VM Placement Strategies for Cloud Scenarios

Nicolò Maria Calcavecchia
Dipartimento di Elettronica e Informazione
Politecnico di Milano
Milano, Italy
calcavecchia@elet.polimi.it

Ofer Biran, Erez Hadad, Yosef Moatti
IBM Haifa Research Lab
Haifa, Israel
{biran, erezh, moatti}@il.ibm.com

Abstract—The problem of Virtual Machine (VM) placement in a compute cloud infrastructure is well-studied in the literature. However, the majority of the existing works ignore the dynamic nature of the incoming stream of VM deployment requests that continuously arrive to the cloud provider infrastructure.

In this paper we provide a practical model of cloud placement management under a stream of requests and present a novel technique called Backward Speculative Placement (BSP) that projects the past demand behavior of a VM to a candidate target host. We exploit the BSP technique in two algorithms, first for handling the stream of deployment requests, second in a periodic optimization, to handle the dynamic aspects of the demands. We show the benefits of our BSP technique by comparing the results on a simulation period with a strategy of choosing an optimal placement at each time instant, produced by a generic MILP solver.

Keywords—Cloud Computing; Placement; Datacenter; VM-migration;

I. INTRODUCTION

Cloud computing can be implemented at various level of abstraction depending on the specific service that is being offered by the cloud provider (i.e. storage, computation, application framework, etc.). In this paper we focus on the Infrastructure as a Service model (IaaS) [20] which deals with provisioning virtual machines (VMs) on top of which clients may construct their own services. A key challenge of implementing IaaS is thus determining the placement of VMs, i.e., the assignment of each VM to a host in the provider’s cloud infrastructure. The placement is subject to many constraints originating from multiple domains, such as the resource requirements of the VMs (both static and dynamic), security (isolation) requirements, availability requirements, etc. Many of the constraints may be defined in a Service Level Agreement (SLA) associated with each customer’s VM.

A fundamental aspect for cloud providers is reducing data center costs while guaranteeing the promised SLA to cloud consumers. Current virtualization technology offers the ability to easily relocate a virtual machine from one host to another without shutting it down (i.e., live migration [11]), thus giving the opportunity to dynamically optimize the placement with a small impact on performance.

The problem of cost reduction becomes even more complex when considering a continuous request stream of VMs to be deployed in the data center. Indeed, in a cloud computing context, customers might deploy and remove VM instances at any time in an unpredictable manner. This behavior may lead the infrastructure toward a suboptimal or unstable configuration (for example not exploiting free space left by VMs that are undeployed).

However, the majority of the existing works ignore the dynamic nature of the incoming stream of VM deployment requests to which the cloud infrastructure is subject over time. Moreover, VMs can show some correlation in the resource usage (i.e., for example a web server and an application server will likely have a similar CPU utilization dependency on the incoming workload). Techniques using statistical multiplexing [16], [13], [8] can exploit statistical characteristics of the resource usage among VMs for example by placing VMs that have highly probable complementary CPU demands together on the same host.

In this paper we focus on the virtual machine placement problem in a real-life dynamic cloud environment; the main contributions of this work can be summarized as:

- We introduce a novel technique called Backward Speculative Placement (BSP) that projects the past demand behavior of a VM to a candidate target host as a base for deployment or relocation decisions. This is a simple and efficient method that captures the aspect of demand correlation between the VMs in the past and use it for future prediction.
- We exploit the BSP technique for the cloud life-cycle optimization, within two scenarios. First we use it for handling the stream of deployment requests, second in a periodic optimization, to handle the dynamic aspects of the demands.
- We effectively evaluate the proposed technique in a realistic cloud scenario by modeling a stream of VM deployment requests, the VMs lifetime and demand patterns. We introduce a reasonable evaluation metrics to capture the benefit of our proposed BSP strategy, and compare the results on a simulation period with a strategy of choosing an optimal placement at each time instant, produced by a generic MILP solver - ILOG.
The remainder of the paper is organized as follows: in Section II we define the placement problem under study, while in Section III we present the proposed BSP-based strategy by explaining the two operations aimed at the placement optimization. Section IV shows the results obtained by our technique compared with a strategy of optimal placement at each time instant, using a generic MILP solver. Section V presents a literature review of works related to the VM placement problem, lastly Section VI concludes the paper.

