Root-cause analysis through machine learning in the cloud

Tim Josefsson
Abstract

Root-cause analysis through machine learning in the cloud

Tim Josefsson

It has been predicted that by 2021 there will be 28 billion connected devices and that 80% of global consumer internet traffic will be related to streaming services such as Netflix, Hulu and Youtube. This connectivity will in turn be matched by a cloud infrastructure that will ensure connectivity and services. With such an increase in infrastructure the need for reliable systems will also rise. One solution to providing reliability in data centres is root-cause analysis where the aim is to identifying the root-cause of a service degradation in order to prevent it or allow for easy localization of the problem.

In this report we explore an approach to root-cause-analysis using a machine learning model called self-organizing maps. Self-organizing maps provides data classification, while also providing visualization of the model which is something many machine learning models fail to do. We show that self-organizing maps are a promising solution to root-cause analysis. Within the report we also compare our approach to another prominent approach and show that our model preforms favorably.

Finally, we touch up some interesting research topics that we believe can further the field of root-cause analysis.
To Christina, who was my pillar throughout this project. Without her this report would not make nearly as much sense. To Andreas and Jawwad, who was always there when I needed advice and helped me see what was important in my work. To Salman, who helped me through the University-maze and helped me get this thesis off the ground and into the world.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Introduction</td>
<td>9</td>
</tr>
<tr>
<td>1.1 Objectives</td>
<td>10</td>
</tr>
<tr>
<td>1.2 Thesis outline</td>
<td>10</td>
</tr>
<tr>
<td>2 Background</td>
<td>11</td>
</tr>
<tr>
<td>2.1 Root-cause Analysis</td>
<td>11</td>
</tr>
<tr>
<td>2.2 Machine learning</td>
<td>11</td>
</tr>
<tr>
<td>2.3 Related work</td>
<td>13</td>
</tr>
<tr>
<td>2.3.1 Supervised Learning</td>
<td>14</td>
</tr>
<tr>
<td>2.3.2 Unsupervised Learning</td>
<td>14</td>
</tr>
<tr>
<td>2.3.3 Other approaches</td>
<td>15</td>
</tr>
<tr>
<td>3 The Self Organizing Map</td>
<td>17</td>
</tr>
<tr>
<td>3.1 Learning Algorithm</td>
<td>17</td>
</tr>
<tr>
<td>3.1.1 Self-Organizing Map Variations</td>
<td>18</td>
</tr>
<tr>
<td>4 Methodology</td>
<td>20</td>
</tr>
<tr>
<td>4.1 Testbed</td>
<td>20</td>
</tr>
<tr>
<td>4.1.1 Tools</td>
<td>20</td>
</tr>
<tr>
<td>4.1.2 Usage scenarios</td>
<td>21</td>
</tr>
<tr>
<td>4.2 Anomaly detection</td>
<td>22</td>
</tr>
<tr>
<td>4.3 Fault localization</td>
<td>22</td>
</tr>
<tr>
<td>4.3.1 Dissimilarity measurement scheme</td>
<td>22</td>
</tr>
<tr>
<td>4.4 Experiment setup</td>
<td>23</td>
</tr>
<tr>
<td>4.4.1 Data trace collection</td>
<td>23</td>
</tr>
<tr>
<td>4.4.2 Data smoothing</td>
<td>24</td>
</tr>
<tr>
<td>4.4.3 Sensitivity Study</td>
<td>24</td>
</tr>
<tr>
<td>4.4.4 Training and Prediction</td>
<td>25</td>
</tr>
<tr>
<td>4.4.5 Localization</td>
<td>26</td>
</tr>
<tr>
<td>5 Results &amp; Discussion</td>
<td>28</td>
</tr>
<tr>
<td>5.1 Sensitivity study</td>
<td>29</td>
</tr>
<tr>
<td>5.2 Training and Prediction</td>
<td>32</td>
</tr>
<tr>
<td>5.2.1 Training time</td>
<td>32</td>
</tr>
<tr>
<td>5.2.2 1-layered SOM vs 2-layered SOM</td>
<td>33</td>
</tr>
<tr>
<td>5.2.3 Specialized vs. Generalized map</td>
<td>34</td>
</tr>
<tr>
<td>5.3 Localization</td>
<td>34</td>
</tr>
<tr>
<td>5.3.1 1-layered SOM vs 2-layered SOM</td>
<td>35</td>
</tr>
<tr>
<td>5.3.2 Specialized vs. Generalized map</td>
<td>36</td>
</tr>
<tr>
<td>5.3.3 Evaluating the final system</td>
<td>37</td>
</tr>
<tr>
<td>5.4 Demonstrator</td>
<td>41</td>
</tr>
<tr>
<td>6 Conclusion</td>
<td>42</td>
</tr>
<tr>
<td>6.1 Future work</td>
<td>42</td>
</tr>
<tr>
<td>References</td>
<td>44</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Related works ................................................................. 16
4.1 Data traces collected ..................................................... 23
4.2 Minimal feature set ....................................................... 23
5.1 Comparison of 1-layered map and 2-layered map with regards to prediction accuracy. ..................................................... 33
5.2 Comparison of specialized map and generalized map with regards to prediction accuracy. ..................................................... 34
List of Figures

2.1 Decision tree showing survival of passengers on the Titanic. The number below each node is the probability of that outcome and the percentage of observations in that leaf [28]. ................................. 12
2.2 Support vector machine showing two possible hyperplanes to classify the data [12]. ................................................................. 13
2.3 A neural network with one hidden layer [26]. .......................... 13
3.1 A simple Self-Organizing Map ........................................ 17
4.1 The testbed setup .......................................................... 20
4.2 The effects of injected faults with respect to registered SLA violation. Teal is the injected fault and red is if a fault is registered at the client machine. ........... 24
4.3 The flow of the RCA system showcasing how each sample is handled. ....... 27
5.1 Prediction mapping of SOM. Ground truth of test set represented as 0 and 1. .. 28
5.2 Effect on the neighbourhood size threshold on different sized 1-layered maps. .. 30
5.3 Effect on the x-weight on different sized 2-layered maps. ....................... 31
5.4 Training and selection time for a SOM. .................................. 32
5.5 Total time taken for both training and selection. ........................... 33
5.6 ROC comparison of 1-layered and 2-layered SOM. ......................... 34
5.7 The impact of the size of a map on the localization performance. ............ 35
5.8 The impact of the x-weight of a map on the localization performance. ...... 36
5.9 Localization performance on memory fault comparing 1-layered and 2-layered SOM. .......................................................... 36
5.10 Localization performance on CPU fault comparing 1-layered and 2-layered SOM. .......................................................... 37
5.11 Localization performance on I/O fault comparing 1-layered and 2-layered SOM. .......................................................... 37
5.12 Localization performance on memory fault comparing specialized and generalized map. ......................................................... 38
5.13 Localization performance on CPU fault comparing specialized and generalized map. ......................................................... 38
5.14 Localization performance on I/O fault comparing specialized and generalized map. ......................................................... 39
5.15 10 hour periodic load pattern used for final system evaluation. ............... 39
5.16 Fault localization frequency for a periodic load data trace containing CPU, memory and I/O faults. ........................................ 40
5.17 Demonstrator example. ..................................................... 41
1. Introduction

Ericsson has predicted that by 2021 there will be 28 billion connected devices and that this connectivity will be matched by a cloud infrastructure that enables connectivity and services [4]. This predicted number by Ericsson is mostly based on the immense increase of devices used for machine to machine communication that accompanies the increase in Internet of Things solutions. In a similar study Cisco Systems have created a forecast that by 2021 internet video, which encompasses services such as Hulu, Netflix and Youtube, will produce more than 80% of the global consumer internet traffic[15]. Both these studies show that in order to support this increase there will be a need for reliable and sturdy data and cloud centers that can handle the increase in both connected devices and users. The inherent complexity of the cloud must be managed and optimized as the user come to expect, and become reliant on, their devices being connected to high-speed networks and high-quality services all the time. This will make real-time service assurance, root-cause analysis and anomaly detection important scientific areas for both present and future cloud infrastructure in order to provide a highly reliable cloud. The need for reliability in telecom clouds is not purely a scientific endeavour but also an economic journey since reliability problems can and will often lead to major economic losses for a company. In 2011 it was projected that IT downtime costs companies more than $26.5 billion in revenue [5] around the world. Five years later the projected losses for companies in North America alone was projected to be $700 billion[13]. In addition to this, it is important for companies to be able to provide assurances that they will be able to uphold their end of service level agreements(SLA) that have been offered to customers.

