SELF-OPTIMISATION IN DATA CENTERS

Submitted by:

SAM O'FLOINN

Supervisor:

PROFESSOR GREGORY PROVAN

MSc in Software and Systems for Mobile Networks

Computer Science Department
University College, Cork

September 6, 2015

Abstract

The present generation is hard pressed to get as much efficiency out of data centers as it can, especially in the cloud computing environment. Self adaptation has proven to be important to modern systems; researchers argue that a self optimisating data center would be able to deal with many kinds of datasets. Using the CloudSim data center simulator and the Naive Bayes classifier alongside Weka, this project creates an ARFF dataset of the features in a data center's tasks and virtual machines, then applies machine learning techniques to analyse data in a way that the system can better organise it.

Declaration

I confirm that, except where indicated through the proper use of citations and references, this is my original work and that I have not submitted it for any other course or degree.

Signed: _____

Sam O'Floinn
September 6, 2015

Acknowledgements

I would like to give special thanks to Prof. Gregory Provan, my supervisor, for taking me on this project. His guidance, advice and support was invaluable for this project's research.
I would also like to give special thanks to Mark Horgan and Brian Irwin for their constant support, and for their advice and their assistance in proofreading this thesis.
Lastly, I would like to thank my family for their constant support.

Dedication

To my brother, whose confidence in my ability to pursue studies in computer science set me on this path. I would not be here today without him.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 State of technology and software

Since the mid-20th century, the field of software has seen tremendous technological advancements, at a far quicker pace than many people predicted, and it is still rapidly growing today. That said, hardware has arguably accelerated at a faster pace, with storage systems being rapidly outdated; a modern USB stick, as large as a thumb, stores more memory than a supercomputer in the 1970s did. However, the rate of technological advancement is beginning to show complications on two fronts.

Firstly, the hardware front has also come to see diminishing returns - where there once were massive gains of perhaps six or three times the improvement in output from newer hardware (which was far larger than the rate of growth in software), there is now less than double the improvement in recent years. This trend suggests that future work should not rely as heavily on hardware advancements in the past as current practices have, and consider turning to software. It also suggests that software has much room to catch up, and that the next major technological advancements may be made on this front.

Second, as both software and hardware have seen rapid growth, so too has their importance in society. Many people use smartphones, which were barely a thing in just 2006, to carry out important day-to-day tasks. There are large businesses operating worldwide that run their entire careers on computers, and have databases full of terabytes of data. Data centers, critical to communication in large industries, deal with communication over the world and hundreds of different types of data.

The presence of even just Microsoft Excel and other office software has saved most businesses significant amounts of time. Because of this convenience, many businesses have tried to do much more with the time given to them, and generally try to accomplish much more, effectively pushing their software tools to their limits.

But having discovered these limitations, there is now a need and demand for the software that can surpass these.

But with these limitations being gradually approached, the problems that wait to be challenged at this state have begun to become clear. As systems like data centers deal with more numerous and more complex activities, the amount of energy they expend has risen, which is significantly increasing their energy costs and maintenance costs.

These aren't the only complications that emerge; for another example, the energy expenditure has a negative impact on the environment, adding further incentive to investigate these issues. Thirdly, and perhaps of most interest to further study, the programs that teams have worked on are simply becoming too big to manage.

Some systems have reached this level already; for instance, the manifesto "IBM's Perspective on the State of Information Technology"[**ibm**] warned that computer systems were becoming too complex to modify, requiring millions of lines of code and thousands of programmers to work on. While to create it is costly enough, modifying it becomes more infeasible as the programmers on the task move onto different roles and new programmers have to examine potentially millions of lines of code to do something. Such systems would become impossible to maintain by humans over time.

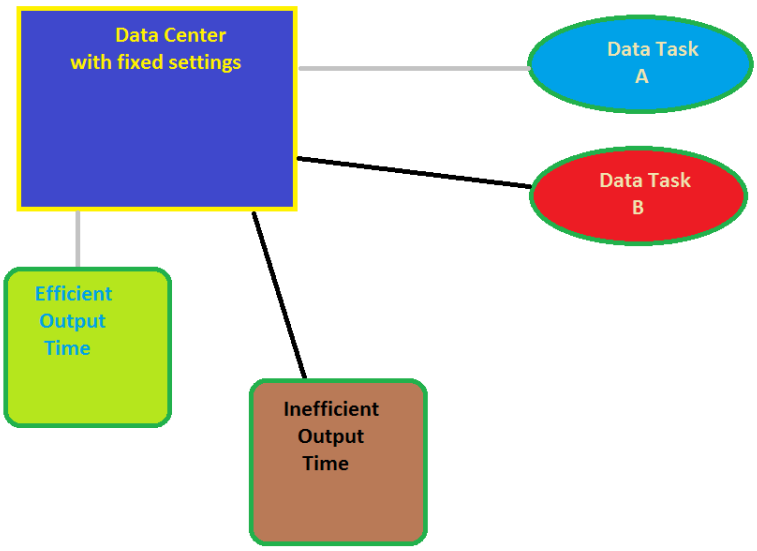


Figure 1.1: In data centers with fixed states, some tasks take longer to compute than others.

But if people cannot efficiently make these sorts of modifications, then what possibly can? The proposed candidate to make this less of a problem is the system itself. The idea may at first seem to be a sharp contradiction - a program can't do anything more than what it is scripted to do, after all - but in reality there is a field of research already dedicated to these sorts of systems. This is referred to as 'self-adaptation', where parts of software modify themselves based on some desired needs as design by the developers.

Bear in mind that this is different to suggesting programs that diverge from their scripted requirements; what is being put forward here is not to create evolutionary algorithms like those seen in AI research. Instead what is proposed is that the system already knows what sorts of elements to look out for and, based on its given understanding of them, dynamically adapt accordingly whenever it identifies such a thing. This kind of element can depend on the system itself.

This is relevant to supervised machine learning, a study where software studies a known dataset in advance and then runs a prediction how well the data would be calculated. If the system has a database, then it can use some variable like the amount of occupied space in the database to decide whether to take an action.

If the system has detection hardware like a thermometer or a camera, it can call for a change to occur if the temperature reaches a certain level or if it sees some important visual element like a great fall just three feet in front of it. In general, self adaptation is not a tool or a library, but a principle on which programs should be designed.

This principle has been further studied and defined than under just 'self adaptation'; it has extended into many various speciality fields, from self-protecting systems which adapt to protect against emergency failures and malicious software attacks, to self-healing systems that take action whenever they sense the system is not performing important tasks properly.

For the purposes of this project, the speciality of self-adaptation that will be explored is called 'self-optimisation'. This is what is described as a system that can adapt its actions in order to create a more efficient use of a situation. This kind of flexibility is valuable in certain areas where data management is important, and improvements in calculations and functional efficiency is merely a second long.

1.1.1 Self Optimisaton and Data Centers

Self optimisation is where some software dynamically adapts itself based on the current conditions of the system in order to generate a more efficient result. To draw an analogy, it's like deciding what shoes would be the best fit for an occasion. If you are going for a run, you will want to wear your running shoes and not some very stiff, formal ones; if you then go to a formal party after the run, though, those

running shoes are no longer the best choice, and so you will want the formal shoes that were not a good fit for the run.

Some programs can be seen as constantly going to different occasions that require different shoes. If shoes are algorithms at the program's disposal for a simulation, then the running shoes would be the most efficient choice for a run and the formal shoes would be the most efficient choice for a party. Knowing and picking the best shoes for the list of occasions you'll be involved in today is a successful case of self optimisation.

