A Decentralized Approach for Homogenizing Load Distribution

Disha Sangar[**]
s.dish92@hotmail.com
Oslo Metropolitan University (OsloMet)
Oslo, Norway

Håvard Haugerud[††]
haugerud@oslomet.no
Oslo Metropolitan University (OsloMet)
Oslo, Norway

Anis Yazidi[†]
anisy@oslomet.no
Oslo Metropolitan University (OsloMet)
Oslo, Norway

Kyrre Begnum[†]
kyrre@oslomet.no
Oslo Metropolitan University (OsloMet)
Oslo, Norway

ABSTRACT

Running a sheer virtualized data center with the help of *Virtual Machines* (VM) is the de facto-standard in modern data centers. Live migration offers immense flexibility opportunities as it endows the system administrators with tools to seamlessly move VMs across physical machines. Several studies have shown that the resource utilization within a data center is not homogeneous across the physical servers. Load imbalance situations are observed where a significant portion of servers are either in overloaded or underloaded states. Apart from leading to inefficient usage of energy by underloaded servers, this might lead to serious QoS degradation issues in the overloaded servers.

In this paper, we propose a lightweight decentralized solution for homogenizing the load across different machines in a data center. In search of better solutions, we have looked outside the field of computer science for inspiration. Inspired by Nobel Peace Prize winners Alvin Roth and Lloyd Shapley's work on Stable Matching [4], we borrow the concept of stable marriage matching problems where we pair pairs of underloaded servers and overloaded servers based on some notion of preferences for the purpose of homogenizing their load through exchange of VMs. Furthermore, our solution is distributed by accommodating this aspect in the original Stable Matching algorithm. We provide some real-life experimental results that demonstrate the efficiency of our approach.

CCS CONCEPTS

• **Hardware → Power estimation and optimization**; Enterprise level and data centers power issues; • **Software and its engineering** → *Virtual machines*;

KEYWORDS

Self-Organization, cloud computing, Stable Marriage, Distributed Load Balancing.

1 INTRODUCTION

Major systems and Internet based services have grown to such a scale that we now use the term âĂIJhyper scaleâĂİ to describe them. Furthermore, hyper scale architectures are often deployed in cloud based environments, which offers a flexible pay-as-you-go model.

From a system administratorâĂŹs perspective, optimizing a hyper scale solution implies introducing system behaviour that can yield automated reactions to changes in configurations and fault occurrences. For instance, auto scaling is a desired behaviour model for websites to optimize cost and performance in accordance to usage patterns.

There are two different perspectives on how an automated behaviour can be implemented within the field of cloud computing. One of the perspectives is to implement the behaviour in the infrastructure, which is the paradigm embraced by the industry. The other alternative is to introduce behaviour as a part of the Virtual Machine (VM), which opens up a possibility for cloud independent models.

Several studies have shown that the resource utilization within a data center varies drastically across the physical servers. Load imbalance situations are observed where a significant portion of servers are either in overloaded or underloaded states. Apart from leading to inefficient usage of energy by the presence of an underloaded servers, this might lead to serious QoS degradation issues in the overloaded servers. The aim of this paper is to present an efficient and yet simple solution for homogenizing the load in data centers. The potential gain with this research is to find an efficient and less complex way of operating a data center. Stable Marriage is a an intriguing theory emanating from the field of economy and holds many promises in the field of computer science and more particularly in the field of cloud management. In this paper, we apply the theory of Stable Marriage matching in order to devise a load homogenizing scheme within data center. We also modify the original algorithm in order to support distributed execution.

Various studies on self-organizing approaches have been emerging in the recent years to efficiently solve computationally hard problems where centralized solutions might not scale or might also create a single point a failure.

The aim of this paper is to provide a distributed solution for achieving distributed load balancing in a data center which is inspired by the the Stable Marriage algorithm [4]. The algorithm implements message exchange between pairs of servers. It is worth emphasizing that modern distributed systems often use gossip protocols to solve problems that might be difficult to solve in other ways, either because the underlying network has an inconvenient structure, is extremely large, or because gossip solutions are the most efficient ones available [5] in terms of communication. We

[*]Author contributed significantly
[†]This Author and all other authors also contributed

shall present two variants of the Stable Marriage algorithm and study their behavior under different scales.

2 STABLE MATCHING

According to Shapley et al. [3] an allocation where no individual perceives any gain from any further trade is called *Stable*. Stability is a central theory in the field of cooperative game theory that emanates from mathematical economics which seeks to know how any constellation of rational individuals might cooperate to choose an allocation.