Due to the increased commercial interest in cloud infrastructure there has also been an increase in the interest for software solutions to help deliver reliability in data centres and cloud services. Thus the research into the field of root-cause analysis has been gaining popularity with hopes to find effective methods and models to provide reliability to cloud services. The hope is that by predicting and localizing faults and service degradations, engineers and technicians can make fact-based decisions on how to improve the system or mitigate the possible faults. This in turn would allow for companies to deliver a more reliable cloud service.

However, understanding and predicting the performance of a cloud service is by its nature a hard thing to do. The services are often a part of a large and complex software system that runs on a general purpose operating system platform [31]. Therefore understanding the performance of a system such as this does not only require expert domain knowledge but also analytical models that often tend to be overly complex.

An often used alternative approach to complex analytical models is to design and implement models based on statistical learning. This allows for models that learn the system behaviour from observation of system metrics and once implemented can make predictions of the system behaviour based on future observations. The downside is that a large amount of data containing observations need to be gathered, however the upside of this is that no knowledge about the system and inter component interactions is needed.

In this master thesis project, the focus is on exploring the possibilities of a machine learning approach and as such we forgo the statistical learning approach. Like statistical learning, machine learning builds models based on observational data and the user does not need to understand the underlying complexity of the system, and can thus provide accurate predictions of future observations. Another major strength found in some machine learning algorithms, is that the model is able to learn the topological properties of the observational data which is something that we leverage in this project in order to provide fault localization in the system.
1.1 Objectives

The main objectives of this project have been to implement, evaluate and further improve state-of-the-art for troubleshooting and root-cause analysis (RCA) through machine learning in order to deliver a highly reliable telecom cloud. In order to achieve these objectives we have developed a testbed environment that replicates a video-on-demand cloud service in a data centre, this testbed is based on the work done in [31]. The testbed has been developed to include monitoring functionality, fault injection and several different load scenarios. To go with the testbed we have also developed a prediction and localization engine based on Kohonen’s self-organizing map [11] that is able to run in both a real-time online mode and an offline mode. This RCA system has then been evaluated in comparison to another similar RCA approach in order to ascertain the effectiveness of our approach.

1.2 Thesis outline

This thesis report is structured as follows: In chapter 2 we present the necessary background to understand the work that we have done during the project. We touch upon the basics of root-cause analysis, machine learning and present the intuition behind the Self-organizing Map algorithm. In chapter 3 we go into the methodology used; more specifically we present the testbed setup and the tools used, we give an overview of the prediction and localization engine and also present the setup used for the different experiments we have performed during our work. Following that, in chapter 4 we showcase the results from the experiments presented in chapter 3. In chapter 5 we provide a discussion around the collected results based on the experimental data and the implications of those results. Finally, in chapter 6, we conclude the report and discuss some possibilities of future work that could stem from the work we have performed.
2. Background

In this chapter we present the necessary material needed to understand the problem at hand and to also understand the work we have done. We start by presenting an overview of Root-cause Analysis and machine learning and we finally, we end this chapter by providing a look into some work related to the work we have done.

2.1 Root-cause Analysis

The process of Root-cause Analysis (RCA) has been used in numerous areas and is usually concerned with finding the root causes of events with safety, health, environmental, quality, reliability, production and performance impacts. RCA is as such a tool to help the user determine what and how something occurred, but also why something went wrong [18]. Having determined these three things, an effective recommendation can then be given as to how to stop the event from occurring again.

The RCA process is usually divided into four major steps [18]:

1. Data collection.
   The first step of RCA will always be to gather the necessary data to be able to localize a fault. This could range from asking every employee involved in a workplace incident to describe their experience, to monitoring the system resources of a server. From a cloud management point of view, one of the big challenges with data collection is the decision from where the data should be collected. If data measurements are collected from the wrong place you will miss the essential data needed. However if you instead try to collect all the data you run the risk of losing the interesting part of the data in all the noise. In addition to this, the overhead of processing a lot of unnecessary data would be unwanted. Thus a good balance needs to be found.

2. Casual factor charting/Prediction.
   A causal factor chart is more or less a simple sequence diagram depicting the actions that led to an occurrence. In a computer related scenario this might be slightly harder to define and can be likened to a prediction algorithm able to detect faults when or before they occur. As such the predictions in a cloud management system should occur in real time and preferably make predictions of the future state of the system.

   When the sequence of events leading to an occurrence have been determined they in turn can be used to determine the actual root cause of the occurrence. For a cloud management system this means finding the root-cause of any prediction that indicates a fault in the system.

4. Recommendation generation and implementation.
   When the root cause of an occurrence has been determined, recommendations on how to minimize or completely remove the cause of the occurrence can be made. The recommendation and method vary with the system that has been analyzed. In a cloud management system the optimal solution would be to point to any possible system metrics or components that could facilitate the maintenance of the system.

2.2 Machine learning

Machine learning is generally divided into three major paradigms: unsupervised learning, supervised learning and reinforcement learning. Within these paradigms we find algorithms and
models such as artificial neural networks (NN), clustering, support vector machines (SVM) and more. Furthermore, additional techniques used for solving optimization problems such as evolutionary computation and swarm intelligence can also be found under the umbrella of machine learning [3].

Supervised learning aims to learn by observing data where each sample in the data has been labeled to show what that specific sample is supposed to be, the correct value of the sample to be more precise. A model will be trained with the data provided, but when evaluating the data it won’t have access to the labels, and the output of the model will then be compared to the expected output from the label. From this a training error can be derived. The goal of supervised learning is thus to minimize this training error.

Unsupervised learning aims to learn by discovering patterns in the training data without any assistance from external sources such as pre-labeled data and the like. Common unsupervised learning algorithms often include some form of clustering where similar data samples will be clustered together. These clusters can then be used to classify new data by looking at which cluster they are closest to when inserted into the model.

Reinforcement learning focuses on training an agent in an environment by rewarding good behavior and penalising bad behavior. By looking at the impact the agents actions has on the defined environment and the reward received for its actions, the agent can be trained by interaction with the environment.

As mentioned, these three major paradigms are comprised of numerous different algorithms and models. Some of the more popular ones are described below [12].

**Decision tree:** Decision tree learning is a supervised learning algorithm that makes use of a predictive model called a decision tree to map decisions and their possible consequences in a tree-like graph. The resulting graph can then be easily followed in order to arrive at a logical conclusion. An example of this from [28] using the passenger information from the Titanic can be seen in figure 2.1. Here a row in the dataset is a passenger and the features of the dataset are the age, sex and number of siblings/spouses (sibsp). The ground truth for the dataset will be if the passenger survived or not.

![Figure 2.1. Decision tree showing survival of passengers on the Titanic. The number below each node is the probability of that outcome and the percentage of observations in that leaf [28].](image)

**Support Vector Machines:** This is another supervised learning method that provides binary classification or multi-class classification of multidimensional data. The goal of SVM is to find a hyperplane of one dimension less than the actual data that separates the points into two classes as accurately as possible. An example of this is seen in figure 2.2. This is done by finding a hyperplane that separates the points while keeping the maximum distance from all points. SVMs have been successfully used in numerous machine learning tasks, notably among these is large-scale image classification [12].

**Artificial Neural Networks:** Neural Networks (NN) is a computational approach that aims to solve problems in the same way as the human brain [26]. This is accomplished by modeling
several layers of interconnected nodes (or neurons as they are called). The NN consists of an input layer, an output layer and one or more hidden layers as seen in figure 2.3. Each node in a layer is connected to each node in the adjacent layers by a connection (called weight) represented by a number. When data is fed through the network via the input unit the data will be multiplied with the weight for each neuron in the hidden layer. If the sum of all connected units to a hidden unit exceeds a threshold that unit will fire and trigger the units in the next layer [30]. In addition to this each node also contains an activation function (such as a sigmoid) to introduce non-linearity. NN is widely used in a myriad of different areas, one notable area is as an intelligent flight control system by NASA [21].

