# An adaptive algorithm for dynamic resource allocation in large heterogeneous Cloud environments

Ryan Lacerna


Submitted as part of the requirements for the degree
of MSc in Cloud Computing
at the School of Computing,
National College of Ireland
Dublin, Ireland.

September 2015

Supervisor Dr. Adriana Chis

# Abstract

Today, nearly everybody is connected to the Internet and consumes cloud services whether to store, process and deliver data. Cloud consists of large networks of virtualized solutions via data centres. In this new era where cloud is at the forefront, a multitude of domains such as healthcare, education, finance, science etc. have established the need for new content-driven applications. These content-driven applications require massive data gathering, generation, processing and then have them all in a large heterogeneous system that consist of a variety of private/public cloud systems that are geographically dispersed. In this context, resource provisioning and allocation becomes a big challenge in modern distributed systems due to the unpredictable fluctuation of service requests and heterogeneity of system types within the cloud environment. In consideration, an intelligent load balancer becomes an indispensable part of cloud computing. In this research paper we propose a novel algorithm to tackle such heterogeneity. This new algorithm takes advantage of the social communication and self-organisation of the intelligent foraging behaviour of Honeybees. Creating a distributed, self-organising, multi-agent system that takes advantage of the Self-aggregation technique. Self-Aggregation attempts to group services together to structure the clouds heterogeneity. In this dissertation, we empirically evaluate this new algorithms UserBase response time and data center processing performance via CloudSim against various state-of-the-industry algorithms that are currently being used in large and heterogeneous distributed systems.

# Acknowledgements

I would like to give my special thanks to Dr. Adriana Chis of the School of Computing, National College of Ireland for her invaluable support, technical advice and staying up 4am in the morning to proof-read and provide me feedback before submission. I have never met a lecturer with such work ethic and care for students to do well.

I would also like to give my sincere thanks to Clara Aggasid for proof reading this dissertation. Her kindness and help in analysing the research paper has given me confidence that the grammar and spelling are up to standard.

I could not have completed this research paper without the unwaivering love and support from my family and friends who supported and encouraged me throughout the process.

The literature review and quality of critial thinking would not have been up to standard without the careful analysis and feedback from Keith Brittle.

Lastly, I want to thank my team and colleagues at H&R Block GTC for their kindness and encouragement at work during the process of my dissertation.

# Submission of Thesis and Dissertation

**National College of Ireland**

**Research Students Declaration Form**

***(Thesis/Author Declaration Form)***

**Name: _____**

**Student Number: _____**

**Degree for which thesis is submitted: _____**

**Material submitted for award**

(a) I declare that the work has been composed by myself.

(b) I declare that all verbatim extracts contained in the thesis have been distinguished by quotation marks and the sources of information specifically acknowledged.

(c) My thesis will be included in electronic format in the College Institutional Repository TRAP (thesis reports and projects)

(d) ***Either*** *I declare that no material contained in the thesis has been used in any other submission for an academic award.

***Or*** *I declare that the following material contained in the thesis formed part of a submission for the award of

_____

*(State the award and the awarding body and list the material below)*

**Signature of research student: _____**

**Date: _____**

**Submission of Thesis to Norma Smurfit Library, National College of Ireland**

Student name: _____ Student number: _____

School: _____ Course: _____

Degree to be awarded: _____

Title of Thesis: _____

_____

_____

One hard bound copy of your thesis will be lodged in the Norma Smurfit Library and will be available for consultation. The electronic copy will be accessible in TRAP (http://trap.ncirl.ie/), the National College of Ireland's Institutional Repository. In accordance with normal academic library practice all theses lodged in the National College of Ireland Institutional Repository (TRAP) are made available on open access.

I agree to a hard bound copy of my thesis being available for consultation in the library. I also agree to an electronic copy of my thesis being made publicly available on the National College of Ireland's Institutional Repository TRAP.

Signature of Candidate: _____

For completion by the School:
The aforementioned thesis was received by_____ Date:_____

This signed form must be appended to all hard bound and electronic copies of your thesis submitted to your school

# Contents

# List of Figures

# List of Tables

# Listings

# Chapter 1

# Introduction

Cloud computing is a completely internet-based computing model where all the services delivered are hosted within the cloud. Cloud is a collection of thousands of computers interlinked together in a complex manner L.D. and Krishna [2013]. This computing model integrates the summation of parallel and distributed computing to deliver on-demand access to shared resources. These shared resources consist of software, hardware or information of devices and computers that use the resource. This model is one of the emerging topics in the IT industry and provides computing as a utility service where consumers can pay as you use.

The shared use of these cloud resources by the consumers does lead to a range of issues in the system. Challenges such as; scalability, fault tolerance, system reliability, high availability and energy efficiency. These challenges occur when multiple concurrent requests to a single server lead to the server malfunctioning due to overload, while other servers are underutilised (idle) Yao and Ju-Hou [2012]. This type of failure is led by an imbalance of load in the system.

Large Data centres are where cloud hosts its services. Large-scale data centres are mainly heterogeneous systems, which means that cloud users are geographically dispersed and use a diverse range of services. This is a big challenge for data centres to deliver and handle these services efficiently where millions of requests can fluctuate frequently. This issue can be tackled by the implementation of a load balancer. A load balancers responsibility is to balance the load effectively across the machines. A survey by Valentini et al. [2013] on energy efficient techniques describes that the main aim for a load balancer is to achieve optimal resource utilization, avoid overloading the system and minimize the response time. However, effective resource allocation in such a large, dynamic and diverse system such as the cloud is a big challenge. An evaluation of

efficient resource allocation techniques by Hameed et al. [2014] highlight that various applications may not be related to each other via workload. Therefore, one of the main research challenges is to discover which applications could be effectively consolidated into one server.

The Honey Bee behavioural algorithm is categorised as a meta-heuristic type of artificial intelligence and has been proven to be a very effective algorithm for load balancing. Depictions of meta-heuristic algorithms by A Sesum-Cavic and Kuhn [2011] demonstrate that it's exploration techniques explore unchartered areas in the search space, while its knowledge accumulation is used and exploited. This is a good balance between contradictory requirements, which eventually leads to finding the optimum solution (global maxima). An overview of the Honey bee algorithm and its applications by Abu-Mouti and El-Hawary [2012] shows that Honey Bee algorithm have been used in various applications such as comparative analysis, electric power systems, parallel and grid computing, data clustering and image analysis, computer science applications and many more. The overview of this aforementioned research on the Honey Bee algorithm highlights that has matched or outperformed some of the other meta-heuristic algorithms. In resource allocation, Honeybee takes into account task prioritizations that have been migrated from an over-utilized virtual machine (VM). It also enhance overall performance of processing and load balancing that aims to reduce the amount of time a task needs to wait on queue of a VM. Thus, decreased latency. However, it has been proven that this improvement in performance works well as the variety of services increased, but its performance seize to increase as the size of the system grows.

It was discovered that the implementation of the Honey Bee on the application layer had led to a certain change in topology at the resource layer. Resulting in the minority of services having disproportionate amount of connectivity, while the majority of services only had a small number of links. Self-Aggregation (Active clustering) technique groups vital, similar or vital services to deal with load balancing. In the normal environment of Self-aggregation as described by Di Nitto et al. [2007] is a network with interconnected nodes characterized by their types and a list of its neighbour nodes. In this situation the Self-aggregation algorithm attempts to evolve the connectivity with its neighbouring nodes in order to reach an optimum configuration. These nodes are characterized by this algorithms technique that is used to rewire the network.

The research question to be answered: will the performance of Honey bee foraging algorithm used in load balancing be improved with the adoption of Self-aggregation in large heterogeneous cloud systems? This question is asked with a view of improving load balancing to tackle such a challenge. To reveal the need for further research on this area, this study reviews the research conducted previously and critically analysed

their discovery and findings. These reviews are included in the main body of the next chapter.

## 1.1 Contribution

Our research aims to improve resource allocation via a combination of algorithms for load balancing that is targeted specifically for large and heterogeneous distributed systems. The contributions of this dissertation are the following:

- A novel hybrid-scheduling algorithm, and a prototype implementation, that aims to improve resource allocation. The aforementioned algorithm is an intelligent self-organising solution that utilizes resources effectively in the heterogeneous nature of Cloud.
- An empirical evaluation of this novel solution is carried-out/performed against existing state-of-the-industry algorithms, considering various performance metrics such as reponse time, VM utilization and processing times of data centers.
- The empirical performance evaluations of our proposed algorithm will be conducted through a simulation using and extending the CloudSim [Calheiros et al., 2011] simulation framework.

# Chapter 2

# Background

The main focus of discussion within this main body is the critical analysis of load balancing and the techniques used which answer some of the challenges faced to deliver a smooth and efficient resource allocation in heterogeneous cloud systems. Here, the algorithms that will be used to answer the research question are also introduced.