Because of the issues mentioned previously in modern and past systems, it is important that they be designed as efficiently as possible. But this is paradoxical because of the different pressures that data centers have faced; there is no one fixed specification for a data center that will make the most out of every task it is asked to compute; it can compute some well, but never all, and sometimes not even most.

For the sake of the project, let us narrow the field down further, however, as the amount of software fields that self optimisation can be applied in is gargantuan. One such field which I believe would benefit tremendously from being able to be able to dynamically pick by itself the most efficient path for any different task would be data centers. Since data centers constantly have to organise, reorganise and collect information from what can be a large pool of data, they must also perform a multitude of different computations at a time, perhaps hundreds within a single minute.

Since it can work with a plethora of different data sets, and with different applications of it possible to be requested, giving a data center only one algorithm to work with can be very taxing. If a data center is given ten tasks to do, then it would give a more efficient output if it could choose the order in which the tasks should be done, and pick an algorithm or branch of coding that would give the optimal outcome for each task.

In other words, data centers need to have a wardrobe of shoes available to them, and they need to be able to change their shoes very often so that they wear the right ones to every occasion. Otherwise their fancy friends might ask why they showed up to a formal outing in their slippers.

The aims of this project were to explore the state of self optimisation in data centers, to analyse the different tools that could be used in this field, and to apply a simulation which tested these principles in a practical manner. Doing this would provide a successful example of a data center dynamically adapting.

1.2 Preliminary Reading

Self-adaptation, in relation to data centers, has been studied previously. Before delving into the work, several of these studies were analysed. The following articles were some which served as inspiration for this project.

1. **"DARE: Adaptive Data Replication for Efficient Cluster Scheduling."Dare**
   This paper showed the effectiveness of using an adaptive algorithm in a data center, notably in positioning data closer to areas where it will be requested. The end result made use of the FIFO (First In First Out) scheduler and reduced the time spent on jobs significantly. It served as an example of a successful self-adaptive system architecture.

2. **"Hedera: Dynamic Flow Scheduling (2010)".** [**al2010hedera**]
   This paper focused more on the network aspect of data management. To give a high level explanation, its architecture has a control loop which can detect large flows of traffic at the edge of network switches, estimate the natural demand of large flows and use placement algorithms to compute good paths for them.

3. **"A Scalable, Commodity Data Center Network Architecture."**[**Scalable**]
   This study focused on leveraging ethernet switches to support full aggregate bandwidth. Its most interesting feature was the architecture it used for its network. It had a two-level architecture which helps switches recognise differences between pods that take advantage of the topology's structure.

   While its algorithm use was not the most valuable, this system's architecture gave me insight on how flexibly one could structure a data center network, which I would take into account when considering data centers later on.

4. **"Task Scheduling in the Cloud Using Machine Learning Classification (by Abhijeet P. Tikar et al)".** [**tikartask**]
   This paper gave a look at task scheduling using tools similar to what would be used in this project. It clearly showed and illustrated how data center software simulators and machine learning tools could interact with one another, and also explained how different scheduling algorithms could be chosen and applied in this environment.

5. **"GreenCloud: a new architecture for green data centers."**
   The GreenCloud architecture was informative because it was able to "dynamically adapt workload and resource utilization through VM life migration". It accomplished this by taking variables that "reflected performance measures including application workload, resource utilization and power consumption". I considered the article both an insight as to how dynamic adaptation could be done and also a reminder of the benefits that can come from this work.

Chapter 2

Tool Analysis

2.1 Data Center Simulation

2.1.1 Overview of Simulators

This project lacked the budget to run its tests on an actual data center with physical hardware servers to work with. Instead, however, I was able to use a data center software simulator. This is a software library which is designed to be able to simulate the effects of a data center in motion, taking into account variables like the file size of a task the simulator wants to execute, or the amount of RAM its virtual machines had.

While it is not the real thing, it is an excellent substitute for testing purposes regarding software, and their simulations provide enough information that the results are useful for work relating to actual data centers.

It's worth mentioning that data center simulators are considered very valuable even to large companies, who prefer to test any adjustments that they make to their data center architecture on them first, before trying it on their actual data centers. These simulators are helpful to anyone working with anything relating to data centers, whether they have one or not.

While data center simulators are a relatively recent creation, there already exist many of them to consider. Some of these are more specialised than others, but a few simulators caught my attention during the research phase. When considering which simulator to use, the primary concerns was how flexible each simulator could be with its inputs, how difficult or easy it would be to implement learning algorithms into its architecture, what specialisations it had if any, and how much control it had over its different components. Other features, such as security, speed over networks or network topology, were a secondary concern.

- **GDCSim.**
  This was developed by researchers at Arizona State University.[**GDCSim**] There are three components to GDCSim: a dynamic CFD emulator, multi-tier resource manager, and a simulator that provides immediate feedback based on different resource management designs. It has been used to optimise the energy efficiency of data centers under different kinds of geometries, workload characteristics, platform power management schemes, and scheduling algorithms.
  I was very interested in GDCSim at the start because of its emphasis on green energy. With environmental impact being a key concern in the present era, any progress that can be made on that front is bound to be valuable. Unfortunately, at the time of the project, it was not available for open downloading like the other software tools were. Bill Loux of the Arizona Science and Technology Enterprises was contacted regarding permission to use GDCSim[**GDCSim_contact**], as but he never responded. With some reluctance, I had to put this aside.

- **SPECI.**
  SPECI stands for "Simulation Program for Elastic Cloud Infrastructures", and has several advantages like a use of scalable middleware in a cloud environment[**SPECI**]. While it was touted for its ability to use predictive models[**SPECI2**], it did not apply direct machine learning tools and so was not predictive of data in the way this project aimed to be. It was considered, but ultimately turned down in favour of other simulators.

- **BigHouse.**
  BigHouse applied techniques like queuing theory and stochastic modeling, but overall it seemed disadvantageous to work with from the get-go. [**meisner2012bighouse**] This is because it was not designed intentionally to work on Windows, which was the primary operating system of my work computer. [**Bighouse_os**] I was not interested in adding the installation of Linux to test if BigHouse would be a better choice than the already appealing alternatives I had found, so I dropped research on quickly.

- **CloudSim.**
  CloudSim is one of the more popular data center simulators in use today. It is implemented as a large library of Java programs, each of them passing Java classes as inputs and outputs. The different features, from cloudlets to network topology to others, are represented as Java classes, and came with multiple prepared examples to quickly demonstrate its effectiveness with multiple hosts, virtual machines and tasks. [**calheiros2011cloudsim**]

- **DCSim.**
  Created and presented by G. Keller et al. from the unversity of Western Ontario [**DCSim**], DCSim has a lot of similarities to CloudSim. For instance,

it too also draws upon Java classes for inputs, giving it a great deal of flexibility. Distinguishing it from CloudSim, however, is its emphasis on virtualisation (running multiple virtual machines on a single physical machine, better making use of a server's otherwise wasted resources). In the case where a server lacks enough resources, DCSim can apply Virtual Machine Live Migration - the action of moving virtual machines across to other servers when one server reaches max capacity.