Clustering: Clustering is an unsupervised learning algorithm that focuses on grouping objects in clusters, so that each cluster consists of objects which have similar properties [27]. There are numerous clustering algorithms all with their own strengths and weaknesses so listing them all here would not be realistic. One of the more well-known algorithms is centroid-based clustering or k-means clustering as it is more commonly know. This clustering algorithm works by creating $k$ centroids, one for each cluster. Each point of data is then associated with the centroid that is closest to it (by Euclidean distance). Each centroid is then moved to the mean point of all data points associated with that cluster. This is repeated until convergence is reached [29]. Clustering algorithms are commonly used in vector quantization.

2.3 Related work
The field of anomaly detection and root-cause analysis (RCA) has been growing over the last couple of years and will continue to attract research attention in the future, especially with regards to cloud services [7].
This section is devoted to reviewing major contributions that have been made in the field of RCA, focusing on contributions that have used a machine learning or statistical learning methods. Machine learning approaches to RCA generally falls into two branches: supervised or unsupervised learning. The choice between the two branches usually depends on if it is possible to label the available data or not. The choice between supervised and unsupervised learning also depends on the needs of the system in question. Supervised learning methods usually perform better on known anomalies, but they lack the ability to detect new anomalies. On the other hand, unsupervised learning methods are able to detect anomalies that were not present in the training phase and as such are more suited to systems where the anomalies are not known during the design process. These two branches might also be combined in order to reap the benefits of both supervised and unsupervised learning. This is sometimes referred to as semi-supervised learning.

2.3.1 Supervised Learning

In [10], Jung et. al. provide an approach for automated monitoring analysis. In their paper, they highlight the importance of staging in distributed n-tier applications and point out that, if done early in a development process, it can provide crucial feedback with regards to the performance of the service under development. In their paper, Jung et. al. present a decision tree based machine learning approach to automated bottleneck detection. By looking at the utilization of the different service metrics, more specifically the change in utilization with increased workload, Jung et. al. are able to filter out service metrics that might, initially, seem to have a big impact on system’s performance, but is in fact not relevant to a potential bottleneck. With these filtered service metrics, Jung et. al. train a decision tree based classifier that is able to predict to which degree a service level objective (SLO) is fulfilled.

2.3.2 Unsupervised Learning

In [2], Dean et. al. present an Unsupervised Behavior Learning (UBL) system for predicting and preventing anomalies. UBL utilizes a Self Organizing Map (SOM) that is only trained with normal, non-anomalous, data. During the training phase, each input vector is presented to the SOM, the closest node to the input vector is found using euclidean distance and the weights of the winning node is updated so that the node moves closer to the input vector. In addition to this, each neighboring node, decided by a neighborhood function, also has its weight updated to move closer to the input vector, albeit to a lesser extent than the winning node. The effect of this training is that nodes that are close to normal data will be updated frequently and thus move closer together. On the other hand, the distance to nodes that do not correspond to normal data will be further apart from the other nodes. By leveraging the distance between neighbors, UBL is able to classify an input vector as normal or anomalous by looking at the total distance to neighbors. If the distance is small, the input corresponds to a normal state, and if the distance is large it corresponds to a failure state. Dean et. al. is able to show that UBL provides a good prediction accuracy for various anomalies in a testbed environment.

CloudPD [20] is an unsupervised fault management framework designed by Sharma et. al. and is designed in order to solve several of the challenges that is present in a cloud environment when performing RCA. More specifically, Sharma et. al. highlight shared resources, the elasticity of the cloud (VM migration, resizing and cloning) and autonomic changes to the workload requirements, as the major challenges when performing fault detection in cloud based systems. By constructing a framework that combines both simple and correlation-based models, Sharma et. al. are able to develop a multi-layered system that finds a good balance between prediction accuracy and time and system complexity.

In [6][23], two similar approaches to anomaly detection is presented. Both approaches rely on Local Outlier Factor (LOF). LOF are similar to k-Nearest Neighbor, but also factors in the local density of a measurement in regards to its neighbors. A point with much lower local density
than its neighbors is considered an outlier (anomalous). However, aside from the similar machine learning models used, the two papers differ in the way they attack the problem of anomaly detection in the cloud. The most notable difference is the amount of overhead work that is required in addition to LOF calculation. In [23], Wang et. al. utilize Principle Component Analysis (PCA), clustering and recognition in order to divide the data into workload patterns, and are then able to detect anomalies in those patterns using LOF without the need to model correlation. On the other hand, Huang et. al. [6] forgo the preprocessing done in [23] and instead opt for an adaptive knowledge base that is constantly updated with the behavior of the system. By comparing the LOF of each new point to the anomaly information in the knowledge base, Huang et. al. are able to both predict known anomalies and, due to the constantly updating knowledge base, identify new anomalies.

2.3.3 Other approaches

In [31], Yanggratoke et. al. pursue a statistical learning approach using several regression methods in order to predict the service level metrics of a video streaming service. With their work, Yanggratoke et. al. make two main contributions. First, they provide a learning method that is able to accurately predict service metrics in a cluster based service. Their learning method is proven to work both in batch learning scenarios and in an online fashion. Secondly, they showcase the importance of feature reduction for reducing computation time and improving accuracy of a learning algorithm.

In [9], Johnsson, Meirosu and Flinta present a novel algorithm for localizing performance degradations in packet networks. By applying an algorithm based on discrete state-space particle filters, they are able to provide an effective algorithm that is able to quickly and automatically identify the location of performance degradations.
<table>
<thead>
<tr>
<th>Work</th>
<th>ML Branch</th>
<th>ML Model</th>
<th>Prediction</th>
<th>Localization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dean et. al. [2]</td>
<td>Unsupervised</td>
<td>SOM</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Jung et. al. [10]</td>
<td>Supervised</td>
<td>Decision Tree</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Sharma et. al. [20]</td>
<td>Unsupervised</td>
<td>kNN, Hidden Markov Models, K-Means Clustering</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Huang et. al. [6]</td>
<td>Unsupervised</td>
<td>Local Outlier Factor(LOF)</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Yanggarote et. al. [31]</td>
<td>Statistical</td>
<td>Regression Analysis</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Johnsson et. al. [9]</td>
<td>Other</td>
<td>Novel algorithm</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Ahmed et. al. [1]</td>
<td>Supervised</td>
<td>Winnow algorithm</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Wang et. al. [23]</td>
<td>Unsupervised</td>
<td>LOF</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

In the above table, prediction refers to if the work done was presenting a method for detecting faults, while localization refers to if the work was presenting a method for localizing faults.
3. The Self Organizing Map

The Self-Organizing Map (SOM) is an unsupervised learning technique introduced in early 1981 by the Finnish professor Teuvo Kohonen. SOMs are closely related to artificial neural networks and was originally designed as a viable alternative to traditional neural network architectures [11]. One big strength of SOMs is their ability to represent high-dimensionality data in a low-dimensionality view, while preserving the topological properties of the data; this makes SOMs very powerful for providing visualizations of complex data. SOMs are frequently used in pattern and speech recognition applications due to its ability to capture properties of input data without any labeling or other aids [11].

Traditionally, a SOM is represented as a rectangular or hexagonal grid in two or three dimensions (see figure 3.1). However the SOM is by no means limited to these configurations.

![Figure 3.1. A simple Self-Organizing Map](image)

3.1 Learning Algorithm

The training of a SOM is done in several steps; these steps are then repeated over numerous iterations. The steps required to train a SOM is as follows:

1. Initialize the weights of each node in the SOM.
   
   *In order to provide an initial map to use for training the SOM, each node in the map needs to be initialized. There are numerous ways this can be accomplished, and one of the more common and simple ways is to assign random values to each weight; these random values are bound by the range of the input values.*

2. Present a randomly chosen vector from the training set to the SOM.

3. Find the node in the entire SOM that most closely resembles the input vector. This node is referred to as the Best Matching Unit (BMU).

   *Measuring distance from one input vector to a node in the SOM is usually done by finding the Euclidean Distance between the two points, given by the following function:*

   \[
   d = \sqrt{\sum_{i=0}^{n} (V_i - W_i)}
   \]  

   *(3.1)*

   *where n is the number of features in the input data, V is the current input vector and W the weight vector of the node.*

4. Find each node that belongs to the neighborhood of the BMU; the neighborhood is defined by the neighborhood function.
Kohonen’s SOM utilizes a neighborhood function to decide which nodes belong to the neighborhood of the found BMU. This neighborhood function is often represented as the radial distance between the coordinates of two nodes. One important feature of the neighborhood function of a SOM is that it should be decreasing with each time-step in order to allow the SOM to reach convergence.

