Improving Resource Utilization in Cloud Environments using Application Placement Heuristics

Atakan Aral and Tolga Ovatman
Department of Computer Engineering, Istanbul Technical University, Istanbul, Turkey

Keywords: Application Placement, Resource Allocation, Cloud Computing, Software as a Service.

Abstract: Application placement is an important concept when providing software as a service in cloud environments. Because of the potential downtime cost of application migration, most of the time additional resource acquisition is preferred over migrating the applications residing in the virtual machines (VMs). This situation results in under-utilized resources. To overcome this problem static/dynamic estimations on the resource requirements of VMs and/or applications can be performed. A simpler strategy is using heuristics during application placement process instead of naively applying greedy strategies like round-robin. In this paper, we propose a number of novel heuristics and compare them with round robin placement strategy and a few proposed placement heuristics in the literature to explore the performance of heuristics in application placement problem. Our focus is to better utilize the resources offered by the cloud environment and at the same time minimize the number of application migrations. Our results indicate that an application heuristic that relies on the difference between the maximum and minimum utilization rates of the resources not only outperforms other application placement approaches but also significantly improves the conventional approaches present in the literature.

1 INTRODUCTION

In addition to software delivered as services over the Internet, hardware and software systems that make the delivery possible are referred as cloud computing (Armbrust et al., 2010). In cloud computing, resources such as CPU, memory, bandwidth and storage are treated as utilities that can scale up and down on demand. It also allows per-usage metering and billing of these resources. By means of cloud computing, cloud users can handle unexpected high demands without over-provisioning and they do not need to invest for abundant hardware resources initially. Cloud providers, on the other hand, have the opportunity of reallocating idle resources for other cloud users.

Cloud computing services are categorized into three layers according to the level of abstraction they provide. These are, in ascending order of abstraction: (1) Infrastructure as a Service (IaaS) that provide virtual raw resources, (2) Platform as a Service (PaaS) that provide virtual development environments, and (3) Software as a Service (SaaS) that provide online applications on demand (Buyya et al., 2011).

One general research challenge in cloud computing is the efficient allocation of cloud resources to users since cloud providers should satisfy quality of service (QoS) objectives while minimizing their operational cost (Zhang et al., 2010). In this paper, we approach this challenge from the perspective of a SaaS provider. Provider receives user applications with heterogeneous resource requests and assigns these applications to the virtual machines (VMs) with finite capacities. Multiple requests can be assigned to a single VM so the SaaS provider would prefer to fully utilize the VM resources at hand before hiring new ones. It is also possible to move applications between VMs, but this would cause a temporary halt of the migrating application and would reduce QoS. Consequently, the objective is to maximize the number of applications assigned to VMs and at the same time, minimize the number of application migrations.

For the sake of simplicity, we make the following four assumptions regarding the characteristics of the applications and VMs.

1. Cloud provider possesses constant number of VMs with homogeneous capacities. These capacities are four dimensional (namely CPU, memory, bandwidth and storage). It is not possible to add new VMs or resize existing ones.
2. Similarly, applications have the same four constant consumptions (requests).

3. Requests of the applications are foreknown.
4. Cloud provider receives one application at a time and after this application is assigned to a VM, it runs infinitely (i.e., it does not have a life span). However, it may temporarily suspend, migrate to another VM and continue its execution there.

These assumptions may seem too restrictive at first but our approach can be easily generalized to real world cloud scenarios. We use static values (VM capacities, counts etc.) because we aim to use current resources at hand in most efficient way. When these resources are almost fully utilized, SaaS provider possibly increases them, and our algorithm immediately adapts and starts to make decisions according to the new configuration. Similarly, when resource consumption changes over time or even terminates, new decisions will be made considering the current utilization. Third assumption of foreknown resource consumptions is based on the Service Level Agreements (SLAs) made between the cloud provider and cloud user where expected performance criteria is defined.

Fundamentally, the problem of optimally placing applications to VMs can be formulated as an NP-hard multidimensional bin packing problem and solved via optimization techniques such as linear programming (Zhang et al., 2010). However, it would be computationally expensive to run the optimization algorithm at each application arrival especially for more than a few VMs. Moreover, optimization would incur large number of application migrations. Our solution is to distribute applications to VMs using an intelligent heuristic until the point that a migration is inevitable (i.e. when no VM has enough capacity to accept incoming application). After this point, an optimization technique can be employed to rearrange the applications.