Shapley [2] introduced the concept of *pairwise matching*. Pairwise matching studies how individuals can be paired up when they all have different preferences regarding who are their best matches. The matching was analyzed at an abstract level where the idea of marriage was used as an illustrative example.

For this experiment Shapley et al. tested how ten women and ten men should be matched, while respecting their individual preferences. The main challenge was to find a simple method that would lead to stable matching, where no couples would break up and form new matches which would make them better off. The solution was *deferred acceptance*, a simple set of rules that always led straight to the stable matching.

Deferred acceptance can be set up in two different ways, either men propose to women or women propose to men. If women propose to men the process begins with each woman proposing to the man she likes the best. Each man then looks at the different proposals he has received, if any, and regards the best proposal and rejects the others. The women who were rejected in the first round, then move along to propose to their second best choice. This will continue in a loop until no women wants to make any further proposals. Shapley et al. [3] proved this algorithm mathematically and showed that this algorithm always leads to stable matching.

The specific way the algorithm was set up turned out to have an important distributional consequence. The outcome of the algorithm might differ significantly depending on whether the right to propose was given to the women or to the men. If men proposed this lead to the worst outcome from the womenâĂŹs perspective. This is because if women proposed, no woman would be worse off than if the men had been given the right to propose [3].

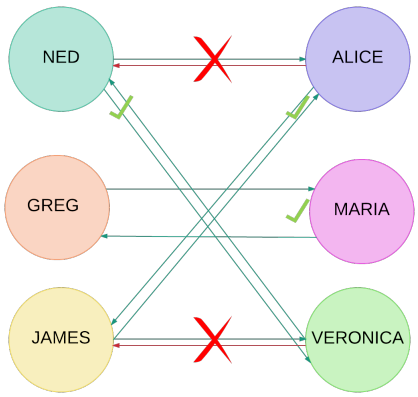

Figure 1: Stable Matching

The model depicted in Figure 1 presents the selection process for Stable Matching. On the right side we find the women with their preferences and to the left the men with their respective preferences.

3 RELEVANT RESEARCH

In this section, we shall review some prominent works on distributed approaches for homogenizing the load in a data center. It is worth mentioning that the related work in this particular area is rather sparse.

Rao et al. [6] present three different but simple techniques to achieve load balancing in a structured system. The three different approaches balance the load by migrating nodes from one place to another. The first technique called one-to-one, picks two random virtual machines, where one is a heavy node and the other one is a light node. Each light node can periodically pick a random ID and then perform a look up operation to find the node that is responsible for that ID. If that node is a heavy node, then a transfer may take place between the two nodes. The second scheme is the "One-to-many" scheme. According to this scheme, a heavy node is allowed to consider more than one light node at the time and migrate to the lightest one after choice. The third and last technique is the many-to-many scheme, which is an extension of the first two schemes. In this method, there is a concept of a global pool where each heavy node drops off their weight. This happens over three phases; unload, insert and dislodge. Without going into much detail, heavy nodes drop their "weight" and unload it until they become a light node. The idea is to transfer all virtual servers from the pool to light nodes without creating any new heavy nodes.

Marzolla et al. [5] propose a decentralized gossip-based algorithm, called *V-MAN* to address to the issues regarding consolidation. V-MAN is invoked periodically to create consolidate VMs into fewer servers. They assume that the cloud system has a communication layer, so that any pair of servers can exchange messages between them. The work of Marzolla et al. [5] yields very promising results which show that using V-MAN converges fast âĂŞ after less than 5 rounds of message exchanging between the servers.

In [1], the authors use scouts which are allowed to move from one PM (physical machine) to another âĂŞ to be able to recognize which compute node might be a suitable migration destination for a VM. This is completely opposite of what V-MAN does. It does not rely on any subset like scouts, instead each server can individually cooperate to identify a new VM location, which makes V-MAN scalable. It is also to be noted that any server can leave or join the cloud at any time.

Sedaghat et al [7] use a Peer-to-Peer protocol to achieve energy efficiency and increase the resource utilization. The Peer-to-Peer protocol provides mechanisms for nodes to join, leave, publish or search for a resource-object in the overlay or network. This solution also considers multi-dimensionality âĂŞ because the algorithm needs to be specified to be dimension aware, each PMs proportionality should be considered. Each node is a peer where a peer sampling service, known as newscast, provides each peer with a list of peers whom are to be considered neighbours. Each peer only know k random neighbours which map its local view. The aim

is to improve a common value which is defined as the total imbalance of each pair at the time of decision-making by redistributing the VMs.The work uses a modified dimension aware algorithm to tackle the multi-dimensional problem. The algorithm is iterative and starts from an arbitrary VM placement. When the algorithm converges, a reconfiguration plan is set so the migration of the VMs can start.

