Load Balancing in Cloud Computing

Nicholas M. Plucker
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
pluck011@morris.umn.edu

ABSTRACT

Cloud computing utilizes many computing resources to provide a wide range of services over the internet, rather than on computing hardware one physically controls. Many organizations are moving their infrastructure to the cloud because it reduces expenditure, provides flexibility, and offers a reliable solution for data storage. As cloud computing continues to grow, the demand on cloud infrastructure also increases. Thus, it is necessary that efficient load balancing of tasks take place. Load balancing distributes tasks on virtual machines to reduce costs and maximize performance. This paper examines two proposed load balancing algorithms that shorten completion time and reduce task migrations.

Keywords

Cloud Computing, Load Balancing, Round-Robin

1. INTRODUCTION

The processing of balancing the load of tasks in a cloud computing environment is complex and essential to ensure the reliable performance that cloud computing is known for.

A common use of cloud computing is the hosting of web servers. There is a load balancing system in place that reduces the likelihood that web servers will slow down or crash due to increased traffic. The load balancer distributes the traffic amongst multiple servers to improve performance and response time for the user. Additionally, the need for load balancing is increasing due to the rise of cloud computing [4].

In this paper, we will explore two proposed load balancing algorithms that aim to improve how quickly tasks execute in a cloud environment. The first algorithm, devised by Devi and Uthariaraj [2] (Section 3), expands on weighted round-robin load balancing. The researchers combine static and dynamic load balancing techniques to reduce the number of task migrations and completion time. Simulations using CloudSim were then used to analyze the performance of this algorithm. The second algorithm, devised by Babu and Krishna [1] (Section 4), uses foraging behavior derived from honey bees as the basis for the algorithm. The goal is to reduce the number of task migrations and completion time by migrating tasks from overloaded to underloaded virtual machines. CloudSim was also used in this study.

UMM CSci Senior Seminar Conference, October 2019 Morris, MN.

In addition to the two algorithms discussed in Sections 3 and 4, necessary background is provided in Section 2. This paper then provides conclusions in Section 5.

2. BACKGROUND

In order to understand the two algorithms in this paper, an introduction to key concepts is required. In this section, background is first given on cloud computing followed by load balancing. This background lays the framework for the problem the algorithms are trying to solve. In addition, background is given on the round robin load balancing algorithm in Section 2.3, the idea of a task migration in Section 2.4 and a tool known as CloudSim in Section 2.5.

2.1 Cloud Computing

Cloud Computing utilizes many computing resources to provide a wide range of services over the internet, rather than on computing hardware you physically control. Generally speaking, cloud computing is the term used to describe content and services hosting on servers located in a data center that is connected to the internet. This information is then delivered to the user through a client, such as a web browser or application [5].

Since cloud computing is primarily used as a service, it can be broken up into three distinct service models that build on top of each other: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) [5]. The first layer is IaaS. Rather than using onsite computing hardware, IaaS providers give customers access to computing resources, such as storage, networking and processing, over the internet. In addition, IaaS providers only charge for what you use. Some examples of IaaS providers include DigitalOcean, Amazon EC2 and Microsoft Azure. The second layer is PaaS. PaaS providers give customers the ability to develop and deploy applications on a cloud service instead of directly managing the underlying infrastructure, such as processing capabilities, operating systems, etc. The PaaS provider will control the underlying infrastructure, which simplifies the development process for application developers [5]. Some examples of PaaS providers include Google App Engine, Oracle Cloud Platform and Heroku. The third layer is SaaS. SaaS providers are ones we interact with everyday – Google Apps suite, Office 365, Apple's iCloud, etc. SaaS providers give customers access to software over the internet and in turn, customers may pay a fee for using them.

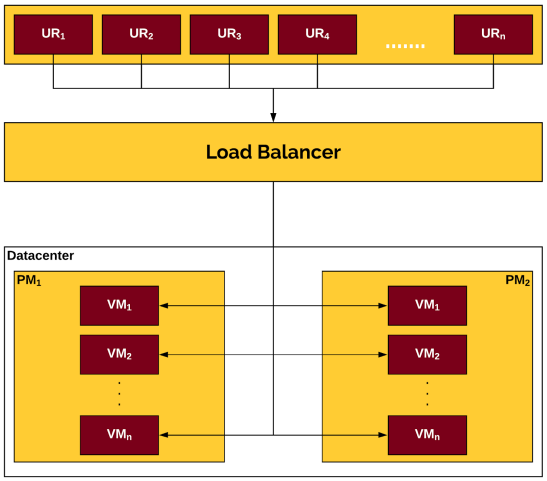

Figure 1: Load Balancing Model based on [4].