II. PROBLEM FORMULATION

In this Section we define a mathematical time-based model for a simple cloud lifecycle. Based on this model we then formulate a placement problem that can be computed at each time instant, taking into account common optimization goals such as load-balancing and the minimization of changes between two consecutive placements.

A. Mathematical model

The problem is formulated as a multi-dimensional bin-packing problem. We consider a cloud offering a simple pay-per-use (or “on-demand”) service to users, similar to e.g., Amazon EC2 [1]. In such a cloud, we observe a stream of VM deployments that a typical cloud infrastructure is subject to [2]. We consider discrete time units where each time instant can be associated with two types of deploy events: (i) a new set of VMs needs to be deployed (i.e., placement request), or (ii) an already deployed set of VMs is turned off, or being “undeployed”. We identify each placement request with the tuple \( q = (t, VMSET) \) where \( t \) is the time instant in which the deploy request is received and \( VMSET \) is the set of VMs to be deployed. Notice that each VM can be undeployed at any time \( t' > t \). An example of this behaviour is shown in Figure 1.

![Diagram](https://via.placeholder.com/150)

**Figure 1:** An example of deployment stream. The stream involves the placement of 8 VMs spread over time instants 2, 5 and 11. In this example only VMs from 1 to 5 are undeployed while VMs 6, 7 and 8 remain in the cloud.

We consider a data center infrastructure composed by \( M \) distinct hosts. Each host is characterized by \( R \) resources (i.e., CPU, memory, IO bandwidth, etc.). Each host \( j \) has a known capacity \( C_{jk} \) for each resource \( k \), where \( j \in \{1 \ldots M\} \) and \( k \in \{1 \ldots R\} \); we identify CPU resource with index 1, \( k = CPU = 1 \).

As shown in Figure 1 a cloud life-cycle involves a sequence of events of deploying and undeploying VMs, which characterize the dynamic nature of the cloud.

Each VM \( i \) requires a minimum amount of \( r_{ik} \) units from each resource \( k \) which we call reservation. Failing to meet the minimum amount of resources required by the reservation will prevent the VM from being deployed on a host.

While the reservation value is statically defined, the demand represents the actual value of consumed resource and typically varies with time. Without loss of generality, but for the sake of simplicity, we focus only on CPU demand. In particular, we call \( D_i(t) \) the CPU demand for VM \( i \) at time instant \( t \).

Migration of VMs from one host to another opens opportunities for placement optimization but also new issues. In particular, VM relocations can introduce a significant overhead in the network infrastructure and thus deteriorate the performance of services hosted in the cloud. In order not to generate too many migrations, we limit the total number of relocations in one optimization cycle to a predefined threshold called \( \text{maxRelocations} \) which can be customized by the infrastructure owner.

B. Placement problem formulation

At each time instant the placement problem can be formulated taking into account only two types of VMs: already placed VMs and VMs that need to be deployed on that time instant; we call this set \( \text{active}(t) \). More formally, \( \text{active}(t) = \{\text{placed}(t) \setminus \text{removed}(t)\} \cup \text{added}(t) \), where \( \text{placed}(t) \) is the set of VMs that are already deployed at time \( t \), \( \text{removed}(t) \) is the set of VMs that are undeployed at time \( t \), \( \text{added}(t) \) represents the VMs that are deployed in this time instant (i.e., they will be active at instant \( t + 1 \)).

The placement solution consists of two variables: VM mapping and CPU resource allocation. The first variable expresses the association between the VM and the machine that is hosting it. More specifically, the binary variable \( x_{ij}(t) = 1 \) indicates that VM \( i \) is placed at host \( j \) at time \( t \).

The second variable we consider is the CPU allocation (i.e., the amount of CPU actually assigned to a VM). We use the symbol \( a_i(t) \) to indicate the amount of CPU assigned to VM \( i \) at time \( t \).

