Modeling for Dynamic Cloud Scheduling via Migration of Virtual Machines

Wubin Li, Johan Tordsson, and Erik Elmroth
Department of Computing Science and HPC2N
Umeå University, Sweden
{wubin.li, tordsson, elmroth}@cs.umu.se

Abstract—Cloud brokerage mechanisms are fundamental to reduce the complexity of using multiple cloud infrastructures to achieve optimal placement of virtual machines and avoid the potential vendor lock-in problems. However, current approaches are restricted to static scenarios, where changes in characteristics such as pricing schemes, virtual machine types, and service performance throughout the service life-cycle are ignored. In this paper, we investigate dynamic cloud scheduling use cases where these parameters are continuously changed, and propose a linear integer programming model for dynamic cloud scheduling. Our model can be applied in various scenarios through selections of corresponding objectives and constraints, and offers the flexibility to express different levels of migration overhead when restructuring an existing infrastructure. Finally, our approach is evaluated using commercial clouds parameters in selected simulations for the studied scenarios. Experimental results demonstrate that, with proper parametrizations, our approach is feasible.

Index Terms—cloud computing, dynamic scheduling, virtual machine placement, migration overhead, linear integer programming

I. INTRODUCTION

As the use of cloud computing grows and usage models [1] become more complex, cloud users are confronted with obstacles in integrating resources from various cloud providers. In this context, the use of efficient cloud brokering mechanisms is essential to negotiate the relationships between cloud service consumers and providers, including integrating cloud services to make up a user’s cloud environment. A cloud broker also helps users prevent potential vendor lock-in problems by means of migrating applications and data between data centres and different cloud providers.

However, current brokering approaches are limited to static scenarios, where changes in characteristics such as pricing schemes, virtual machine (VM) types, and service performance throughout the service life-cycle are ignored. Conversely in dynamic scenarios, it is arguable that either the offers of the cloud providers or the requirements of the service owner change over time. When conditions change, it is necessary to analyse how to optimally reconfigure the service to adapt it to new situations. For example, if a vendor retreats from the market, cloud users may be forced to migrate some resources from one cloud provider to another so as to guarantee the service availability. Similarly, when a price reduction occurs, the current configuration may become suboptimal, and it may be possible to obtain a better placement of resources by restructuring the virtual infrastructure.

In this paper, we focus on modeling for dynamic scheduling in the context of cloud brokerage where cloud users employ multiple cloud infrastructures to execute their VMs in which business services are encapsulated. In dynamic scheduling scenarios, the ability to efficiently migrate VMs between servers or data centres is crucial for the efficient and dynamic resource management. VM migration is essential to increase the flexibility in VM provisioning, avoid vendor lock-in problems, and guarantee the service availability, etc. One of the key issues for dynamic cloud scheduling is finding a suitable metric for VM migration overhead, a metric that captures the distance between two infrastructures in order to estimate the feasibility of restructuring an existing infrastructure. Possible infrastructure distance metrics include number of VMs to migrate, number of VMs to migrate weighted with VM size, and total migration downtime, etc. Another issue is how to express and embody the migration overhead metric in an objective function that can equivalently represent the user’s goal. To tackle these problems, we investigate and classify multiple dynamic scenarios and propose a linear integer programming model. With proper parametrization and selections of objective functions and constraints, our model can be used in a wide range of scenarios. The optimization problem is finally encoded in a mathematical modeling language and solved using a linear programming solver.

In summary, our contributions are the following. We investigate dynamic cloud scheduling use cases and propose a linear integer programming model for dynamic cloud scheduling via VM migration across multiple clouds. Evaluations based on characteristics of current commercial cloud offerings demonstrate that our model provides the flexibility of expressing different levels of migration overhead when restructuring an existing infrastructure. By proper parametrizations, our approach can be used to accurately decide optimal VM migration strategies for elasticity scenarios, as well as handling changes in provider offers and prices.

The remainder of the paper is organized as follows: Section II describes background about cloud brokerage, placement optimization for cloud resources, and VM migration. Section III introduces cloud brokering mechanisms for optimized placement of VMs across multiple providers, describes the proposed model, and elaborates its flexibility for expressing different levels of migration overhead for restructuring an existing infrastructure. Section IV presents experimental evalu-
ations against commercial clouds offerings. Finally, some conclusions are presented in Section V followed by a presentation of future work, acknowledgments, and a list of references.