2.2 Load Balancing

As cloud computing continues to grow, load balancing is essential to ensure that the quality of service isn't compromised for end users [4]. Load balancing is the process of distributing workload amongst a collection of servers in a data center. It's an essential operation that improves the reliability and performance of a cloud service. Improved reliability is achieved by routing incoming traffic away from unhealthy servers to healthy servers. Improved performance is achieved by routing traffic away from busy servers to idle servers.

In a cloud environment, tasks are run on a virtual machine (VM), an emulation of a physical computer. A VM is hosted on a server and there can be many VMs on a single server. The VMs each have a chunk of physical computing resources allocated to it, such as processing capabilities, RAM, storage, etc., from the server's hardware. For example, if a developer wants to deploy a web server hosted on a cloud service (such as DigitalOcean), it will be hosted on a VM. For a large company, their website may be hosted on multiple VMs across multiple physical servers and potentially spanning multiple data centers. Whenever a user goes to that website, the load balancer will decide which VM gets the request and the VM will respond with the content for the website.

Figure 1 illustrates a load balancing model based on [4]. Users make a request to a cloud service, denoted by $UR_1 ... UR_n$. The first point of arrival is the load balancer. The load balancer decides which VM gets which task. As shown in the diagram, there are two physical machines (servers), denoted as PM_1 and PM_2, that each contain n virtual machines, denoted as $VM_1 ... VM_n$. This resides within a data center.

There are two types of load balancing algorithms: static and dynamic. Static load balancing doesn't depend on the current state of the virtual machine. The distribution of tasks depends on information gathered (processing capabilities, storage capacity etc.) about a virtual machine before run-time. This is potentially problematic as static load balancers can lead to an uneven distribution of resources. This is because the load on the virtual machine can change during run-time and the information gathered prior to run-time can become inaccurate [4]. Dynamic load balancing considers the current state of a virtual machine, but can be complex to implement as more information about the system is required [4].

2.3 Round Robin Algorithm

The Round Robin (RR) algorithm is a widely used static load balancing algorithm [4]. It's a simple algorithm that assigns tasks to VMs in a cyclical manner. For example, suppose there are two VMs and six user requests (tasks). Task 1 will be assigned to VM 1, task 2 will be assigned to VM 2, task 3 to VM 1, task 4 to VM 2 and so on until all tasks are assigned to a VM. RR works well in a homogeneous environment, where the VMs are more or less the same. If some VMs are more capable than others (i.e. more processing capabilities), then the system can become imbalanced [4]. If VM 1 was more capable than VM 2, then VM 2 might become overloaded because it will, on average, be given the same number of tasks as VM 1.

To account for this, a weight can be assigned to a VM. Continuing the example from above, suppose the weight for VM 1 is 5 and the weight for VM 2 is 1, then for every 5 tasks VM 1 can take, VM 2 can take 1. This idea is known as the weighted round robin algorithm – it accounts for the capabilities of each VM.

There is another problem that arises, however. If a task is executing for a long period of time, then the tasks waiting to get executed will also have to wait. At the same time, a task on a VM with less weight may finish.

Both algorithms in this paper aim to prevent this situation from happening. The first algorithm in section 3 aims to effectively allocate tasks before run-time by also considering task length in the load balancing decision. The second algorithm in section 4 migrates tasks from an overloaded VM to an underloaded VM.

2.4 Task Migration

A task migration is the process of moving a task from an overloaded VM to an underloaded VM at run-time before a VM executes that task. This is a common technique used in dynamic load balancing algorithms [4]. Task migrations are an expensive process because it takes time to move a task from one VM to another, therefore the goal in both algorithms in this paper is to minimize the number of task migrations that occur.