We assume that the data center is always big enough to accommodate each new deploy request. Given this, Equation 1 states that each one of the active VMs must be placed in a single host.

\[
\sum_{j=1}^{M} x_{ij}(t) = 1, \forall i \in \text{active}(t) \tag{1}
\]
For each resource type, each VM must be exclusively guaranteed its reservation. Consequently, the sum of reservations for VMs placed on a host should not exceed the capacity of that host.

\[ \sum_{i \in \text{active}(t)} x_{ij}(t) \cdot r_{ik} \leq C_{jk}, \forall j = 1 \ldots M, k = 1 \ldots K \] (2)

In our model the allocation of CPU is flexible in the sense that the actual allocation may be higher than the reservation declared at deployment time. Equation 3 states that the total allocated CPU between the CPU reservation and the actual CPU allocated must remain below the host capacity, where \( a_i \) refers to the CPU allocated on VM \( i \).

\[ \sum_{i \in \text{active}(t)} x_{ij}(t) \cdot a_i(t) \leq C_j CPU, \forall j = 1 \ldots M \] (3)

As allocating more CPU than the actual VM demand might bring to over provisioning of resources, we explicitly limit the allocated CPU between the CPU reservation and the demand (see Equation 4).

\[ r_{iCPU} \leq a_i(t) \leq D_i(t), \forall i \in \text{active}(t) \] (4)

Due to relevant costs involved in VM relocations (i.e., bandwidth, CPU) the new placement should be reached without generating a large number of VM migrations. This aspect is captured by Equation 5 where \( rel_i(t) \) represents the binary variable assuming value 1 if VM \( i \) is relocated to a new position (i.e., \( x_{ij}(t) \neq x_{ij}(t - 1) \) for some active host \( j \)).

\[ \sum_{i \in \text{active}(t)} rel_i(t) \leq \text{maxRelocations} \] (5)

C. Placement optimization goals

In this Section we present the optimization goal (i.e., objective function) for the placement problem. It consists of several components, each commonly-used optimization goal on its own.

The first captures the CPU demand satisfaction aspect component (i.e., how much CPU is allocated to VMs with respect to the desired amount). We refer to this component with the symbol \( AR(t) \) and its definition is given in Equation 6. Notice that, the value is within the range \([0, 1]\) thus representing the fraction of allocated demand with respect to the demand required (i.e., a value of 1 means that all the desired demand is satisfied)

\[ AR(t) = \frac{\sum_{i \in \text{active}(t)} a_i(t)}{\sum_{i \in \text{active}(t)} D_i(t)} \] (6)

In order to minimize the number of VM migrations caused by the new placement, we included a second component quantifying the number of relocations with respect to the maximum allowed. As shown in Equation 7 also in this case the value is within the range \([0, 1]\).

\[ RC(t) = \frac{\sum_{i \in \text{active}(t)} rel_i(t)}{\text{maxRelocations}} \] (7)

The last component takes into account load balancing among hosts. More specifically, we consider the actual load, (i.e., CPU allocated for each host normalized by the host capacity) according to the Equation \( LA_j(t) = \frac{\sum_{i \in \text{active}(t)} a_i \cdot x_{ij}(t)}{C_{jCPU}} \) for machine \( j \) (notice that \( a_i(t) \) is the value of CPU resource assigned by the placement algorithm, potentially different from the actual demand \( D_i(t) \)). The optimization component is defined such that the difference between the most and the least loaded host is minimized. Since some of the hosts might be empty (i.e., powered-off or in stand-by) we only consider active hosts. See Equation 8 where \( \text{hosts}(t) \) is the set of active hosts at time \( t \).

\[ LB(t) = \max_{j \in \text{hosts}(t)} \{LA_j(t)\} - \min_{j \in \text{hosts}(t)} \{LA_j(t)\} \] (8)

High values of \( LB(t) \) refer to unbalanced settings while a value close to 0 indicates a good level of balancing.