Contributions in this paper may be summarized as follows.

1. A Mixed Integer Programming (MIP) formulation of the application placement problem that minimizes the number of migrations
2. Novel heuristics to assign applications to VMs efficiently and intelligently
3. Comparison of the heuristics to the existing ones
4. Analysis of the heuristic approach on various VM counts and capacities

The following section gives an overview of previous work in resource allocation and application placement literature. Section 3 explains the details of our method including MIP formulation and heuristics, while Section 4 contains the experiments, their results and discussion. Finally, we conclude the paper with final remarks in Section 5.

2 RELATED WORK

Allocation of a cloud provider's resources to its customers is entitled as resource allocation, resource selection or application placement depending on the context and what is meant by the term resource. In this section, we review different approaches to the problem in the cloud computing literature. It should be noted that similar problems are encountered and extensively studied before the arrival of cloud computing (Urgaonkar et al., 2005). However cloud computing brings some unique challenges to the problem such as inclusion of network as a resource and as a request, geo-diversity and location limitations (jurisdiction), multi-tenancy, more heterogeneous resources and less predictable demands.

2.1 VM Assignment Strategies

Selection of physical machines to host virtual machines while fulfilling the requirements and optimizing resource usage is a problem that can be solved via optimization and approximation algorithms. Some examples use heuristics, linear programming, artificial intelligence, nature inspired computing and game theory (Endo et al., 2011).

In a recent study (Papagianni et al., 2013), the problem is formulated as two phases that must be optimized together: node mapping and link mapping. User request come in the form of VMs with defined capacities and their connectivity as a graph. The aim is to optimally assign these capacity request to physical machines (node mapping) and connectivity requests to network entities (link mapping). They apply a relaxed MIP approximation for the node mapping and shortest path (or minimum cost flow) algorithm for the link mapping in a coordinated way.

Another approach (Xiao et al., 2013), introduced the skewness metric to measure unevenness in the utilizations of various resources within a server. They aim to find a trade-off between overload avoidance and green computing concepts. Using a set of threshold based heuristics and a prediction algorithm they dynamically create a list of migrations that relieves overloaded servers to ensure QoS and evacuates underloaded ones to exploit green computing.

Following the improvements of the cloud systems, VM migration is being used to increase utilization of the virtual machines on the cloud infrastructure. (Yang et al., 2011) considers the different VM migration strategies adopted by the host machines that have different load states by taking into account four different resource types. In a more recent study (Wang et al., 2013), migration is performed regarding the queue model of the time of application deployment requests from the clients. A centralized control management mechanism has been defined to inform the client which server is available at a time.

2.2 Application Assignment Strategies

Literature in the area of allocation within a VM is not as extensive as the one in allocation within a server, as the most studies consider resource allocation for the benefit of IaaS providers. A detailed model of SLAs is given in the study (Wu et al., 2011) which falls into the application placement category. SLAs between cloud providers and users are taken into consideration to quantitatively measure the service quality. They also suggest a simple heuristic: 'assign the request to the VM with the lowest (or the highest in other variation) utilization'.

In another study (Tang et al., 2007) optimal load balancing for dynamically changing demands is considered. Their approximation method aim to maximize satisfied demand and minimize application migrations. Proposed application placement controller estimates future demand and migrates applications accordingly to avoid overloading.

A recent work (Espadas et al., 2013) offers a resource allocation mechanism that exploits multi-tenancy and provides tenant isolation, load balancing and VM allocation. They also formally measure over and underprovisioning of VM resources.

3 APPLICATION PLACEMENT ALGORITHM

3.1 Overview

A high level pseudo code of both parts is given in Algorithm 1. In part I (lines 4 to 12), migrations are not allowed, and each arriving application is assigned to the VM determined by an heuristic of evenness. Goal of this part is to keep the utilization of four resources in a VM approximately even. By increasing the resource evenness of all VMs, overall utilization of the resources are maximized (Xiao et al., 2013).

To decide which VM is the best for a given application, it is transiently considered to be assigned to the VMs that has enough remaining capacity one by one and evenness is calculated. Then, the application is actually assigned to the VM with the best evenness value (line 12). At one point of the part I, incoming application's resource request will not fit into the remaining capacity of any VM (the application is not directly assignable and condition in line 11 is not satisfied). However, rejected application can still be assigned to a VM if some applications are re-assigned.