2.5 CloudSim

Experimenting with new load balancing techniques in a real cloud environment is not practical because service could potentially be disrupted for users if the new technique proves to be flawed [1]. In addition, it's expensive to implement an actual data center. Therefore, simulations are used to experiment with new load balancing techniques. One such simulation tool is known as CloudSim – a modeling and simulation framework for cloud computing services and infrastructure. It allows researchers to simulate various aspects of a cloud system, such as load balancing and scheduling algorithms. CloudSim is an open source Java-based application available on GitHub[1]. Both studies covered in this paper use CloudSim to analyze the performance of their respective algorithms.

[1] `https://github.com/Cloudslab/cloudsim`

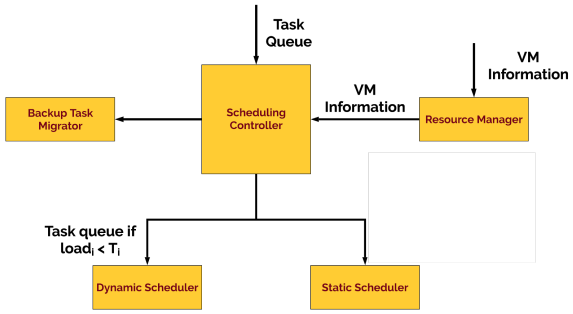

Figure 2: LWRR Architecture based on [2]

3. LENGTH BASED WEIGHTED ROUND ROBIN ALGORITHM

The first algorithm discussed, known as Length Based Weighted Round Robin (LWRR), was developed by Devi and Uthariaraj [2]. It improves upon weighted round robin by considering the length of the task when making load balancing decisions. LWRR combines static and dynamic load balancing techniques to effectively allocate tasks to VMs before run time, reducing the need for task migrations and improving the overall completion time.

Section 3.1 discusses the architecture of LWRR, Section 3.2 introduces the idea of a threshold value and how it influences the load balancing decision and Section 3.3 discusses the simulation results of LWRR.

3.1 LWRR Architecture

There are five main components to LWRR, as shown in Figure 2. When tasks arrive at the load balancer, they go to the scheduling controller. Its job is to determine which of the two schedulers will be used to schedule the current task. The resource manager gathers information about each VM, such as the current load and total capacity and passes it to the scheduling controller.

LWRR combines static and dynamic load balancing techniques to make VM allocation decisions. As mentioned in Section 2.2, static load balancing doesn't depend on the current state of the system. It makes placement decisions based on information gathered before run-time, such as a VM's maximum capacity, its processing capabilities etc. Eventually, an imbalance will likely occur, causing some VMs to be overloaded and some to be underloaded. For LWRR, when an imbalance occurs, the scheduling controller switches to the dynamic scheduler. The dynamic scheduler takes into account the current state of the system and can make better decisions on where to place new tasks with a VM. An imbalance occurs when the current load on a VM is less than its threshold value, which will be discussed more in Section 3.2. New tasks will be placed to the underloaded VM until the load is above that VM's threshold value. When the dynamic and static schedulers are used in conjunction, tasks are effectively allocated to VMs without the need for further task migrations [2].

In the event that there is a further imbalance in the system, the backup task migrator takes over. It's run after every task executes and if it finds any further imbalance in the system that wasn't taken care of with the static and dynamic schedulers, it migrates a task from and overloaded VM to an underloaded VM. As discussed in Section 2.4, the fewer task migrations, the better. The researchers use this component to analyze how many task migrations take place when using LWRR in comparison to using weighted round robin and plain round robin.

3.2 Threshold Value

The *threshold value* is the determining factor for which scheduler is used for VM placement. If the current load on any VM falls below its threshold value, the dynamic is scheduler is used until the load is above the threshold value. Each VM has its own unique threshold value. The details of the threshold value are described below.

The sum of the current loads on all VMs is defined as:

$$L = \sum_{i=1}^{k} l_i \tag{1}$$

where i is the VM number, k is the number of VMs in the system, and l_i is the current load on VM i.

The sum of capacities for all VMs defined as:

$$C = \sum_{i=1}^{k} c_i \tag{2}$$

where i is the VM number, k is the number of VMs in the system and c_i is the capacity of VM i.

