftopbotdiff)
data = {}
data['metric'] = 'rack.differantials.temp'
data['timestamp'] = ts
data['value'] = value
data['tags'] = {}
data['tags']['sensor_description'] = sensor_description
opentsdb_data.append(data)

for key, value in mydict.iteritems(): ## loop through they dicitnary.
    #print key, value
    stringvar = "%.2f" % (key, value)
g = re.findall('P5_F[A-Z]{1},(.*)\s', stringvar)
if (g):
P5FSUM.append(float(value))
g1 = re.findall('P5_R[A-Z]{1},(.*)\s', stringvar)
if (g1):
P5RSUM.append(float(value))
P5diff = sum(P5RSUM) - sum(P5FSUM)
sensor_description = 'FRONT_TOP/BOT_DIFF'
value = abs(Ftopbotdiff)
data = {}
data['metric'] = 'rack.differantials.temp'
data['timestamp'] = ts

Ftopbotdiff = sum(P1FSUM) - sum(P1RSUM)
ftopbot_leap = open('diff_ftopbot_leapdetection.txt', 'a')
leapstring = "%.2f" % (Ftopbotdiff)
ftopbot_leap.write(leapstring)

""
data['value'] = value
data['tags'] = {}

data['value']['sensor_description'] = sensor_description
opentsdb_data.append(data)

topbotdiff = sum(P5RSUM) - sum(P1RSUM)

topbotLeap = open('diff_topbotLeapdetection.txt', 'a')
leapstring = "%s
" % (topbotdiff)

topbotLeap.write(leapstring)

sensor_description = 'REAR_TOP/BOT_DIFF'

value = abs(topbotdiff)

data = {}
data['metric'] = 'rack.differantials.temp'
data['timestamp'] = ts
data['value'] = value
data['tags'] = {}
data['tags']['sensor_description'] = sensor_description
opentsdb_data.append(data)

httpServ = httpplib.HTTPConnection("128.39.120.212", 4242)

httpServ.connect()

http_headers = {'Content-Type': 'application/json; charset=UTF-8'}

json_data = json.dumps(opentsdb_data)

httpServ.request('POST', '/api/put', json_data, http_headers)

response = httpServ.getresponse()

if response.status == 204:
    log_message = "{0}
".format(time.ctime(ts))
    Data has been stored successfully.

else:
    log_message = "{0}: An error occurred. Response code
".format(time.ctime(ts), response.status)
httpServ.close()
print "\nStopping data collection...\n"
exit(0)

9.3 Leap Detection Test script

#!/usr/bin/perl

# the size of the short term memory:
my $MEMORY_LENGTH = 10;

# The memory buffer:
my @MEMORY = ();

# Max
my $MAX;

# Average:
my $AVG;

# Sum:
my $SUM;

# The index for the buffer-rotation:
my $index = -1;

# The currently read data item
my $data;

# The result of the CHI squared computation:
my $chi = 0;

# counter
my $counter = 0;

# Application specific variables. Not part of the algorithm
use strict "vars";
use Getopt::Std;
my $opt_string = 'vm:c:qaDl:e:';
getopts("$opt_string", \my %opt ) or usage() and exit;

my $QUIET;
my $QUIET_CHECK;
my $DRAMATIC = 0;
$DRAMATIC = 1 if $opt{D};

$QUIET = 1 if $opt{q};
my $VERBOSE;
$VERBOSE = 1 if $opt{v};

my $CHI_LIMIT;
$CHI_LIMIT = $opt{c} if $opt{c};

my $ALL_EVENTS;
$ALL_EVENTS = 1 if $opt{a};

my $MEMORY_LENGTH = $opt{m} if $opt{m};

my $EXP_START = $opt{e};

my $EXP_LENGTH = $opt{l};

#######################

while ( $data = <STDIN> ) {

    # removing newline:
    chomp $data;
    $data = ( $data * -1 ) if $data < 0;
    $index++;
    $counter++;

    # $MAX = $data if $data > $MAX;
    verbose("n--- read data: $data\n");
    if ( $#MEMORY + 1 < $MEMORY_LENGTH ){

        # we have not filled up the entire memory yet:
        $SUM += $data;
        $AVG += $data/($MEMORY_LENGTH + 1);
        verbose("inserting $data at position \
" . (#MEMORY + 1) ." (sum: $SUM) [avg: $AVG]\n");
        $MEMORY[$index] = $data;

    } else {

        # we have filled up the memory, and can now start the detection

        # lets find the adaptive treshold
        $MAX = find_max();

        my $MAXAVG = $AVG + $MAX/($MEMORY_LENGTH + 1);

        # we compute the chi squared:
        my $Tchi =
        sqrt(
          ($SUM - $MEMORY_LENGTH * $MAX)**2
          /
          ($MEMORY_LENGTH * ($MEMORY_LENGTH + 1) * $MAXAVG )
        );

        verbose("Adjusting new chi treshold to $Tchi with max: $MAX\n");
        $CHI_LIMIT = $Tchi;