This chapter presents relevant related work organized as follows:

- Section 2.1 Load Balancing: discusses why load balancers play a critical role in cloud computing and the challenges that confronts load balancers in heterogeneous cloud systems.
- Section 2.2 Honey Bee Behaviour algorithm: analyses the Honey Bee algorithm and its effectiveness and weaknesses in balancing loads.
- Section 2.3 Self-Aggregation: this section justifies the Self-aggregations effectiveness used in the Active clustering algorithm in balancing loads and points out its weaknesses. In this section it is also detailed why a combination of algorithms is required to enable an effective load balancing technique.

## 2.1 Load Balancing

This section reviews the previous research on load balancing and the challenges faced by the algorithms in the different system environments. This is a critical section in which we reveal the motivation why the research question chooses to explore dynamic algorithms.

Load balancing is one of the indispensable feature of cloud computing. According to

the definition by Soni and Kalra [2014], load balancers are essential for cloud systems to achieve high throughput and shorten the response time. High throughput is one of the fundamental goals for load balancing. Throughput is the measure of the amount of work that the system can process in a given amount of time. A similar view by L.D. and Krishna [2013] describe the main objectives of load balancing to improve the speed of execution time of applications that uses a cloud resource. The workload of these applications behave in an unpredictable manner during run time. An investigation by Devine et al. [2005] follows the view that applications behave unpredictably or change during computation. These studies make a strong argument for measuring the performance load balancing on heterogeneous cloud systems.

Heterogeneous cloud systems as described by Topcuoglu et al. [2002], are a diverse and geographically dispersed collection of computing resources interconnected through a high speed network. A study conducted by Yao and Ju-Hou [2012] express that in a cloud computing environment, homogenous servers in the cloud system are less. Homogenous servers (nodes) refer to systems that have a uniform set of hardware and software working in parallel to achieve a specific goal. A great deal of research has been established in load balancing on homogeneous nodes. However, the major drawback, Randles et al. [2009b] argue from this approach is that, it is unrealistic for most cloud computing instances. For most cloud computing instances are defined as diverse, and it is paramount that a heterogeneous and dynamic system is needed to provide optimal delivery of service. The study by Devine et al. [2005] shares this concern and highlights that it is important that algorithms need to be adaptive and state-of-the-art, heterogeneous, tailoring work assignments proportionate to communication, memory and processing resources. This vision is strengthened by another investigation on load balancing by Yao and Ju-Hou [2012]. If the system collapses due to the maintenance of the servers, it could bring great losses to the cloud users; the research emphasises that dealing with such requests in a heterogeneous system environment is an important challenge. To accommodate the needs for a heterogeneous system Devine et al. [2005] suggest that it requires a dynamic load balancer which can adapt to such unpredictable workloads.

Load balancing techniques are extensively examined for both homogenous and heterogeneous environments. L.D. and Krishna [2013] illustrate two types of load balancing techniques. They are called static and dynamic.

- Static load balancing algorithms as described by L.D. and Krishna [2013], perform well when servers have low variations in the applications workload. This type of load balancing thrives on homogenous systems. However, approaches of this kind carry with them various well known limitations. From existing reviews from the

previous paragraph, we find that these type of algorithms will not work well in heterogeneous cloud systems where the workloads vary frequently. This concern is shared by Yao and Ju-Hou [2012] where a description of the three conditions in which load balancing algorithms need to follow; (1) When the workloads are not too demanding, the algorithms should be able to self-organise to scale down the exchange of information, (2) Load balancing mechanisms should be able to thrive on heterogeneous cloud environments, and (3) The system throughput should not be affected, therefore the load balancing mechanisms should enhance the system throughput high as possible. Yao and Ju-Hou [2012] emphasise that high throughput is the most essential requirement in load balancing.

- Dynamic load balancing algorithms are techniques which can autonomously adapt to varying workloads at run time. The study by L.D. and Krishna [2013] states that dynamic load balancing algorithms have dominance over static load balancing algorithms. But in order to achieve such advantage, it requires the additional cost of collecting and maintaining load information.L.D. and Krishna [2013] highlights that there type of load balancing techniques are exceedingly good in load balancing for heterogeneous environments. Likewise,Sesum-Cavic and Kuhn [2010] share this view that self-organisation solutions are proven to be a useful approach when handling complexity. And that dynamic load balancing allows the highest level of productivity and leads to a dramatic increase in the overall distributed system performance. Our proposed load balancing algorithm called the Honey Bee is also a dynamic technique. Based on the three essential conditions followed by effective load balancing techniques, described from the previous paragraph. We can interpret that the heterogeneous cloud computing system and load balancing has a certain resemblance with a colony of Honey bees and harvesting for food Yao and Ju-Hou [2012].

This section has reviewed load balancing and its system environments. We have discovered that there are two types of load balancing algorithms for different system environments in cloud computing and found dynamic algorithms to be the most effective in heterogeneous systems. Building from this section, the next section shall further elaborate a dynamic load balancing algorithm called the Honey bee. Detailing its technique, and examining its strengths and weaknesses in terms of load balancing.

## 2.2 Honey Bee Foraging Behaviour algorithm

This section elaborates on the review of the Honey Bee algorithm from past studies in the area. This section also highlights the strengths and weaknesses of the algorithm

which points to the need of improving the algorithm based on the research question for this review.

The honey bee colony and how it forages for food is described in research by Randles et al. [2010]; the forager bees are sent for sufficient food sources; the forager returns to the hive and advertises the find to the hive using a method called a waggle dance. The suitability of the food source is measured by its quantity, the quality of the nectar that has been harvested or its distance away from the hive. Once the food source has been communicated through the waggle dance, the other honey bees follow the forager back to the discovered food source and begin harvesting. When the bees return to the hive, the remaining quantity of the food source is advertised by each honey bee through the waggle dance. This enables the bees to be sent out to a more plentiful food source and abandon less plentiful food sources. This method is well perceived in other research into honey bee inspired load balancing techniques including Sahu et al. [2013], L.D. and Krishna [2013],Yao and Ju-Hou [2012] Ghafari et al. [2013]. In the previous section it is mentioned that we can interpret that heterogeneous cloud systems and load balancing have a striking resemblance to the behaviour of honey bees foraging for food.

The Honey Bee Foraging Behaviour algorithm is an optimization algorithm that imitates the intelligent behaviour of a swarm of honey bees foraging for food L.D. and Krishna [2013]. The principles in which the use of this intelligent algorithm for Load balancing are proposed. Sesum-Cavic and Kuhn [2010]represents the honey bees as software agents at each individual servers. A server contains exactly a single hive and one flower which has several units of nectar. A task is represented as unit of a single nectar. Individual hives consists of a limited number of forager (outgoing) and follower (receiver) bees. In the initial stages, all outgoing bees are foragers. These foragers scout for the location policy partner server of their server to decide whether to take/-give nectar to/from the server, and call followers. The goal is to locate the optimum location policy partner server by following the optimum path that is advertised to be the shortest path. Randles et al. [2010] express that given this robust profit calculation method, the pattern of this intelligent behaviour offers a distributed and global mechanism of communication; making sure that profitable virtual machines seem to be an attractive candidate and are allocated to underutilized servers. On a performance evaluation by Karaboga and Basturk [2008] compared Honeybee algorithm against other evolutionary algorithms. Empirical results show that under various control parameters, the algorithm outperforms the algorithms and emphasizes that this technique can be implemented to tackle multimodal engineering problems with dimensionality. The promise of this robust and dynamic algorithm strengthens the reason for further research on its load balancing capabilities.

The throughput performance of the Honey Bee algorithm is examined by a series of empirical tests by Randles et al. [2009b]. The algorithm is evaluated against 2 other effective dynamic algorithms called Biased random sampling (BRS) and Self-aggregation (Active Clustering). The empirical results show that, for the Honey bee algorithm, increasing the server resources does not improve its equivalent throughput, its performance remains consistent. Conversely, the honey bee algorithm performs progressively well showing an increase in performance when the diversity of the workloads increase inline with the servers workload increase. A more recent study by Randles et al. [2010], demonstrates empirical results which shows similar results to that of the previously mentioned experiment. The result demonstrates that the honey bee algorithm performs in a consistent manner. Although it does not increase throughput performance in-line with the increase in system size. A third experiment conducted by Yao and Ju-Hou [2012] aimed to improve the current honey bee algorithm and the empirical results that emerged from the experiments strengthens the argument presented in Randles et al. [2009b] and Randles et al. [2010]. Yao and Ju-Hou [2012] agree that increasing the system size has deteriorated the performance of honey bee load balancing. An observation by Randles et al. [2009a] show that the self-arrangement at one layer causes an effect on other layers which was not predictable from the implementation. The findings present some weaknesses of the honey bee algorithm. Where the algorithm arranges only an insufficient link between requests in the same server node queue, therefore, the improvement of the throughput in the system is sub-optimal. These results are critical and builds upon the purpose of the research question mentioned in this review. To improve this sub-optimal throughput performance in the honey-bee algorithm.