5. For each node found in step 4, update the weights so that the nodes more closely resemble the input vector. Weights of the BMU are updated the most and the factor of update for the other nodes is dependent on how close they are to the BMU.

When updating the weights of each node the following function is used:

\[
W(t + 1) = W(t) + \eta(t) \ast N_c \ast (V(t) - W(t))
\] (3.2)

where \(\eta(t)\) is the learning rate and \(N_c\) is the neighborhood function centered on the node \(c\). \(N_c\) will have a value between 1 and 0 depending on how far away from node \(c\) the node to be update is.

6. Repeat steps 2-5 \(N\) times, where \(N\) is number of iterations chosen.

3.1.1 Self-Organizing Map Variations

As previously mentioned, the Self-Organizing Map is traditionally an unsupervised learning method. However, there have also been numerous variations on the popular SOM algorithm that instead utilize a supervised learning method.

In this project, two different SOM implementations have been designed and compared against each other. The first SOM is an unsupervised version originally proposed by Dean et al [2], while the other is a supervised version originally proposed by Ron Wehrens [14]. Both SOMs have been implemented in the programming language R and uses the Kohonen package to provide SOM functionality [25].

From this point on we take a step away from using the terms unsupervised and supervised when discussing the two different SOMs used in the project. This is due to the fact that both versions contain aspects of both supervised and unsupervised learning and therefore we refer to them by the structural properties of the map instead. The map by Dean et al will be referred to as a 1-Layered SOM, while the map by Wehrens will be referred to as a 2-Layered SOM.

**1-Layered SOM**

The 1-layered SOM works in a very similar fashion to the original SOM described in chapter 2, in that it uses only a single map for all the data and uses the same exact formulas for updating the weights as in a traditional SOM. The main difference here is that the 1-layered SOM will evaluate the performance of the map during training in order to present the best possible map after training. This is done by splitting the data (containing only non-faulty samples) into K-folds and present K-1 folds to the map as training data; this is commonly know as K-fold cross validation. When the map has been trained, a neighbourhood distance is calculated for each node, in the map and then each node is classified as either a faulty node or a non-faulty node by a pre-set threshold on what constitutes a faulty node or not. After this, the classified map is presented the data that was not used for training and each sample in the test data is mapped to the map. By looking at where each sample is mapped, an accuracy can be calculated. For a perfect map each sample would be mapped to a non-faulty node since the data only contains samples without faults. This process is repeated K times, once for each fold in the data. This will give K trained maps from which the one with the highest accuracy is selected.

After the map has been trained, it can be used to predict the outcome of new samples mapped to the map. This is done as follows:

1. Present a new sample \(X = [x_1, x_2, \ldots, x_n]\) to the trained SOM.
2. Compare \(X\) to the weight vector \(W_i = [w_1, w_2, \ldots, w_n]\) of each node in the trained map. The best matching node \(M_i\) to \(X\) is found using the Euclidean distance:

\[
M_i = \min_i \sqrt{\sum_{i=1}^{n}(x_i - w_i)^2}
\]
3. Calculate the neighbourhood size $S$ of the node $M_i$ that $X$ is mapped to. $S(M_i)$ is calculated as the sum of the Manhattan distance $D$ between $M_i$ and the nodes adjacent to it $M_T, M_B, M_L, M_R$. Thus $D(M_i, M_j) = |W_i - W_j|$ and $S(M_i) = \sum_{Y \in \{M_T, M_B, M_L, M_R\}} D(M_i, Y)$.

4. Compare $S$ to a pre-set threshold $T_S$. If $S \geq T_S$ then $X$ is predicted as faulty (SLA violation), otherwise $X$ is predicted as healthy (non-SLA violation).

### 2-Layered SOM

The 2-layered SOM is also similar to the traditional SOM, but with some key differences. The main difference here lies in that the 2-layered SOM contains more than one map, two maps to be exact, one for the $X$-features and one for the $Y$-features (class labels). During training of the 2-layered SOM, the distance from each new sample to each node is calculated by finding the shortest combined distance to both layers. This is done by first calculating the distance to each node on the $X$ layer using only the $X$-features of the input sample, and then calculating the distance to each node on the $Y$ layer using only the $Y$-features. The combined distance is then the weighted sum of those two distances. The weight is a value between 0 and 1 (commonly referred to as the $x$-weight) and is pre-set by the user before training. The $x$-weight is a measure of how much of the distance should be taken from the $X$-layer. As an example, an $x$-weight of 0.7 would mean 70% of the distance of the $X$-layer and 30% of the $Y$-layer is used when combining the distances.

After the map has been trained it can be used to predict the outcome of new samples mapped to the map. This is done as follows:

1. Present a new sample $X = [x_1, x_2, \ldots, x_n]$ to the trained SOM.
2. Compare $X$ to the weight vector $W_i = [w_1, w_2, \ldots, w_n]$ of each node in the trained map. The best matching node $M$ to $X$ is found using the Euclidean distance $M = \min_i \sqrt{\sum_{i=1}^{n} (x_i - w_i)^2}$.
3. Determine the predicted value of $X$ by looking at the value of best matching unit. $X$ is predicted to have the same value as that node, either non-healthy (SLA violation) or healthy (non-SLA violation).
4. Methodology

This chapter is dedicated to describing the approaches we took to design and implement the experiments that we performed. We touch upon each facet of our experiment setup and provide explanations of why we made the choices that we did.

4.1 Testbed

In order to provide a sufficient environment to perform and evaluate experiments, a testbed has been designed as a part of this thesis project. The testbed consists of three major components (seen in figure 4.1) and have been built to replicate a video-on-demand (VoD) cloud service. The testbed is built upon the work done in [31] and has been expanded to include additional functionality.

![Figure 4.1. The testbed setup](image)

**Host machine:** The host machine acts as a platform for the VoD service and is responsible for spawning and maintaining virtualized containers, which is done using Docker [8]. Each of these containers provide a server to which clients can connect, request and stream videos. The host machine is also responsible for monitoring the service metrics of both itself and of each container that has been spawned. These metrics include, but are not limited to, things such as CPU, memory, IO operations, network and more.

**Load generator:** The load generator, as the name suggests, is responsible for generating and maintaining a load towards the VoD servers, in order to simulate different work-loads on the server. In addition to this, the load generator is responsible for scheduling and executing fault injection into the host machine in order to simulate faults that occur on the server side of the service.

**Client machine:** The client machine is used to initiate a connection to one of the VoD servers and then continuously stream videos for the duration of the connection. The client machine is also responsible for collecting client-side statistics for the session, such as display frames, audio buffer rate and more.

4.1.1 Tools

**Stress [24]:** A simple stress testing tool for POSIX systems, written in C. Stress allows for putting a configurable stress on a system by imposing different resource hogs. The resource hogs that are available to the stress tool are CPU, memory, I/O and disk hogs.
System Activity Report (SAR) [17]: A system monitoring tool for Linux provided in the “sysstat” package. SAR allows for tracking and reporting on different system loads such as, but not limited to, CPU, memory, network and I/O operations. SAR also allows for exporting the results of a monitoring session to a csv file, which in turn allows for easy generation of data traces for a system that can be used for data analytics and machine learning.

VLC media player [22]: An open source media player that can be set up as a media server with streaming capabilities and as a video client that can connect to a server and stream video. The VLC client used in this project has also been modified to allow for the gathering of service level metrics such as display frames per second, audio buffer rate, number of lost frames, among others.

Docker [8]: A tool which allows for easy creation and deployment of applications using virtualized containers. This allows for developers to ship applications with all libraries and dependencies needed in a complete package and guarantees that the application will run on any Linux system [16]. For this project, Docker provided an excellent way of creating multiple instances of a containerized video streaming service.

4.1.2 Usage scenarios
In order to for the testbed to be able to simulate a real world VoD service, several different load scenarios and fault scenarios have been designed in order to mirror common usage patterns and problems.