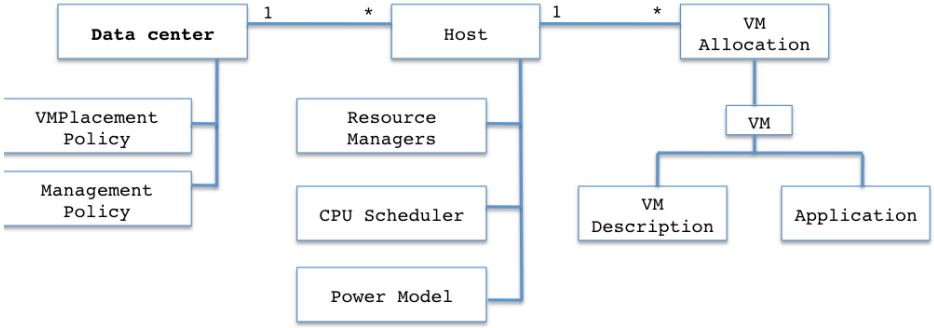


Figure 2.1: Architecture of DCSim

2.1.2 Why CloudSim?

After investigating multiple data center simulators, the two most promising choices were CloudSim and DCSim. Ultimately, CloudSim seemed more appealing because of its more thorough documentation and proven reliance in the research field.

Although the supposedly heightened flexibility of DCSim with virtual machines was tempting to work with, virtual machines were not going to have a decisive role in the project; for reasons that will be explored, the emphasis is on the tasks of the simulator rather than the machines it can allocate to them. CloudSim's proved adequately flexible and had enough thorough documentation own right that it proved too reliable to pass up on.

2.1.3 CloudSim Architecture

CloudSim is a simulation software tool, and as such it operates in abstraction. It has Java classes representing the core elements of a data center, from the data center itself to the hosts, their virtual machines and brokers, and more. Since these are all abstractions, they can be edited and modified to meet specifications. Each instance of CloudSim follows a similar pattern. In each iteration, the following steps occur, preferably in this order:

- *A registry of cloud resources is created.* This is sometimes referred to as a Cloud Information Service (CIS). This registry basically ensures that everything made from here on out will be able to use CloudSim tools.
- *A data center object is created.* This object is an astraction of the hardware services that a physical data center provides, and must be registered with the CIS once created. With the data center, a list of hosts is also created for the data center to give characteristics to. In this case, one host with certain variables - such as the amount of processing elements (PE), RAM (memory) and file size allocated - is sufficient.

  Just as the virtual machine object represents a virtual machine in a data center, the host object represents a physical machine in a data center that can run virtual machines. Hosts have their own variables for memory, storage and bandwidth (sometimes referred to as BW), and each host is virtualised into one or more virtual machines.
- *A broker is created.* The broker's purpose is to submit tasks to the data center. Initially it talks to the CIS to tell it the characteristics of the registered data center. The broker collects these tasks and their virtual machines and submits them to the data center. The broker, data center and registry are sometime called 'entities'.
- *A task, or 'cloudlet', is created.* Cloudlets represent any task that a data center might be asked to do, which is instrumental to the project. They are not nearly as complex as a full-fledged program that may be passed into a real data center, but the variables they are given to describe themselves means they can represent the cost of computing a proper task. This will be explored in deeper detail later.

  It is also worth noting is that cloudlets are assigned two IDs on creation; the ID of the data center broker they are to be processed through, and the ID of the cloudlet itself so as to more easily distinguish it from other cloudlets. This organisation both help the simulation, and anyone using it for their own programs, to better apply it.
- *One or more virtual machines (VMs) are created.* Their creation involves detailing their attributes, from their size and ID to the amount of bandwidth/MIPS/RAM they may have, adding the VM to the VM list, and then submitting that list to the data center broker.

In motion, these elements have a framework resembling the following image:

2.1.4 CloudSim Variables

CloudSim has numerous variables in each of its abstractions. But out of these, which ones should be analysed? Those would belong to the cloudlet and virtual

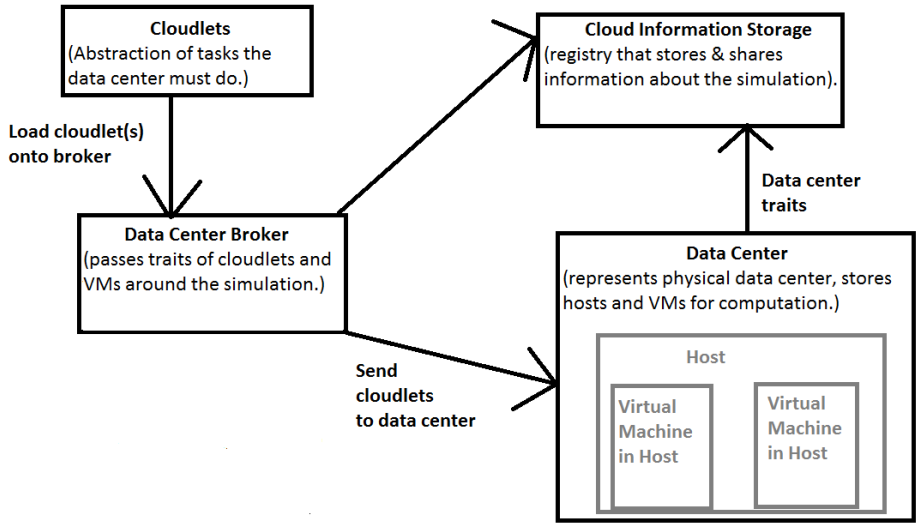


Figure 2.2: CloudSim system framework

machine classes, the two classes that tell us, respectively, the characteristics of the tasks the data center must execute, and the capacity under which they will be executed.

2.1.4.1 Cloudlet variables

- **Cloudlet Length.**
  The length/size of the cloudlet in terms of millions of instructions (MI). Hence, if this is divided by the MIPS rate of the virtual machine this task is scheduled on, the value is the length of time it will take for this task to be processed.
- **Cloudlet File Fize.**
  The file size of the cloudlet, in bytes, *before* it is executed by a data center.
- **Cloudlet Output Size.**
  The file size of the cloudlet, in bytes, *after* it is executed by a data center.

2.1.4.2 Virtual Machine variables

Each task in a data center is computed by a virtual machine. Different virtual machines have different resources allocated to them, which influences their relationship with the cloudlet they have. As a result, cloudlets cannot be considered separate from the virtual machines they are partnered with.

- **MIPS.**
  The number of MIPS in the virtual machine. MIPs stands for 'millions of instructions per second' and represents both the CPU load and the total capacity of the host. In other words, it is their processing capability. Since MIPS are assigned upon host creation, they can be said to be a preconfigured element.
  This has a direct relationship with the 'length' variable of cloudlets as mentioned above, which are measured in millions of instructions (MIs). If a cloudlet's length is 500 MIs, and the virtual machine computing that cloudlet has a MIPS value of 50, then 50 of those 500 millions of instructions are computed in a second.

- **Size.**
  The amount of storage space allocated to the virtual machine.

- **Ram.**
  The amount of random access memory (RAM) allocated to the virtual machine.

- **Bandwidth.**
  The amount of bandwidth assigned to the virtual machine. Important for network communications.

- **Number of PEs.**
  The number of processing elements (PEs) represented in this virtual machine. Each PE represents a CPU unit, as defined in the terms of the MIPS rating.

Note that these aren't the only variables in the virtual machines or cloudlets. For instance, in the cloudlet class there exists a method to return the cost of running the Cloudlet in the latest simulation resource. However, the most relevant ones to describing the data are the ones listed above.