Part II (lines 13 to 16) migrates some of the applications and tries to make room for the new application. If part II succeeds to assign the application (line 16), part I of the algorithm continues to receive new applications; otherwise, it is certain that there is no placement that can assign received applications to the VMs and the algorithm terminates (line 17).

Algorithm 1: Pseudo code for the application placement strategy.

```
1  rejected ← false
2  while rejected = false do
3      receive application A
4      assignable ← false
5      foreach virtual machine VM do
6          if VM has enough capacity for A then
7              assignable ← true
8              assign A to VM
9              calculate unevenness of VM
10             remove A from VM
11     if assignable = true then
12         assign A to VM with min unevenness
13     else
14         run optimization algorithm
15         if optimization succeeds then
16             assign A and migrate others
17         else rejected ← true
```

One variation of part I can be obtained by calculating unevenness twice, i.e. before and after tentatively assigning the application. In that case, actual assignment is made to the VM whose unevenness decreases the most, instead of the one whose unevenness becomes the minimum.

3.2 Heuristics

We propose four heuristics to approximate the unevenness in the utilization of the resources on a VM. We also adapt the skewness heuristic originally suggested for assigning VMs to servers (Xiao et al., 2013). Finally we implement the greedy approximation algorithm, round-robin, to use as the baseline.

3.2.1 Standard Deviation (*SD*)

Standard deviation is a natural choice to find the unevenness of data points since it shows the amount of dispersion from the average. *SD* of the resource utilizations on a VM v can be calculated as follows.

$$SD(v) = \sqrt{\sum_{i=1}^{m} (r_i - \bar{r})^2} \tag{1}$$

Here, r_i denotes the utilization of the ith resource, while $\bar{r}$ is the average utilization of all resources of v. m is the number of resources.

3.2.2 Span (*SP*)

For the trivial case of only four instances, a simpler heuristic may be used. Since we are interested more in outliers than inliers, difference between the maximum and minimum utilization rates can be a good candidate for an unevenness heuristic.

$$SP(v) = \max_{i\in[1,m]} (r_i) - \min_{i\in[1,m]} (r_i) \tag{2}$$

3.2.3 Cumulative Difference (*CD*)

Another candidate is the total difference of the utilization rates of all resource pairs on v, as given by the formula below. This heuristic also considers the inliers similar to *SD*, but it is simpler.

$$CD(v) = \frac{\sum_{i=1}^{m} \sum_{j=1}^{m} |r_i - r_j|}{2} \tag{3}$$

3.2.4 Cumulative Difference from Minimum (*DM*)

A similar heuristic to *CD* is to calculate differences from the minimum utilization rate. This approach tries to ensure that no resource is under-utilized (low outlier) relative to others, while previous approaches are trying to prevent both low and high outliers.

$$DM(v) = \sum_{i=1}^{n} \left(r_i - \min_{j\in[1,m]} (r_j) \right) \tag{4}$$

3.2.5 Skewness (*SK*)

Formulation of *SK* is given in (Xiao et al., 2013).

$$SK(v) = \sqrt{\sum_{i=1}^{m} \left(\frac{r_i}{\bar{r}} - 1 \right)^2} \tag{5}$$

3.2.6 Round-Robin (*RR*)

Round-robin algorithm is a very straightforward solution to the application placement problem. For each application, the greedy algorithm attempts to assign it to the next VM. If that VM is not available, consecutive ones are tested in order. In the ideal case where all VMs are always available, it assigns equal number of applications to each of them. Naturally, it does not consider the evenness of resource types within a VM as opposed to the heuristics mentioned above.

3.3 MIP Formulation

To determine an optimal application placement scheme, we use MIP techniques. We minimize the number of migrations when determining the place (VM) of each application from the pool of VMs. In achieving this, for each solution we compare the place of each application with its former place and try to maximize the number of applications that don't change their place.

We have used the native library lp_solve and Java ILP API to solve the MIP problem introduced earlier. We have simulated four VMs residing in a host machine which let us to run our simulations on a decent personal computer. Heavier analysis on a large number of VMs residing in federated cloud environments can also be simulated, however MIP performance exponentially decrease for such cases.