The load per unit capacity (LPC) defined as:

$$LPC = \frac{L}{C} \tag{3}$$

The threshold value for each VM i is then defined as:

$$T_i = LPC * c_i \tag{4}$$

where c_i is the capacity for VM i.

Each VM has its own unique threshold value that is calculated every time a task is added or removed from a VM. The resource manager calculates l_i and c_i which are then passed to the scheduling controller for the threshold value calculation.

3.3 Results

In this study, the researchers use CloudSim. The researchers analyze LWRR based on two criteria, task migrations and overall completion time, using homogeneous and heterogeneous tasks in a homogeneous environment. Task length is known ahead of time. For heterogeneous tasks analysis, task length is randomly chosen to be between 500,000 and 200,000,000 executed instructions. The researchers compare LWRR against weighted round robin and plain round robin.

Figures 3 and 4 compare the number of task migrations to the number of VMs for both heterogeneous and homogeneous tasks, respectively. LWRR is compared against round robin (RR) and weighted round robin (WRR) algorithms. The number of tasks used is not specified in the study. The x-axis represents the number of VMs, ranging from 10 to 100 in increments of 10 for both Figures 3 and 4. The y-axis represents the number of task migrations. Note that the units range from 0 to 20 in Figure 3 and from 0 to 15 in Figure 4.

Both RR and WRR perform well when tasks are the same (homogeneous), therefore there are more task migrations when tasks are of different type (heterogeneous). Since

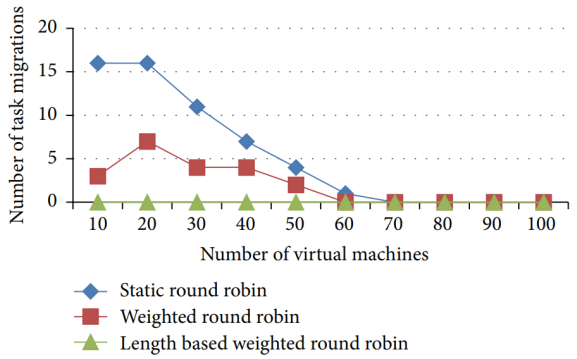

Figure 3: Number of Heterogeneous Task Migrations vs Number of VMs [2]

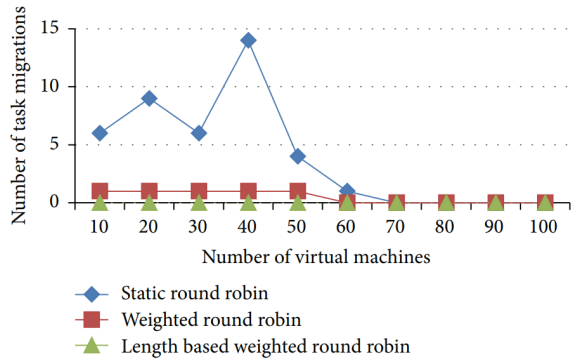

Figure 4: Number of Homogeneous Task Migrations vs Number of VMs [2]

LWRR effectively allocates tasks to VMs without the need for task migrations, the number of task migrations is minimal. In Figure 3, the number of task migrations for RR with 10 VMs is 16, as compared to 6 in Figure 4. For WRR, the number of task migrations in Figure 3 is 3, as compared to 1 in Figure 4. The number of task migrations for all three algorithms converge to zero when there are ≥ 70 VMs, which suggests that there are more VMs than tasks at that point.

Figures 5 and 6 compare the overall completion time to the number of VMs for both heterogeneous and homogeneous tasks, respectively. LWRR is compared against RR and WRR. The y-axis represents the overall completion time in seconds ($\times 10^4$), ranging from 0 to 15 in increments of 5. The x-axis represents the number of VMs, ranging from 10 to 100 in increments of 10. For both Figures 5 and 6, it is clear that LWRR performs better than RR and WRR. This is because LWRR allocates tasks to VMs better than RR and WRR, therefore tasks will complete faster due to less waiting time.

