Empirical Evaluation of Vector Bin Packing Algorithms for Energy Efficient Data Centers

Lei Shi, John Furlong, Runxin Wang
Telecommunications Software & Systems Group,
Waterford Institute of Technology, Ireland
Email: {lshi, jfurlong, rwang}@tssg.org

***Abstract*—As the proliferation of cloud data centers has been progressing in recent years, the power consumed in cloud enterprises has increased also. Strategies to reduce the power consumption of a data center can be modelled into variants of vector bin packing algorithms. The application of virtualization technology allows a large number of virtual machines (VMs), treated as items, to be concurrently hosted into relatively fewer number of Physical Machines (PMs), treated as bins, as long as the sum capacity of the co-located VMs does not exceed the capacity of a hosting PM. This paper addresses the problem of which vector bin packing strategy to use in attempting to first consolidate VMs in a data center, the aim of which is to reduce power consumption by powering off unused PMs, then going forward, which strategy to adopt when packing new VM requests to PMs as optimally as possible.**

I. Introduction

The problem of how to minimize energy consumption in data centers has been an area of much research over the last number of years as the use and construction of new data centers has rapidly increased. Figures for 2010 show that of the total energy consumed in the world, between 1.1% and 1.5% was accounted for by data centers, with this figure rising to between 1.7% and 2.2% in the United States [1]. Hence a very important objective is the reduction of the energy consumption used to operate and manage data centers, especially with the development of efficient solutions for consolidating virtual resources.

Virtualization technology enables multiple virtual machines (VMs) to co-exist on a physical machine (PM), sharing the networking and computing resources. These VMs can be migrated from one PM to another PM, while maintaining their states. VM migration allows VMs to be consolidated into a smaller number of PMs, leading to a better PM utilization, which in turns reduces power consumption, as after consolidation those PMs hosting no VMs can be shut down. To achieve a better PM utilization, the objective is to find the minimum number of PMs to accommodate the VMs requests demanded by cloud users. This problem can be formulated as an optimization problem, known as the "vector bin packing problem" [2], where the VMs that are treated as items are packed onto PMs that are treated as bins.

First fit decreasing (FFD) method is the common method applied to find an approximated solution to a vector bin packing problem, and it has also been applied by data centers to solve the VM placement problem. The advantage of using FFD method is that FFD can solve the corresponding problem of large size in a reasonable time with a near optimal solution. FFD needs to sort the packing items in a decreasing order, for VM placement VMs can be sorted in different ways according to different strategies. In practice, data centers are often already running, with some PMs being partly assigned with VMs, and therefore it is in an intermediate state; VMs are dispersed randomly over PMs due to the dynamics of VM requests over time. In some circumstances some VMs have to be migrated from one PM to another PM, subsequently the PMs with no VMs after migration can be powered off, thus saving energy. However, VMs migration is also a cost to data centers in terms of the bandwidth and operations required to do so, thus the number of VM migrations (particularly live migrations in which VMs are migrated without stopping their services) must be reduced.

In this paper, given the initial placement of the VMs and PMs, we show that by calculating the CPU and memory components of each and then sorting the VMs and PMs in different ways, that a significant energy saving could be achieved. For the concern of PMs in an intermediate state, a series of different algorithms are designed in order to consolidate the VMs onto as few PMs as possible, while also aiming to minimize the number of VM migrations necessary for this consolidation. For the online scenario that VMs requirements are arriving incrementally over time, a number of online algorithms are designed for dynamic placement, aiming to place new VMs onto the PMs as efficiently as possible.

Previous work has studied variants of FFD to obtain the optimized VM consolidation regarding the constraint and objective in their own problems and scenarios. Most previous work assumes PMs are the same in capability, yet in today's cloud data centers, the type of PMs could be diverse. In addition, previous research was only performed in relation to consolidation alone for a data center, in which VMs had become dispersed across PMs over time due to VM requests coming online and dying out. However, the impact of different sorting strategies for FFD algorithms and the consolidation considering dynamic VM requests are not well investigated. In this work, we evaluate six different sorting strategies of FFD in order to identify which are the best performing algorithms in relation to various consolidation metrics, for both the offline and online scenario. We find that different sorting strategies could lead to different levels of optimized consolidation. Besides measuring which sorting strategy yields the minimum number of PMs to host VMs, we also design algorithms to minimize the number of migrations to realize its consolidation.

The remainder of this paper is structured as follows, in §II previous work in this area of research is discussed. The

978-1-4799-3755-4/13/$31.00 ©2013 IEEE

problem of VM placement and an integer linear programming model are illustrated in §III. In §IV the FFD based VM consolidation algorithms which we developed are explained in detail. In §V we present the experiments that evaluate the VM consolidation algorithms, our findings are discussed with the experimental results. Finally we conclude the paper in §VI.