The combined optimization function is defined by Equation 9 where \( |\alpha_1| \gg |\alpha_2| \gg |\alpha_3| \) represents weights among components. Since the last two components should be minimized we impose that \( \alpha_2, \alpha_3 < 0 \).

\[ \text{maximize} \quad \alpha_1 \cdot AR(t) + \alpha_2 \cdot RC(t) + \alpha_3 \cdot LB(t) \] (9)

III. BACKWARD SPECULATIVE PLACEMENT

Due to the NP-Hard nature of the previously described placement problem, it is not possible to devise an algorithm providing optimal solutions in reasonable time. In this Section we describe Backward Speculative Placement (BSP), a placement technique which is based on a simple and fast heuristic producing high quality placements with respect to the optimization goal described in the previous Section. Notice that, due to the unpredictable nature of

\footnote{Note that the score does not capture the imbalance of load among VMs but only its total allocation. For example, given a scenario with two VMs with equal resource requirements, the situation in which the available load is entirely assigned to one VM is indistinguishable from the one in which the same load is equally spread between the two VMs. However, the imbalance effect is mitigated by the minimum assignable resource (as imposed by constrain in Equation 5).}
the deploy stream, the problem cannot be solved using traditional scheduling techniques.

We distinguish two phases for cloud placement optimization:

Continuous deployment: Cloud infrastructures continuously receive an unpredictable stream of deploy requests over time [2]. This kind of behavior demands for (i) quick placement of newly arrived deploy requests, (ii) limited number of time expensive operations (i.e., computation of new placement, relocations), and (iii) smart placement decision so that future state of the infrastructure remains stable and efficient. Due to the high responsivity requirement, the continuous deployment phase does not allow relocations of VMs (i.e., all existing VMs are locked in their host, except those with expired lifetime). The placement decisions deal only with the mapping of newly arrived VMs to hosts.

Ongoing optimization: Long periods using a continuous deployment setting might bring the infrastructure to a sub-optimal state which is far from the optimization goal defined in Section II-C. For this reason, in BSP we periodically re-optimize the placement also considering migration of VMs.

Similarly to [16], [12] we developed a placement technique inspired by statistical multiplexing and analysis of historical workload traces. In particular, BSP assumes the knowledge of past CPU demand for each virtual machine deployed in the cloud infrastructure (this data can be easily obtained with common monitoring tools). Demand traces are collected at each VM for past TM time units and conveyed toward a central software component.

A. Demand risk score

Before presenting the algorithms for each optimization phase, we introduce a scoring function which is at the core of BSP. We call it the “demand risk” and its purpose is to measure the level of demand dissatisfaction of an host in the last TM time instants. First of all, we define the function $U$ measuring the amount of unsatisfied demand for a given time instant, see Equation 10. Specifically, the function $U(h,V,t,\delta)$ measures the amount of non satisfied demand for host $h$ at time $t$ when the set of VMs $V$ is placed on it and its capacity is reduced by $1 - \delta$ percent.

$$U(h,V,t,\delta) = \sum_{l \in V} d_l(t) - \delta \cdot C_{hCPU}$$ (10)

We denote the “demand risk” with the term $DR$ and its formulation is reported in Equation 11. The demand risk contains two terms; the first measures the demand non satisfied by VMs in the last TM time instants while the second measures the unsatisfied demand as if the host would have a capacity reduced by $1 - \delta$. The two terms are linearly combined through coefficients $A$ and $B$ with $A \gg B$. Introducing the risk threshold allows to better capture spikes in demand which have a limited duration (for example due to periodic trends in demand).

$$DR(h,V,T) = A \cdot \sum_{t=T-1}^{T} U(h,V,t,1) + B \cdot \sum_{t=T-1}^{T} U(h,V,t,\delta)$$ (11)

It must be noticed that when computing the “demand risk” we modify the historical tracing in two ways: first, VMs that were deleted or were relocated from the host during the last TM instants are not considered. Second, we extrapolate the historical tracing of VMs that were deployed or were relocated to the host during the last TM instants by approximating their CPU usage prior to their effective deployment to be equal to their respective minimum reservations.

![Figure 2: Example of demand dissatisfaction](image)

B. Continuous deployment

Each time new requests to deploy VMs are received, Algorithm 1 is executed to compute their placement. The algorithm follows a decreasing best fit strategy according to the “demand risk” score.

**Algorithm 1** Continuous deployment heuristic at time $T$

1. sort $VMSET$ by decreasing demand;
2. for all $i \in VMSET$ do
   - $bestHost \leftarrow \arg \min_h DR(h, vmsInHost(h) \cup i, T)$;
   - deployVmlInHost($i, bestHost$);
3. end for

For each VM in the deploy set we evaluate the “demand risk” for each available host considering VMs already deployed plus VM $i$.