4 SOLUTION

4.1 Overview of a functioning framework

As the algorithm implemented will be based on a real life inspiration, it is important to understand that the outcome can end in two different cases. Just as each relationship does not end in marriage neither will the decision of the PMs. Each PM can be viewed as individuals making their own âĂIJlife choicesâĂİ.

Figure 2 and 3 enhances the different outcomes the algorithm can opt for and how the framework is set up to work around the execution of the algorithm. Note that the environment later implemented is not in an actual data center. However, for the sake of showing that the main goal is to achieve load balance in a distributed cloud data center, the intention is to create a framework that may work in any given scenario and setup.

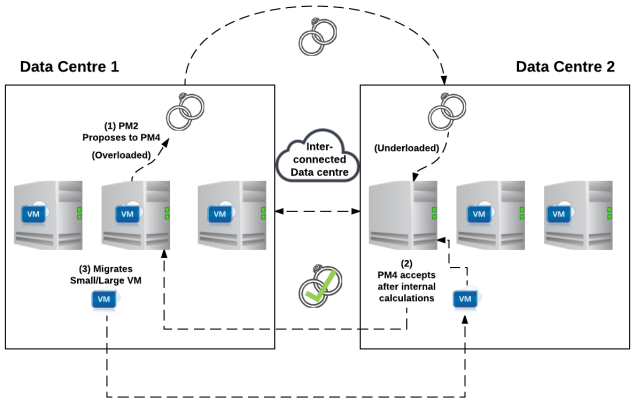

Figure 2: Proposal accepted

The basic framework for both scenarios are the same, it is a data center consisting of PMs with different weight. However, as explained in the section above, based on the calculations of the underloaded server in the second scenario the proposal is rejected and the PM moves on to the next best on their list. This process is supposed to be a continuous process, unless the target load for each PM is achieved, then the process stops entirely.

4.2 Bin Packing with Stable Marriage

The bin packing problem is the challenge of packing an amount n of objects in to as few bins as possible. In this case, the servers are the bins and the VMs are the objects. This terminology fits the stable marriage algorithm well, as the bins are the humans who are in quest of a partner (bin) which represents a good match while satisfying some constraints in terms of capacity. A constraint can

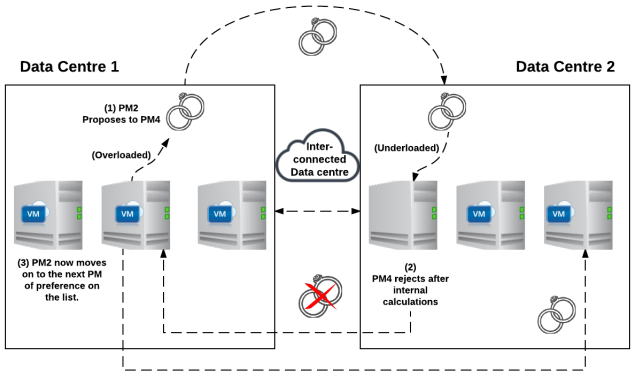

Figure 3: Proposal Rejected

be defined in many ways, but for a bin some common constraints would be the height of the box, its width and depth. Our algorithm will focus on Virtual CPUs (VCPUs) and memory as constraints. The aim of the algorithm is to even and equalize the load of the data center by evenly distribute the weight of the VMs across the bins in such a way that the bins should neither be overfilled nor underfilled.

4.3 Stable Marriage Animation

A known set of servers is divided into two groups of overutilized (men) and underutilized (women). The goal is to find a perfect match for the overutilized servers. The matchmaking is based on three values, the average CPU load, the imbalance before and after migration (calculated before the eventual migration) and the profit of such a marriage.

The figures below demonstrate the expected outcome of implementing the Stable Marriage algorithm. This approach is mainly centralized and the PMs know the allocated values of each other. This means that each PM, both over and underutilized, has a list of preferred men and women they want to propose to or receive a proposal from.

Figure 4 shows that PM1 has reached full capacity as marked by the red line. The red line represents the average capacity that each PM can handle. Assume that each group of men can only handle four or six full servers, in this case PM1 has then reached its full capacity and so has PM2. They need to migrate the load to a underloaded PM of preference, so that they can balance the load equally. Hence, the overloaded server PM1 proposes to his first choice, PM3.