Several of the others are either for other workings (like the IDs) or aren't written until after the simulation has stopped, (i.e before the project can optimise the set of tasks), which limits their usefulness. Hence, none of the excluded variables have any impact on what about the simulated data we're aiming to learn. Suffice it to say, the variables which matter in both objects are already in use.

Also note that all the listed variables are numeric, which is an important distinction in a dataset.

2.2 ARFF Dataset Files

Before getting to Weka - the set of machine learning tools used for this project - the features from the CloudSim environment must be presented in some way that the machine learning tools can read them. On the outset one might assume this would be straightforward, but diving into the details during an attempt at implementation leads to an obstacle - CloudSim and Weka cannot naturally interface with each other.

Trying to put a Weka tool inside of a cloudlet and expecting it to run is like attempting to mix oil and water; they may be similar and share an environment, but inherent properties are opposite enough that they directly mingle. In this same sense, the inner environments of CloudSim and Weka are too different that they can't interact directly.

Hence, an intermediary is needed. The best approach is to collect the data from the data center simulator and present it in a way that the machine learning tools can easily digest. The solution is a dataset that is made up of the values of the tasks and virtual machines in play from our simulation. Why? Because those values describe the CloudSim data. If data is like a forest, then an ARFF file is like a path through that forest.

There exist more than one type of data file this could be formatted into and saved it as, but the best fit for this was an Attribute-Relation File Format, or ARFF file for short. ARFF was created by the Machine Learning Project, at the Department of Computer Science of The University of Waikato, explicitly for use with Weka software. [**arff**]
ARFF files can be broken down into two parts: 'Header' sections and 'Data' sections.

2.2.1 Header

The header items are used to define both the ARFF file itself and the nature of the data that will follow. The first field that appears is the *@relation* field, which defines how the file should be referred to or called.

The second is the *@attribute* field, or fields, as it is strange for ARFF files typically have many attributes. As attributes do in general, they describe the data in the dataset. The attributes can be one of four data types: numeric, nominal, string or date. Numeric attributes represent numbers, nominal represents one of a given array of values (e.g. if the nominal attribute is defined as one, two, three, then the value written to match that attribute in the data field can be either one, two or three, but nothing else). String attributes are for arbitrary text values, which is useful for mining text-based applications. Date attributes are used to represent dates.

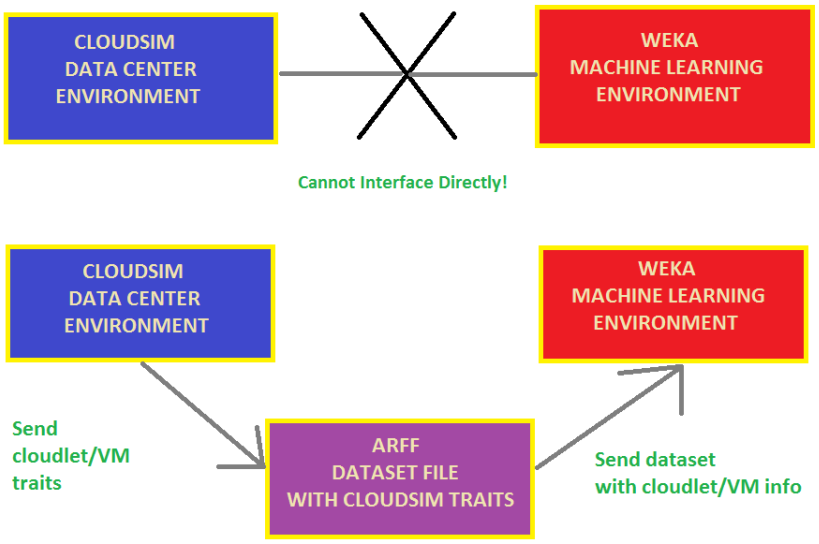


Figure 2.3: The initial problem, and the solution that ARFF provides.

```
@RELATION iris

@ATTRIBUTE sepallength  REAL
@ATTRIBUTE sepalwidth   REAL
@ATTRIBUTE petallength  REAL
@ATTRIBUTE petalwidth   REAL
@ATTRIBUTE class    {Iris-setosa,Iris-versicolor,Iris-virginica}
```

Figure 2.4: Sample ARFF header field, with the relation name and list of attributes

Of particular importance are the numeric and nominal data types. Numeric because the values in CloudSim are numeric in themselves, and nominal because it that is the type which the *target attribute* must be. This attribute will be discussed in the machine learning section, but for now, understand that it represents whether an instance is classified as 'correct' or not, and that it is required for classification.

2.2.2 Data

The data field is the actual data of the file, and is started with the '@data' declaration. Everything that comes after this declaration can be considered an instance of actual data.

Each instance of data is represented on a single line, and new lines denote a new instance. The instances have a correlation with the attributes that were defined in the header field above. Each data attribute value is separated by a comma - for instance, "123" is one single attribute value, whereas "1, 2, 3" denotes three separate values. Secondly, the order in which the attributes were defined in the header field must be matched perfectly by the order the attributes in every instance of data is presented. That is to say, the first value written for an instance should be the value matching the first attribute written in the ARFF file. If a date value is written for the third attribute in a data instance, and the third attribute defined in an ARFF file is a string value, then that is a type mismatch.

```
@DATA
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
```

Figure 2.5: Sample ARFF data. Each line is one instance of data.

Given that all of the data that should be transferred into a dataset from

CloudSim is numeric in type, the numeric value was the most important. However, the classifer later used in Weka would require a nominal value to classify each instance with for predictions' sake.

2.3 Learning Algorithm Tools - Weka

2.3.1 Weka Overview

Consider that the ideal state for our data center is to be able to adjust its course of action based on what it predicts would work better for a given input, with this input being the dataset. This practice is called machine learning; this exploration of algorithms that can learn from and make predictions on data is its own sub-field of computer science.

There exist several machine learning algorithm tools out there, but ultimately the search for a fitting software tool to explore such with was a simpler one than the earlier journey for an appropriate data center simulator. The most popular and recommended, by far, is Weka, developed by the University of Waikato. If nothing else, it's the most downloaded on Sourceforge by a good ten thousand or so.

Weka is a collection of different machine learning algorithms and data preprocessing tools, used for the sake of making data mining and machine learning tasks easier to use and more efficient. It can be used inside its own software application, but it can also be put into a command line or Java environment and executed with code just as efficiently. This means it an easy fit for our project and is compatible in the same environment as CloudSim, which also operates in Java.

Weka has numerous applications at its disposal, notably techniques that are part of the supervised learning discipline. Supervised machine learning is when a program infers a function from labelled data, like from the attributes of an ARFF file. Going back to our earlier map analogy, if data attributes are part of a path through a forest, then supervised learning techniques are about looking at a map that shows these paths and then walking through the forest.

One supervised learning technique is called classification, which refers to organising data into categories that best suit that data's needs. This principle ties back to our goals - if the CloudSim data can be categorised appropriately, then the system has an easier time of optimally dealing with it all.

In conjunction with this, learning methods can be applied to a dataset and then its output can be analysed in order to learn more about it. These learning methods are called classifiers.

Another is to apply multiple learners and compare their performance in order to choose one for prediction. These learning methods are called classifiers, and Weka has multiple at its disposal to classify data with. After that, a predictive model can be generated to understand the results.