Load scenarios
Constant load: This load pattern has a fixed amount of clients that connect to one or more media servers. Once connected, the clients start requesting videos for streaming and after a video has finished streaming, a new video is requested. This process is repeated for the entire duration of the experiment. This load scenario might not reliably mirror a real world load, since a constant load on a VoD service is not very probable. However, this load scenario provides an excellent baseline for fault prediction and can also be used for debugging and testing the system.

Periodic load: This load pattern allows for clients requests to arrive following a Poisson process. The arrival rate of the clients changes according to a sinusoid function. This more closely resembles the usage of an actual service, since there will usually be a peak sometime during the day, and be significantly less during other parts of the day, much like a sinusoid curve. The periodic load pattern allows the testbed to simulate the effects of faults in the system under both high and low load.

Fault scenarios
A number of fault scenarios have been developed, though they differ in how often and with what probability they inject faults. All scenarios are able to inject the same three faults, which is CPU hog, memory hog and disk hog.

Probabilistic injection: This fault scenario is built around the idea that every fault has some probability of occurring in a system during a specific time window. The fault injection has been modeled after a binomial distribution. This is accomplished by generating a random number every n time units (seconds, minutes, hours), and if that number is lower than a predefined fault probability, then a fault will be injected into the system for a predefined time period.

Spike injection: This fault scenario has been built in order to provide increased control over the fault injection, as opposed to the fault scenario above. Instead of injecting faults based on a defined probability, the user will instead specify when they want the fault injected and for how long. This scenario will only inject a single fault spike.
4.2 Anomaly detection

The first step of performing root-cause analysis (RCA) in a cloud environment, or in any environment for that matter, is the anomaly detection. Before the data can be analyzed, and the process of finding the root-cause of a service degradation can be started, the system must first be able to detect that an actual anomaly is present. Two different variations of the self-organizing map have been chosen as the prediction engine in the RCA system and both variations have been compared against each other in order to ascertain which provides the most accurate prediction. The key differences between these two engines have been described in chapter 3.

4.3 Fault localization

After the system has ascertained that a fault/service degradation is present in the host machine, the process of determining the root-cause of that service degradation can begin. For this RCA system, a dissimilarity measurement scheme has been implemented. This dissimilarity measurement scheme has not been proven, but is rather a proposed method that has shown promise which will be shown in the results of this report. The localization scheme is described in detail below.

4.3.1 Dissimilarity measurement scheme

Dissimilarity measurement is quite simple in practice and is rooted in the fact that when you compare a healthy sample to fulfill your Service Level Agreement (SLA) to a sample that does not fulfill the SLA, you can observe the differences in the features and the features that have the largest difference, or dissimilarity, likely to be the cause of SLA violation.

The SOM has an inherent property, that during training the weight vectors of nodes connected to each other will be updated to more closely resemble each other. This leads to the nodes clustering closer together and it is this property that makes the SOM well suited for the dissimilarity measurement scheme.

1. Present a new sample \(X = [x_1, x_2, \ldots, x_n]\) to the trained SOM.
2. Compare \(X\) to the weight vector \(W_i = [w_1, w_2, \ldots, w_n]\) of each node in the trained map. The best matching node \(M\) to \(X\) is found using the Euclidean distance \(M = \min_i \sqrt{\sum_{i=1}^{n} (x_i - w_i)^2}\)
3. Determine the predicted value of \(X\) by looking at the value of \(M\). \(X\) is predicted to have the same value as \(M\), either non-healthy (SLA violation) or healthy (non SLA violation).
4. If \(X\) is mapped to a healthy node, i.e. predicted as healthy, then return to step 1. Otherwise proceed to step 5.
5. Locate the \(N\) nearest healthy nodes to \(X\) using Manhattan distance.
6. Calculate the dissimilarity vector \(DS_{tot}\) for \(X\) where \(DS(X, N_i) = [|x_1 - w_{1,i}|, |x_2 - w_{2,i}|, \ldots, |x_1 - w_{n,i}|]\) and \(DS_{tot} = \sum_{i=1}^{n} DS(X, N_i)\).

The dissimilarity vector can then be used to interpret the nature of the fault present in the sample \(X = [x_1, x_2, \ldots, x_n]\). Since each element in the dissimilarity vector \(DS_{tot}\) corresponds to a feature \(x_i\) in \(X\). By sorting \(DS_{tot}\) in descending order, one receives a ranking list where the top ranked results are more likely to be the cause of the detected fault.
4.4 Experiment setup

4.4.1 Data trace collection

During the course of the project, several data traces have been generated in order to ascertain the performance of the VoD service under different scenarios. Each of the traces in table 4.1 ran for a total of 10 hours which amounted to approximately 36000 samples for each trace. Every 30 seconds, a fault was injected with a probability equal to 0.2 (unless a fault was already present) and ran for the specified fault duration. In case several faults were present in the scenario, such as for trace #4 and #6, then the type of fault was selected at random with equal probability for each fault type.

Table 4.1. Data traces collected

<table>
<thead>
<tr>
<th>#</th>
<th>Load Pattern</th>
<th>Fault injected</th>
<th>Fault duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Constant</td>
<td>CPU</td>
<td>45s</td>
</tr>
<tr>
<td>2</td>
<td>Constant</td>
<td>Memory</td>
<td>45s</td>
</tr>
<tr>
<td>3</td>
<td>Constant</td>
<td>I/O</td>
<td>45s</td>
</tr>
<tr>
<td>4</td>
<td>Constant</td>
<td>CPU+Mem+I/O</td>
<td>45s*</td>
</tr>
<tr>
<td>5</td>
<td>Periodic</td>
<td>CPU</td>
<td>30s*</td>
</tr>
<tr>
<td>6</td>
<td>Periodic</td>
<td>CPU+Mem+I/O</td>
<td>30s*</td>
</tr>
</tbody>
</table>

* For each injected fault, duration is taken from a Gaussian distribution with the provided duration as mean and 8s as standard deviation.

Feature selection

The original data traces contain a large number of features, 648 to be exact. Due to this, there was a need to perform some feature selection in order to minimize the feature set to a more manageable number. The feature selection was done using domain knowledge of the problem at hand, which brought down the feature set to the 14 features below. This feature selection set has been proven to be effective in [1] and has been chosen with this in mind.

Table 4.2. Minimal feature set

<table>
<thead>
<tr>
<th>Memory [kB]</th>
<th>CPU [%]</th>
<th>I/O [per sec.]</th>
<th>Block I/O [per sec.]</th>
<th>Network [per sec.]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory used</td>
<td>CPU host*</td>
<td>Total Trans.</td>
<td>Blocks read</td>
<td>Received packets</td>
</tr>
<tr>
<td>Memory committed</td>
<td>CPU container*</td>
<td>Bytes read</td>
<td>Blocks written</td>
<td>Transmitted packets</td>
</tr>
<tr>
<td>Memory swap used</td>
<td></td>
<td>Bytes written</td>
<td></td>
<td>Received data(kB)</td>
</tr>
</tbody>
</table>

* The stats for utilization by user, system, and wait have been summed to one feature.

All features presented in table 4.2 were collected at host level, except for CPU container that was collected from the video streaming service container that the client machine was connected to.

Data normalization

Due to the nature of SOMs and how data is mapped to each node, the SOM experience problems when training the map with features of wildly varying ranges[2]. Since the servers we are running have system metrics that express both CPU utilization in the range [0,100] and, for example memory used in the range [0,300000], steps had to be taken in order to improve the training
process of the SOM. The two choices generally taken in order to combat this is either to increase
the size of the map, or to normalize the features to a range between either [0,1] or [0,100][2].
Since increasing the size of the map will also lead longer training times, and does not necessarily
improve the accuracy of the map we opted to go with feature normalization.

In order to normalize the feature data to the range [0,1], the following formula was applied to
each feature in the dataset:

\[ z_i = \frac{x_i - \min(x)}{\max(x) - \min(x)} \]  

(4.1)

where \( x = [x_1, x_2, \ldots, x_n] \) and \( z_i \) is the \( i^{th} \) normalized data.

4.4.2 Data smoothing

When designing a video streaming service, one common feature that is often included is video
buffering. This allows for parts of the video to be temporarily stored in the client’s memory, to
mitigate performance problems that can occur in both the host machine and the network connec-
tion. This is a behaviour that helps provide a more stable service to the client. However, it also
complicates the anomaly detection process. As seen in figure 4.2, the introduction of a fault (blue
line) does not immediately lead to an SLA violation being registered by the client (red line).