In this section we have given a brief overview of how the colony of foraging honey bees behave and have described how this intelligent behaviour is adopted for load balancing in heterogeneous cloud systems. We have also discovered that although the throughput for honey bee performs very well when the service types are diverse, its throughput performance is sub-optimal when system size increases. In the next section we shall further elaborate another algorithm mentioned in the last paragraph called Self-aggregation (Active-clustering). Defining its strengths and weaknesses with the view of combining honey bee and self-aggregation to improve system throughput.

## 2.3 Self-Aggregation

This section describes the technique of Self-aggregation, revealing its strengths and weaknesses based from the experiments in which the algorithm was tested. This chapter also reveals the need for a combination of algorithms to improve load balancing

Self-aggregation is a dynamic load balancing technique that rewires the network Randles et al. [2009b]. This algorithm aims to group similar services together. Whereas many load balancers only work well in situations where the nodes are aware of similar (like) nodes and delegate workloads to them. This algorithm is particularly important to this review as Randles et al. [2010] demonstrate the concern where an implementation of the honey bee algorithm in the application layer induced a particular topology to arise in the resource layer. This led to a number of services having an unequal amount of network connectivity from its collaborating services, at the same time most services have only received a low number of links. Saffre et al. [2009] demonstrated a series of experiments that proves regular rewiring process in this situation could implicitly fix the problem, resulting in an established optimum cooperation in the long term for similar processes. An experiment by di Nitto et al. [2008] demonstrate that a number of jobs processed using self-aggregation have shown an improvement as a result of the Load balancing algorithms being able to work on a large homogeneous environment.

Empirical results from the experiment by Randles et al. [2009b], detail self-aggregation as having a contrasting result from the Honey bee algorithm. It shows that the performance of self-aggregation increases as the number of processing nodes increase. Whereas self-aggregations performance decreases when the types (diversity) of nodes increase. This is where the Honey bee algorithm performs well. A more recent study by Randles et al. [2010] illustrates similar results where the self-aggregation algorithms performance is decreased as the diversity of the nodes increase. One question that needs to be asked, however, is why Randles et al. [2009b] and Randles et al. [2010] did not try to combine some of these algorithms in order to root out some of their weaknesses. Another drawback with the research by Yao and Ju-Hou [2012] is that it relies too heavily on changing the parameters of the different Honey bee algorithms. All these studies reviewed so far, however, did not combine the algorithms but did suggest combining them and improving the algorithm based on their weaknesses.

A combination of a variety of load balancing algorithms is proposed by Randles et al. [2009b], when using a selection of the types of algorithms; either combined to provide an optimal solution or utilised independently when different scenarios arise. Yao and Ju-Hou [2012] share this view stating, within a certain number of server nodes, the increased size of the system does not lead to an increase in throughput performance, therefore, an improved Honey bee algorithm is suggested to solve this issue. This review follows the research question where we adopt self-aggregation to improve the honey bee algorithm. Vasile et al. [2014] have developed a similar idea to this research where self-aggregation and a combination of algorithms have been implemented. The conclusion coming from the novel algorithm implemented from their research shows that

clustering had an overhead effect, however, using a specialized algorithm combined with the self-aggregation have resulted in a good improvement in results.

In this section we have demonstrated how the self-aggregation algorithm works, pointing out its strengths and revealing its weakness. Here we have discovered that a combination or a variety of algorithms working together is needed to solve the underlying issues of some load balancing algorithms.

# Chapter 3

# Design

This chapter explores in detail the design of the proposed algorithm for this research. First we will explain the architecture of CloudSim and explain how we integrate our algorithm to this well used simulator. Thereafter, we explain the self-aggregation phase of our solution, identifying its parameters and explaining the algorithm. Finally we examine the scheduling algorithm to be utilized in order to allocate task to resources and also responsible for maintaining resource utilization of VMs.

## 3.1 CloudSim: Simulation Framework

Although there are a variety of commercial cloud computing infrastructures, such as, Azure, Aneka, Google App Engine and EC2, building a cloud testbed on a real infrastructure is time consuming and expensive. It is almost impossible to test performances of different scenarios of the application in a controllable and repeatable manner. For the purpose of this research, we therefore implement simulation methods to evaluate performance of our algorithms.

CloudSim is a framework designed by Calheiros et al. [2011], it is used to emulate application services and cloud-based infrastructures and can also be used on economy-driven resource management policies on large cloud computing systems. With the use of this toolkit we can focus on resource allocation problems rather than the implementation of the low level details of the cloud-based infrastructures and services Calheiros et al. [2011].

CloudSim supports both system and behaviour modeling of Cloud system components such as data centers, VMs, and resource provisioning policies. Several researchers and

organisations, such as HP Labs are using CloudSim for their investigation of resource provisioning and energy-efficient management of data center resources. CloudSim also supports the modeling and simulation of Cloud computing environments consisting of both single and federated Clouds [Calheiros et al., 2011]. There are a number of simulators(Grenchmark, GreenCloud, NS3, REPAST, iCanCloud) that can be used for simulating a Cloud enviroment, but these simulators do not fully support a realistic modeling of systems and behaviour of Cloud environments as well as CloudSim.

We use and extend CloudSim to provide a prototype implementation of our novel algorithm. The framework is written in the Java programming language, therefore, our system will be implemented in Java.

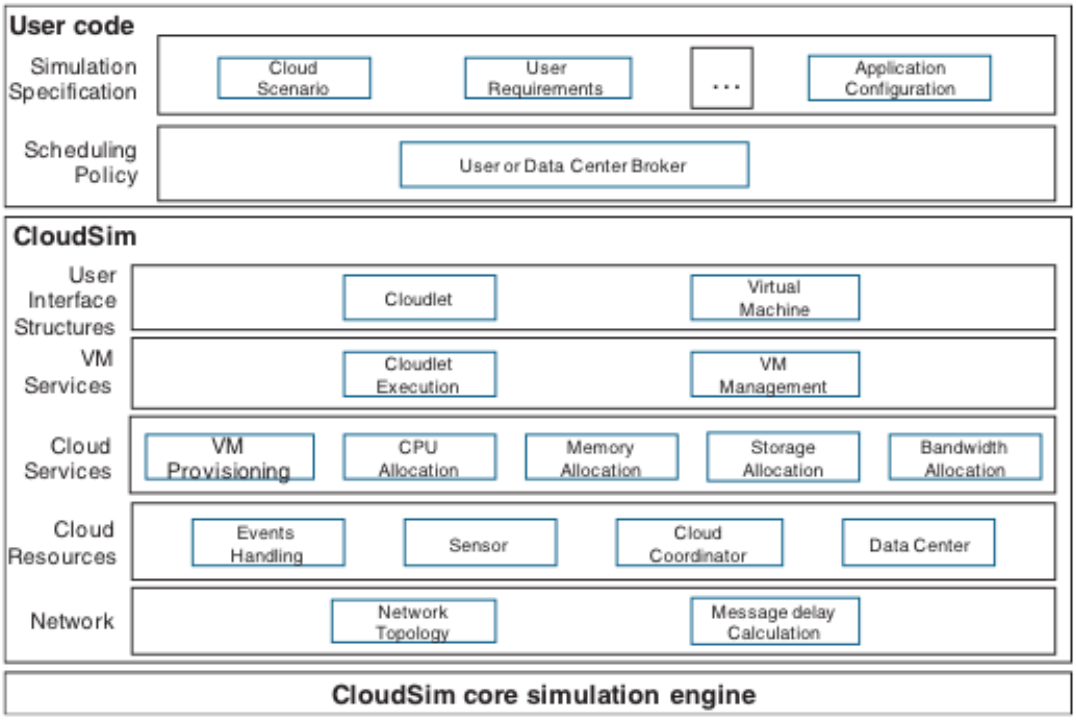


Figure 3.1: Layered CloudSim Architecture [Calheiros et al., 2011]

Figure 3.1 shows the layered architecture of the CloudSim toolkit and its architectural components. By extending the basic functionalities already available in CloudSim, we will be able to perform test beds on specific configurations and scenarios, therefore leading to the development of best practices in all of the critical aspects in cloud computing.

For this research, we implement the Self-aggregation phase on the Service Broker which dynamically allocates the load to a data centre based on best response time and user proximity to the Data Centers. The Honey Bee algorithm is then implemented in

the data center where the Data Centre Controller consults the load balancer for the next VM to allocate the cloudlet(task). The following sections will further detail these architectures.

## 3.2 Design Challenges

The original design for the self-aggregation phase was to implement a clustering algorithm called K-means by MacQueen [1967]. This algorithm is one of the simplest and most popular unsupervised learning algorithms that solve well-known clustering problems. For this research, the purpose of using K-means was to cluster characteristics of resources and feed this information to the VM load balancer.

The algorithm works by:

- Placing k points into the space represented by the objects being clustered. These points represent initial group centroids.
- Assign each object to the group which has the nearest centroid.
- Once all the objects have been assigned, re-calculate the positions of k centroids.
- Repeat step 2 and step 3 until the centroids no longer move. This creates a separation of the objects into groups.