The female set of servers have their own method to calculate the advantage/disadvantage of such a marriage. If the underutilized PM calculates a higher imbalance than before the marriage, she sees this as a disadvantage and rejects the proposal. This method also avoids that the proposing party becomes underutilized in the future.

After being rejected, PM1 proceeds to his next best choice which is PM4. PM4 then calculates the imbalance before and after the proposed marriage to check if it improves after a potential migration.

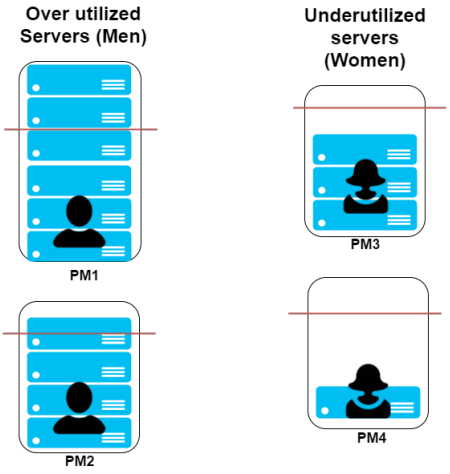

Figure 4: Set of over/under utilized servers

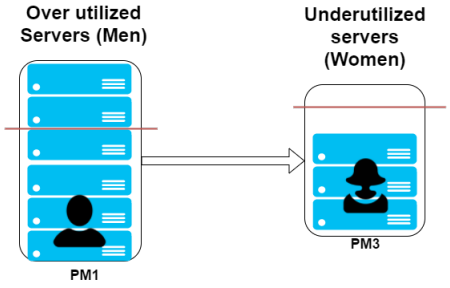

Figure 5: PM1 proposes to PM3

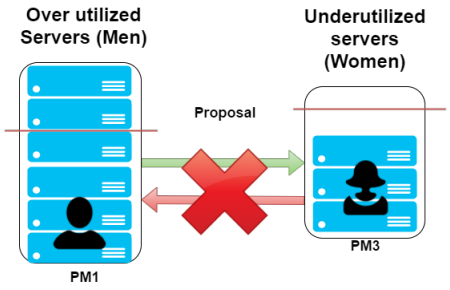

Figure 6: PM3 rejects PM1 seeing no benefit to this marriage.

In this case the imbalance factor improves, and PM4 accepts the proposal. The migration can now take place.

Since PM4 has the same amount of capacity to accept load, the server is not over-utilized and the load has been balanced between the married PMs.

This particular animation doesn't have any scheme implemented, it just gives an idea of how the algorithm is supposed to work. The schemes will only make a difference in terms of the size of the VMs that is migrated. The figure below shows how the VM sizes may

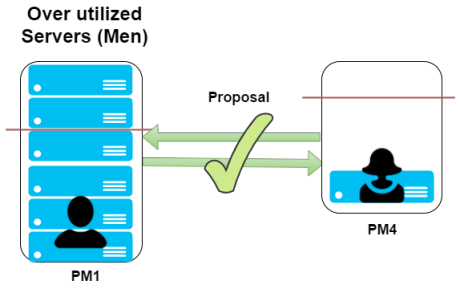

Figure 7: PM4 accepts PM1's proposal

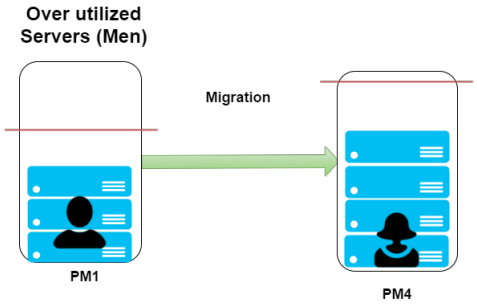

Figure 8: Migration successful

differ on each PM and how the migration process may look inside each server.