        $AVG += $data/($MEMORY_LENGTH + 1);

    }
}

$AVG += $data/($MEMORY_LENGTH + 1);

# we compute the chi squared:
$chi =
sqrt(
  ($SUM - $MEMORY_LENGTH * $data)**2
  /
  ($MEMORY_LENGTH * ($MEMORY_LENGTH + 1) * $AVG )
)
$index = $index % $MEMORY_LENGTH;

$SUM = $SUM - $MEMORY[$index] + $data;
$AVG = $AVG - ($MEMORY[$index] / ($MEMORY_LENGTH + 1 ));
$MEMORY[$index] = $data;

}

print_summary();

}

sub print_summary {
  if ( $QUIET ){
    if ( $CHI_LIMIT and $chi > $CHI_LIMIT and not $QUIET_CHECK){
      print "\n";
      $QUIET_CHECK = 1;
      return;
    }
    return;
  }
  if ( $ALL_EVENTS ){  
    print "$counter , $chi Vs $CHI_LIMIT, ";
    if ( $CHI_LIMIT and $chi > $CHI_LIMIT){
      if ( $DRAMATIC ){  
        if ( $EXP_START and $EXP_LENGTH and ( $counter > $EXP_START and $counter < ($EXP_START + $EXP_LENGTH))){
          print "1 TRUE LEAP DETECTED\n";
        } elsif ( $EXP_START and $EXP_LENGTH and ( $counter < $EXP_START or $counter > $EXP_START + $EXP_LENGTH )){
          print "1 FALSE LEAP DETECTED\n";
        } else {
          print "1\n";
        }
      } else {
        print "0\n";
      }
    } else {
      return;
    }
  }
  if ($VERBOSE) {
    print_memory();
    verbose("chi($MEMORY_LENGTH) = $chi [".($chi**2) \\n"]
    verbose("new index: $index (sum: $SUM) [avg: $AVG]\n")
  } else {
    print "$chi";
    if ($CHI_LIMIT and $chi > $CHI_LIMIT ){
      print " *";
    }
    print "\n";
  }
}

sub verbose {
  if ($VERBOSE) {
    print $_[0];
  }
}
### A) Installing and loading required packages

```bash
if (!require("gplots")) {
  install.packages("gplots", dependencies = TRUE)
  library(gplots)
}
if (!require("RColorBrewer")) {
  install.packages("RColorBrewer", dependencies = TRUE)
  library(RColorBrewer)
}
```

### B) Reading in data and transform it into matrix format

```r
data <- read.csv("alltest_timeline.csv", comment.char="#")
rnames <- data[,1] # assign labels in column 1 to "rnames"
mat_data <- data.matrix(data[,2:ncol(data)]) # transform column 2-5 into a matrix
rownames(mat_data) <- rnames # assign row names
```

### C) Customizing and plotting the heat map

```r
# creates a own color palette from red to green
my_palette <- colorRampPalette(c("green", "yellow", "red"))(n = 299)
# (optional) defines the color breaks manually for a "skewed" color transition
col.breaks = c(seq(0,20,length=100), # for red
    seq(21,25,length=100), # for yellow
    seq(26,100,length=100)) # for green
# creates a 5 x 5 inch image
png("all_cases.png", # create PNG for the heat map
    width = 5*300, # 5 x 300 pixels
    height = 5*300,
    res = 300, # 300 pixels per inch
    pointsize = 8, # smaller font size
    heatmap.2(mat_data, # heatmap function
        cellnote = mat_data, # same data set for cell labels
        main = "improved", # heatmap title
        notecol="black", # change font color of cell labels to black
        col = my_palette,
        colometers = TRUE,
        breaks = col.breaks,
        scales = TRUE,
        densitycol="black",
        densityr NULL,
        )
```
9.5 3D Heatmap

```python
import matplotlib
matplotlib.use("Agg")
import os
os.environ['MPLCONFIGDIR'] = '/home/migjen/
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

#readfile = start_completeobstruction.csv
xs = []
ys = []
zs = []
the_fourth_dimension= []
import csv

with open ('3dtopcpu.txt', 'rb') as csvfile:
    reader = csv.reader(csvfile)
    for row in reader:
        xs.append(int(row[0]))
        ys.append(int(row[1]))
        zs.append(int(row[2]))
        the_fourth_dimension.append(float(row[3]))
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
#n = 100
import csv

colors = the_fourth_dimension/max(the_fourth_dimension)
#colors = [the_fourth_dimension*2**n for n in range(len(xs))]
colmap = cm.ScalarMappable(cmap=cm.hsv)
colmap.set_array(the_fourth_dimension)
yg = ax.scatter(xs, ys, zs, s = 500, c=colors, marker='o')
#cb = fig.colorbar(colmap)
ax.set_xlabel('Cords')
ax.set_ylabel('Cords')
ax.set_zlabel('Planes')
plt.savefig("3dtopcpu")
```