Although the K-means algorithm is considered to be one of the simplest and popular unsupervised learning algorithms, one of its disadvantages is that it can be computationally expensive. First, the number of clusters(k) needs to be pre-defined. This pre-definition of k could mean that it can affect the dynamic nature of what we are trying to achieve. To extend K-means to accommodate the adaptiveness of finding k will lead to increase in complexity. This increase complexity can affect the execution time of our proposed solution that can then lead to reduced response time and performance. The next section explains the design of the implemented alternative to K-means.

## 3.3 Self-aggregation

Self-aggregation technique is a load-balancing technique that groups services together depending on certain characteristics. In our proposed solution, we will re-configure this technique to accommodate our novel algorithm.

Our Self-aggregation algorithm phase is implemented on the Service Broker component of CloudAnalyst. At this level we can dynamically re-configure the load based on the best response time achieved for the user, the proximity of that user base and current load of the data center. The main purpose of implementing this algorithm in the service broker is to have a level of control over user request routing to a data center. This is to address the problem raised in the background section where the Honey Bee's performance degrades as the number of user requests and system increase.

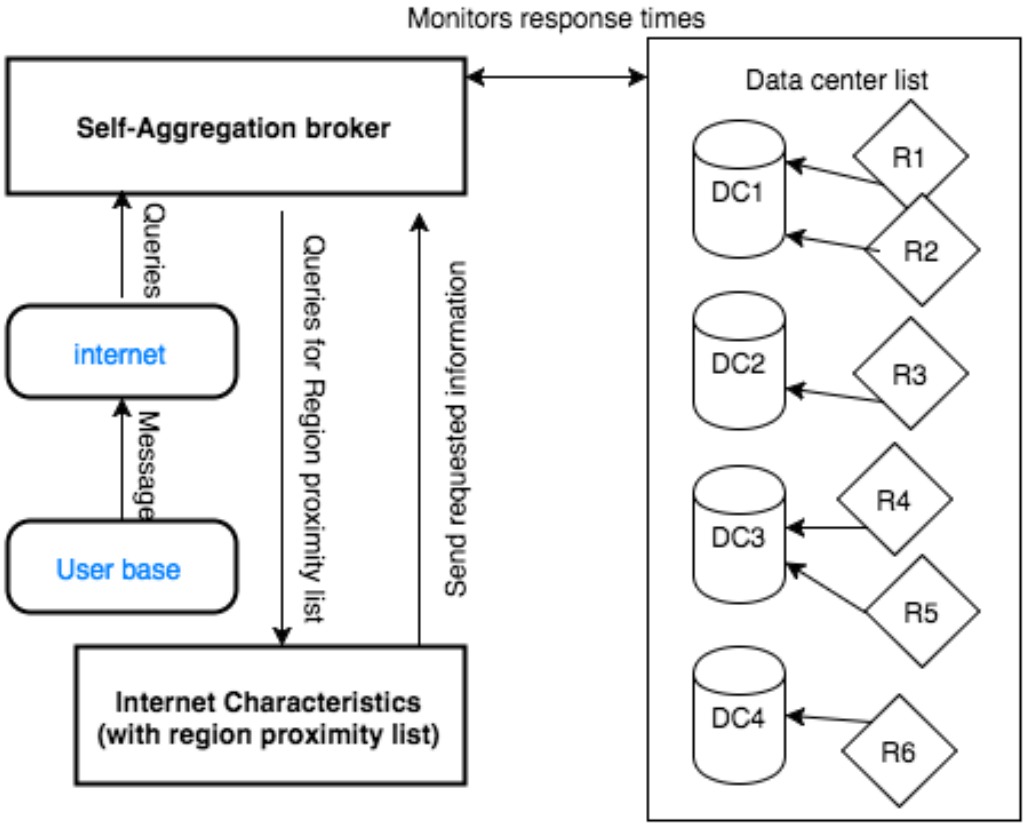


Figure 3.2: Flow Diagram of Self-Aggregation

## 3.4 Honey Bee

The system performs load balancing by pre-emptive scheduling. Load balancing is used to distribute the load evenly across machines to avoid overloaded, under-loaded or idle machines. For our proposed system we use the Honey bee algorithm for VM load balancing.

The task removed from the overloaded VMs becomes the Honey bees. Consequence of a submission to an under-loaded VM, a number of various tasks and load of tasks

assigned to that VM will be updated. This will be advantageous for other tasks since whenever a high task need to be assigned to a VM, it should address the VM which has a minimum number tasks so that the particular task will be executed quicker. Considering that all the VMs are sorted in an ascending order, the removed task will be assigned to under-loaded VMs.

### 3.4.1 Load balancing

We follow the Honey Bee formula used by L.D. and Krishna [2013]

Let $VM = VM_1, VM_2, VM_3, ..., VM_m$ be the set of $n$ virtual machines which should process the tasks outlined in the set $T = T_1, T_2, T_3, ..., T_n$.

Overall capacity of all VMs is given in (1)

$$(1)\ C = \sum_{i=1}^{m} C_i$$

The total length of the task that are assigned to a VM is to be considered as the load on a single VM. Formula given in (2)

$$(2)\ L_{VM_i,t} = \frac{N(T,t)}{S(VM_i,t)}$$

Where the $N(T,t)$ is the number of tasks at the time $t$ on the service queue of $VMi$, while $S(VMi,t)$ is the Service rate of the $VMi$ at time $t$.

Load of all the VMs is given in (3)

$$(3)\ L = \sum_{i=1}^{m} L_{VM_i}$$

Processing time of a VM is calculated by (4)

$$(4)\ PT_i = \frac{L_{VM_i}}{c_i}$$

Processing time of all VM are calculated by (5)

$$(5)\ PT = \frac{L}{C}$$

All of the VMs load are divided by the capacity of all VMs Following on the values presented from the above parameters, the scout bee checks weather the host is in a balanced or unbalanced state. This is calculated used standard deviation of the load given in (6)

$$(6)\ \sigma = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(PT_i - PT)^2}$$

If the $\sigma$ value falls within the condition of the threshold set (T s) $\in [0, 1]$, then the system is balanced, otherwise it is in an unbalanced state.

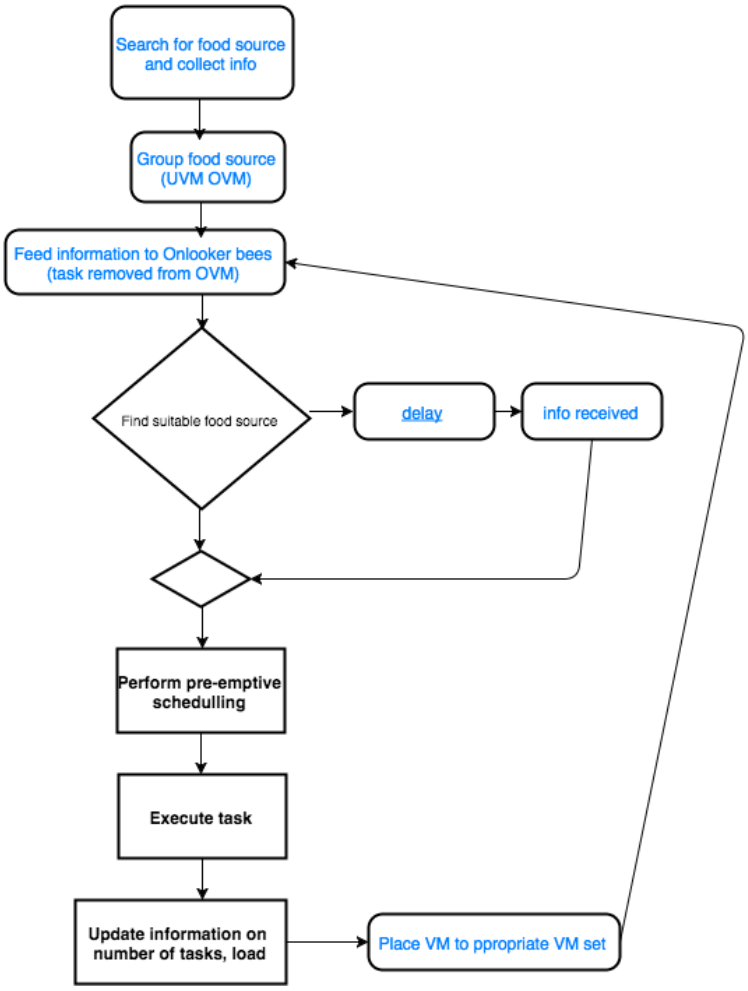


Figure 3.3: Illustrates the flow diagram of the honey bee algorithm for VM load balancing.

In this chapter we examined the design of the proposed solution for this research, and reviewed the Self-Aggregation phase and VM load balancing phase of our proposed novel algorithm. In the next chapter we delve into the actual implementation.