C. Ongoing optimization

Actions generated during the continuous deployment phase might bring the system to a sub-optimal allocation of resources to VMs. In order to improve the current placement we periodically activate an optimization task involving VM migrations, we call it “ongoing optimization”. The main idea behind the Algorithm 2 is to identify the most loaded hosts...
and to perform VM migrations toward hosts which are most likely to ensure a better demand satisfaction according to the function defined in Equation [11]

**Algorithm 2** Ongoing optimization heuristic at time $T$

```plaintext
for iteration = 1 → maxRelocations do
    // Identify most loaded host;
    worstHost ← arg minj DR(h, vmsInHost(h), T);
    for all vm i deployed in worstHost do
        for all j ∈ 1...M, j ≠ worstHost do
            current ←
            DR(worstHost, vmsInHost(worstHost), T) +
            DR(j, vmsInHost(j), T);
            migration ←
            DR(worstHost, vmsInHost(worstHost \ i), T) +
            DR(j, vmsInHost(j) ∪ i, T);
            migrationScorei,j ← current − migration;
        end for
    end for
    i, j = arg maxi,j migrationScore(i, j);
    if migrationScorei,j > 0 then
        add migration of VM i to host j to the list of migrations;
    else
        // Stop migrations;
        return;
    end if
    for the rest of computation, consider VM i in host j;
end for
perform migrations;
```

The loop is executed at most maxRelocations times in order to limit the number of migrated VMs. Once a host with high load is identified, all possible migrations between VMs deployed on it and all other hosts are evaluated. In each iteration of the second and third loop we compute the demand risk of the most loaded and the candidate host with the current placement. Then we evaluate the demand risk of the scenario where VM i is removed from the most loaded host and is placed in the candidate host. The score of the move is defined as the difference between the two demand risks. We then extract the maximum score and consider the migration only if its score is positive (i.e. both hosts gain from the migration).

**D. Interaction between the two phases**

As previously described, the ongoing optimization is periodically triggered in order to re-optimize the status of the cloud. It must be noticed that from the moment an ongoing optimization is triggered to the moment in which changes are effective (i.e., migrations) a sensible amount of time might elapse. We call this the critical interval. As a consequence, newly arrived deploy requests might not be placed correctly because the current placement is going to change in the future. In order to avoid this problem, whenever a new request is received within the critical interval, we delay the ongoing optimization and switch back to the continuous deployment phase. The ongoing optimization is restarted as soon as deploy requests are satisfied.

**IV. Evaluation**

In this Section we describe how the system was implemented and validated showing also the obtained experimental results.

**A. Experimental settings**

We implemented our placement heuristic and the scenario generation in Java language. Results obtained by our heuristic were compared with the ones produced by an optimization model solved through the commercial ILOG CPLEX optimization engine [3]. In particular, we only replace our heuristics (Algorithm 1 and 2) and invoke the external solver. In our experiments we considered two kinds of resources (i.e., CPU and memory). However the approach can be easily extended to any number of resources. In each experiment configuration of hosts were chosen randomly from a set of predefined capacities according to the Table I.

**Table I: Host configurations**

<table>
<thead>
<tr>
<th>Configuration</th>
<th>CPU</th>
<th>RAM</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMALL</td>
<td>100</td>
<td>100</td>
<td>0.2</td>
</tr>
<tr>
<td>MEDIUM</td>
<td>180</td>
<td>180</td>
<td>0.4</td>
</tr>
<tr>
<td>LARGE</td>
<td>450</td>
<td>450</td>
<td>0.4</td>
</tr>
</tbody>
</table>

The input stream of VMs is modeled with three probabilistic distributions capturing three different aspects of deploy workload: (i) inter-arrival time of deploy requests, (ii) deploy size (i.e., number of VMs in the deploy request), and (iii) duration of each deploy request (i.e., lifetime). In particular, the inter-arrival time between two deploy requests is modeled with a Poisson process with average $\lambda$ time units, the deploy size is modeled with a Poisson variable with average $\mu$ VMs. Similarly, the duration of each deploy request follows a Poisson process with average $\nu$. Values for these parameters are given below for each experiment. The choice of a poisson distribution is motivated by the fact that many real systems shows this property in request workloads. However a more effective evaluation would be possible using real workload traces. Similarly to hosts, VMs are probabilistically extracted from a predefined set of sizes as reported in Table II.