To help combat the challenges introduced by the data buffering, we implemented data smooth-
ing to make the actual SLA violation easier to detect. This was done by having each sample be
the mean of 15 previous samples that came before it.

![Figure 4.2. The effects of injected faults with respect to registered SLA violation. Teal is the injected fault and red is if a fault is registered at the client machine.](image)

4.4.3 Sensitivity Study

When configuring the SOM for training, there are several parameters that affect both training
time and accuracy of the resulting SOM. In order to ascertain the optimal parameters, a sensitivity
study was performed on both the 1-layered SOM and the 2-layered SOM.

One major factor for the performance of both 1-layered and 2-layered maps was the actual
size of the map and the optimal parameter needed to be found. This was done by repeatedly
presenting the same data to the maps several times and only varying the size of the map while all
other parameters were kept fixed. The same random seed was also used in order to make sure any
pseudo-random elements of the system (such as initialization of the map) were exactly the same
for each experiment.

For the 1-layered SOM, the other major factor that affected performance the neighbourhood
distance that is pre-configured before running the training process. For the 2-layered SOM, the
other major factor was the weight given to the X-layer; by varying this, the performance of the map
varied as well. In order to find the best possible parameter settings for the problem at hand, the
same experiment was repeated numerous times, while only changing the neighbourhood distance
or X-layer weight between each experiments.
The performance of the maps were then evaluated by using ROC (Receiver operating characteristic) curves. The best performing map is the one that achieves the highest area under the curve (AUC).

4.4.4 Training and Prediction
Since the data traces collected were heavily biased towards non-faulty data, the classification accuracy (CA) commonly used when presenting the accuracy of a machine learning method was not sufficient. Therefore there was a need for an accuracy that more accurately reflected the performance of the SOM. For this purpose we consider both the CA and balanced accuracy (BA) when evaluating the performance of the SOM.

\[
CA = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Test Samples}} \quad (4.2)
\]

\[
BA = \frac{TPR + TNR}{2} \quad (4.3)
\]

where,

\[
TPR = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}
\]

\[
TNR = \frac{\text{True Negatives}}{\text{True Negatives} + \text{False Positives}}
\]

In order to evaluate the system with regards to training and prediction performance, two different types of maps were evaluated. The first was a specialized map trained to detect one specific type of fault (such as CPU, memory, I/O etc.) and only be subjected to data containing that type of fault during testing. The other map was a more general map that was subjected to many different types of faults during training. Both maps are described in more detail below.

Comparing the specialized map to the generalized map is of particular interest since the intuition is that a specialized map will provide better prediction performance for the specific fault the map is specialized towards but not be able to accurately detect other faults. On the other hand, the generalized map would likely perform better on average for data containing several types of faults, but would not reach the same performance for data containing just one type of faults. The generalized map is also more desirable from a real-world scenario, since training just one map for an entire system is more efficient than training one for each type of expected fault in the system.

**Specialized map**
The specialized map is a map where the training data and test data used contain the same type of fault. The map is trained using one set of trace data containing fault $F$, and then another set of trace data containing only the same fault $F$ is used to evaluate the prediction performance. Each sample in the second trace is presented to the SOM and resulting prediction is compared to the ground truth of that sample in order to determine if a correct prediction was made.

**Generalized map**
In contrast to the specialized map, the generalized map is a map that has been trained to distinguish several faults. During training, a data trace containing varied faults is used to train the map. After training, the non-specialized map can be evaluated in two ways. Either the map is fed a data trace containing only one type of fault and thus the ability of the generalized map to detect that specific fault is evaluated, or the map is fed a data trace containing varied faults in which case the ability of the map to detect faults in general is evaluated.
4.4.5 Localization

In order to evaluate the localization performance of the developed localization engine, the same approach for localization described in section 4.3.1 was used on a data trace that was not used for training the map. However, in order to avoid running the localization engine on outliers, alarm filtering was implemented.

Alarm filtering
Due to the inherent nature of the data and due to the fact that it was collected from a live system introduced certain outliers in the system. These outliers would manifest as a one-second SLA violation on the client side of the testbed, even though no fault was present on the service side. To avoid running the localization engine for these outliers, a simple alarm filtering method was used.

For the system to start the localization engine, there must be enough samples in a short amount of time that are predicted as SLA violations. This is achieved by implementing a pre-set alarm threshold and an alarm counter. Each time a sample is predicted to be a SLA violation, and no alarm is currently active, the alarm counter will increase by one. Once the alarm counter reaches the alarm threshold, the alarm will be raised and the localization engine will start. The localization engine will then run until a fixed number of consecutive non-SLA violations have been predicted. Once this set number is reached, the localization engine will be suspended until the next alarm is raised.

Evaluating the localization
The evaluation of the localization engine is done in a similar way to that of the prediction engine. The whole process is shown in figure 4.3. A trace that was not used to train the SOM is presented to the prediction engine, one sample at a time. Whenever an alarm is raised, each sample will also go through the localization engine. After applying the procedure described in section 4.3.1, we look at the resulting dissimilarity vector and consider the top three metrics. If any of the top three metrics are related to the same system group (e.g., Memory, CPU, IO, etc.) as the injected fault, we consider the localization successful. Otherwise, we consider the localization a failure. We then evaluate the performance of the localization engine based on the number of samples successfully localized out of the total number of samples presented to the system to be localized. This is defined as the localization accuracy.
Figure 4.3. The flow of the RCA system showcasing how each sample is handled.
5. Results & Discussion

In this chapter we present the results achieved from the experiment setup described in chapter 4. Below we cover three major areas that have been investigated and after that we also present a prototype demonstrator that exemplifies how it could be applied to a production environment.

In chapter 3, we stated that one of the strengths with self-organizing map was the visualization capabilities it could provide. A problem with many machine learning algorithms is that it is usually not a trivial task if you want to change something inside a trained model in order to improve the prediction accuracy. This stems from the fact that most machine learning models are black box solutions where it is often hard to understand why the trained model ended up the way it did by inspection. Tweaking the trained model is also hard due to the fact that it is problematic to see connections between input data and output data when the model is applied.

On the other hand, the SOM provides an easy to understand model that is accompanied with a nice, visual representation. This allows the trained model to be inspected and the underlying cause of any possible misclassifications can be corrected.

As an example, we have trained a SOM comprised of 10 by 10 nodes. For this SOM we have plotted the prediction mapping of the model, this mean we have plotted each node and the corresponding node classification. Finally, we have the ground truth of each sample of a test set not used for training on top of the prediction mapping. In figure 5.1 below we see the results, where red colours are connected to SLA violations and green/black colours are connected to non-SLA violations.

As we can see by studying the mapping, it is easy to see some nodes where the training process has most likely misclassified the node and this could in turn be used to improve the map further after training.

* represents a possible misclassification of a node

Figure 5.1. Prediction mapping of SOM. Ground truth of test set represented as 0 and 1.
5.1 Sensitivity study

As we described in section 4.4.3, before we started evaluating the system we needed to decide on the optimal configuration of the system. We therefore performed a sensitivity study to find the best parameters for the problem at hand.

For the 1-layered SOM, the first major parameter studied was the neighbourhood threshold, a too high threshold would mean that each node in the map would predict non-SLA violations, while a too low threshold would mean the opposite. Therefore it is important to find the optimal threshold that will provide the highest possible prediction accuracy to the map.

Since we want to find both the optimal neighbourhood size threshold and map size for the 1-layered SOM, we have repeated the same experiment for several different map sizes. For each map size we produced a ROC (receiver operating characteristic) curve where we varied the neighbourhood size threshold from 0.5 to 6 in increments of 0.25. In each case the data set consisted of data with a CPU fault present.
As we can see in figure 5.2, the optimal threshold for the neighbourhood size will decrease as the size of the map increases. Therefore, this would suggest that a low neighbourhood threshold would be preferable. This is generally true, however, the 1-layered SOM is unfortunately quite sensitive to the data used to train the map, and variations in how much the nodes cluster together is common. For example, when training a 20x20 map from a data set containing memory fault, the optimal neighbourhood threshold was 1.5, while one trained from data containing I/O faults suggested the optimal threshold was 2.5. This would mean that the system would need to be trained differently for each type of fault.