II. Related Work

In relation to bin packing, Wilcox *et al.* [3] formulate a new algorithm called Reordering Grouping Genetic Algorithm (RGGA), which is first tested on conventional bin packing problems, before being applied to the specific bin packing problem of migrating and assigning VMs to PMs. Gupta *et al.* [4] view the problem of PM consolidation as falling into three categories: centralization, physical consolidation and application integration, with the focus being on physical consolidation. Physical consolidation is stated as being closely related to the bin packing problem and a solution is sought using a novel two stage heuristic algorithm for taking care of the bin-item incompatibility constraints. Murtazaev and Oh [5], proclaim a PM consolidation algorithm, Sercon, for use in cloud data centers. The Sercon algorithm is specifically designed to use with live migration in which applications are migrated from VM to VM without stopping their services. While these migrations are necessary for optimization, the energy costs involved in a migration are large, therefore the Sercon algorithm will not only try to minimize the number of PMs used but also try to minimize the number of migrations also. In a further study by Panigrahy *et al.* [6] the approach is taken to study variants of the FFD algorithm in a heuristic approach to the vector bin packing problem.

In relation to data center optimization, Wang *et al.* [7] decide to focus on the bandwidth constraints of network devices, stating that it is difficult for the traditional schemes to make a reliable, deterministic estimate of bandwidth demand. The solution offered is to first formulate a stochastic bin packing problem in order to model the volatility of the demands of the network and subsequently to propose an online packing algorithm to optimize it. Data centers which are "power proportional" are investigated by Lin *et al.* [8]. Cost metrics are developed relating to toggling of PMs into power saving mode, and an algorithm is developed to minimize the number of PMs in use, while also minimizing the costs.

An investigation into the power-aware provisioning of VMs for real-time services by Kim *et al.* [9] focuses on the acknowledged Service Level Agreements (SLAs) between the cloud service providers and the customers. The need to provide energy savings in the cloud computing environment without impinging on the SLAs is noted and a solution is proposed in two ways; firstly an attempt is made to model a real-time service as a real-time VM request using a cloud service framework which requests a virtual platform for real-time applications. Then an attempt to provision VMs in cloud data centers using Dynamic Voltage Frequency Scaling (DVFS) schemes along with some other schemes to reduce energy consumption is made.

Beloglazov *et al.* [10] use "live-migration" and VM consolidation, but only focus on the QoS of such an approach even in heterogeneous infrastructure containing heterogeneous VMs. An attempt is made to solve both the traditional bin packing problem as well as the "intermediate state" data center optimization problem and four heuristics are proposed for choosing which VM to migrate. Beloglazov *et al.* [11], focus on the dynamic consolidation of VMs using live migration in which idle nodes are switched to sleep mode. Also proposed are novel adaptive heuristics for the dynamic consolidation of VMs based on the analysis of historical data from resource usage by the VMs.

Tarighi *et al.* [12] aim to find a more intelligent way to migrate VMs from underutilized PMs to more utilized ones. They propose a new method to migrate VMs between cluster nodes using the TOPSIS (Technique for Order Preference by Similarity to Ideal Solution) algorithm. Ajiro *et al.* [2] propose using the FFD algorithm which is already widely used with regard to one and two dimensional bin packing problems but use it in conjunction with another algorithm called Least Loaded (LL) which is used for load balancing. This technique reduces the number of destination PMs by re-ordering the existing PMs and then retrying the packing procedure using either FFD or LL before a new PM is added.

III. Integer Linear Programming Problem Formulation

We consider the problem of placing VMs on PMs subject to capacity constraints specified by the cloud user (IaaS tenant). Let us assume that we have I available PMs and we wish to satisfy J requests for placement of VMs. Let $i = 1, \ldots, I$ denote an arbitrary PM and let $\mathbf{v_j}$, where $j = 1, \ldots, J$, denote an arbitrary VM placement request. Let $\mathbf{V} = \{\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_J}\}$ be a vector of the J requests that are to be satisfied at a given time. Each request, $\mathbf{v_j}$, is a vector of k' VM specifications: $\mathbf{v_j} = \{v_j^1, v_j^2, \ldots, v_j^{k'}\}$, where $k' = 1, \ldots, K_{max}$ and K_{max} denotes the maximum number of VMs that a cloud user is allowed to request in a request. Let v_j^k, where $k = 1, \ldots, k'$, denote an arbitrary VM specification in $\mathbf{v_j}$.