2.3.2 Dataset Classifiers

Naive Bayes is not the only classifier, but it is one of the most commonly recommended ones for classifying data. It is probabilistic in nature and is based on Bayes' theorem, carrying the key trait of assuming the different predictors have independence from each other. Naive Bayesian models work effectively on large datasets since its parameter estimation is very simple; just give it a dataset to classify and it will do so. [**leung2007naive**]

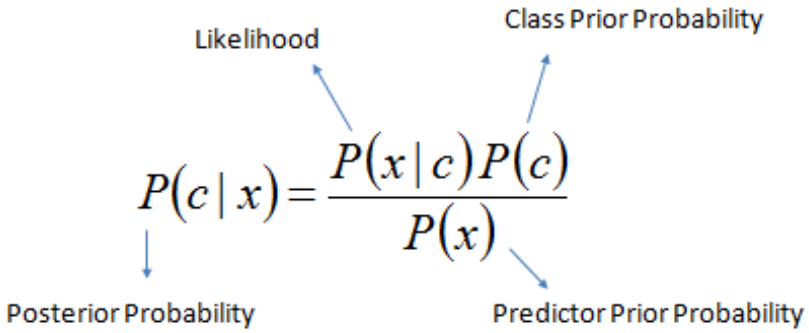
The only big criteria it asks for of an ARFF dataset is that it have a categoric 'class' attribute to classify each instance of data by according to its findings - this is the target attribute that was mentioned earlier. The Naive Bayes classifer it easy to build and was intended for smaller datasets, yet many applications have seen it outperform more complicated classifiers, even in large datasets.

2.3.2.1 Naive Bayes

Bayes theorem calculates the posterior probability (the probability of an observation fitting into a certain group, based on the given data) based on what it predicted the class of a given feature would be, and on the probability a given feature has to be a certain target attribute value (i.e the probability it reads as 'yes' or 'no).

For instance, given a nominal yes-no variable of some 9 instances, with 5 saying 'yes' and 4 saying 'no', then the overall probability can be said to be 'yes'. In order to calculate the probability of 'yes' given an attribute reads a certain way, then $P(c|x)$ P. Note that the probability is multiplied because this is working with independents.

In short, the classifier uses the values that an instance is expected to have, and then applies the bayesian equation to find a newer, more informed prediction value.[**sayadBayesTheorem**]
With all the unique tools for this project having been fully reviewed, it is time for the implementation.



$$P(c \mid \mathrm{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Figure 2.6: Naive Bayes theorem with labels.

Chapter 3

Implementation and Testing

3.1 Implementation: CloudSim and Weka Integration

3.1.1 Overview

As mentioned earlier, the goal of the project is to find a way a data center could dynamically analyse or adjust according to a prediction on data(ideally to make something more efficient). The tools that have been presented - CloudSim, ARFF and Weka - are able to do just that when put together. The system architecture that we've reviewed for each helps to give structure, but the following diagram shows an overall system architectural structure.

This section details the steps of the implementation.

1. The CloudSim environment as shown in past architecture notes is created.
2. Once the elements of the CloudSim environment - the broker, the datacenter, the cloudlets and virtual machines - are created, the elements of the cloudlets and virtual machines are extracted from the broker and arranged into an ARFF dataset file.
3. This dataset is then classified using the Naive Bayes classifier and evaluated using a weka model.

3.1.2 Implementing CloudSim

The first step is to create the CloudSim environment. This aspect of the project is, for the most part, as straightforward as the architecture we've created. In the start of the program dedicated to running CloudSim code, the CIS registry is initialised like normal so every CloudSim object will have access to the basic library.

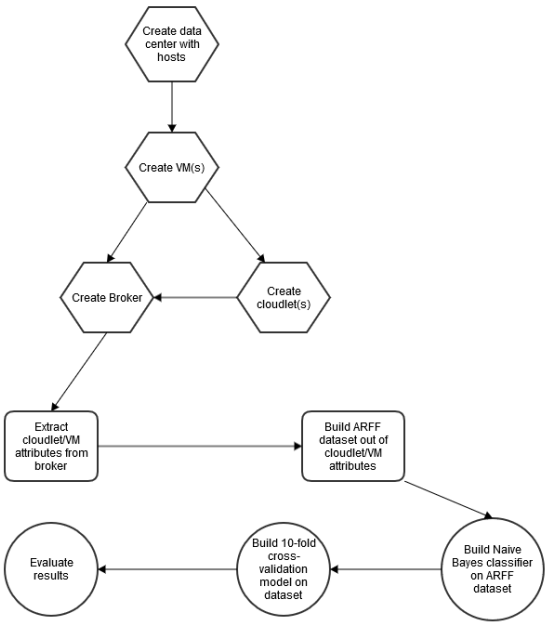


Figure 3.1: System Architecture, broken down into three tiers. Tier 1 = Data Center Simulation, Tier 2 = Data Extraction, Tier 3 = Classification

This gives them everything that the CloudSim parts of the architecture require to complete their basic tasks.

```
int num_user = 1; // number of cloud users
Calendar calendar = Calendar.getInstance();
boolean trace_flag = false; // mean trace events
// Initialize the CloudSim library
CloudSim.init(num_user, calendar, trace_flag);
```

3.1.2.1 Data Center and Host

The next step is to create a single data center. It is possible to create and run more than one data center in a single simulation, but it is of greater interest to see the project's effectiveness on a single data center.

The host is also created alongside the data center. This host, again, is not intended to be altered, so it is enough to leave it with some default values for ram, storage space and bandwidth size; the emphasis should remain on the tasks of the data center. Like with having multiple data centers in a simulation, there can be multiple hosts in a data center, but for this project only one host is sufficient.

These characteristics represent the energy cost of using processors, bandwidth, memory and storage on the data center. Since the different policies were discussed in our previous overview of CloudSim, let me repeat that no alterations were made with the CloudSim policies; the data centers just had a standard VM Allocation policy with the standard host list. Finally, variables to represent the key data center characteristics are created before moving onto the next step.

3.1.2.2 Broker and Virtual Machines

The broker object is created and assigned an ID - a simple integer variable - so that it can be easily referred to later on. Only one broker is used in this implementation, but an ID is important for other objects to refer to it and to distinguish it.

Next is to create the virtual machines. There can be multiple virtual machines in an implementation if desired, but note that - although the virtual machine attributes are uploaded into the ARFF file - the decision was made to keep the virtual machine values constant for each virtual machine in the tests.

That is to say, aside from their IDs, each virtual machine would have the same amount of MIPS, RAM, amount of bandwidth, number of processing elements as any other virtual machine. This way, the results generated would focus on the variation between cloudlet tasks, which would be the hardest to predict part in a real data management environent.

3.1.2.3 Cloudlet creation and generation

The true variance in the implementation from a standard CloudSim program comes from the cloudlets, the tasks that the data center is simulating. After creating the virtual machines and loading them onto the broker's VM list, a certain number of cloudlets are created.