For the 2-layered SOM, we studied the weight given to the X-layer during the training. Here a low X-weight would mean that the importance of the X features from the host machine would be lessened, and instead the SLA violation data from the client would heavily affect the SOM mapping. On the other hand, with an X-weight of one, the 2-layered map will not utilize any information from the Y-layer.

Here we see that the optimal x-weight is without a doubt 1.0, meaning that no information from the Y-layer is utilized during training. Now, this might seem strange that the added layer in the
2-layered map does not provide any benefit to the model. However, bear in mind that almost all the information is present in the X feature data, while the Y features only contain the ground truth of each sample. Despite these sensitivity results, there is value in allowing for some information from the Y-layer to be used, which we will show when we present our localization results.

If we look at what would be the optimal size, it is clear to see that the 5x5 map provides the best performance judged by looking at the area under the curve. This is true for both the 1-layered SOM and the 2-layered SOM. However, while this is true from a pure prediction accuracy perspective, we will later show that this is not true from a localization perspective where the larger sized maps will allow for distinguishing between similar samples.
5.2 Training and Prediction

One major part of the system that we have designed during this project is the prediction engine. Therefore, the system has been evaluated from several key points. The results of these evaluations are presented below.

5.2.1 Training time

An important consideration when evaluating the potential value of a system that will run in a data centre environment is scalability. The amount of data that needs to be processed is enormous, and as such the, potential RCA system needs to be able to scale in a satisfactory manner. For a Self Organizing Map, one way to scale the ability of the map is to increase the number of nodes of the map. This allows for the map to detect more varied trends in the, data and by finding a good balance between the number of nodes in the map and the data available, one can design a map with optimal performance.

The training time of self organizing map can be broken down in two different steps. The first step is the actual training step, as described in section 3.1, and this step is performed once for every K-fold defined in the cross validation. The total training time taken is the combined time for the training of each K-fold. The second step is the selection step, where the best map out of all the maps trained in the previous step is selected. The selection time is equal to the combined time taken for each K-fold to evaluate a provided set of test data.

Since we use K-fold cross validation in our system, the training set is comprised of 2/3 of the data and the test set is comprised 1/3 of the data.

![Training and selection time for a SOM.](image)

Figure 5.4. Training and selection time for a SOM.

In figure 5.4, we see that there is not much difference in the training time for the 1-layered and 2-layered map (a); the 1-layered has a slightly faster training time compared to the 2-layered map. The increased time taken for the 2-layered map can be attributed to the fact that during the training process the 2-layered map needs to compare each sample to both the X layer and the Y layer of the map.

When looking at the time taken to select the map with the performance performance, we see in (b) that the 2-layered map scales with linear time and clearly outperforms the 1-layered map that scales with closer to quadratic time as the number of nodes in the map increases. This behaviour is due to the neighbourhood calculation needed by the 1-layered map. In the 2-layered map, information is stored with each node during training as to what value is associated with that node.
(in our case SLA violation or not). However, in the 1-layered map the prediction of a node is decided by the neighbourhood size of that specific node, thus after the map has been trained we must go through each node and determine the neighbourhood size in order to decide what that node should predict. This adds a fair bit of complexity to the selection process.

![Figure 5.5. Total time taken for both training and selection.](image)

By looking at the combined time taken, as seen in figure 5.5, we see that with up to 400 nodes in the map, both methods perform about the same with the 1-layered map being slightly faster. However, after 400 nodes the 2-layered map starts to outperform the 1-layered map and it is clear to see that when looking for a map that will scale well when increasing the size of the map, the 2-layered map is the prime choice.

5.2.2 1-layered SOM vs 2-layered SOM

As a part of this project we have evaluated our novel 2-layered SOM approach to RCA against the 1-layered SOM implemented in [2]. Our initial sensitivity study showed that the 2-layered map performed better than the 1-layered map, and in this section we expand upon that initial finding. By looking at the ROC curves in section 5.1 in the same graph, this initial finding becomes even clearer.

In order to delve deeper into this finding, we have analyzed the performance of each map for the different data traces presented in table 4.1. For the 1-layered map we used a neighbourhood threshold of 1.25 and for the 2-layered map we used an x-weight of 1.0. Both maps were of size 20x20 nodes. The results from the experiment can be seen below in table 5.1.

<table>
<thead>
<tr>
<th>Trace</th>
<th>1-layered SOM</th>
<th>2-layered SOM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CA</td>
<td>BA</td>
</tr>
<tr>
<td>CPU</td>
<td>0.725</td>
<td>0.605</td>
</tr>
<tr>
<td>Memory</td>
<td>0.818</td>
<td>0.601</td>
</tr>
<tr>
<td>I/O</td>
<td>0.695</td>
<td>0.543</td>
</tr>
<tr>
<td>All</td>
<td>0.756</td>
<td>0.597</td>
</tr>
</tbody>
</table>

By comparing the results we can see that it is quite clear that our 2-layered map achieves better prediction accuracy for each data trace compared to the 1-layered map.
5.2.3 Specialized vs. Generalized map

In order to compare the performance of the specialized map against the generalized map, we train one specialized map for each type of fault (CPU, memory and I/O). We then look at how capable the map is at detecting faults of the same type that the map was trained for. This is done by feeding the map a test set containing just one type of fault, that corresponds to the specialization, and observing the detection rate.

For the generalized map, we train just one map with a data set containing all types of faults, and then evaluate the map in the same manner as the specialized map.

Table 5.2. Comparison of specialized map and generalized map with regards to prediction accuracy.

<table>
<thead>
<tr>
<th>Trace</th>
<th>Specialized</th>
<th>Generalized</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA</td>
<td>0.924</td>
<td>0.918</td>
</tr>
<tr>
<td>BA</td>
<td>0.749</td>
<td>0.765</td>
</tr>
<tr>
<td>CA</td>
<td>0.898</td>
<td>0.892</td>
</tr>
<tr>
<td>BA</td>
<td>0.876</td>
<td>0.873</td>
</tr>
<tr>
<td>CA</td>
<td>0.878</td>
<td>0.873</td>
</tr>
<tr>
<td>BA</td>
<td>0.574</td>
<td>0.605</td>
</tr>
</tbody>
</table>

From studying table 5.2, we see that the performances of the specialized map and the generalized map is very similar and only differ between 0-3% from a balanced accuracy perspective, which is what we want to maximize. This means that a system using one map trained to recognize different faults will perform just as good, and sometimes even better, than a map specialized to recognize one type of fault. By only having to design and train one map, we also bring down the complexity of the system as opposed to training several different maps. Therefore, we can conclude that the generalized map is the optimal choice for this system.

5.3 Localization

The second major part of the system is the localization engine, and in this section we present our results and evaluations of the performance of the system with regards to localization accuracy. When evaluating the localization, we have looked at the frequency of each type of fault localized when that fault is ranked as the primary fault. A good localization result is when the injected fault type is the most frequent. We have also considered the localization accuracy which we define as:

$$loc.acc = \frac{\text{Fault Frequency}}{\text{Total number of localizations performed}}$$ (5.1)
where “Fault Frequency” is equal to the number of localizations where the injected fault is ranked as one of the top three faults.

The avid reader might have noticed that for prediction purposes, the optimal map size would be 5x5. This would provide the best prediction accuracy and also the fastest training time. However, as we briefly mentioned during the sensitivity study, this is not true for the localization process. Some system faults are by nature quite connected in how they affect the system, and if the map is of insufficient size detecting the difference between two similar faults will become very hard. For instance, if a system is experiencing a memory fault, this will have a direct impact on the cpu utilization of the system and vice versa. As an example of this we have trained a 5x5 and a 20x20 2-layered generalized map, and then presented the same test set containing CPU faults to the trained maps.

![Fault occurrences per feature (Loc. acc: 0.907)](image1)

(a) 5x5 nodes.  

![Fault occurrences per feature (Loc. acc: 0.928)](image2)

(b) 20x20 nodes.  