Let us assume that each PM i has associated with it a D-dimensional vector $\mathbf{c_i} = \{c_i^1, c_i^2, \ldots, c_i^D\}$ representing the capacities of the PM's D resource types (CPU capacity, memory, network bandwidth, etc.) that can be assigned to VMs. Let $d = 1, \ldots, D$ denote an arbitrary PM resource type. We assume that VMs are assigned a fixed minimal amount of such resources when placed on a PM. Let each $v_j^k \in \mathbf{v_j}$ have an associated D-dimensional resource requirement vector $\mathbf{u_j^k} = \{u_j^{k1}, u_j^{k2}, \ldots, u_j^{kD}\}$ specifying its resource requirements. Let $x_i(v_j^k)$ indicate VM placement, such that $x_i(v_j^k) = 1$ if v_j^k is assigned to PM i and $x_i(v_j^k) = 0$ otherwise. We now introduce our objective function and the constraints we apply. The objective of our ILP formulation for the VM to PM placement problem is to minimize the number of active PMs to host the requested VMs and the VM migrations, subject to resource capacity constraints (specified below):

$$\min \quad \sum_{i=1}^{I} H_i + \alpha \sum_{j=1}^{J} \sum_{k=1}^{|\mathbf{v_j}|} m(v_j^k)$$

In this objective function, the first component is the number of PMs for satisfying the VM requests. H_i is assigned to 1

978-1-4799-3755-4/13/$31.00 ©2013 IEEE

if there is at least one VM running on PM i, otherwise 0, as shown in following constraint.

$$\forall i \in \{1, \ldots, I\} : \sum_{j=1}^{J} \sum_{k=1}^{|\mathbf{v_j}|} x_i(v_j^k) \geq 1 \Longrightarrow H_i = 1$$

$$x_i(v_j^k) \in \{0, 1\}, H_j \in \{0, 1\}$$

The second component represents the number of migrations introduced by the migrations of the VMs allocated during the lifecycle of the placement. VM migration is an energy intensive affair so the algorithms need to balance the number of PMs powered off with the number of migrations. This component is introduced to discourage excessive VM migrations during VM placement lifecycle. The parameter $\alpha > 0$ is used to tune the impact of each component in the objective function.

VM migration means that previously the VM was placed and now it is still placed but on a different PM. Let F denote the existing VMs that cannot be removed, which can be only migrated or remained in their current PMs. Let $h(v_j^k)$ represent the previous PM upon which that VM v_j^k was placed. Let $m(v_j^k)$ be an indicator variable, indicating if VM v_j^k is to be migrated or not; thus:

$$\forall \mathbf{v_j} \in \mathbf{V}, \forall k \in \{1, \cdots, |\mathbf{v_j}|\}, v_j^k \in F :$$

$$m(v_j^k) = \sum_{i=1, i \neq h(v_j^k)}^{I} x_i(v_j^k)$$

For a VM v_j^k, if it is migrated then $m(v_j^k)$ is 1, otherwise 0. We want to minimize the total number of migrations; therefore, it is also integrated into the objective mainly to act as a penalty for placement plan updates that cause widespread migration of VM instances between PMs.

The resource capacity constraint is specified below, which ensures that physical machine resources are not over-allocated:

$$\forall d \in \{1, \ldots, D\}, \forall i \in \{1, \ldots, I\} :$$

$$\sum_{j=1}^{J} \sum_{k=1}^{|\mathbf{v_j}|} x_i(v_j^k) \cdot u_j^{kd} \leq c_i^d$$

For the VM placement problem with energy efficiency and migration constraints, although the ILP formulation yields an optimal solution, it is not suitable for the *dynamic* runtime placement of newly arrived VM requests. Because of its non-polynomial complexity, it might take several hours or even days (using commercial optimization software without parameter configurations), to produce a placement solution even for a relatively small problem. Therefore, fast, even if sub-optimal, approaches to compute the placement are required for provisioning of VMs.

IV. Bin Packing Algorithms

A cloud data center consisting of VMs hosted on PMs can be approached as a variant of the vector bin packing problem, where the resources used by each item are additive in each dimension. VMs that are treated as items are packed onto PMs that are treated as bins, as long as the sum of the items (VMs) does not exceed the capacity of the bin in any dimension. For this VM placement problem, this work considers two scenarios: 1) the offline scenario, some VMs are already placed on some PMs, assuming no new VM requests would arrive, a bin packing algorithm needs to consolidate the VMs into a smaller number of PMs subject to the VM's requirements, migration may be required to make space for the new VM. 2) the online scenario, new VM requests are arriving incrementally over time, while some PMs have already been placed with some VMs, a bin packing algorithm needs to place a new VM at each time, subject to the new VM's requirements.