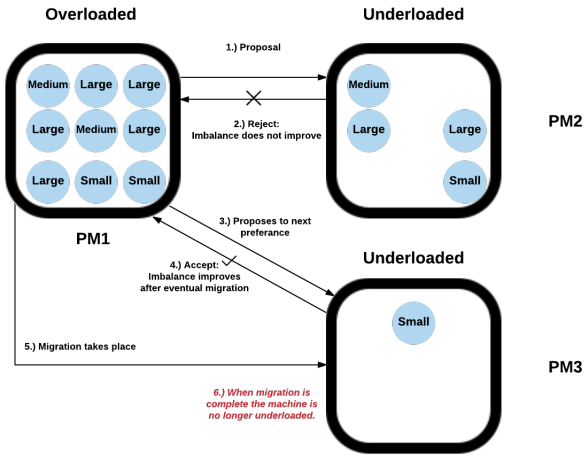

Figure 9: PM with various VM sizes

4.4 Stable Marriage Algorithm - Migrate Largest and Migrate Smallest

The two different schemes that will be implemented are *Migrate Largest* and *Migrate Smallest*. These two schemes are very similar,

but depend on the size of the VM. As shown in Figure 10, the resources allocated to the VMs are different. In fact, we tested in our experiments either to migrate the largest VM in the overloaded PM or the smallest VM in the overloaded PM. As will be explained in the next section, both schemes lead to a reduction in the overall imbalance, however, usually migrating smallest VM first results in a better overall result at the cost of a higher number of migrations. Migrating Largest VM first requires less number of migrations to reach a final state where no more gain can be achieved by the algorithm in terms of imbalance. However, this takes place at the cost of higher final imbalance compared to Migrating Smallest First.

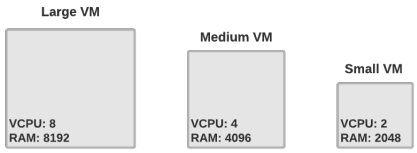

Figure 10: VMs with their allocated values

5 IMPLEMENTATION OF STABLE MARRIAGE

In this section we give details about the implementation of the Stable Marriage algorithm which aims to find the perfect match for gaining load balance. Each node is considered an individual with preferences and demands. These are taken into consideration to be able to find the perfect balance for each individual node.

The flow diagram in Figure 11 gives an insight into how the Stable Marriage Algorithm operates and the different procedures involved.

The following describes the most important parameters of the algorithm. We define the CPU load to simply be the number of VCPUs of a VM. Let C_i be the total CPU load of server i contained in its VMs and let c_j be the number of VCPUs assigned to VM_j:

$$C_i = \sum_{VM_j} c_j$$

Each server i has a maximum CPU capacity ${C_i}^{max}$ which is its number of physical CPUs and we define this to be the maximum number of VCPUs a server can contain:

$$C_i \leq {C_i}^{max}$$

When it comes to consolidation, most algorithms take into account the bottleneck resource as a sole criterion for achieving better consolidation decisions. Similarly, when it comes to load balancing, one can base the algorithm on the imbalanced resource whether it is CPU or memory. For the sake of simplicity, we assume that the CPU is the most imbalanced resource in our data center, which in real life is often the case.

Average load $\overline{C}$ is defined as the average CPU load of the N Physical Machines:

$$\overline{C} = \sum_{PM_i} C_i/N = \sum_{i=1}^{N} C_i/N$$

If the system was perfectly balanced and all servers of same size, each server would have this number of VCPUs. Furthermore, we define the average capacity of a server as

$$\overline{{C^{max}}} = \sum_{PM_i} {C_i}^{max}/N$$

We define the target load to be the result when evenly distributing the load according to the capacity of the servers. Let T_i be the the target load at PM_i when there is no imbalance:

$$T_i = \frac{{C_i}^{max}}{\overline{{C^{max}}}}\overline{C}$$

If all the machines have the same capacity, this would reduce to equal load on each server:

$$T_i = \overline{C} = \sum_{PM_i} C_i/N$$

We define the imbalance I_i of a server or physical machine PM_i in terms of CPU load as the deviation of the load of machine PM_i from the target CPU load:

$$I_i = |T_i - C_i|$$

The following pseudo code shows how the possible gain of a migration between an overloaded server and an underloaded server is calculated:

```
Gain_of_Migration_Couple

# Calculate imbalance before an eventual migration
<calculate imbalance of overloaded server>
<calculate imbalance of underoaded server>

Ib = I overloaded + I underloaded

# Calculate imbalance after an eventual migration
# Assume moving largest VM from overloaded
# to underloaded server

<calculate imbalance of overloaded server>
<calculate imbalance of underoaded server>

Ia = I overloaded + I underloaded

# Calculate the gain

gain = Ib - Ia

# Positive result shows a gain
```