Considering that the three core cloudlet variables at work are the cloudlet length, the file size and the output size, each cloudlet that is created can be characterised both by having different IDs and by having different values in these fields.:

- The cloudlet length is calculated by generating a random number between 1 and 16 and multiplying it by 5,000. This gives us some five-digit multiple of 5,000 to apply for the cloudlet length. (i.e the highest possible value will be 80,000 and the lowest possible cloudlet length will be 5,000.)
- The cloudlet file size is defined similarly; some random, positive, single digit value is multiplied by 100 - following the same principle as above, this means the highest possible value will be 900 and the lowest will be 100.
- The output size is defined with the same value as the file size. This is to maintain consistency.

```
//create the cloudlets
   cloudletList = new ArrayList<Cloudlet>();
   int cloudletAmount = 5000;
   for (int cloudletId = 0; cloudletId < cloudletAmount;
       cloudletId++)
   {
      cloudletList.add(createCloudlet(brokerId,cloudletId));
   }
```

With that created, it is time to create a for-loop that makes as many cloudlets as each test case should have. Each cloudlet receives an ID according to their creation (i.e the first cloudlet has an ID of 0, the tenth has an ID of 9) and adds each one to the list of cloudlets. Each cloudlet is assigned to a broker using the broker ID variable that was mentioned earlier.

```
   int cloudletAmount = 5000;
   for (int cloudletId = 0; cloudletId < cloudletAmount;
       cloudletId++)
   {
      cloudletList.add(createCloudlet(brokerId,cloudletId));
   }
```

With all the cloudlet and VM data prepared, their two lists are loaded onto the broker object.

```
   //...after vms are made and defined
   broker.submitVmList(vmlist);
   //...after cloudlets are made and defined
   broker.submitCloudletList(cloudletList);
```

Now to extract the data from this broker so the machine learning tools can classify it. The best way to do this is with a class created specifically for this project.

3.1.3 Implementing Data Extraction

The BrokerExtractor object is an object made fresh for this project, in order to work with the CloudSim tools. It takes two objects as input - the data center broker (with the two lists of cloudlets and virtual machines) and a string. The string represents the file name desired for the .ARFF file that the object will ultimately create.

```
//create an extractor to extract info from the broker (first arg)
```

```
//and put it into an ARFF file with the given string as its name.
BrokerExtractor extractor = new BrokerExtractor(broker,
    "extract_"+cloudletAmount+"_instances");
```

The constructor sets up the filename and the broker for reference for the core function that carries out the real computation, which is called 'writeFile()'. This method carries out the objectives of the class:

```
   Datacenter Broker broker;
   String filename;
//Constructor
   public BrokerExtractor(DatacenterBroker bro, String name)
   {
      filename = "C:/datasets/" + name + ".arff";
      broker = bro;
      this.writeFile();
   }
```

1. It fetches the cloudlets and virtual machines of the broker assigned to this instance of BrokerExtractor. This is done with two simple getters, one for each list.

2. It creates two special writer objects which exist in Java; a FileWriter object and a PrintWriter one. The FileWriter can take a string as an input and use that to create a file at a specified location by the string - in this case, the filename variable from the constructor can be used. The PrintWriter - if it is an assigned a file like the one our FileWriter creates - can be used to write the values of string inputs into a field, using 'print("Example string input.")' as an argument.

   ```
   //set up tools to write the file
      FileWriter fw = new FileWriter("C:/datasets/" + filename +
          ".arff");
      PrintWriter pw = new PrintWriter(fw);
   ```

3. Next, the PrintWriter can be used to fill in the file created by the FileWriter. This process starts with the header field, the attributes and the '@data' attribute (but none of the instances just yet). The header field's value should be the filename value as given from the CloudSim call.

   ```
      \centering
      //have the PrintWriter, pw, write an attribute field for
          the header section
   ```

```
pw.println("@attribute cloudletLength numeric");
\par
```

4. Lastly the data must print the values of each cloudlet and the virtual machine assigned to that cloudlet. Each cloudlet gets one line to represent each instance of data, with its attributes written in an order matching the order the attributes are presented in the header section; if they were out of order, the data would be inaccurate. The last step is to generate a randomised result for the class value of each instance;

```
//pred is a random object
//pred generates a random number
pred = rand.nextInt(10);
pw.print(getPrediction(pred));

//getPrediction(int x) method.
public String getPrediction(int x)
{
   if (x > 2)
   {
      return "yes";
   }
   else
   {
      return "no";
   }
}
```

The end result may resemble something like this:

Once this is all done, the dataset is ready to be classified and evaluated using machine learning tools, which was done inside a different class that is called at the end of the BrokerExtractor.

```
DataEval data = new DataEval();
data.wekaEval(filename);
```

```
@relation 'ExtractData'

@attribute cloudletLength numeric
@attribute cloudletFileSize numeric
@attribute cloudletOutputSize numeric
@attribute vmMips numeric
@attribute vmSize numeric
@attribute vmRam numeric
@attribute vmBw numeric
@attribute vmPesNum numeric
@attribute class {yes, no}

@data
400000,300,300,1000.0,10000,512,1000,1,yes
200000,300,300,1000.0,10000,512,1000,1,no
300000,300,300,1000.0,10000,512,1000,1,no
500000,300,300,1000.0,10000,512,1000,1,yes
320000,300,300,1000.0,10000,512,1000,1,yes
450000,300,300,1000.0,10000,512,1000,1,yes
600000,300,300,1000.0,10000,512,1000,1,yes
550000,300,300,1000.0,10000,512,1000,1,yes
700000,300,300,1000.0,10000,512,1000,1,yes
900000,300,300,1000.0,10000,512,1000,1,yes
```

Figure 3.2: Standard ARFF file

3.1.4 Implementing Weka Classification and Evaluation

3.1.4.1 Preparing and instantiating data

BrokerExtractor is not the only class that was created for our project, but just one of two; there was also a class dedicated to reading the dataset from the ARFF file that the extractor created, determining how it should be approached by using machine learning tools as mentioned earlier in this thesis.

To read the data, Java has two library objects available to use - a FileReader and a BufferedReader. A FileReader can read character files conveniently. It can know which file to read based on constructor inputs, which can be either a file, a file descriptor or a filename presented as a string. The latter is used because of the format of the project thus far, where the filename has been a string: using the filename string first set up in the CloudSim environment, this file can now be referred to. This way, the file that is read is the ARFF data file required for the classification process.

A buffered reader can read text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays, and lines. It can take the file reader as a constructor input, and by doing so it can read the ARFF data file.

With the file now ready, an instance must be trained on it. This is done using an Instance object, a weka class designed to handle an ordered dataset of instances. It is written to be able to carry over an ARFF file if it is passed in as a constructor argument, so the buffered reader from above is passed in.

The "setClassIndex()" line defines which of the ARFF attributes it has read is the class attribute. The classification approach requires a nominal attribute to classify the data by; without this, it does not know how to analyse the data or read the results.

With that, the instantiated data is set up, and so the buffered reader does not any more, so it is closed as to avoid the runtime errors that come when readers in programs aren't closed.

```
BufferedReader breader = null;
breader = new BufferedReader(new FileReader(filename));
Instances train = new Instances (breader);
train.setClassIndex(train.numAttributes() -1);
breader.close();
```

3.1.4.2 Building the Naive Bayes Classifier

The next step is to establish an instance of the Naive Bayes classifier itself. In Weka, it is represented as a Java object, and there are actually four different Naive Bayes classes to choose from, like NaiveBayes and NaiveBayesUpdateable. In the regular NaiveBayes class, numeric estimator precision values are chosen based on an analysis of the training data. To protect against errors in the dataset, it is good to have something more reliable with numeric values should any be missing.

Since effectively all of the values from the data center simulator are numeric values, NaiveBayesUpdateable is a reliable choice has, by default, a predictor value of 0.1 for numeric types, which is good on the occasion where someone tries to build a classifier with zero training instances.