# Chapter 4

# Implementation

The implementation chapter details all the components of our proposed novel algorithm. Firstly, the CloudSim simulator was extended to work with the implementation of our solutions. In the beginning of this chapter, the CloudSim architecture and its components are explained, this is important as the proposed algorithms are built to work with these components. Thereafter, the implementation of the Self-Aggregation technique on the Service Broker is explained. Lastly, The Honey Bee VM load balancer implementation and integration are demonstrated.

To achieve our proposed implementation we have used the IntelliJ IDEA as our Integrated Development Environment(IDE) using Java version 1.8.0 for creating and executing our code. We imported the CloudAnalyst package components that are implemented in Java into our IDE.

## 4.1 CloudSim components and extension

This section briefly describes the main components of the CloudSim simulator and the flow of these components to fully understand the integration of our algorithm. CloudAnalyst is an extension of CloudSim that provides a graphical user interface(GUI) package to assist the ease of experimental modification and configurations.

The main components and its responsibilities are:

- GUI: As discussed in the previous paragraph, this package is responsible for the UI, and manages the controls of front end for the application. It manages the screen transitions and all other user interface activities. For us to be able to

use our implemented algorithm we have included BROKER POLICY SELF AGGREGATION to the $ConfigureSimulationPanel$ class so that we can select the algorithm in the GUI.

```
1
2  cmbServiceBroker = new JComboBox(new ↩
       String[]{Constants.BROKER_POLICY_PROXIMITY,
3  Constants.BROKER_POLICY_OPTIMAL_RESPONSE,
4  Constants.BROKER_POLICY_SELF_AGGREGATION});
5  cmbServiceBroker.setSelectedItem(simulation.getServiceBrokerPolicy());
6  cmbServiceBroker.setBounds(x, y, compW, compH);
7  mainTab.add(cmbServiceBroker);
```

Listing 4.1: Adding our policy to the GUI

- UserBase: This component is responsible for modelling groups of users and generates the traffic that represents the users.
- Simulation: This component is in charge of the simulation parameters, generating and executing the simulations. In the *Simulation* component we have extended the code to integrate the algorithm within the simulation.

```
1      CloudAppServiceBroker serviceBroker;
2      if (serviceBrokerPolicy.equals(Constants.BROKER_POLICY_PROXIMITY)){
3        serviceBroker = new ServiceProximityServiceBroker();
4      } else if ↩
           (serviceBrokerPolicy.equals(Constants.BROKER_POLICY_SELF_AGGREGATION)){
5      serviceBroker = new SelfAggregationBroker(dcbs);
6    }else {
7    serviceBroker = new BestResponseTimeServiceBroker();
8  }
9  internet.addServiceBroker(DEFAULT_APP_ID, serviceBroker);
```

Listing 4.2: Integrating the algorithm to the simulation

- Internet: This component is responsible for modelling the internet and the implementation of routing behaviours.
- DataCenterController: This component is responsible for the data center activities. The Data Center controller queries the Load balancer for the next available VM to allocated a cloudlet(task) to, if the VM is busy($-1$) it queues the cloudlet for that VM.

```
1    int nextAvailVM = loadBalancer.getNextAvailableVm();
2
3    if (nextAvailVM == -1){
```

```
4       //All VM's are busy. Put it in queue
5       //System.out.println("VM's busy, queueing " + cl);
6       waitingQueue.add(cl);
7
8       queuedCount++;
9       } else {
10      submitCloudlet(cl, nextAvailVM);
11      }
```

Listing 4.3: Get the next available VM

- VmLoadBalancer: This component is responsible for modelling the load balancing policy used by the data centers when allocating requests. We entend this component in our $BeeVmLoadBalancer$ class.
- InternetCharacteristics: This component is reponsible for defining the internet characteristics applied during the simulation, such as available bandwidths between regions, latency, the current traffic levels and the current level of performance information for each of the data centers.
- CloudAppServiceBroker: This component is responsible for modelling the service brokers which manages traffic routing between the user bases. We implement this interface in our $SelfAggregationBroker$ class that uses $getDestination$ method and takes in a $GeoLocatable$ as input which then returns a name of the datacenter, based on our Self-Aggregation policy.

```
1 public class SelfAggregationBroker extends ServiceProximityServiceBroker
2   implements CloudAppServiceBroker
```

Listing 4.4: Extending the ServiceProximityServiceBroker

A high level illustration of the system flow for these components are shown in Figure 4.1:

## 4.2 Self-Aggregation

This section provides the description of the final state of implementation of the Self Aggregation policy. As mentioned from section 3.1 we have implemented this phase in the Service Broker to give us the level of control over user requests routing to a data center. At this level, we can also monitor all the data centers, the proximity of the UserBases to the data center and latency.

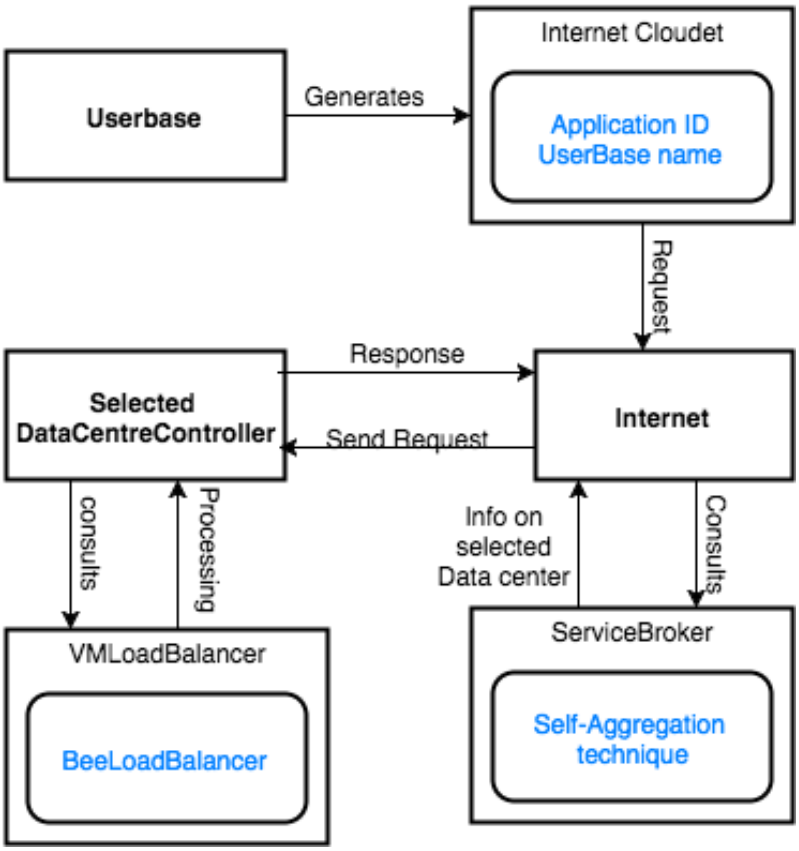


Figure 4.1: Flow Diagram of the system architecture

To achieve the implementation of the Self -Aggregation component, the $SelfAggregationBroker$ is implemented within the *cloudsim.ext.servicebroker* package of CloudAnalyst.

The algorithm works by:

- Maintaining a list of all the data centers and also a list of all the best response time currently recently recorded for each of the data centers.

```
1 private Map<String, Double> bestResponseTimesRecord;
2 private Map<String, DatacenterController> DataCenterList;
```

Listing 4.5: List of Data centers and List of Best Response time recorded

- Once the internet receives a message from the user base it consults the Self-Aggregation algorithm to determine a data center for the request.

```
1   int appId = cloudlet.getAppId();
2   CloudAppServiceBroker serviceBroker = serviceBrokers.get(appId);
3   destName = serviceBroker.getDestination(originator);
```

Listing 4.6: Get destination for the user request

- Our Self-Aggregation algorithm then queries the ServiceProximity broker for the destination

```
1  public String getDestination(GeoLocatable inquirer) {
2    List<Integer> proximityList =  ↩
         InternetCharacteristics.getInstance().getProximityList(inquirer.getRegion());
3
4  int region;
5  String dcName;
6  for (int i = 0; i < proximityList.size(); i++) {
7    region = proximityList.get(i);
8    dcName = getAnyDataCenter(region);
9    if (dcName != null) {
10   return dcName;
11   }
12 }
```

Listing 4.7: Querying for the next destination the user request

- The Self-aggregation algorithm then updates its best response time records if the new response time has performed better than the previous.

```
1  for (String dc : serviceLatencies.keySet()){
2  currLatency = serviceLatencies.get(dc)[0];
3  bestSoFar = bestResponseTimesRecord.get(dc);
4   if (currLatency != null){
5      if (bestSoFar != null){
6        if (currLatency <= bestSoFar){
7        bestResponseTimesRecord.put(dc, currLatency);
8      } else {
9        DatacenterController dcb = DataCenterList.get(dc);
10       if (dcb.getVmStatesList().size() <= maxVms){
11       dcb.createNewVm();
12       }
13       }
14 } else {
15 bestResponseTimesRecord.put(dc, currLatency);
16 }
```

Listing 4.8: Updating response time record of data centers

## 4.3 Honey Bee

This section details the implementation of the Honey Bee algorithm for VM Load balancing. The *BeeVMLoadBalancer* class is implemented within the *cloudsim.ext.datacenter* package of CloudAnalyst.