Please note that as explained in the previous section, we have two different schemes: migrate largest VM first and migrate smallest VM first. In fact, we tested in our experiments both schemes: largest VM in the overloaded PM or the smallest VM in the overloaded PM. Intuitively, migrating large VM first would result in a smaller number of migrations before the algorithm stops. While migrating small VM first would result in larger number of migrations but a better final overall result in terms of balance. In more informal terms, we could choose to operate with small changes in the data center by migrating the smallest VMs in the overloaded PMs or we

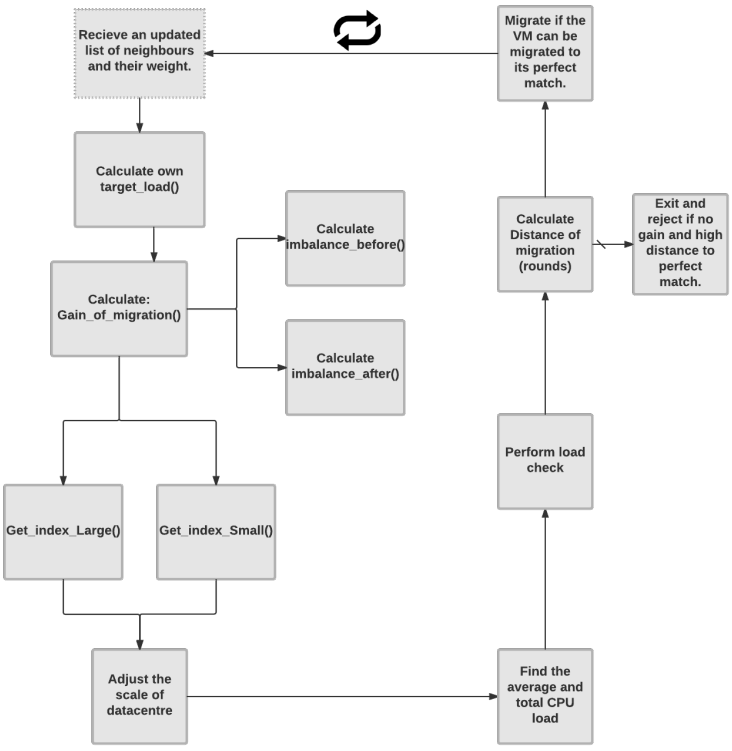

Figure 11: Flow Diagram of the Stable Marriage Implementation

could operate with more crude and less cautious steps by making larger changes in the data center by migrating the largest VMs.

The Stable Marriage algorithm operates in rounds and it stops when no more âĂIJgainâĂİ in terms of reducing imbalance can be achieved. The algorithm will then exit. In other terms, if there is no beneficial proposal that reduces the imbalance or the proposals will increase the imbalance, the algorithm will stop whenever there are no possibilities to reduce further imbalance. It also restricts overloaded servers to become underloaded, which means that PMs may also decline a proposal if the overloaded server becomes underloaded. This means that a node can never become overbalanced again or underbalanced to take more VMs on board. This is an important part of the implementation, as the point of the Stable Marriage algorithm is to *stabilize* the system, this algorithm contributes to the stability factor.

6 EXPERIMENTS

6.1 Experimental Set-up

Figure 12 is a model which gives an overview of the structure in which the project will be implemented. This is a figure which shows how the different components from entirely different worlds are paired together. The bottom layer is the physical hardware consisting of PM1-PM3 or Lab01-Lab03 which are the assigned name on the OS. This layer is controlled by the hypervisor KVM, which is in control of the virtual environment, also the network of VMs which are later spawned in layer 3.

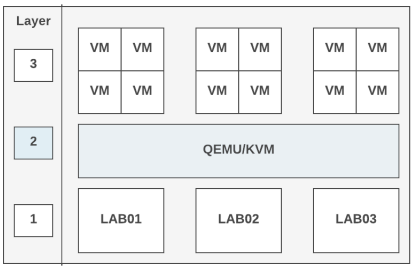

Figure 12: Design

6.2 Environment Configuration

Evidently, a framework is built with several services and components, which are necessary for an environment to work. To set up a virtual environment for this project several physical and virtual technologies were necessary.

The physical servers in this project are stored in a server room at Oslo Metropolitan University. There are three dedicated servers for this project, as seen in figure 13. The setup consists of a dedicated gateway to connect to the outside. All of the PMs are interconnected through a dedicated switch.