**Figure 5.7.** The impact of the size of a map on the localization performance.

As we can see in the figure above, the smaller map is unable to detect the CPU faults present in the data and instead misclassified them as memory faults due to the similarity of the faults. The 20x20 map on the other hand has no problem localizing the CPU faults due to the fact that the larger map contains more nodes, and is thus able to contain more subtle differences in the data.

Continuing on the same track, we have found that using an x-weight of 1.0 is not as optimal as the sensitivity study would suggest. When evaluating a test set containing memory faults, we found that a map with an x-weight of 1.0 had problems distinguishing between memory and CPU faults during localization. However, if we allowed for some information from the Y-layer to be utilized, we saw a great improvement in localization performance. Below we show a comparison between an x-weight of 1.0 and 0.9 to prove this finding.

### 5.3.1 1-layered SOM vs 2-layered SOM

Having previously shown that the 2-layered SOM performs better than the 1-layered SOM with regards to fault prediction, we now present our findings on how well both approaches perform from a localization perspective.

For the experiment we used the same exact data from both the 1-layered map and the 2-layered map. Both maps were of size 20x20 and were trained as generalized maps. For the 2-layered map we used an x-weight of 0.9 and for the 1-layered maps we used a neighbourhood threshold of either 1.25, 1.5 or 2.5, depending on the fault as specified in section 5.1.

From the results of our performance comparison, we can see that the 1-layered map struggles with accurately localizing the faults found during the evaluation. Even though the 1-layered map
is able to localize the injected fault (as in figure 5.9 and 5.11), it often misinterprets the fault as something else, and for CPU faults it suffers from the same problem as a too small map, and is unable to distinguish between CPU and memory faults.

On the other hand the 2-layered map is able to identify the injected fault as the most frequent fault in each experiment, even though it has slight problems with memory faults as seen in figure 5.9. This is probably due to the side effects of a memory fault on the system which could cause both changes in network and CPU utilization, even though the fault stems from a memory problem.

5.3.2 Specialized vs. Generalized map

In order to evaluate the localization performance of the specialized and generalized map, we apply the same approach described in section 5.2.3.

36
As we showed in section 5.2.3, the generalized map is on equal grounds with the specialized map when it comes to prediction accuracy. However, when we start looking at localization performance, we clearly see that the generalized map performs much better than the specialized map. For the specialized map, we see that it has problems detecting both memory and I/O faults. Therefore, we can conclude that the generalized map is the optimal choice from both a prediction perspective and a localization perspective.

5.3.3 Evaluating the final system

By combining the findings from the sensitivity study, the prediction evaluation and the localization evaluation above, we put together a complete system that will serve as the final product of this thesis. This system serves as the basis for the demonstrator described in section 5.4.
For this final system we have found that using a 2-layered generalized map of size 20x20 nodes with an x-weight of 0.9 provides the optimal performance for the system with regards to both prediction and localization accuracy.

In order to evaluate the system in a setting close to an actual VoD cloud service, we have used the more complex data trace (number 5) shown in table 4.1. These traces contain a periodic load pattern, as implemented in [31], where the load generator will start clients according to a Poisson process, with an arrival rate of 70 clients/minute, and then change according to a sinusoid function with a period of 60 minutes. The amplitude of the sinusoid function is set to 50 clients/minute. This gives the following load pattern over a 10 hour period.
During the 10 hour runtime of the experiment faults, were present in the system for a total of 113.8 minutes (or 6828 samples) where CPU faults stood for 38.65 minutes, memory faults stood for 36.5 minutes and I/O faults stood for the remaining 38.65 minutes. The system registered a total of 1538 samples with SLA violations, of which 1160 were caused due to injected faults in the system, and 378 were the cause of the load put on the service from the load generator or were classified as outliers. By looking at the frequency of faults localized by the system, we see the following:

From figure 5.16 we can see that the system is able to identify both CPU and memory faults in the system. However, the I/O faults are noticeably absent. By looking at the distribution of SLA violations and what caused the SLA violation, we see that out of 1160 samples with SLA violations, 460 were due to CPU faults, 671 were due to memory faults and only 29 were due to I/O faults. Further exploration into this phenomenon has shown that for the periodic load pattern, the I/O stressors that we inject experience problems with producing service degradations even as we increase the number of stressors on the system. We have been unable to determine the underlying cause of this behaviour, but we have our suspicions that it could possibly be due to I/O operations being buffered in memory and handled during periods of low-load on the system. However, this is just a theory and future work need to be performed into this behaviour in order
Figure 5.16. Fault localization frequency for a periodic load data trace containing CPU, memory and I/O faults.

to be able to provide a complete RCA system. Due to time limitations in this project this is left as an open research question.
5.4 Demonstrator

In order to showcase the potential of the complete system, a prototype demonstrator has been designed. This demonstrator presents what the system could achieve in an industry setting. The idea behind the demonstrator is to provide a dashboard that allows for real-time monitoring of different system resources along with live fault prediction and fault localization. The dashboard designed can be seen below in figure 5.17. The prototype provides a time series visualization of the current state of the system as seen in the “System resources”. Along with this, the demonstrator has showcased either the predictions made by the system, or the ground truth of SLA violations and fault injections in the system, as seen in “Fault injection & SLA Violation”. There is also a prediction accuracy measurement displayed for the last 15 minutes. Finally, the demonstrator displays any localized faults and the ranking of each system resource for that fault (found under “Fault rank”); resources with a higher ranking is more likely to be the cause of the service degradation.

The entire prototype is customizable and the user can easily change the dashboard to highlight interesting areas.

![Figure 5.17. Demonstrator example.](image)
In this thesis project we have presented a root-cause analysis system built upon self-organizing maps. This system has been tested and evaluated using a testbed which mirrors a cloud video-on-demand service. This testbed has also been designed to allow for different load simulations and fault injections into the system to provide different usage scenarios.

We have performed a study on both the underlying components of the system, as well as related research that tries to solve similar problems.

With our work we have shown that the system we have designed is able to achieve both good prediction accuracy with regards to detecting faults in the system, as well as good localization accuracy with regards to localizing any faults found. The system has also been shown to be able to handle varied faults without the need to employ differently trained maps for each specific fault.

In order to compare our system, which utilizes 2-layered maps, and to our knowledge, is the first of its kind, we have compared it to a system developed in [2]. Within this paper we are able to show that our 2-layered approach performs better than the 1-layered approach in [2] in both prediction and localization. Our 2-layered system has also been proven to scale favorably as the size of the map increases, which is valuable if the system will be deployed in a larger environment.

Finally, we have identified interesting research questions that might serve as the basis for future work done within the field of RCA.

6.1 Future work

During the course of this thesis there have been a number of topics that have sparked our interest. However, due to the time limited nature of the project these topics had to be left out of the research. In this section we present potential research topics that stem from the research we have done into the field of RCA and our thoughts on how to delve deeper into each research topic.

The system we have designed in this thesis project have the ability to predict and localize performance degradations as they happen in a system. Even though this has uses in a production environment, it is somewhat limited. For a truly useful system, we would want to be able to predict future behaviour which would allow the system to preempt the service degradation before it occurs. We have theorized that this may be possible by creating a third type of node in the SOM that will act as a pre-failure node. By defining the nodes that are located close to the border between anomalous and non-anomalous nodes as pre-failure nodes, the system might be able to predict service degradations that are about the happen ahead of time. With changes such as these to the system, it will also be of interest to evaluate the system from the perspective of lead-time. Lead-time would be a measurement of the time from when a service degradation happens, to when it actually occurs in the system, and here longer lead-time would be better. Therefore the evaluation of the system would be expanded to see if we find a good balance between prediction accuracy, localization accuracy and lead-time. Something similar has been briefly discussed in [2], although we believe there is a need to expand upon that research.

As mentioned in section 5.2.1, scalability is an important aspect of any software that will be deployed in a data centre. This thesis project has been focused on a testbed containing a single host machine and single VoD service and therefore cross server scalability has not been considered. However, for a RCA system to be truly valuable inside a large data centre, the scalability and distributability of the system need to be carefully evaluated.

With our work we have focused on evaluating data over a short period of time and during this time there have not been any significant changes in the host system. However in a real world data
centre there will be changes to both architecture, hosted data and service usage that might have a significant impact on the performance of the system. Therefore research need to be made into how long a trained map will stay accurate inside a changing system and how often the map needs to be re-trained and updated in order to stay relevant as a RCA system.
References