### 4.3.1 VM Grouping

In addition to the Self-Aggregation component mentioned previously where we have grouped the resources and tasks according to their characteristics, this particular phase aggregates the virtual machines based on their load. These groups are Overloaded VMs and Under-loaded VMs. Each group contains a number of VMs. New tasks are allocated to the under-loaded VM. The task removed from an overloaded VM has to make a decision to be moved to an under-loaded VM based on the load and tasks available in that under-loaded VM. In this algorithm, this is referred to as the Honey Bee and the under-loaded VM is refered to as the food source (destination) of the honey bees. The information that the bees (tasks) updates are the number of VMs on each group (OVM, UVM) and number of tasks of each VMs.

```
1       List<Map.Entry<Integer, Integer>> list =
2       new LinkedList<Map.Entry<Integer, Integer>>(suitableNectarSource.entrySet());
3
4       //compare OVM and UVM
5       Collections.sort(list, new Comparator<Map.Entry<Integer, Integer>>() {
6       public int compare(Map.Entry<Integer, Integer> o1,
7       Map.Entry<Integer, Integer> o2) {
8       return (o1.getValue()).compareTo(o2.getValue());
9       }
10      });
11
12      Map<Integer, Integer> sortedMap = new LinkedHashMap<Integer, Integer>();
13      for (Iterator<Map.Entry<Integer, Integer>> it = list.iterator(); ↩
            it.hasNext();) {
14      Map.Entry<Integer, Integer> entry = it.next();
15      sortedMap.put(entry.getKey(), entry.getValue());
16      suitableNectarSource.put(entry.getKey(), entry.getValue());
17      }
```

Listing 4.9: Grouping UVM and OVM

### 4.3.2 Task transfer

If the decision is to balance the load, the scheduler will prompt the load balancing function. For the load balancer to perform, the overloaded VMs need to be found, under-loaded VMs and supply (the available load). Following this, remove the tasks from the overloaded VMs. Tasks that have been removed previously (scout bee) from the overloaded VM are essential in finding the appropriate under-loaded VM for the current tasks (forager bee). This forager bee then becomes the scout bee for the following task. This process is repeated until load balancing is successful.

```
1       if (e.getId() == CloudSimEvents.EVENT_CLOUDLET_ALLOCATED_TO_VM){
2       int vmId = (Integer) e.getParameter(Constants.PARAM_VM_ID);
3       Integer currCount = suitableNectarSource.remove(vmId);
4       if (currCount == null){
5       currCount = 1;
6       } else {
7       currCount++;
8       }
9       suitableNectarSource.put(vmId, currCount);
10
11      } else if (e.getId() == CloudSimEvents.EVENT_VM_FINISHED_CLOUDLET){
12      int vmId = (Integer) e.getParameter(Constants.PARAM_VM_ID);
13      Integer currCount = suitableNectarSource.remove(vmId);
14      if (currCount != null){
15      currCount--;
16      suitableNectarSource.put(vmId, currCount);
17      }
18      }
```

Listing 4.10: Allocating tasks to a VM

To conclude this chapter, we have demonstrated the implementation of the proposed algorithms on CloudAnalyst by extending the simulator to integrate our code. The vital components of CloudAnalyst have been explained, detailing its system flow and how each components interact, followed by the explanation of the Self-aggregation algorithm implementation. Lastly, we have demonstrated the implementation of the Honey bee to work as a VM load balancer. In the next section we delve into the evaluation of data and analysis. Proving the feasibility of our proposal by examining the results from the analysis of tools and techniques that will be used for the purpose of answering this research question.

# Chapter 5

# Evaluation

This chapter deeply examines the analysis methodologies and tools that will be used to gather and analyse data generated by our proposed solution. The first section dicusses the setup and analysis for the empirical experiment. The second section evaluates results of response times of various scenarios. The third section evaluates the results for processing times, and lastly, we examine the VM utilization results gathered from our empirical evaluations.

## 5.1 Setup and analysis

A typical large scale application on the internet that benefits from Cloud technologies are social network applications. These type of applications presents various and un-uniform usage patterns. Access to such application varies throughout the day which the geographical locations of the users also varies. In addition, a feature in the social application may cause a sudden fluctuation in the service leading to the number of requests arriving unpredictably to servers which may only be temporary.

One popular social networking site is Facebook[1] that boast over 1 billion registered users. According to Wickremasinghe et al. [2010], in 2009, the distribution of facebook users base throughout the globe was:

- North America: 80 million
- South America: 20 million
- Europe: 60 million

[1] www.facebook.com

- Asia: 27 million
- Africa: 5 million users
- Oceana: 8 million users

In this evaluation we shall model the behaviour of social networks such as Facebook and configure the CloudAnalyst simulator parameters to mirror this behaviour to analyse the performance of our adaptive algorithm. We configure the simulation parameters according to the data used by Wickremasinghe et al. [2010]

| User base | Region | Time Zone | Peak Hours (Local time) | Peak Hours (GMT) | Simultaneous Online Users During Peak Hrs | Simultaneous Online Users During Off-peak Hrs |
|---|---|---|---|---|---|---|
| UB1 | 0 – N. America | GMT – 6.00 | 7.00–9.00 pm | 13:00-15:00 | 400,000 | 40,000 |
| UB2 | 1 – S. America | GMT – 4.00 | 7.00–9.00 pm | 15:00-17:00 | 100,000 | 10,000 |
| UB3 | 2 - Europe | GMT + 1.00 | 7.00–9.00 pm | 20:00-22:00 | 300,000 | 30,000 |
| UB4 | 3 - Asia | GMT + 6.00 | 7.00–9.00 pm | 01:00-03:00 | 150,000 | 15,000 |
| UB5 | 4 - Africa | GMT + 2.00 | 7.00–9.00 pm | 21:00-23:00 | 50,000 | 5,000 |
| UB6 | 5 - Ocenia | GMT + 10.00 | 7.00–9.00 pm | 09:00-11:00 | 80,000 | 8,000 |

Figure 5.1: UserBase data based on social network behaviour [Wickremasinghe et al., 2010]

For our empirical evaluations, we define three UserBases that mirrors the three regions around the world(N. America, S. America, Europe) including their parameters as illustrated in Figure 5.1. For our experiment we conduct the experiment at $1/10th$ of the size of Facebook.

| Data Center | Num of VMs | Image Size(Mb) | Memory(Mb) | BW(bits/s) |
|---|---|---|---|---|
| DC1 | 25 | 10000 | 2048 | 10000 |
| DC2 | 50 | 1000000 | 512 | 1000000 |
| DC3 | 100 | 10000 | 4046 | 10000 |

Table 5.1: Application Deployment Configurations

| ID | Memory(Mb) | Storage(Mb) | Available BW(Mb) | Num of Cores | MIPS |
|---|---|---|---|---|---|
| 0 | 2048 | 1000000 | 100000 | 4 | 10000 |
| 1 | 2048 | 100000000 | 1000000 | 4 | 10000 |
| 2 | 2048 | 100000000 | 1000000 | 4 | 10000 |

Table 5.2: Physical Hardware details of a Data Center

In terms of Cloud service costs we will assume that it follows the pricing plan of Amazon AWS EC2. The price plan is as follows:

- Cost per VM per hour(1024Mb, 100 MIPS): 0.10c
- Cost per 1 Gb of data transfer(request/receive): 0.10c

In terms of the configurations of Data Centers:

- Number of Data centers: 3 (Region 0,1,2)
- Data Center configurations:
  - Architecture: x86
  - OS: Linux
  - VMM: Xen

## 5.2 Results

After running the simulator with the above parameters we have achieved the following results for Throlled, Active VM, Round Robin and Honey Bee in terms of overall response time and data center processing times.