Each server is allocated with same specifications:

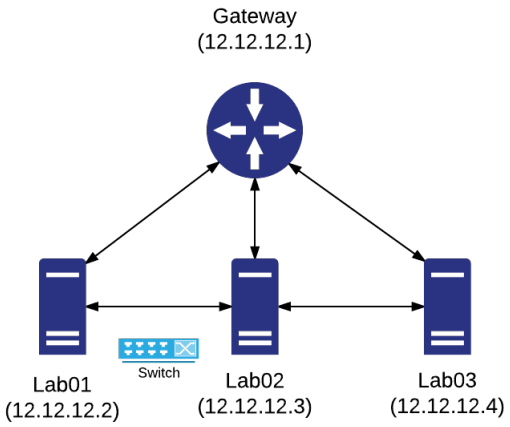

Figure 13: Overview of the Physical lab structure

Hardware:	Design:
Processor	Intel(R) Core(TM)2 Duo CPU E7500 @ 2.93GHz
Architecture	x86_64
Memory	2048 MB
CPU	2
Operating System	Ubuntu 16.04.2 LTS (Xenial)
Virtual	QEMU/KVM, Libvirt

Figure 14: Physical attributes

These are the details for the physical hardware which are dedicated for the virtual implementation. The PMs run Ubuntu which is easier to work with especially with QEMU and KVM for virtualization of the environment.

6.3 Virtual Configuration

The next step is to configure the virtual network. This network will also ensure that when migrating VMs from one host to another, this happens within the same virtualized network. Figure 15 below shows how the PMs are connected and how the VMs reside inside the PMs. The VMs are attached to a virtual bridge by birth. This is a actually a virtual switch, however it is called a bridge and used with KVM/QEMU hypervisors to be able to use live migration for instance.

To connect the PMs together, a physical switch is used.

The different VM flavors that will be used in the experiments are given in Figure 16. Each PM will be given a combination of VMs of different flavors.

6.4 Migrating Smallest First

The aim of this experiment is to see how migration can take place in a real virtualized environment with different flavours of VMs. The figure below displays the results extracted from the experiment when we apply migrate smallet first scheme:

6.4.1 Analysis. Figure 17 displays the results extracted from 3 rounds of migration between host PM 1 and PM2. Test1 had an imbalance of 3.0 before migration and the imbalance after went

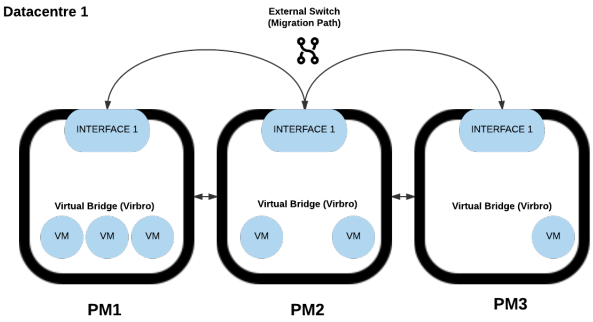

Figure 15: Physical Lab details

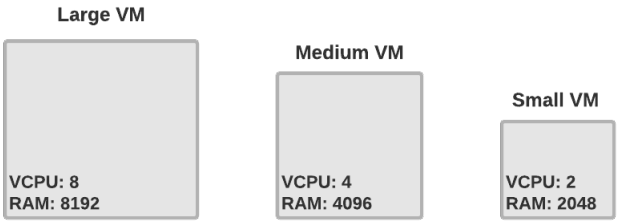

Figure 16: VMs flavors

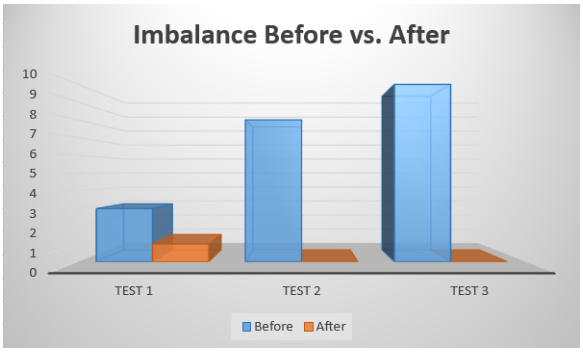

Figure 17: Small Imbalance Before vs. After

down to 1.0. The test had one overloaded server with five small flavoured VMs. The destination host only had two VMs, and space for more. PM1 sent over 2 VMs to PM2.

Test 2 consisted of seven VMs in total, where only two small VMs resided with five medium VMs. The destination host consisted of four VMs, but all small flavoured. This resulted in 3 migrations in total from PM1 til PM2. In this particular migration, two of the VMs were small flavoured and one was medium flavoured.