3.3.1 Variables

When using MIP, it is required to build formulae that represents the constraints of the system and an objective function to be optimized during the process. In constructing such formulae variables that represent the overall properties of the system shall be used. Below, the variables we used in our model and their meanings are explained briefly:

- #VMs: Number of VMs present in the cloud environment.
- #Apps: Number of applications to be placed upon the present VMs.
- oldAsgn: A boolean matrix that holds the present assignment of each application upon a VM.
- newAsgn: A boolean matrix that holds the resulting assignment by MIP.
- resNeed: An integer matrix that holds the amount of resource needed by each application for the four different type of resources mentioned before.
- resAv: An integer matrix that holds the amount of resource available for each VM and each resource.

3.3.2 Objective Function

In modeling resource consumption for a cloud environment, we provide the MIP solver with Equation 6 to maximize. We try to maximize the sum of old and new application-VM assignment products which produce a value of 1 if the assignment didn't change and 0 if a migration is present.

$$\sum_{i=0}^{\#VMs}\sum_{j=0}^{\#Apps}(\text{newAsgn}[i][j]\times\text{oldAsgn}[i][j]) \qquad (6)$$

3.3.3 Constraints

Naturally, we also apply a number of constraints in order to drive the MIP to produce meaningful results. In Equation 7 we guarantee that each application has been assigned to exactly one VM. Additionally in Equation 8 resource needs of each application that has been assigned to a specific VM for each type of resources are summed up to be less than the available resource assigned to the VM.

$$\bigwedge_{i=0}^{\#VMs}\left(\sum_{j=0}^{\#Apps}\text{newAsgn}[i][j]=1\right) \qquad (7)$$

$$\bigwedge_{i=0}^{\#VMs}\bigwedge_{j=0}^{\#Res}\left(\left(\sum_{k=0}^{\#Apps}\text{newAsgn}[i][k]\times\text{resNeed}[j][k]\right)\leq\text{resAv}[i][j]\right) \qquad (8)$$

4 EXPERIMENTS

A simple simulator is implemented to conduct the experiments explained in this section. Simulator generates applications with uniformly random resource requests within a given range and manages VMs with user defined count and capacity. For all experiments, applications are generated with resource requests within the range of $[0,30]$ units while VM count and capacities varied.

4.1 Postponement of Migration

This first experiment aims to compare the power of heuristics to assign maximum number applications before a migration is necessary. As mentioned before, migrations should be avoided since they may cause temporary downtimes for an application. So, a good heuristic should place applications in a way that migrations become compulsory as late as possible.

4.1.1 Experimental Setup

Part II of the algorithm is not required for this experiment since we are only interested in the performance of the heuristics before the optimization starts. Excluding the computationally expensive part of the algorithm allowed to test heuristics for a great number of configurations and repetitions. Two variants (minimum and most decreasing) of five heuristics as well as round-robin are compared on 100 different VM configurations. This consists of 10 VM counts ranging from 3 to 12 and 10 VM capacities ranging from 75 to 300 units per resource.

All 11 strategies are run on each configuration 30.000 times to avoid randomness of applications affect the results. Execution of a strategy on a configuration is stopped when no VM is available for the received application. After that, the total number of applications assigned to all VMs are logged and the average of 30.000 runs are calculated.

4.1.2 Results and Discussion

Due to the great number of configurations tested in this experiment, it is not possible to include all results here. We instead present five configurations with a fixed VM count (8 VMs) on the left half of the Table 1 and five configurations with a fixed VM capacity (200 units per resource) on the right half. Remaining 90 configurations follow the same pattern as them.

For each configuration, Table 1 contain average number of applications assigned by 11 strategies, expected number of applications to fully utilize VMs and the rate of improvement by the best performing strategy in comparison to the baseline (*RR*). Subscripts *min* and *dec* stand for the minimum and most decreasing variants of the strategies, respectively.

Our results indicate that in 477 of 500 cases (95,4%), *min* variant outperformed the *dec* variant of the same strategy. In most of the 23 cases where *min* variant performed worse, tiny VM capacities and the *DM* strategy are used. This clearly demonstrates that arriving application should be assigned to the VM that becomes the most even with the assignment, instead of the one whose evenness increases the most.

It is also evident that intelligent heuristics always perform better than the greedy assignment. In all 100 configurations, all 10 strategies manage to assign more applications than the round-robin. The worst improvement rate is $4,3\%$ (3 VMs with 300 units of capacity per resource) while the best one is $12,1\%$ (12 VMs with 75 units of capacity per resource). Utilization rate of round-robin is between $73,8\%$ and $90,2\%$ while utilization rate of the heuristics is between $80,5\%$ and $96,4\%$.