FFD method is the common approach for solving bin packing problem for both online and offline scenarios, this method involves a step to sort the items in a decreasing order, according to a defined sorting strategy. Different sorting strategies will lead to different placement solutions. In this work, we present six bin packing algorithms based on FFD using six different sorting strategies, each of which has an offline version and an online counterpart, thus in total there are twelve algorithms being proposed.

A. Offline Algorithms

The most common sorting strategy in FFD is to sort the item according to their resource requirements. This work considers two dimensions of resources (CPU and memory) and a VM is sorted based on its magnitude $Rq(v)$ computed using equation 1, where the CPU_v denotes the number of CPUs required by VM v and $Memory_v$ denotes the amount of memory required by v.

$$Rq(v) = \sqrt{(CPU_v)^2 + (Memory_v)^2} \tag{1}$$

Similarly, the total capability $Cp(i)$ of a PM i can be measured by using equation 2.

$$Cp(i) = \sqrt{(CPU_i)^2 + (Memory_i)^2} \tag{2}$$

To measure the utilization of a PM, the magnitude of every VM on a given PM will be summed to determine the utilization of the PM. Let us have a PM i, a sef of VMs V_i, $v \in V_i$, which is the set of VMs placed on PM i. The measurement of PM utilization $Ut(i)$ is given in equation 3:

$$Ut(i) = \sqrt{(\sum_{v \in V_i} CPU_v)^2 + (\sum_{v \in V_i} Memory_v)^2} \tag{3}$$

In addition to the common sorting strategies, we propose six offline algorithms for VM placement, including FFD (the general FFD method), PM Load (ServerLoad), Percentage

Utilization (PercentageUtil), Absolute PM Capacity (AbsoluteCapacity), FFD Sorted PM (FFDSorted), and FFD Residual Load (FFDResidual) algorithms.

The inputs to the offline algorithms are VM set V, PM set P in the data center and the initial VM/PM mapping M. The outputs are an updated VM/PM mapping M, and VM migration schedule S including VM ID, destination PM and source PM.

Once there is an insight into how densely each PM is populated, it is then a matter of going about assessing the relative size of each VM, which is the magnitude of each VM. Next, a placement algorithm can be applied in order to firstly calculate the number of VM migrations needed to consolidate the PMs. The movement will not actually take place until the final migration schedule is determined.

Algorithm 1 Offline ServerLoad Algorithm

Input: V, P, M
Output: S, M

1: $FailedMig = 0$, $S = \emptyset$
2: **for** $i \leftarrow 1$ **to** $length[P]$ **do**
3: Calculate the occupied magnitude O[i]
4: **end for**
5: Sort P in descending order of occupied magnitude
6: **for** $i \leftarrow length[P]$ **to** 1 **do**
7: Backup P, S, M
8: Sort VMs $V[i]$ in PM i in decreasing order of magnitude
9: **for** $j \leftarrow 1$ **to** $length[V[i]]$ **do**
10: **for** $k \leftarrow 1$ **to** i **do**
11: **if** $V[i][j]$ fit into PM k **then**
12: $S \leftarrow S \cup \{V[k][j], i, k\}$
13: update $O[k]$ of PM k, M
14: Sort P in descending order of occupied magnitude
15: **end if**
16: **end for**
17: **end for**
18: **if** VMs$V[i]$ are not fully migrated **then**
19: Restore P, S, M
20: $FailedMig$++;
21: **end if**
22: **if** $FailedMig \geq FailedMigLim$ **then**
23: Report error and return
24: **end if**
25: **end for**

ServerLoad is specified in Alg. 1. A variable $FailedMig$ is initialized to 0, which counts the number of unsuccessful migration attempts. An unsuccessful migration attempt is when all the VMs on the least loaded PM are unable to be fully migrated to other PMs. The VM migration schedule S is initialized to $\emptyset$ in line 1. In lines 2-4, the occupied magnitude $O[i]$ of every PM i is calculated using equation 3. In line 5, PMs P are sorted in decreasing order based on the occupied magnitude. In lines 6-25, every PM will be processed to facilitate the migration of the VMs on the least loaded PM to the most loaded PMs. In line 7, copies of P, S and M are taken so in a situation where not all the VMs on the least loaded PM are fully migrated, the state of P, S and M can be restored. In line 8 the VMs $V[i]$ on each PM are sorted in decreasing order so that the VM with the largest magnitude is the first candidate for migration.