The data gathered and shown in Table 5.3, 5.4, 5.5 and 5.6, shows that there are improvements in performance by the Honey Bee over all the other algorithms tested in this experiment in terms of Overall response time and Data Center processing time.

| Honey Bee Results | | | |
|---|---|---|---|
| | Avr (ms) | Min (ms) | Max (ms) |
| Overall Response time: | 54.54 | 37.44 | 75.20 |
| Data Center processing time | 3.54 | 0.00 | 15.39 |

Table 5.3: Honey Bee Results

| Active Load Balancer Results | | | |
|---|---|---|---|
| | Avr (ms) | Min (ms) | Max (ms) |
| Overall Response time: | 54.70 | 37.44 | 76.52 |
| Data Center processing time | 3.70 | 0.00 | 15.75 |

Table 5.4: Active Load Balancer Results

Bar charts are illustrated in Figure 5.2 and 5.3 to further visualize the data center processing times and overall response time of each algorithm.

| Round Robin Results | | | |
|---|---|---|---|
| | Avr (ms) | Min (ms) | Max (ms) |
| Overall Response time: | 55.07 | 37.44 | 79.74 |
| Data Center processing time | 4.08 | 0.00 | 22.30 |

Table 5.5: Round Robin Results

| Throlled Results | | | |
|---|---|---|---|
| | Avr (ms) | Min (ms) | Max (ms) |
| Overall Response time: | 54.90 | 37.44 | 75.44 |
| Data Center processing time | 3.88 | 0.00 | 12.98 |

Table 5.6: Throlled Results

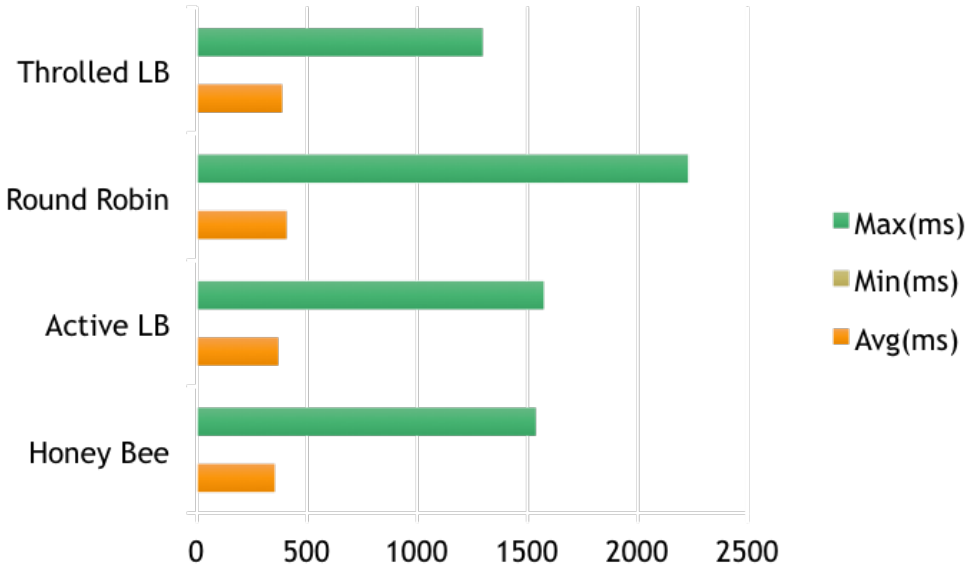


Figure 5.2: Graph illustrating DC Processing time

Figure 5.4 illustrates a graph of VM utilization for each algorithm. In this bar chart we can visualize the resources utilization by the number of task allocated to a VM. By analysing the results, we can tell that Bee have balanced the load between the virtual machines better compared to the competition. Throlled load balancer has allocated a significant amount of task to one VM but balanced the load to all other VMs. Round Robin results shows that it allocates tasks to the first VMs it finds, thus resulting to the first VMs found having more allocated tasks than the VMs found in a later stage

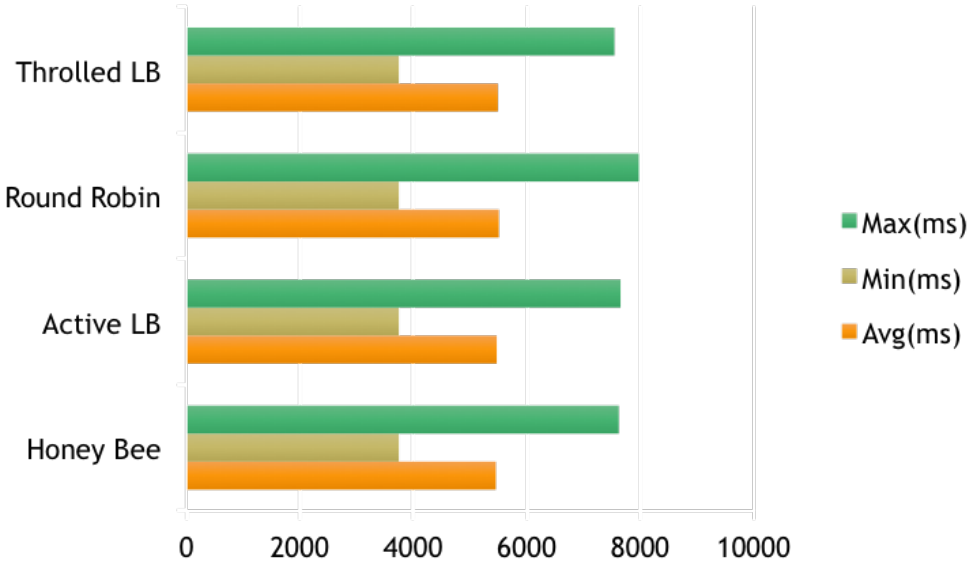


Figure 5.3: Overall response times

resulting in an unbalanced system. Active Load balancers' results illustrates that it has not performed as well as the other algorithms.

These results gives us knowledge that the prototype algorithm that we have developed for this dissertation has improved the performance Honey Bee in terms of overall response time, data center processing time and has performed significatly better in terms of balancing the load between VMs.

This chapter have demonstrated the analysis techniques and tools used to generate data to measure the performance and prove the claim of this research paper. Results show that the Bee algorithm performed better than the competing state-of-the-industry algorithms tested in terms of Response times and data center processing times and resource utilization. The next chapter will conclude the findings of this research, followed by future opportunities for further continuing the work conducted.

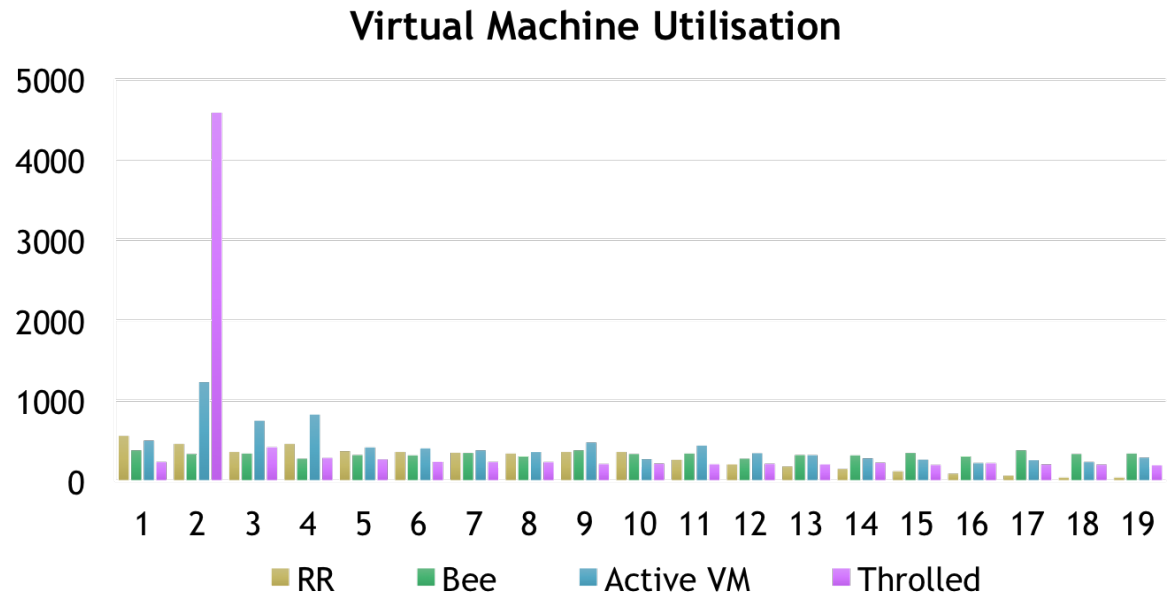


Figure 5.4: Graph illustrating the number of task allocated to a VM for each algorithm. The x-Axis represent virtual machines while the y-Axis represent the number of allocated task.

# Chapter 6

# Conclusion

This research paper proposed a solution to improve the performance of the Honey Bee Foraging algorithm used in load balancing by integrating algorithm called Self-Aggregation to assist Honey Bee perform well in large heterogeneous Cloud computing environments. The traditional Honey Bee algorithm performed well in diverse environments but its performance degraded as the system size increased. The proposed algorithm includes a Honey bee VM load balancer within the Data Center and a Self-Aggregation service broker. The Self-Aggregation broker receives the requests from Userbases and monitors the best reponse times of each data centre, UserBase vs Data Center proximity and current load of the Data Center to generate allocation decisions to a data center. Thereafter, the Honey Bee VM load balancer groups VMs according to their load, wether it is Overloaded(OVM) or Underloaded(UVM) and allocates new requests to the UVM, while de-allocating tasks from the OVM.

To verify that our hyphothesis is effective, we have integrated these algorithms in Cloud-Analyst that uses CloudSim as its core simulation engine. This is popular simulator that is widely used in scientific and industrial research in scheduling and resource allocation. This simulator allows us to configure user and vendor side parameters to mirror certain scenarios. For the purpose of this experiment, we have chosen to mirror the behaviour of social media applications as it provides an un-uniformed and unpredictable user behaviour.