4. HONEY BEE BEHAVIOR INSPIRED LOAD BALANCING

The second proposed algorithm, developed by Babu and Krishna [1] is derived from the foraging behavior of honey bees. The researchers base this algorithm on work done by Johnson and Nieh [3], in which they describe honey bees as

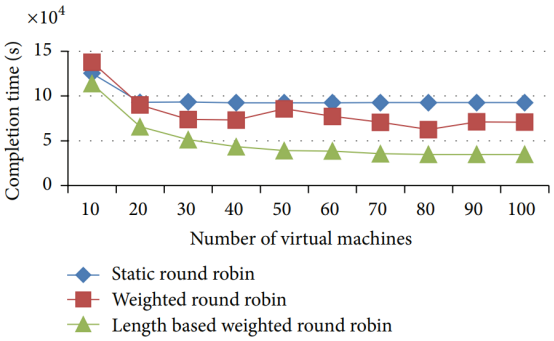

Figure 5: Completion Time vs Number of VMs with Heterogeneous Tasks [2]

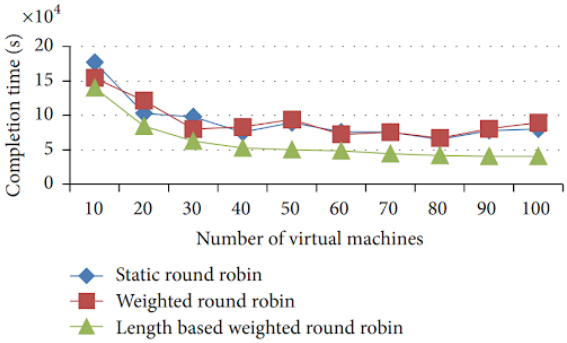

Figure 6: Completion Time vs Number of VMs with Homogeneous Tasks [2]

social insects that communicate using a network of feedback cycles.

The algorithm in this work, known as Honey Bee Behavior Inspired Load Balancing (HBB-LB), differs from LWRR in that it identifies tasks in an overloaded VM and migrates them to an underloaded VM at run-time. The goal is to reduce the number of task migrations and completion time.

Section 4.1 describes the idea of honey bee foraging behavior and how it relates to a cloud environment, Section 4.2 analyzes the HBB-LB algorithm and Section 4.3 analyzes the results of HBB-LB on two criteria: task migrations and completion time.

4.1 Honey Bee Behavior

The HBB-LB algorithm is based on honey bee foraging behavior as described in the work by Johnson and Nieh [3]. They describe bees as social insects that communicate using a network of feedback cycles. Honey bees, for example, have several roles within a colony. Once such role, and perhaps the type we see the most, are the forager and scout bees. A scout bee's job is to search for a food source. Once a suitable food source is found, the scout bee returns to the hive and does a dance. This dance notifies other bees in the hive where the food source is located, the quality and quantity of the food source, etc. A forager bee then follows the scout bee's directions to the food source that was discovered. The forager bee's job is to then collect the food, return to the hive, dance and drop off the food. This dance tells other scout and forager bees the status of a food source, i.e if the

food source is depleted, they should abandon it and find a new food source [1].

A honey bee can be thought of as a task, and its food source is considered a VM. A honey bee foraging a food source can be thought of as a task being assigned to a VM. In addition, a honey bee running out of food at a food source would be a VM in an overloaded state. Finally, a scout bee finding a new food source would be a task being moved from an overloaded VM to an underloaded VM. The hive in this analogy would be the load balancer itself. A resource manager, similar to that in Section 3.1, calculates information about each VM. The resource manager can be thought of as doing a dance whenever a VM is identified as underloaded or overloaded. This causes the load balancer to find a new, underloaded VM for a task to move to and migrates it accordingly [1].

Algorithm 1 HBB-LB Algorithm based on [1]

```
1: Find capacity and loads of all VMs
2: if Load > Maximum Capacity then exit
3: if σ ≤ T_s then exit
4: while both UVM and OVM are not empty do
5:     Group VMs based on load:
6:         Underloaded VMs (UVM)
7:         Balanced VMs (BVM)
8:         Overloaded VMs (OVM)
9:     Sort VMs in UVM set in ascending order by load
10:    Sort VMs in OVM set in descending order by load
11:    Remove a task from first VM in OVM set
12:    Assign task to first VM in UVM set
13:    Update UVM and OVM sets
```

4.2 HBB-LB Algorithm

The HBB-LB algorithm is illustrated, at a high level, in Algorithm 1. It's important to note that HBB-LB isn't concerned with the initial placement of tasks, rather it's a task migration algorithm. First, line 1 finds the capacity and loads of all VMs. After that, there are two checks that take place to determine if load balancing is necessary. In line 2, if the total load in the system is greater than the maximum capacity in the system, then the entire system is overloaded and load balancing won't help. In line 3, if the standard deviation (σ) of the loads in the system is less than the threshold condition set (T_s), then the system is balanced and load balancing is not necessary. After these checks, load balancing takes place.