In lines 9-17, VM $V[i][j]$ which has the largest magnitude on PM i, (which is the PM with the lowest occupied magnitude) is the first candidate for migration. This VM is attempted to be placed on the PM with the highest occupied magnitude, if this is unsuccessful the PM with the second highest occupied magnitude is tried, and so on. If there are sufficient resources for this VM to be placed, then S is updated with that VMs identifier $V[i][j]$, the source PM i, and the destination PM k. In addition, the occupied magnitude $O[k]$ of the destination PM k is also updated to reflect the placement of the new VM $V[i][j]$, and PMs are again sorted in descending order based on occupied magnitude. This process repeats until all the VMs on the least loaded PM are migrated, if all the VMs are migrated successfully, then the process moves on and the next least loaded PM is selected, and so on. If not all the VMs on the least loaded PM are fully migrated then we will need to revert back to the previous states of P, S and M and the number of $FailedMig$ is incremented. If $FailedMig$ is greater or equal to a predefined value $FailedMigLim$ then the progress is terminated, as shown in lines 18-24.

The remaining offline algorithms all operate in a similar fashion, with only variations mentioned below:

- FFD: VMs are sorted in decreasing order and we suppose initially all the PMs are empty. Each VM in turn is then attempted to be placed on the first PM that will accommodate it, where PMs are sorted in terms of their PM IDs. The initial mapping and the final mapping created by FFD are then compared. If a VM is mapped to the same PM on both mappings then no migration is necessary, if not then the VM ID, the original PM ID and the new PM ID are added to the migration schedule S.
- PercentageUtil: The PMs are sorted in decreasing order based on the ratio of absolute capacity of the PM against the total magnitude of any VMs hosted on it.
- AbsoluteCapacity: The PMs are sorted by absolute PM magnitude in decreasing order.
- FFDSorted: operates the same way as FFD except the PMs are also sorted in decreasing order based on their absolute capacity.
- FFDResidual: operates the same way as FFD except the PMs are also sorted in decreasing order by their absolute residual capacity. Initially, the absolute residual capacity for a PM is its total magnitude.

B. Online Algorithms

After the offline VM placement or consolidation, the PMs in cloud are in intermediate state, an online packing strategy is then adopted in order to ensure that new VM requests are packed as optimally as possible with the minimum number of active PMs and high PM resource utilization. New VM requests arrive and are attempted to be placed in turn based on whatever way the particular algorithm sorts the VMs or

PMs, which are defined by the specific online algorithm. If a VM can be placed on a PM, the VM/PM mapping is updated to reflect the new VM placement. If the VM cannot be placed on any of the PMs then the request is rejected. The online ServerLoad algorithm is presented in Alg. 2.

The inputs to the online algorithms are a new VM request v, PM set P in the data center and the initial VM/PM mapping M. The outputs are an updated VM/PM mapping M.

Algorithm 2 Online ServerLoad Algorithm

Input: v, P, M
Output: M

```
for i ← 1 to length[P] do
    Calculate the occupied magnitude O[i]
end for
Sort P in descending order of occupied magnitude
for i ← 1 to length[P] do
    if v can fit into P[i] then
        place v onto P[i]
        update O[k] of PM k, M
        Sort P in descending order of occupied magnitude
        return;
    end if
end for
reject request of v
```

The remaining online algorithms all operate in a similar fashion compared to their offline counterparts described previously.

V. Performance Evaluation

A. Experiment Setup

The initial placement is created with 500 randomly generated VMs from the four VM types. Each of these VM types has a different memory and CPU capacity, which is used to calculate its magnitude. The total magnitude of all 500 VMs is calculated, then tripled, and a number of PMs are generated at random from four different PM types, until their available magnitude is greater or equal to the tripled total VM magnitude. Therefore in this experiment, the number of active PMs in the data center is dependent on the magnitude of the randomly generated VMs, and hence we can measure which algorithm uses the minimum number of PMs to host the given VMs.

VMs are then placed on PMs in a random mapping to recreate the dispersed state of a data center, which may occur over time as new VM requests arrive and old VM requests expire. Once the initial VM/PM mapping is in place, each of the six consolidation algorithms are applied. Each of these algorithms receives the same initial VM/PM mapping as an input so its performance can be accurately measured. Each algorithm will devise a new VM/PM mapping in their own way as they attempt to migrate VMs from one set of PMs to another. This balance between consolidation and VM migrations is measured using the migration efficiency metric. The migration efficiency is the ratio of the number of PMs released to total migrations.

A number of other metrics also measure various facets of the algorithm performance such as number of migrations, number of PMs used and PM utilization percentage. Number of migrations refers to the number of VMs, which need to be migrated in order to achieve a consolidated data center environment; this number should ideally be as small as possible. Number of PMs refers to the number of active PMs after the consolidation has occurred, this figure should ideally be as small as possible. PM utilization percentage is a measure of the ratio of used magnitude on all the PMs to the total magnitude of all the PMs. Ideally we would like this figure to be as close to 100% as possible.