After running the simulator using the parameters for the social network scenario we have achieved the following results for Throlled, Active VM, Round Robin and Honey Bee. The data gathered from the simulation shows that there are no significant improvements in results. However, it terms of the data center processing times we can see that Bee performed slightly better average(ms) response time than the other three

algorithms. In addition, it has also shown slight improvements in terms of overall response time compared to all the other algorithms. For resource utilization, Honey Bee have significatly performed better than the competition, balancing the load effectively across available virtual machines.

Although there are minor improvements found with the performance of the Honey Bee when compared to the other state-of-the-industry algorithms, it is not significant. However, this behaviour will be further investigated as part of future work. In addition, a selection of CloudSim extensions can be experimented with to find a more heterogeneous behaviour in the simulation, that may result in a more significant output for our prototype algorithm. Another limitation faced was that the CloudAnalyst extension of CloudSim does not contain an interface or an easier way to extend our algorithm further to distinguish which cloudlets(task) have a higher priority. Again, experimenting with other extensions of CloudSim may allow us to further characterize Tasks and Resources to develop a more intelligent Honey Bee that may provide significantly better results.

# Bibliography


Vesna A Sesum-Cavic and Eva Kuhn. Chapter 8 self-organized load balancing through swarm intelligence. In Nik Bessis and Fatos Xhafa, editors, *Next Generation Data Technologies for Collective Computational Intelligence*, volume 352 of *Studies in Computational Intelligence*, pages 195–224. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-20343-5. doi: 10.1007/978-3-642-20344-2_8. URL http://dx.doi.org/10.1007/978-3-642-20344-2_8.

F.S. Abu-Mouti and M.E. El-Hawary. Overview of artificial bee colony (abc) algorithm and its applications. In *Systems Conference (SysCon), 2012 IEEE International*, pages 1–6, March 2012. doi: 10.1109/SysCon.2012.6189539.

Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, Csar AF De Rose De Rose, and Rajkumar Buyya. Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exper.*, 41(1):23–50, January 2011. ISSN 0038-0644. doi: 10.1002/spe.995. URL http://dx.doi.org/10.1002/spe.995.

Karen D. Devine, Erik G. Boman, Robert T. Heaphy, Bruce A. Hendrickson, James D. Teresco, Jamal Faik, Joseph E. Flaherty, and Luis G. Gervasio. New challenges in dynamic load balancing. *Applied Numerical Mathematics*, 52(23):133 – 152, 2005. ISSN 0168-9274. doi: http://dx.doi.org/10.1016/j.apnum.2004.08.028. URL http://www.sciencedirect.com/science/article/pii/S0168927404001631. {ADAPT} '03: Conference on Adaptive Methods for Partial Differential Equations and Large-Scale Computation.

E. di Nitto, D. Dubois, R. Mirandola, F. Saffre, and R. Tateson. Self-aggregation techniques for load balancing in distributed systems. In *Self-Adaptive and Self-Organizing Systems, 2008. SASO '08. Second IEEE International Conference on*, pages 489–490, Oct 2008. doi: 10.1109/SASO.2008.38.

Elisabetta Di Nitto, D.J. Dubois, and R. Mirandola. Self-aggregation algorithms for autonomic systems. In *Bio-Inspired Models of Network, Information and Computing Systems, 2007. Bionetics 2007. 2nd*, pages 120–128, Dec 2007. doi: 10.1109/BIMNICS.2007.4610096.

S.M. Ghafari, M. Fazeli, A. Patooghy, and L. Rikhtechi. Bee-mmt: A load balancing method for power consumption management in cloud computing. In *Contemporary Computing (IC3), 2013 Sixth International Conference on*, pages 76–80, Aug 2013. doi: 10.1109/IC3.2013.6612165.

Abdul Hameed, Alireza Khoshkbarforoushha, Rajiv Ranjan, PremPrakash Jayaraman, Joanna Kolodziej, Pavan Balaji, Sherali Zeadally, QutaibahMarwan Malluhi, Nikos Tziritas, Abhinav Vishnu, SameeU. Khan, and Albert Zomaya. A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems. *Computing*, pages 1–24, 2014. ISSN 0010-485X. doi: 10.1007/s00607-014-0407-8. URL http://dx.doi.org/10.1007/s00607-014-0407-8.

D. Karaboga and B. Basturk. On the performance of artificial bee colony (abc) algorithm. *Applied Soft Computing*, 8(1):687 – 697, 2008. ISSN 1568-4946. doi: http://dx.doi.org/10.1016/j.asoc.2007.05.007. URL http://www.sciencedirect.com/science/article/pii/S1568494607000531.

Dhinesh Babu L.D. and P. Venkata Krishna. Honey bee behavior inspired load balancing of tasks in cloud computing environments. *Applied Soft Computing*, 13(5):2292 – 2303, 2013. ISSN 1568-4946. doi: http://dx.doi.org/10.1016/j.asoc.2013.01.025. URL http://www.sciencedirect.com/science/article/pii/S1568494613000446.

J. MacQueen. Some methods for classification and analysis of multivariate observations, 1967. URL http://projecteuclid.org/euclid.bsmsp/1200512992.

M. Randles, D. Lamb, and A. Taleb-Bendiab. Experiments with honeybee foraging inspired load balancing. In *Developments in eSystems Engineering (DESE), 2009 Second International Conference on*, pages 240–247, Dec 2009a. doi: 10.1109/DeSE.2009.19.

M. Randles, E. Odat, D. Lamb, O. Abu-Rahmeh, and A. Taleb-Bendiab. A comparative experiment in distributed load balancing. In *Developments in eSystems Engineering (DESE), 2009 Second International Conference on*, pages 258–265, Dec 2009b. doi: 10.1109/DeSE.2009.20.

M. Randles, D. Lamb, and A. Taleb-Bendiab. A comparative study into distributed load balancing algorithms for cloud computing. In *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*, pages 551–556, April 2010. doi: 10.1109/WAINA.2010.85.

Fabrice Saffre, Richard Tateson, José Halloy, Mark Shackleton, and Jean Louis Deneubourg. Aggregation dynamics in overlay networks and their implications for self-organized distributed applications. *Comput. J.*, 52(4):397–412, July 2009. ISSN 0010-4620. doi: 10.1093/comjnl/bxn017. URL http://dx.doi.org/10.1093/comjnl/bxn017.

Y. Sahu, R.K. Pateriya, and R.K. Gupta. Cloud server optimization with load balancing and green computing techniques using dynamic compare and balance algorithm. In *Computational Intelligence and Communication Networks (CICN), 2013 5th International Conference on*, pages 527–531, Sept 2013. doi: 10.1109/CICN.2013.114.

V. Sesum-Cavic and E. Kuhn. Comparing configurable parameters of swarm intelligence algorithms for dynamic load balancing. In *Self-Adaptive and Self-Organizing Systems Workshop (SASOW), 2010 Fourth IEEE International Conference on*, pages 42–49, Sept 2010. doi: 10.1109/SASOW.2010.12.

G. Soni and M. Kalra. A novel approach for load balancing in cloud data center. In *Advance Computing Conference (IACC), 2014 IEEE International*, pages 807–812, Feb 2014. doi: 10.1109/IAdCC.2014.6779427.

H. Topcuoglu, S. Hariri, and Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *Parallel and Distributed Systems, IEEE Transactions on*, 13(3):260–274, Mar 2002. ISSN 1045-9219. doi: 10.1109/71.993206.

GiorgioLuigi Valentini, Walter Lassonde, SameeUllah Khan, Nasro Min-Allah, SajjadA. Madani, Juan Li, Limin Zhang, Lizhe Wang, Nasir Ghani, Joanna Kolodziej, Hongxiang Li, AlbertY. Zomaya, Cheng-Zhong Xu, Pavan Balaji, Abhinav Vishnu, Fredric Pinel, JohnatanE. Pecero, Dzmitry Kliazovich, and Pascal Bouvry. An overview of energy efficiency techniques in cluster computing systems. *Cluster Computing*, 16(1):3–15, 2013. ISSN 1386-7857. doi: 10.1007/s10586-011-0171-x. URL http://dx.doi.org/10.1007/s10586-011-0171-x.

Mihaela-Andreea Vasile, Florin Pop, Radu-Ioan Tutueanu, Valentin Cristea, and Joanna Koodziej. Resource-aware hybrid scheduling algorithm in heterogeneous distributed computing. *Future Generation Computer Systems*, (0):–, 2014. ISSN 0167-739X. doi: http://dx.doi.org/10.1016/j.future.2014.11.019. URL http://www.sciencedirect.com/science/article/pii/S0167739X14002532.

B. Wickremasinghe, R.N. Calheiros, and R. Buyya. Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 446–452, April 2010. doi: 10.1109/AINA.2010.32.

Jing Yao and He Ju-Hou. Load balancing strategy of cloud computing based on artificial bee algorithm. In *Computing Technology and Information Management (ICCM), 2012 8th International Conference on*, volume 1, pages 185–189, April 2012.