II. BACKGROUND AND RELATED WORK

A. Cloud Brokerage

Cloud brokerage aims to bridge the gap between the cloud service consumer and the provider. Gartner Research divides the responsibility of cloud brokers into three main categories: cloud service intermediary, aggregation and cloud service arbitrage [2]. On-going research on cloud brokerage has caught substantial attention, including efforts that target cloud management middleware (e.g., Emotive Cloud [3] and OpenNebula [4]), virtualization APIs (e.g., libvirt [5]), and cloud interoperability and standardization.

Grivas et al. propose a central Cloud Broker component responsible for the management and the governance of the cloud environment [6]. However, this approach is mainly focusing on business process management. It should be remarked that this approach is still in the phase of comprehensive architecture design. A cloud broker with an optimal VM placement algorithm is presented by Chaisiri et al. [7]. This algorithm can minimize the cost for hosting VMs in a multi-provider environment. This work is however limited to static scenarios where the number of required virtual resources is constant, and the cloud provider conditions (resource prices, resource availability, etc.) do not change throughout the service life-cycle.

B. VM Placement Optimization for Clouds

Virtual machine placement in distributed environments has been studied in the context of cloud computing extensively, e.g., by Bobroff et al. [8] who present a management algorithm for dynamic placement of VMs to physical servers, which provides substantial improvement over static server consolidation in reducing the amount of required capacity and the rate of Service Level Agreement (SLA) violations. Their algorithm pro-actively adapts to demand changes and migrates VMs between physical hosts thus providing probabilistic SLA guarantees. Another SLA-driven dynamic VM placement optimization approach is proposed by Iqbal et al. [9], who describe the problem of bottleneck detection and resolution of multi-tier Web applications hosted on a cloud. They present a solution to minimize the probability that tiers (hosted on VMs) become bottlenecks by optimizing the placement of VMs.

For VM placement optimization in a single cloud, Andreolini et al. [10] present a management algorithm to reallocate the placement of VMs for better performance and resource utilization by considering the load profile of hosts and the load trend behaviour of the guest instead of thresholds, instantaneous or average measures that are typically used in literature. VM placement optimization for multi-cloud scenarios is studied e.g., by Chaisiri et al. [7], Moreno-Vozmediano et al. [11] [12] and Tordsson et al. [13]. However, so far, most of efforts that target VM placement optimization for clouds have focused on either scenarios of one single cloud provider or static scenarios in multi-cloud environments. VM placement issues for dynamic scenarios across multiple cloud providers remain largely unexplored.

C. Virtual Machine Migration

Leveraging the ability of VM migration, cloud users are able to switch data and services between different physical machines in a cloud or even different clouds. In this paper, we consider a VM to be migrated if either it is moved from one cloud to another, or its hardware configuration is changed. VM migration is inevitable when reconstructing virtual infrastructure for cloud users in cloud brokerage scenarios.

Heterogeneous live migration of virtual machines is studied by Liu et al. [14]. Their work demonstrates that due to high variances of memory page dirtying rate, it is possible to get very slow migrations (result in long downtime) although a VM uses only 156MB of memory. Another comprehensive study of VM migration research, as well as an evaluation of methods for efficient live migration of VMs is presented by Svärd et al. [15], who also demonstrates how live migration of large VMs or VMs with heavy load can be done with shortened migration time, migration downtime, and reduced risk of service interruption.

While VM migration research has currently focused on single-cloud scenarios where data and services are located within the same cloud infrastructure, we expect that VM migration across different cloud providers will become a reality in a near future. In our work, the time and cost for VM migration are approximated by looking at the time required to shut down a VM in one cloud provider and start a new VM with the same configurations in another provider.

III. SYSTEM MODEL AND PROBLEM DEFINITION

A. Cloud Brokerage and Modeling

Figure 1 illustrates three roles in the studied cloud brokerage scenario: the User, the Cloud Providers, and the Cloud Broker. The user requests a virtual infrastructure by submitting a service description, which contains a manifest of required resources (e.g., number of VMs, size of storage, etc.), optimization criteria, and a set of constraints to the cloud broker.
The Scheduling Optimizer component of the broker generates an Execution Plan based on requirement criteria provided by the user, the offerings of the available cloud providers, and the change of situation (e.g., service performance scales up or down, cloud providers’ offers change and so forth). The Execution Plan includes either a list of VM templates that can equivalently represent the implementation of the user’s abstract infrastructure request, or a description that represents an adjustment of an existing infrastructure. Finally, the Execution Plan is enacted by the Virtual Infrastructure Manager component that is built on a cloud management middleware such as Emotive [3] and OpenNebula [4]. We remark that in this paper we focus on problem formulations and modeling, while difficulties and challenges involving cloud interoperability, robustness of migration, and similar practical matters, although important for a full implementation, are out of scope.