For the dynamic placement, the test regime was run four times which meant four different sets of inputs. For each test case, we start with a data center with initial placement from time 0, and the number of VM requests is modeled as a Poisson process with average $\lambda = 20$. The VM types are randomly generated from the four different VM types. Each VM has two randomly generated numbers to represent the starting time at which the VM request arrives and the VMs lifespan. The random "seed state" is set so that the list of new VM requests generated is the same for every online algorithm so that each can be compared accurately.

B. Experiment Results

In the consolidation phase, the overall best performing algorithm was ServerLoad. This algorithm was the best performing by a small but clear margin in relation to migration efficiency as shown in Fig. 1. It was the second best performing in relation to both the number of migrations in Fig. 2 and the number of physical machines used in Fig. 3. Finally ServerLoad was one of a number of algorithms which had almost 100% server utilization as shown in Fig. 4, and therefore was one of the best performing algorithms in this respect too.

The next best performing algorithm was PercentageUtil, this was a close second behind ServerLoad regarding migration efficiency as shown in Fig. 1, the best in relation to number of migrations in Fig. 2 and along with a group of algorithms, one of the best in relation to server utilization with almost 100% in Fig. 4. However, in relation to the number of physical machines, PercentageUtil was actually the worst performing as shown in Fig. 3.

The third best performing algorithm for migration efficiency was AbsoluteCapacity as shown in Fig. 1. This algorithm was also the third best in relation to number of migrations as shown in Fig. 2 ahead of the three FFD based algorithms. In relation to number of physical machines this algorithm was actually the best performing in Fig. 3, and was one of a number of algorithms which were the best in relation to server utilization as shown in Fig. 4.

The next best performing algorithm (and best performing of the FFD based algorithms) was FFDSorted. This was the fourth best performing in relation to migration efficiency as shown in Fig. 1. In fact if the number of migrations had been approximately 25% lower, FFDSorted would have actually been the best performing in relation to migration efficiency, it is the fact that the number of migrations needed was almost the maximum possible that reduced the performance of FFDSorted. However, all of the FFD based algorithms (FFD, FFDSorted, FFDResidual) have similarly poor results in relation to number of migrations as shown in Fig. 2,

978-1-4799-3755-4/13/$31.00 ©2013 IEEE

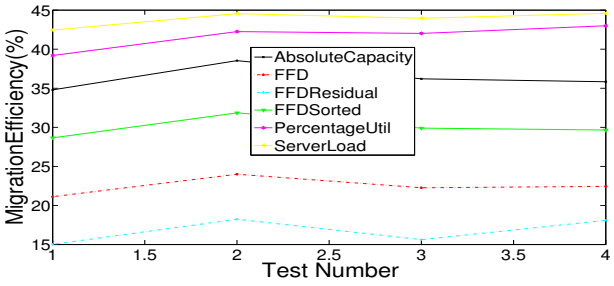

Fig. 1: Migration efficiency

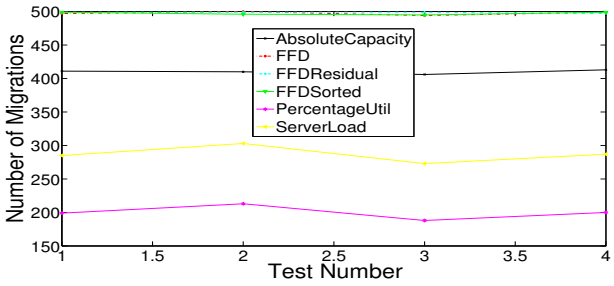

Fig. 2: Number of migrations.

and FFDSorted is no different. The poor performance of the FFD based algorithms is perhaps unsurprising given that they operate by comparing the initial random VM/PM mapping with a FFD generated mapping. In relation to the number of physical machines used, FFDSorted was actually the best performing, slightly ahead of AbsoluteCapacity in Fig. 3. In relation to server utilization, FFDSorted was among a number of algorithms which had just under total utilization as shown in Fig. 4.

The second worst performing algorithm was FFD. This was the second worst performing in relation to migration efficiency as shown in Fig. 1. As previously stated, FFD is one of the joint worst performers in relation to the number of migrations in Fig. 2 and was the fourth worst regarding the number of physical machines in Fig. 3. FFD was among a number of algorithms which had just under total utilisation as shown in Fig. 4.