Lines 4-13 illustrate the load balancing logic. Lines 5-8 group VMs based on their load – underloaded VMs (UVM), balanced VMs (BVM) and overloaded VMs (OVM). Line 9 sorts the VMs in the UVM set in ascending order by load, such that the VMs that are the most underloaded will be first. Line 10 sorts the VMs in the OVM set in descending order by load, such that the VMs that are the most overloaded will be first. Line 11 removes a task from the first VM in the OVM set (the most overloaded VM) and line 12 assigns that task to the first VM in the UVM set (the most underloaded VM). Line 13 updates the UVM and OVM sets. This is an important step as lines 11 and 12 could cause the VMs in either set to become balanced, thus removing the VM from the set.

Lines 4-13 then repeat until either the UVM or OVM set is empty. This process runs after every task executes.

4.3 Results

In this study, the researchers used CloudSim. They analyzed HBB-LB based on two criteria: task migrations and overall completion time. As mentioned in Section 2.4, a task migration is the process of moving a task from an overloaded to an underloaded VM. The overall completion time is the time the system takes for all tasks to finish executing. The goal is to reduce the number of task migrations and overall completion time. Figures 7 and 8 compare the number of task migrations vs number of tasks for 4 VMs and 7 VMs, respectively. HBB-LB is compared against dynamic load balancing (DLB) and dynamic load balancing using a heuristic (HDLB). DLB and HDLB aren't described in detail; it's only mentioned that they are existing load balancing algorithms [1]. The x-axis represents the number of tasks, ranging from 10 to 40 in increments of 5 for both Figures 7 and 8. The y-axis represents the number of task migrations. Note that the units range from 3 to 12 task migrations in Figure 7 and from 0 to 7 task migrations in Figure 8.

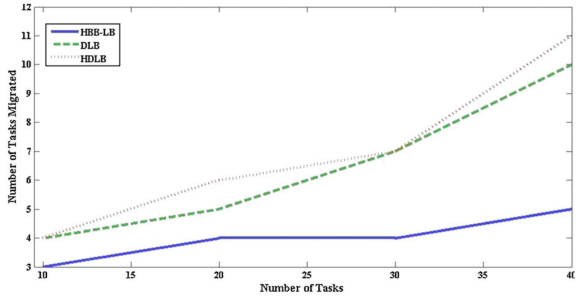

Figure 7: Number of Task Migrations vs Number of Tasks for 4 VMs [1]

.

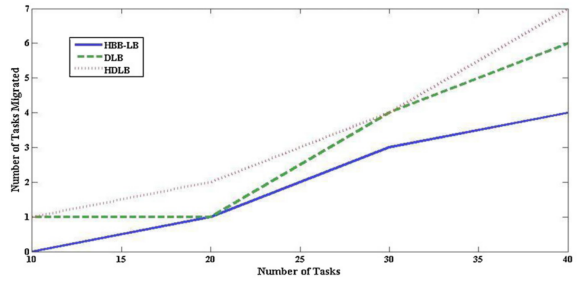

Figure 8: Number of Task Migrations vs Number of Tasks for 7 VMs [1]

.

In both figures, it is clear that HBB-LB is better than DLB and HDLB. The maximum number of task migrations for HBB-LB when there are 4 VMs (Figure 7) and a maximum number of tasks (40 in this experiment) is 5, as compared to 10 for DLB and 11 for HDLB. When there are 7 VMs (Figure 8), the maximum number of task migrations for HBB-LB in this scenario is 4, as compared to 6 for DLB and 7 for HDLB.