Each cloud provider supports several VM configurations, often referred to as instance types. An instance type is defined in terms of hardware metrics such as main memory, CPU (number of cores and clock frequency), the available storage space, and price per hour. Our model has no limitations on the number of instance types. While we currently use five instance types, i.e., micro, small, medium, large, and xlarge (see Table I in Section IV) for the evaluation in this paper, it is straightforward to extend or decrease the number of instance types.

More formally, in a static scheduling scenario, our goal is to deploy \( n \) VMs, \( v_1, v_2, \ldots, v_n \), across \( m \) available clouds, \( c_1, c_2, \ldots, c_m \), which provide \( l \) possible instance types, \( IT_1, IT_2, \ldots, IT_l \), to improve criteria such as cost, performance, or load balance. The hourly computational capability of a given instance type is denoted \( C_{jk} \), \( 1 \leq j \leq l \). The hourly price for running a VM of instance type \( IT_j \) in cloud \( c_k \) is denoted by \( P_{jk} \). One of the most common used objective function is to maximize the Total Infrastructure Capacity (TIC) of the deployed VMs given by:

\[
TIC = H \sum_{j=1}^{l} C_{j} \left( \sum_{i=1}^{m} \sum_{k=1}^{n} x_{ijk} \right),
\]

where \( H \) specifies the expected lifetime of the infrastructure, i.e., how long the virtual infrastructure is to be deployed, \( x_{ijk} \) is a decision variable that satisfies \( x_{ijk} = 1 \) if \( v_i \) is of type \( IT_j \) and placed at cloud \( c_k \), and 0 otherwise. Users can specify the following types of deployment restriction constraints:

- **Budget constraints** - Let Budget denote the maximum investment that can be used. Now, the deployment is restricted to solutions where the total infrastructure price (TIP) satisfies

\[
TIP = H \sum_{j=1}^{l} \sum_{k=1}^{m} P_{jk} \left( \sum_{i=1}^{n} x_{ijk} \right) \leq \text{Budget}. \quad (2)
\]

- **Placement constraints** - Each VM has to be of exactly one instance type and placed in exactly one cloud:

\[
\forall i \in [1..n] : \sum_{j=1}^{l} \sum_{k=1}^{m} x_{ijk} = 1. \quad (3)
\]

- **Load balancing constraints** - can be encoded as:

\[
\forall k \in [1..m] : \quad LOC_{\min} \leq \frac{\sum_{i=1}^{n} \sum_{j=1}^{l} x_{ijk}}{n} \leq LOC_{\max}, \quad (4)
\]

where \( LOC_{\min} \) and \( LOC_{\max} \) are the minimum and maximum percent of all VMs to be located in each cloud.

Note that additional constraints, such as for example network resource requirements, and data locality restrictions can also be added to the model.

As studied by Tordsson et al. [13], the static cloud scheduling problem on performance goals can be formulated as a linear programming model with objective function (1) and constraints (2), (3), and (4). In static scenarios, parameters \((n, l, m, P_{jk}(1 \leq j \leq l, 1 \leq k \leq m), \text{and Budget})\) are constant throughout the service life-cycle where placement decisions can be taken off-line, once only, and in advance to service deployment.

**B. Dynamic Cloud Scheduling**

In dynamic scenarios, any of the previously discussed parameters may change. We identify two main categories of dynamic scenarios of cloud scheduling, which respectively reside in cloud providers and service providers: varying cloud providers offers, and service performance elasticity.

- **Examples in the first category include varying offers:**
  - A new provider appears or withdraws from the offer space. For example, Heroku [16] proclaimed the availability of the commercial version of its new cloud hosting and deployment service on 2009-04-24.
  - Price changes, e.g., in form of new agreements, spot prices, special discount during night time, etc.
  - New instance type offers are introduced, e.g., Amazon announced Micro Instances for EC2 on 2010-09-09 [17].

- **Examples in the second category (service elasticity):**
  - In this case, the service owner wants to scale up or down the performance while optimizing the cost.
    - The service owner adds or removes a mail server from a current infrastructure.
    - The service owner increases or decreases the budget investment, e.g., budgetary reduction during recession.