By far the worst performing algorithm across a number of metrics was FFDResidual. In relation to migration efficiency FFDResidual was clearly the worst performer as shown in Fig. 1. As an FFD based algorithm, FFDresidual is one of the joint worst performers in relation to the number of migrations in Fig. 2 and was the also the joint worst performer along with PercentateUtil when it came to the number of physical machines used in Fig. 3. Even in relation to server utilization, while the other five algorithms had an almost identical level of utilization of just under 100%, FFDResidual only had a server utilization of around 40%. The poor performance of FFDResidual can be explained simply due to how the algorithm operates; VMs are placed onto the PM with the most spare capacity each time. This approach means that VMs are dispersed across a wide range of PMs with no PM very consolidated.

In relation to the online placement, as shown in Fig. 5 to 8, there were small differences in how well most of the algorithms packed dynamic VM requests in terms of PM utilization. However, it is clear that online FFDResidual was the worst performing algorithm. While this algorithm is the least effective for this particular application, there may exist other applications requiring this feature, such as load-balancing.

Although the online algorithms are based on the sorting and packing strategies of their consolidation counterparts, it is important to note that they are independent of each other, so the most effective consolidation algorithm may be unrelated to the most effective dynamic placement algorithm, and both these algorithms can work together.

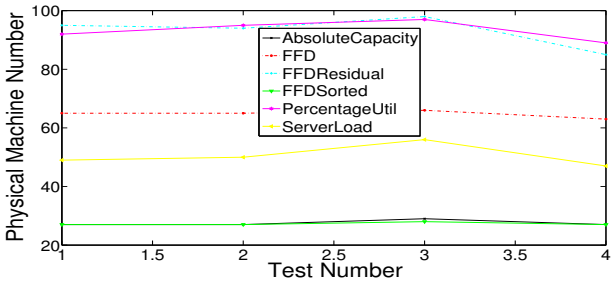

Fig. 3: Number of PMs

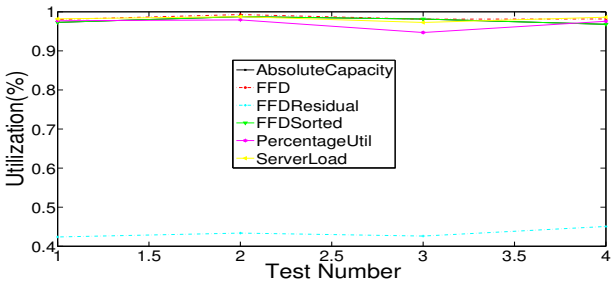

Fig. 4: PM utilization

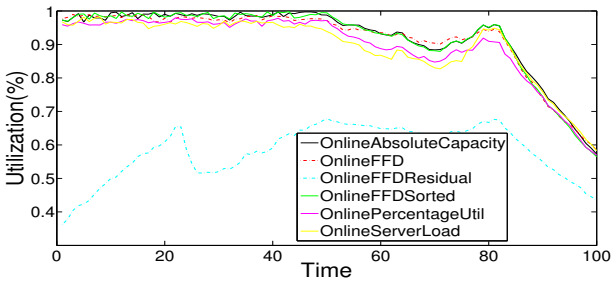

Fig. 5: Results of the online testing of the six packing algorithms showing PM Utilization Vs Time for test 1.

978-1-4799-3755-4/13/$31.00 ©2013 IEEE

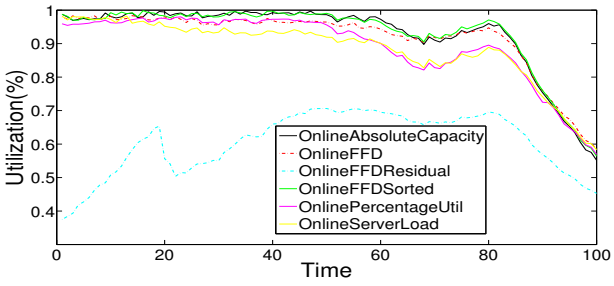

Fig. 6: Results of the online testing of the six packing algorithms showing PM Utilization Vs Time for test 2.

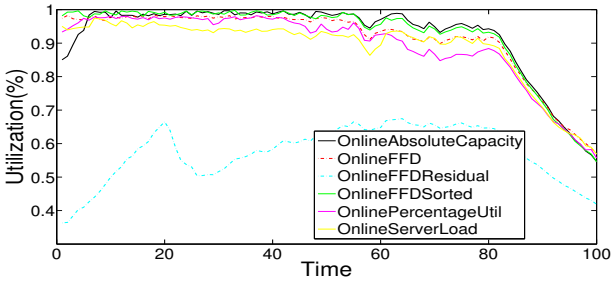

Fig. 7: Results of the online testing of the six packing algorithms showing PM Utilization Vs Time for test 3.