Improvement rate gets better as the number of VMs increase (as seen in the right half of the final row in Table 1) but their capacities decrease (as seen left half of the same row). With high capacities and small number of VMs, randomness of resource requests tend to cover imperfect assignments while with low capacities and large number of VMs, decisions are more critical and should be made with more care. In cases that are ideal for cloud computing benefits (i.e. many distributed VMs with relatively low capacities), heuristics show their real strength.

Table 1: Partial result of the application count experiment.

VM Capacity	**100**	**150**	**200**	**250**	**300**	**200**	**200**	**200**	**200**	**200**
VM Count	**8**	**8**	**8**	**8**	**8**	**4**	**6**	**8**	**10**	**12**
RR	42,0	67,2	92,7	118,4	144,3	46,0	69,3	92,7	116,2	139,7
SK_{min}	45,8	72,3	98,8	125,3	151,9	48,1	73,4	98,8	124,3	149,9
SK_{dec}	45,5	72,0	98,6	125,2	151,9	48,1	73,2	98,6	124,1	149,7
SP_{min}	**46,2**	**73,2**	**100,2**	**127,2**	**154,3**	**48,7**	**74,4**	**100,2**	**126,2**	**152,3**
SP_{dec}	46,0	72,6	99,3	126,0	153,2	48,4	73,8	99,3	124,9	150,7
SD_{min}	**46,2**	**73,2**	**100,2**	**127,3**	**154,3**	**48,7**	**74,4**	**100,2**	**126,2**	**152,3**
SD_{dec}	45,8	72,3	98,9	125,5	152,1	48,2	73,4	98,9	124,4	150,1
CD_{min}	**46,2**	**73,2**	**100,2**	**127,3**	**154,3**	**48,8**	**74,4**	**100,2**	**126,2**	**152,3**
CD_{dec}	45,9	72,5	99,1	125,9	152,7	48,3	73,7	99,1	124,8	150,5
DM_{min}	45,7	72,6	99,6	126,5	153,6	48,3	73,8	99,6	125,4	151,5
DM_{dec}	45,8	72,5	99,2	126,2	153,1	48,3	73,6	99,2	125,0	150,8
Expected	53,3	80,0	106,7	133,3	160,0	53,3	80,0	106,7	133,3	160,0
Improvement	**10,0%**	**8,9%**	**8,1%**	**7,5%**	**6,9%**	**6,1%**	**7,4%**	**8,1%**	**8,6%**	**9,0%**

Since it is already demonstrated that heuristics make better assignments than the greedy algorithm and the *min* variants are better than *dec* variants, we can finally make a comparison among them. In trivial cases where capacity per resource is less than 100 units, all heuristics perform more or less the same, while in all other cases *SP*, *SD* and *CD* perform significantly better than *SK* and *DM*.

Although being still much better than greedy, skewness algorithm is outperformed by the best three heuristics by $1,2\%$ on average. Our observations show that both variants of *SK* are sensitive to average resource utilization of the VMs. As a result, they tend to assign new applications to the VMs that contain more applications although better alternatives in terms of evenness are available. This causes their placements resemble greedy first-fit algorithm.

4.2 Minimization of Migration Count

Goal of this experiment is to assess the quality of application placement of the strategies. A good placement is expected to allow assignment of a new application after a small number of migrations, or in the ideal case, fully utilize VMs without any migrations. It should be noted that the objective of the optimization algorithm is to minimize the number of migrations so the migration counts are guaranteed to be the smallest possible values.

4.2.1 Experimental Setup

Both part I and II are included in this experiment and that forces us to decrease repetition count. That's why, only the best three heuristics of the first experiment, round-robin and skewness algorithms are executed for a single configuration: three VMs with capacities of 300 units per resource. Greater number of VMs causes a significant increase in running time.

An execution is stopped on the following two conditions: (1) In part I, when remaining capacities of all VMs are less than the expected value of the four resource requests (i.e., 15 units), (2) In part II, when the optimization algorithm runs once. We call the first condition a perfect placement since all VMs are almost fully utilized without any migrations. Second condition is only considered when a perfect placement is not achieved and the algorithm continues to part II. In that case, if the algorithm succeeds to assign the application to a VM, number of migrations are logged. Otherwise, if the application is rejected, it is also considered as a perfect placement because MIP proved that it was impossible also for the heuristic to make that placement.

Each strategy is executed 1.000 times calculating the average number of migrations and the number of perfect placements. Number of migrations for a perfect placement is regarded zero.