In some scenarios, e.g., price reduction, the cloud user is offered an opportunity to obtain a better placement of VMs, while in other scenarios, e.g., an in-use cloud vendor withdraws from the market, the cloud user is forced to reconstruct the current infrastructure, striving to guarantee the service availability. Therefore, possible objectives can be identified as follows:

I. **Maximize** the possible new \( TIC \) with consideration of VM migration overhead under new situations.

II. **Minimize** the possible new \( TIP \) while obtaining a new \( TIC \) that can satisfy new performance demands.

III. **Minimize** the overhead of reconstruction a current configuration. The rationale behind this is service continuity. The
more VMs the broker has to start and/or shut down, the more
the service is impacted by the change.
To model dynamic scenarios, we introduce several notations:
- \( MC \) - denotes the overhead of changing the current place-
  ment to a new service layout. It can be defined in terms of
system downtime, the number of VMs migrated, etc.
- \( \beta_i \) - denotes the service downtime penalty per time unit,
  which can be defined either in terms of system failure, or
  monetary loss.

\( MC \) is given by:

\[
MC = \sum_{i=1}^{n} (\beta_i \ast \Delta_i),
\]

where \( \Delta_i \) denotes the overhead of migrating VM \( v_i \). For VM
\( v_i \), \( \Delta_i \) depends on its previous instance type, its new instance
type, the previous cloud it is placed in, and in which cloud
it is about to be located. To calculate \( \Delta_i \), we introduce \( M \),
where \( M_{i,j'}^{k'} \) denotes the overhead for migrating VM \( v_i \) of
instance type \( IT_{j'} \) in cloud \( c_{k'} \) to cloud \( c_k \) with instance type
\( IT_j \).

\[
x_{ij'k'} \ast \Delta_i
\]
denotes the current placement status for VM \( v_i \), Notably, here \( x_{ij'k'} \)
is a constant that denotes the current placement status for VM \( v_i \),
despite \( x_{ij'k'} \) is a decision variable for the new model. Now we get:

\[
\Delta_i = \sum_{j'=1}^{l} \sum_{k'=1}^{m} x_{ij'k'} \ast \Delta_i
\]

We remark that both \( x_{ij'k'} \) and \( x_{ij} \) are sparse 0-1 matrices
that satisfy \( \sum_{j'=1}^{l} \sum_{k'=1}^{m} x_{ij'k'} = 1 \) and \( \sum_{j'=1}^{l} \sum_{k'=1}^{m} x_{ij'k'} = 1 \) for each \( i, 1 \leq i \leq n \). Consequently, the expression for
\( \Delta_i \) is neat and fast to compute although the formulation in
equation (6) is in the form of four-layer nested accumulated
operation. Now, Objective III can be modelled and formulated using equations (5) and (6). Objective I can be expressed as

maximize the the TIC that is given by:

\[
TIC = H \ast \sum_{j=1}^{l} C_{i,j} (n_i \sum_{k=1}^{m} x_{ijk}) - MC
\]

\[
= H \ast \sum_{j=1}^{l} C_{i,j} (n_i \sum_{k=1}^{m} (x_{ijk} \ast \beta_i \ast \Delta_i)).
\]

Hence, Objective I is formulated using equations (6) and (7).
We remark that the TIC can also be a constraint and TIP can
be an objective function in the dynamic model. For example,
a new model can be formulated as:

\[
\text{Minimize : } TIP = H \ast \sum_{j=1}^{l} P_{jk} (\sum_{i=1}^{n} x_{ijk}).
\]

\text{Subject to :}

\[
TIC = H \ast \sum_{j=1}^{l} C_{i,j} (n_i \sum_{k=1}^{m} x_{ijk}) \geq \text{Threshold}
\]

where the user wants to minimize the TIP while maintaining
the TIC in a certain level. To conclude, three forms of the
model are identified:
- Model I: maximize objective function (7), with equation (2),
  (3), and (4) as constraints. A cloud broker employs this model
to obtain an optimal infrastructure capacity that also takes migration overhead into account.
- Model II: minimize objective function TIP (2), with equation
  (3) and (4) as constraints. The goal of this model is minimization of the price of the new infrastructure, while
keeping the capacity above than a certain threshold.
- Model III: minimize objective function (5), with equation
  (2), (3), and (4) as constraints. This model is used when
the cloud broker minimizes the migration overhead,
and meanwhile fulfills the constraints for budget, placement,
and load balancing.

C. Model Parametrization and Application

In our model, \( \beta_i \) signifies the weight of migrating VM \( v_i \). We
argue that the overhead for migrating different VMs differs,
e.g., the overhead of migrating a backup server is lower than
that of migrating a server running an ERP system.