VI. Conclusion and Future Work

With the increased cost of energy and the sharp growth of demand, the need of energy-aware solutions has appeared as an imperative for the cloud data centers. In this work, we propose an integer linear programming model for the optimal VM placement with the objectives of minimizing the active PMs and the number of migrations for PM consolidations. We designed and evaluated a number of bin packing algorithms using different sorting strategies to determine firstly which one is most effective at consolidating VMs in data center enabling unused PMs to be turned off. Furthermore, various strategies were developed to pack dynamic VM requests in the most efficient way. Each of the six algorithms offered some improvement in the number of PMs which could be powered off due to consolidation, even when the number of VM migrations needed for this consolidation was also taken into account. The overall best performing algorithm was ServerLoad.

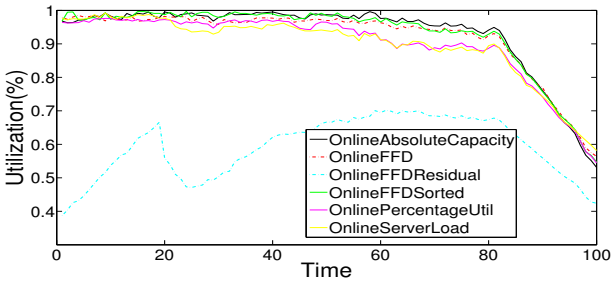

Fig. 8: Results of the online testing of the six packing algorithms showing PM Utilization Vs Time for test 4.

There are several interesting future research directions motivated by our work. Firstly the initial data center placement instead of being generated randomly could perhaps take the guidance of a combinatorial auction [13] similar to Amazon's EC2 spot instances [14]. This would ensure that the initial data center placement is a more realistic resemblance to a real world data center. Second, a number of constraints relating to fault tolerance or security could also be considered. Last but not least, PM failures should be considered for the dynamic VM placement.

Acknowledgment

This work is funded by the Higher Education Authority under the Programme for Research in Third-Level Institutions (PRTLI) Cycle 5 and co-funded under the European Regional Development Fund (ERDF) via the Telecommunications Graduate Initiative (TGI) project in Ireland.

References

[1] J. Koomey, *Growth in data center electricity use 2005 to 2010*. Analytics Press, 2011.

[2] Y. Ajiro and A. Tanaka, "Improving Packing Algorithms for Server Consolidation," in *International CMG Conference*. Computer Measurement Group, 2007, pp. 399–406.

[3] D. Wilcox, A. W. McNabb, and K. D. Seppi, "Solving virtual machine packing with a Reordering Grouping Genetic Algorithm," in *IEEE Congress on Evolutionary Computation*, 2011.

[4] R. Gupta, S. K. Bose, S. Sundarrajan, M. Chebiyam, and A. Chakrabarti, "A Two Stage Heuristic Algorithm for Solving the Server Consolidation Problem with Item-Item and Bin-Item Incompatibility Constraints," *2008 IEEE International Conference on Services Computing*, pp. 39–46, Jul. 2008.

[5] A. Murtazaev and S. Oh, "Sercon : Server Consolidation Algorithm using Live Migration of Virtual Machines for Green Computing," *IETE Technical Review*, vol. 28, no. 3, pp. 212–231, 2011.

[6] R. Panigrahy, K. Talwar, L. Uyeda, and U. Wieder, "Heuristics for vector bin packing," Microsoft Research, Tech. Rep., 2011.

[7] M. Wang, X. Meng, and L. Zhang, "Consolidating virtual machines with dynamic bandwidth demand in data centers," in *IEEE INFOCOM*, 2011, pp. 71–75.

[8] M. Lin, A. Wierman, L. Andrew, and E. Thereska, "Dynamic right-sizing for power-proportional data centers," in *IEEE INFOCOM*, Apr. 2011, pp. 1098 –1106.

[9] K. H. Kim, A. Beloglazov, and R. Buyya, "Power-aware provisioning of virtual machines for real-time Cloud services," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 13, pp. 1491–1505, 2011.

[10] A. Beloglazov and R. Buyya, "Energy Efficient Allocation of Virtual Machines in Cloud Data Centers," in *IEEE CCGrid*, 2010, pp. 577–578.

[11] ——, "Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual Machines in Cloud Data Centers," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1397–1420, 2011.

[12] M. Tarighi, S. A. Motamedi, and S. Sharifian, "A new model for virtual machine migration in virtualized cluster server based on Fuzzy Decision Making," *Journal of Telecommunications*, vol. 1, no. 1, pp. 40–51, 2010.

[13] S. Zaman and D. Grosu, "Combinatorial auction-based allocation of virtual machine instances in clouds," in *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, Washington, DC, USA, 2010, pp. 127–134.

[14] Amazon EC2 Instance Types. [Online]. Available: http://aws.amazon.com/ec2/instance-types/