Right after the migration, the small VMs were up and running quite quickly, using around 0.4 of CPU, while the medium VM spent a quarter of a second more than the small VMs. The imbalance

before migration was 8.0, and after it went down flat 0.0. Which is a great result, concluding the bin in complete balance.

The aim for Test 3 was to see if now PM2 was overloaded, does it locate PM1 and continue migration, as well as how many VMs it would migrate if it only was small flavoured. With an imbalance of 10.0 before and 0.0 after migration.

This test consisted of sixteen small VMs on PM2, while on PM1 there were two small and one large VM. The acceptance of the proposal ended with four migrations. Now, one would think that this results in PM2 still being overloaded with 12 VMs, however, PM1 has one large VM, which evens the imbalance out for both bins. The average CPU load was only 22.

6.4.2 Migrating Largest First. Figure 18 displays the results extracted from 3 rounds of migration between host PM1 and PM2, based on the large flavour scheme.

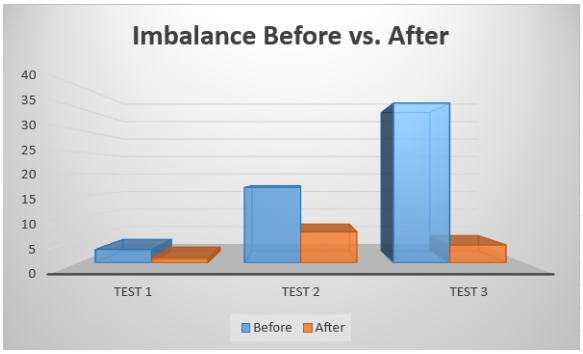

Figure 18: Large Imbalance Before vs. After

6.4.3 Analysis. The tests were performed exactly the same way as the previous small flavour experiments were performed. The first Test, had an imbalance of 17.0 before migration and 3.0 after.

Test 2 had an imbalance of 17.0 before migration and 7.0 after. Compared to the other columns, it is easy to notice that the two imbalances do not differ much from each other. There is an improvement, however PM1 was heavily loaded with seven large VMs, while PM2 had five VMs where three were small and two were large. The algorithm decided to migrate three of its largest VMs from PM1 to PM2, as this would be the best choice.

Test 3 has an imbalance of 36.0 before and resulted in 4.0 after migration. This test had fourteen large VMs on PM1 and eight VMs on PM 2. Out of the eight VMs, four were small and four were large flavoured. The migration resulted in five large VMs from PM1 to PM2. There was more room on PM2 because of the smaller VMs.

7 CONCLUSION

The aim of this paper was to check whether implementing the Stable Marriage algorithm load balancing could be possible in a cloud data center. To the best of our knowledge, this is the first attempt in the literature to apply the latter algorithm in such context. The results are promising and demonstrate the ability of our approach to efficiently distribute the load across different physical servers. As a future work, we would like to investigate an approach for both performing consolidation and homogenizing the load by increasing the target load we would like to achieve in each machine. In fact, if we increase the target load we would achieve in each machine, the number of active machines should decrease in a similar manner to consolidation.

REFERENCES

[1] Barbagallo, D., Di Nitto, E., Dubois, D. J., and Mirandola, R. A bio-inspired algorithm for energy optimization in a self-organizing data center. In *Self-Organizing Architectures*. Springer, 2010, pp. 127–151.

[2] Levine, D. K. Introduction to the special issue in honor of lloyd shapley: Eight topics in game theory. *Games and Economic Behavior 108* (2018), 1 – 12. Special Issue in Honor of Lloyd Shapley: Seven Topics in Game Theory.

[3] Lloyd Shapley, A. R. "stable matching: Theory, evidence, and practical design".

[4] Manlove, D. F. *Algorithmics of matching under preferences*, vol. 2. World Scientific, 2013.

[5] Marzolla, M., Babaoglu, O., and Panzieri, F. Server consolidation in clouds through gossiping. In *2011 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)* (2011), IEEE, pp. 1–6.

[6] Rao, A., Lakshminarayanan, K., Surana, S., Karp, R., and Stoica, I. Load balancing in structured p2p systems. In *International Workshop on Peer-to-Peer Systems* (2003), Springer, pp. 68–79.

[7] Sedaghat, M., Hernández-Rodriguez, F., Elmroth, E., and Girdzijauskas, S. Divide the task, multiply the outcome: Cooperative vm consolidation. In *2014 IEEE 6th International Conference on Cloud Computing Technology and Science (CloudCom)* (2014), IEEE, pp. 300–305.