With that done, it is time to build the classifier, giving the newly created instance as an argument. A classifier takes the following steps:

1. It verifies if the classifier can handle the given data.
2. It removes instances with missing classes
3. It copies instances and makes them more discrete if necessary.
4. It reserves space for distributions
5. It determines estimators for any numeric attributes
6. It counts the instances in the dataset
7. Lastly, it saves the space.

```
NaiveBayesUpdateable nb = new NaiveBayesUpdateable();
nb.buildClassifier(train);
```

With the classifier now built, it is time to evaluate the data. This is done by creating and defining an Evaluation object, a class specifically for evaluating machine learning models.

```
Evaluation eval = new Evaluation(train);
eval.crossValidateModel(nb, train, 10, new Random(1));
```

3.1.4.3 Cross-Validation Model

To complete our work, a model must be used to illustrate the data with. A cross-validation model is a technique for assessing how the results of a statistical analysis will generalise. This model is effective when there is no test model to work with, as was the case in this project where the data was taken from the data center simulator directly. Cross validation requires some key inputs in order to work:

- The classifier that it should apply to the dataset. In this instance, it is the Naive Bayes object.
- The dataset itself to be operated on. At this point, it has been classified by Naive Bayes.
- A number that will represent the amount of folds it will form over the data. If there are eight folds, it will partition the data into eight folds of equal size; seven of these will be used for training the data, and one for testing it. This partitioning is repeated for all combinations of train/test with the eight folds (e.g in one run, 4 is the test fold, but in the next one it might be 3). The aggregate from all runs is collected for results.
- A a random number for randomisation to ensure model variety. Note that this number is generated with a starting seed of 1, where a seed is a number that initialises a random generator. Note the advantage of using a seed is that it makes the randomisation pseudo-random, making it easier to reproduce these results in the future.

```
//run cross-validation model
eval.crossValidateModel(nb, train, 10, new Random(1));
```

The cross-validation model acts not unlike this. [**crossValidation**] To finish off,

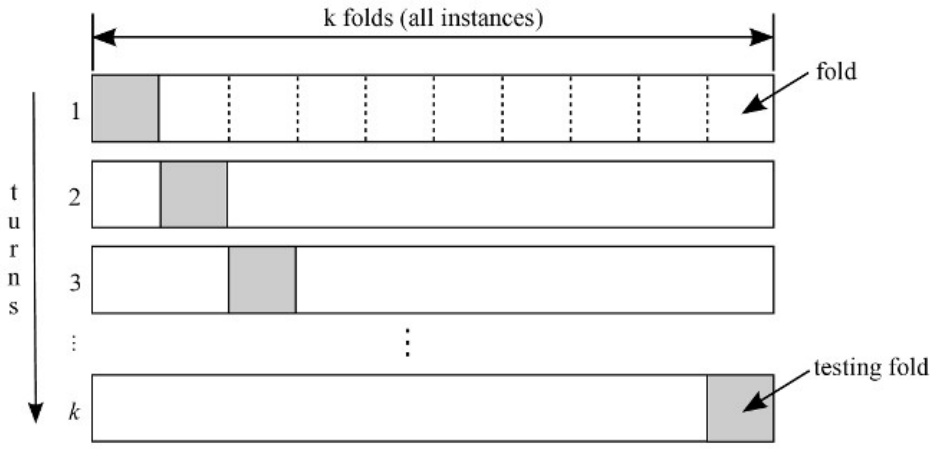


Figure 3.3: K-fold cross validation. Source: "Intro to AI material" by Joshua Eckroth

the evaluation object prints the results so they can be analysed.

3.2 Testing

3.2.1 CloudSim Testing

When testing CloudSim, there were some elements that were not altered across each instance. These included the values of the virtual machines, data centers or hosts, as well as the amount of virtual machines or data centers in the simulation. The idea is that the dataset can optimise the order of the tasks which are done, and those tasks are covered by the cloudlets.

While the tasks are indeed relegated between different virtual machines, data centers and hosts, it cannot be assumed that altering these features would give us honest data about the influence that classifying the cloudlets has on the simulation. Even if altering the features of these parts of the simulation, or the amount of them in play, would give our simulation a better overall result, it would not give us a more valuable information result.

The testing of CloudSim centred primarily on the cloudlet creation and the different attributes. The cloudlets, remember, represent the tasks that a data center must execute, and so these are what must actually be changed in order to see the desired effects.

A reminder that, in order to see a wide and credible variety of datasets, the cloudlets' variables were generated through pseudo-random means, following guidelines outlined earlier in the implementation section. The lengths are a number between 1 and 16 and multiplied by 5,000. The file and output sizes are a number between 1 and 16 and multiplied by 50. This was repeated for each cloudlet, meaning that there was a low likelihood of too much similarity between the clouds.

```
//create a cloudlet with the given ID for the broker of the given
    broker ID
private static Cloudlet createCloudlet(int brokerId, int id)
{
   Random rand = new Random();
   long length = ( (rand.nextInt(16)) + 1 ) * 5000;
   long fileSize = ((rand.nextInt(16) + 1)) * 50;
   long outputSize = fileSize;
   int pesNumber = 1; //number of processing elements.
   UtilizationModel utilizationModel = new UtilizationModelFull();

   Cloudlet cloudlet = new Cloudlet(id, length, pesNumber,
      fileSize, outputSize, utilizationModel, utilizationModel,
      utilizationModel);
   cloudlet.setUserId(brokerId);
   return cloudlet;
```

```
}
```

These value ranges were considered appropriate because of the variety of values presented to each variable. Pseudo-random tests were chosen so as to cover a more general, and more reliable, case of values for our simulator. The values were not left to be purely randomised under the prediction that the sheer numerical range of each variable meant that the dataset generated would be impractical. If a 16-sided dice is rolled a thousand times, it is likely that a table formed of the numbers it lands on will feature each value reliably. If a 90,000-sided die is rolled one thousand times, the chance of some value ranges being left out is slanted.

The idea is that our data center can self optimise under any circumstance, when dealing with any variety of datasets, not just two or three different types. Note that the results could be observed by opening the ARFF file generated in an editor; each time the results appeared to be varied.

The last thing to vary is the amount of cloudlets created, added to the broker and ran to the simulator. In terms of our ARFF file, this affects the number of instances under the data field.

3.2.2 Testing Weka

The major component to vary in our testing of the Weka components was the number of folds created in the cross-validation model. In general, each dataset was tested using 10-fold validation, which is the standard amount of folds in most cross-validation models.

Chapter 4

Results and Conclusion

4.1 Results

Upon completion, the cross-validation model prints a table of results. These results measure the accuracy of the predictions made, the margins of error, the overall accuracy of classification and the complexity of the program. The following list is the list of results fields that the model yields. [**evalResultErrors**]

4.1.1 Overview of Results Fields

- **Correctly Classified Instances.**
  The NaiveBayes classifier classifies each instance it reads. This percentage measures how many of the classifications formed match the classifications the datasets were given. Naive Bayes typically returns a high to above-average percentage of correctly classified instances.

- **Kappa Statistic.**
  The classifier draws two different categorisations for each dataset: one for what it predicts the outcomes will be before evaluation, and one for what the actual outcomes were. The Kappa statistic (or value) is a metric to analyse the probability that these two analyses agree with each other. It calculates the value of the times where they agree divided by the overall amount of instances each reviewed in the dataset. Hence, the Kappa statistic relays information about the probability of the observed and expected agreements agreeing.

- **The K and B Information Score.**
  Named after the researchers Igor Kononenko and Ivan Bratko, this score evaluates the information score of a classifier's answers without relying on probability as an estimation performance. Hence, it's an evaluation that ignores the influence of prior probabilities.