**Table II: VM configurations**

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Min CPU</th>
<th>Max CPU</th>
<th>RAM</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMALL</td>
<td>19</td>
<td>20</td>
<td>20</td>
<td>0.3</td>
</tr>
<tr>
<td>MEDIUM</td>
<td>50</td>
<td>90</td>
<td>50</td>
<td>0.25</td>
</tr>
<tr>
<td>LARGE</td>
<td>75</td>
<td>120</td>
<td>75</td>
<td>0.25</td>
</tr>
<tr>
<td>HI-CPU</td>
<td>80</td>
<td>140</td>
<td>40</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Experiments were performed on a Ubuntu virtual machine equipped with 4 Intel Xeon E5530 running at 2.40 Ghz and 8 GB of RAM. Experiments are performed using discrete time instants (i.e., deploy requests and optimizations only occur in these
time instants). In all experiments we fixed the ongoing optimization cycle to 25 time instants and the maximum number of VM migrations to 4. Demand traces are collected for $TM = 100$ previous time instants. Coefficients for optimization goal were fixed to $\alpha_1 = 100, \alpha_2 = −10$ and $\alpha_3 = −1$ while coefficients for the demand risk were fixed to $A = 10, B = 1$ and $\delta = 90\%$.

B. Workload generation

Beside the incoming stream of VM requests, we also generate CPU demands according to the typical night-day variation found in data centers \[12, 8\]. In particular, VMs alternate high and low demand periods having a duration extracted from two different uniform processes. Similarly to what done in \[10\], demand values are extracted from two Poisson distributions with average the minimum and maximum values allowed value for CPU as reported in Table II. Minimum and maximum values are also used as bounds for the demand such that variations are limited within a range. Moreover, VMs belonging to the same VMSET have the same night-day variation pattern (i.e., high and low periods are aligned). This assumption is motivated by the fact that these VMs are likely to implement the same service (a typical configuration is a load balancer dispatching requests to a set of web servers).

C. Experimental results

The objective of the experimental phase has been to evaluate both the quality and performance of BSP under different load conditions. In our first set of experiments we evaluate the quality of solutions produced by our technique. The evaluating scenario involves a data center of fixed dimension (150 hosts), which is put under a stream of deploy requests. Experiments have been performed using two different level of load (i.e., required CPU): the first one with $\lambda = 5, \nu = 110$ and $\mu = 15$ producing a cloud utilization of 60%. In the second level we only increase the size of deploy request to $\mu = 20$ bringing the cloud utilization to 80%.

We evaluate the quality of placement by firstly looking at the goal score (see Formula 7). This metric represents a cumulative level of satisfaction for the whole placement. As depicted in Figure 3a and 3b BSP reaches a score value between 85 and 100 in both levels of load. Downward spikes in the figure corresponds to ongoing optimizations having the effect to reduce the score due to VM migrations. Differently from the CPLEX engine, BSP takes advantage of historical demand traces by co-locating complementary VMs into the same physical machine. The effect of this approach is a sensible gain in demand satisfaction level. In Figure 3c and 3d we also report the fraction of VMs a fully satisfied demand for each time instant. While in the case with 60% load BSP is able satisfy almost 100% of VMs in the case high load (80%) the fraction of satisfied VMs ranges between 75% and 100%, still remaining superior to placements computed by CPLEX.

Table III reports cumulative results concerning the previously described experiment. Execution time refers only to ongoing optimizations as these are the most computational intensive tasks. With respect to CPLEX engine, BSP greatly reduces the computation time of ongoing optimizations while maintaining an high quality of satisfied demand (i.e., and a similar number of relocations). For example, in the heavy load case the average ongoing optimization in CPLEX takes 50 seconds while BSP only takes 0.054 seconds.