4.2.2 Results and Discussion

Table 2 gives the average number of migrations introduced by the optimization after each five strategies (*RR*, SK_{min} and three best performers of experiment 1) failed to assign any more applications. The table also contains the number of cases (among 1.000 executions) where heuristics made a perfect placement so an optimization was not required.

Results demonstrate that placements made by the heuristics allow assignment of new applications with less changes since SD_{min}, SP_{min} and CD_{min} strategies required $34,5\%$, $33,3\%$ and $31,0\%$ less migrations than *RR*, respectively. Similar to the first experi-

Table 2: Results of the migration count experiment.

Strategy	Avg. Migr. Count	Perfect Count
RR	8,4	26
SD_{min}	5,5	108
SP_{min}	5,6	100
CD_{min}	5,8	86
SK_{min}	7,2	88

ment SK_{min} performed somewhere in the middle with $14,3\%$ better than *RR* but $27,8\%$ worse than the others in average. SD_{min} and SP_{min} made perfect placements in $10,8\%$ and $10,0\%$ of executions, followed by SK_{min} with $8,8\%$ and CD_{min} with $8,6\%$. Coincidental perfect placements are made by *RR* in only $2,6\%$ of cases.

It is clear that intelligent heuristics make better placements that can be optimized with significantly less migrations and that are already optimum in roughly four times more incidents than the greedy approach. Suggested heuristics also outperform SK_{min} clearly with the average number of migrations. As a result, they allow applications to run with less suspension as well as decreasing the need for time consuming optimization algorithm.

Due to its simplicity, we choose SP_{min} heuristic to represent heuristic approach in the future performance analysis.

4.3 Analysis on Constant Total Capacity

As explained before, partitioning total hardware capacity to higher number of virtual machines provides benefits of cloud computing but reduces overall utilization. This reduction can be especially significant when different application types are not combined appropriately and a certain resource of a VM runs much shorter than the others. In this experiment, we aim to observe utilization performance of our algorithm in comparison to greedy approach given various number of VMs sharing a fixed hardware capacity.

4.3.1 Experimental Setup

SP_{min} heuristic which is one of the best performing and simplest strategies, is compared to the greedy round-robin algorithm using 3 to 12 VMs. The VMs share a total capacity of 1.000 units per resource. Similar to the first experiment, part I of the algorithm is executed 30.000 times for both strategies to calculate average count of applications that are assigned. Executions are stopped when a migration is necessary.

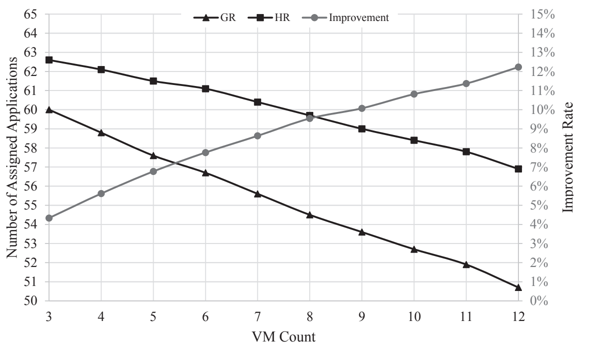

Figure 1: Results of the constant capacity analysis of the heuristic approaches (*HR*) in comparison to gready (*GR*).

4.3.2 Results and Discussion

Chart in Figure 1 displays the number of applications that are assigned by heuristic (SP_{min}) and greedy (*RR*) approaches to the same constant total capacity that is partitioned to varying number of VMs. It also shows the rate of improvement provided by the heuristic strategy on its secondary y-axis.

As expected, both strategies assign less applications as the number of VMs increase, however the rate of decrease is different. *RR* starts with $60,0$ applications and encounters a decrease to $50,7$ applications with a slope of $-1,03$. SP_{min}, on the other hand, starts with slightly more applications, $62,6$, and ends around $56,9$ with a less steep slope around $-0,63$.

Other heuristics are also tested but the results are omitted since their performance is nearly as good as SP_{min}. That's why we can consider this experiment as a comparison of heuristic and greedy approaches rather than SP_{min} and *RR*. Experimental results indicate that using a more intelligent heuristic instead of the greedy algorithm slows down the decrease in resource utilization while partitioning increases. Accordingly, improvement provided by heuristics gets more significant. This is especially important when we consider that benefits of cloud computing becomes more evident with higher number of VMs.