- **Class complexity**
  "| scheme" This is the log-loss for the learning scheme being evaluated. Log-loss helps measure the accuracy of a classifier; in this case, the value is represented in bit form.
- **Class complexity.**
  | order 0" is the log-loss when using the prior probabilities for the class, estimated from the training data.
- **Complexity improvement.**
  (sf) This tells us how much the classification has improved on the class prior using the learning scheme. The complexity improvement is measured in terms of bits. It returns two values; the amount of bits it improved/reduced each instance by, and the total amount of bits it improved/reduced from the entire dataset.
- **Mean absolute error (MAE).**
  This measures the difference between the predicted values from the observed values. It is calculated as the average of the magnitude of each individual error without accounting for their sign. The mean absolute error doesn't exaggerate the effects of instances with larger prediction errors than others; it is a linear score which means all the individual differences are weighted equally in the average.
- **Root mean squared error (RMSE).** The RMSE is the average of the square of all of the error. This value is used very often and is considered an excellent general purpose error metric for numerical predictions. Compared to the similar Mean Absolute Error, RMSE amplifies and severely punishes large errors.

  Note its synergy with the MAE for comparing results. The RMSE will always be larger or equal to the MAE; the greater difference between them, the greater the variance in the individual errors in the sample. To illustrate that, if only one pair of shoes in a shoe closet of hundreds has a broken pair of laces, the gap between the RMSE and MAE for evaluating that closet is much higher than if. If they are the same value, then there is no variation in the error margin for each instance.
- **Relative absolute error.**
  This is the total absolute error, just normalised into a relative field rather than an absolute one. To illustrate the difference: if a 10 percent error is equally important regardless of how large the overall pool is (be it 50 out of 500 instances or 0.2 out of 2), then getting the absolute is meaningless compared to the relative error.
- **Root relative squared error.** This is the squared error relative to what it would have been if a simple predictor had been used. This value is obtained

by dividing the total squared error that was used by the total squared error had a default predictor (the average of the actual values from the data).

There was also the prediction value results, which presented the percentage of correctly and incorrectly classified instances. In each of our tests, the majority of instances were classified correctly.
Of the listed value fields, particular attention was paid to complexity improvement. The end goal was to assess how much the classification has helped organise the dataset and see if it has made our work more efficient or less. The complexity measurements seemed to be the most valuable to this end; it's measurement of how many bits per instance the algorithms chosen by our classification have made the dataset more efficient to read.

4.1.2 Results from Tests

Below is a table compiled from five different test datasets, ranging from fewest number of instances to the largest number of instances from left to right. The smallest set has 50 instances while the largest has 10,000.

Table 4.1: Cross Validation Model Results

Instances	50	100	1000	5000	10000
Correctly Classified Instances	64%	60%	69.7%	70.38%	70.68%
Kappa Statistic	-0.1421	-0.0689	0	0	0
K and B Info Score	-5.0109	-3.1129	-14.21	-39	-66.7353
Class Complexity(order 0)	43.1668	91.6537	884.9446	4382.982	8728.1924
Complexity Improvement	-8.261	-7.0067	-4.7248	-5.7452	-2.8
Mean Absolute Error	0.4225	0.4428	0.4235	0.4171	0.4143
RMSE	0.496	0.4881	0.4611	0.457	0.4553
Relative Absolute Error	103.1977%	99.7545%	100.2195%	100.0209%	99.965%
Root RSE	109.8617%	103.68%	100.3246%	100.081%	100.0191%

4.1.2.1 Results Analysis

- All of the datasets return a high classification accuracy, almost never going lower than 70% correctly classified instances and with some of them going much higher than that. Hence, the results from these tests are accurate.
- Measuring the complexity improvement values, it can be seen that each dataset had strictly negative values. Comparing this to the class complexity of each dataset shows a percentage alteration; the set with 50 instances has

the largest value at -8.261 bits, which is a sizeable percentage of its class' complexity of over 43 bits.

The largest set with 10,000 instances, however, only has a complexity improvement value of -2.8 bits even though its class complexity is over 8700 bits; this is a very small value in comparison, suggesting it has had very little effect. However, the total bit level stayed at the decimal level, even though their respective class complexities ventured far above that. From this, it is clear that as the dataset became larger, the evaluation had less of an impact on its complexity.

- Each test also had a comfortably narrow margin between the RMSE and MAE values. The largest gaps between them were about 0.04 in size, which is a very small margin given the size of most of our datasets.
- The values of the RAE and root RSE fields were very high in each area, ranging from 99.965% at 10,000 instances to 103.1977% at 50 instances. The values were consistent across all the datasets, from large to small, implicating that size had little effect on this field.
- Note that these results stayed consistent throughout the different sizes; even a dataset with 5,000 instances had roughly the same complexity improvement value, correctly classified instance percentage and margin between RMSE and MAE as a dataset with only 1000 instances.

Chapter 5

Conclusions & Future Work

5.1 Conclusions

5.1.1 Analysis of positive outcomes

Based on the results, the Naive Bayes classifier returned primarily satisfactory results. In short, the majority of our instances were classified correctly, and the dataset was rather manageable.
While there was room for improvement beyond the typical 70% result, this outcome is still positive and not unreliable.

5.1.2 Accuracy Analysis

One recurring feature in each test case is that the magnitude of error was spread relatively evenly across all the dataset. Even though the datasets were formed through randomisation, there were little to no instances that were dramatically more erroneous than the others.

5.1.3 Criticisms

The use of Naive Bayes can be criticised, however, as not always being an ideal benefit for each dataset; while there were technically improvements, the margins of such were rather small as a whole; multiple results returned little in the way of varying results.

The complexity improvement kept consistent as a value, but was inconsistent as a percentage. The majority of them remained at single digit values before the decimal space even as the class complexity increased by the thousands.

An improvement of -8 bits has a noticeable impact on a dataset of 43 bits; an improvement of -4 bits has less of an impact on a dataset with a class complexity of 4000 bits than it does on a dataset of 8000 bits.

This implies that the classification has a very neutral influence on the results altogether, and that the simulation would not be drastically different had this adjustment been left out altogether.

5.2 Future Work

5.2.1 Thoughts on data center simulator

CloudSim was a satisfactory, reliable tool in this project. At no point was it a hinderance to the work of the project, as it was a relatively straightforward process to integrate the means to form a dataset from its cloudlets and virtual machines and then apply machine learning tools.

However, future research in this field with other simulators may turn up interesting results. The likes of SPECI may be a better fit for more specialised works than self-optimisation, though DCSim could have promise in this field. Instead of CloudSim in case the greater VM control would be more valuable.

5.2.2 Alternative Machine Learning Tools

Weka is so powerful and easily applicable a software compilation that it is difficult to criticise. It can only be recommended for future work with data center simulators, particularly in the field of self-optimisation.

NaiveBayes delivered accurate recordings, but its underwhelming influence on the complexity of some datasets results should prompt consideration for other supervised machine learning techniques.

While Naive Bayes was the classifier of choice for this project, there are other supervised machine learning tools available that would likely have other such as decision tree branching. A similar project to this one experimented with the use of a decision tree classifier in order to predict the appropriate task scheduling algorithm**tikartask** to give one example of different ways supervised classification can help a data center self-optimise itself. Future research could consider the effectiveness of this against the classification approach that was appled in this project.