In order to better evaluate the scalability of BSP we performed a second set of experiments by varying the size of the data center. The number of VMs is iteratively increased while maintaining the same level of load (70%) in the data center. Due to the extremely long computation times needed by CPLEX we were not able to obtain solutions in a reasonable time (i.e., less than 15 minutes). Conversely, BSP is able to compute a new placement within an average execution time of 2 seconds. Results, depicted in Figure 4 show that BSP is able to maintain an average demand satisfaction level close to 95%. Similarly, the fraction of VMs with a satisfied demand remains high despite the increasing difficulty of the problem.

V. RELATED WORK

The continuous advancements in virtualization technology opened a brand new research area in data center optimization both in performance and in cost. Both the industry and the open source community currently offer solid solutions which are able to provide isolation and migration features for VMs \[21, 7\]. Moreover, the growing diffusion enabled by the cloud computing paradigm, raised the need for tools able to manage the whole cloud infrastructure at higher levels of control; an example of these tools are IBM WebSphere CloudBurst \[4\], Novell PlateSpin Recon \[6\] and Lanamark Suite \[5\].

An important aspect of data center management is the cost involved in maintaining it up and running without violating the SLA stipulated with customers. VM consolidation together with dynamic placement techniques are the main approaches to reduce costs as they are able to find (sub) optimal placement and resource allocation of VMs. The problem of VM placement is typically modeled as a multidimensional bin-packing problem which is NP-hard. The research community made an important contribution
in this sense by proposing various heuristics able to scale realistic sizes of modern data centers while maintaining a high level of quality for solutions. For example the technique proposed in [18], strives to optimize the CPU demand of each VM by shifting load among VMs in order to reduce the placement change task performed afterwards. In [9] the placement problem is reformulated as a multi-unit combinatorial auction and then solved with a column generation technique which is common for solving large integer programming problems. Similarly, the authors of [19] reduce the solution search space by identifying a subset of physical machines that better suit placement reconfigurations and then apply a constraint satisfaction technique to this subset of infrastructure.

VMs tend to follow patterns in their resource demands. A common example is CPU utilization of information services which is higher during the day and lower during the night. Such patterns can be exploited to derive more efficient placement solutions. A study [8] of a large number of CPU traces from production servers shows that most of the demand traces have high correlation with a periodic behaviour. The key concept of statistical multiplexing is to exploit possible correlations among VMs in order to better consolidate VMs with complementary resource demands [16], [13], [8]. Other approaches, perform a continuous monitoring on all running VMs and extract significant resource usage patterns which are then exploited by placement heuristics [12].

Recently the problem of VM placement has been extended to include other important aspects of the data center infrastructure such as network traffic and storage facilities. In particular, VMs deployed in cloud infrastructures typically show a network traffic dependency which can be better accommodated by consolidating interdependent VMs in the same physical machine [22]. Interestingly, network topology and the structure of the data center highly influence the choice of the placement under traffic optimization goal [17]. Similar dependencies appears also between VMs and storage resources which must serve different clients. In this case, applications requiring higher IO throughput can be placed closer to storage [14].

Similarly to our work, the authors of [15] consider a continuous stream of deploy requests directed to the cloud; however, they exploit the knowledge of the exact duration of each deploy set. We argue that this assumption is too simplistic in large cloud providers, where deploy requests come in an unpredictable manner and have unknown durations.

VI. CONCLUSION AND FUTURE WORK

In this paper we presented BSP a placement technique that optimizes the placement of Virtual Machines in a cloud environment under a continuous stream of deploy requests.
By monitoring historical demand traces of deployed VMs, BSP projects the past demand behavior of a VM to a candidate target host, and capture the VMs correlation aspect in an efficient way. We divide the placement decisions in two different phases: (i) continuous deployment, where the placement decision is taken upon the arrival of a deploy request and no migrations are allowed, and (ii) ongoing optimization where the current placement is re-optimized by relocating VMs to other hosts. Results show that our technique is able to place VMs such that a high level of demand satisfaction is obtained.

Possible future works involve studying the network relations among VMs belonging to the same deploy set. This knowledge may be useful for further infrastructure optimizations. Finally, another interesting study is the pattern/trend analysis of deploy requests belonging to the same user.

ACKNOWLEDGMENTS
This research has been partially funded by the European Commission, under projects SMSCom (IDEAS-ERC 227977) and S-Cube (FP7/2007–2013 215483).

REFERENCES