5 CONCLUSION

In this paper we propose a method for assigning applications with multi-dimensional resource requests to virtual machines with finite capacity. The method employs heuristics and mixed integer programming to optimally solve the application placement problem. Our goal is to allow SaaS providers to host maximum number of applications with their resources at the same time increasing their service quality.

While optimization algorithms such as our MIP formulation guarantee to assign maximum number of applications, they decrease QoS in two ways: (1) They are extremely time consuming for high number of instances so they delay the decision making process while the request is pending, (2) Their decisions cause migrations of applications which require temporary downtimes.

Suggested heuristics approximate to the optimal placement before the optimization is inevitable. They do so by combining workloads together to keep utilization of resources as even as possible. Previously suggested unevenness metric, skewness, fails to yield good combinations while greedy algorithms such as round-robin do not consider evenness at all.

Heuristics are evaluated via a simple simulator that supports random generation of demands and maintains VM capacities. Our experiments demonstrate that heuristics provide the following improvements to QoS in comparison to greedy approximation.

1. They make the optimal placement and almost fully utilize VMs up to $10,8\%$ of cases. This is four times more than the rate provided by the greedy algorithm. In such cases, service quality is preserved since the execution of MIP algorithm and application migrations are not required.

2. In the rest of the cases, they delay the requirement for MIP algorithm up to $12,1\%$ applications. This means more applications can be accepted without migrations and any effect on service quality.

3. When the execution of MIP algorithm is mandatory, their placements require up to $34,5\%$ less application migrations causing less applications to suspend and less harm to service quality.

As future work, we aim to reproduce our results on a state of the art cloud computing simulator and to test our heuristics on other scenarios. These include other random distributions of resource requests and dynamic resource demands.

ACKNOWLEDGEMENTS

This work is performed in joint with “mCloud” project of Simternet Iletisim Sistemleri Reklam San. ve Tic. Ltd. Sti. “mCloud” project is supported by The Scientific and Technological Research Council of Turkey (TUBITAK) – TEYDEB project number 7130115.

REFERENCES

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. (2010). A view of cloud computing. *Commun. ACM*, 53(4):50–58.

Buyya, R., Broberg, J., and Goscinski, A. M. (2011). *Cloud computing: Principles and paradigms*, volume 87. Wiley. com.

Endo, P., de Almeida Palhares, A., Pereira, N., Goncalves, G., Sadok, D., Kelner, J., Melander, B., and Mangs, J.-E. (2011). Resource allocation for distributed cloud: concepts and research challenges. *Network, IEEE*, 25(4):42–46.

Espadas, J., Molina, A., JiméNez, G., Molina, M., RamíRez, R., and Concha, D. (2013). A tenant-based resource allocation model for scaling software-as-a-service applications over cloud computing infrastructures. *Future Gener. Comput. Syst.*, 29(1):273–286.

Papagianni, C., Leivadeas, A., Papavassiliou, S., Maglaris, V., Cervello-Pastor, C., and Monje, A. (2013). On the optimal allocation of virtual resources in cloud computing networks. *Computers, IEEE Transactions on*, 62(6):1060–1071.

Tang, C., Steinder, M., Spreitzer, M., and Pacifici, G. (2007). A scalable application placement controller for enterprise data centers. In *Proceedings of the 16th international conference on World Wide Web*, WWW ’07, pages 331–340, New York, NY, USA. ACM.

Urgaonkar, B., Shenoy, P., Chandra, A., and Goyal, P. (2005). Dynamic provisioning of multi-tier internet applications. In *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, pages 217–228.

Wang, Y., Chen, S., and Pedram, M. (2013). Service level agreement-based joint application environment assignment and resource allocation in cloud computing systems. In *Green Technologies Conference, 2013 IEEE*, pages 167–174.

Wu, L., Garg, S., and Buyya, R. (2011). Sla-based resource allocation for software as a service provider (saas) in cloud computing environments. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pages 195–204.

Xiao, Z., Song, W., and Chen, Q. (2013). Dynamic resource allocation using virtual machines for cloud computing environment. *Parallel and Distributed Systems, IEEE Transactions on*, 24(6):1107–1117.

Yang, K., Gu, J., Zhao, T., and Sun, G. (2011). An optimized control strategy for load balancing based on live migration of virtual machine. In *Chinagrid Conference (ChinaGrid), 2011 Sixth Annual*, pages 141–146.

Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18.