It's important to note that when the number of VMs increase, the number of task migrations decrease, as long as the number of tasks stay constant. This is because there would be a smaller degree of imbalance between the VMs, thus making the number of task migrations necessary to balance the system lower. If there were 40 VMs and 40 tasks, there would be no task migrations because each task would have its own VM. This is not feasible in a real-world setting however, because the number of tasks will outnumber the number of VMs.

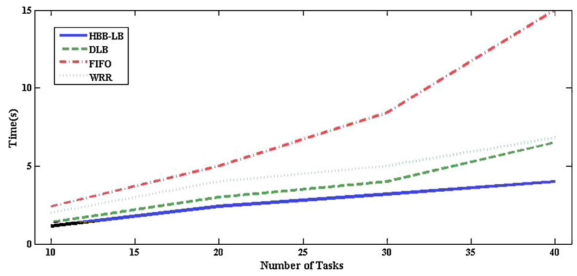

Figure 9: Overall Completion Time vs Number of Tasks [1]

.

Figure 9 compares the overall completion time with the number of tasks. HBB-LB is compared against dynamic load balancing (DLB), first-in-first-out load balancing (FIFO) (also known as round robin), and weighted round robin (WRR). The x-axis represents the number of tasks, ranging from 10 to 40 in increments of 5. The y-axis represents the overall completion time in seconds, ranging from 0 to 15 in increments of 5.

It is clear that HBB-LB performs better than DLB, FIFO and WRR. When there are a small number of tasks (such as 10), the completion time for all four algorithms is 2 $\pm$ 1 seconds. When the number of tasks increase, however, the variance in completion times between the algorithms increases. For 40 tasks, the completion time for HBB-LB is 4 seconds, as compared to 6.5 seconds for DLB, 7 seconds for WRR and 15 seconds for FIFO.

5. CONCLUSIONS

Load balancing is the process of distributing workload amongst multiple servers. This increases reliability and performance by routing incoming tasks away from unhealthy servers and balancing tasks amongst multiple healthy servers. This increases the likelihood that cloud servers run efficiently and uninterrupted. As discussed in Section 3, LWRR improves load balancing by using a combination of static and dynamic load balancing techniques to distribute load at arrival time. As a result, LWRR minimizes task migrations in the experiments presented in this paper and decreases the overall completion time. In Section 4, an algorithm based on the foraging behavior of honey bees is discussed and analyzed. It identifies VMs as either underloaded, balanced or overloaded and migrates tasks from an overloaded VM to an underloaded VM until the system is balanced. This process minimizes task migrations and overall completion time

While both algorithms aren't directly compared to one another, it's useful to consider how the two algorithms might compare. As discussed in both algorithms, the number of task migrations is an important metric in determining the performance of the algorithm. Since task migrations are expensive, the goal is to reduce the number of them. Both LWRR and HBB-LB do this, but I believe that LWRR would perform better than HBB-LB because LWRR has almost zero task migrations where HBB-LB still has some, although the number is minimal. This is due to the idea that LWRR initially allocates tasks to VMs in such a way that very few task migrations are necessary, whereas the entire idea of HBB-LB is to migrate tasks.

Acknowledgments

I would like to thank my advisor Nic McPhee and my senior seminar instructor Elena Machkasvoa for their feedback, flexibility and support. I would also like to thank Laverne Schrock for his time and review of this paper.

6. REFERENCES

[1] D. Babu and P. Venkata Krishna. Honey bee behavior inspired load balancing of tasks in cloud computing environments. *Appl. Soft Comput.*, 13(5):2292–2303, May 2013.

[2] D. Devi and V. Uthariaraj. Load balancing in cloud computing environment using improved weighted round robin algorithm for nonpreemptive dependent tasks. *The Scientific World Journal*, 2016.

[3] B. R. Johnson and J. C. Nieh. Modeling the adaptive role of negative signaling in honey bee intraspecific competition. *Journal of Insect Behavior*, 23(6):459–471, Nov 2010.

[4] P. Kumar and R. Kumar. Issues and challenges of load balancing techniques in cloud computing: A survey. *ACM Comput. Surv.*, 51(6):120:1–120:35, Feb. 2019.

[5] Wikipedia. Cloud computing — Wikipedia, the free encyclopedia, 2019. `https://en.wikipedia.org/wiki/Cloud_computing` [Online; accessed 04-March-2019].