By assigning suitable values to \( \beta_i \), we have \( 0 \leq \beta_i \leq \beta \).
In our model, \( \beta_i \) is a constant that denotes the current
placement status for VM \( v_i \) where \( \beta_i \) is a decision variable
for the new model. Now we get:

\[
\beta_i = \sum_{j'=1}^{l} \sum_{k'=1}^{m} (x_{ij'k'} \ast \Delta_i)
\]

Infeasible migration can be modelled through \( \infty \)-assignment,
e.g., assignment \( M_{i,j',k'} \) \( \infty \) specifies that it is
impossible to migrate VM \( v_i \) of instance type \( IT_{j'} \) placed
in cloud \( c_{k'} \) to cloud \( c_k \) and of instance type \( IT_j \).
In practical applications, \( \beta_i \) and \( M \) can be estimated based on practical
experience of the used cloud platforms and data collection
to learn the behaviour of the migration applications.

Algorithm 1 Cloud re-scheduling

Input:
- Environment changes, e.g., updated prices, VM numbers,
  budget, cloud provider configurations, etc.

Output:
- New placement after reconfiguration;
1: Select optimization criteria (including objective and
  constraints selection);
2: Determine parameter \( \beta \);
3: Determine parameter values in matrix \( M \);
4: Solve problem for criteria selected in step 1;
5: Migrate VMs if the solution is feasible;

This VM placement problem is known to be \( \text{NP hard}.\)
Existing approximation and heuristic algorithms can scale to
at most a few hundred machines, and may produce solutions
that are far from optimal when system resources are tight [18].
An in-depth study of integer programming scalability is given
by Alper et al. [19]. Instead of proposing new approximation
algorithms, we encode our model using a mathematical mod-
ing language and use state-of-the-art linear programming
solvers. Optimizations for improving the scalability problem
and complexity investigation are left to future works.
IV. EVALUATION AND DISCUSSION

We evaluate our approach using imaginary service scenarios based on performance figures from contemporary clouds offerings. Notably, our goal is not to evaluate the various providers but rather to investigate how well our proposed cloud brokering approach can adapt to changes in realistic scenarios. Two commercial cloud providers are used to model in our experiments. The first one is GoGrid [20], and the second is Amazon EC2 [21]. EC2 offers two separate clouds, one is in the USA, the other in Europe. These three clouds are henceforth referred to as EC2-US, EC2-EU and GoGrid. To solve the optimization problem, we use AMPL [22] as the modeling language and Gurobi [23] as the binary integer programming problem solver.

A. Experimental Setting and Parameter Estimation

We consider five different VM instance types, their hardware characteristics and prices are listed in Table I.

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>micro</th>
<th>small</th>
<th>medium</th>
<th>large</th>
<th>xlarge</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU (# cores)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>CPU (GHz/core)</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Memory (GB)</td>
<td>0.613</td>
<td>1.7</td>
<td>3.5</td>
<td>7.5</td>
<td>15</td>
</tr>
<tr>
<td>Storage (GB)</td>
<td>50</td>
<td>160</td>
<td>300</td>
<td>850</td>
<td>1700</td>
</tr>
<tr>
<td>Computing Capacity</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
</tr>
</tbody>
</table>

In the new instance type scenario (see Section IV-B ) and the price change scenario (see Section IV-C), we set $LOC_{min} = 30\%$ and $H = 1\, hour$. These two scenarios are evaluated for one hour, and each cloud should host at least $30\%$ of the VMs. To estimate parameter $B_i$ and the values in matrix $M$, we use the service downtime statistics (see Table II) presented by Iosup et al. in [24] and [25] to calculate the computation capacity losses of the infrastructure.

More specifically, $B_i = C_{j'}$, $M_{i,j',j,k'} = Downtime\, of\, VM_{i}$, where $Downtime\, of\, VM_{i}$ is the sum of Release Time of $v_i$ of instance type $IT_{j'}$ placed in $c_k$ and Allocation time of $v_i$ of instance type $IT_{j}$ placed in $c_k$. Notably, $M_{i,j',j,k'}$ can be ignored if $H$ is large enough.

In the following, three dynamic cloud scheduling scenarios are selected to evaluate our proposed model. In all experiments, the number of VMs ($n$) to be deployed is 32. 

B. Scenario I: New instance type offers

In this case, we consider a service owner who has a limited budget, $5$ per hour to run 32 VMs. At first, there are only four instance types available - small, medium, large and xlarge, and then we simulate the event that the micro instance type is introduced [17].

![Fig. 2. VM placement with and without the micro instance type.](image)

In our experiment, the user obtains an optimal total infrastructure capacity ($TIC$) of 78 by placing 31 VMs of small instance type and 1 VM of xlarge (at EC2-US) instance type. This situation changes when the micro instance type is announced. As Figure 2 illustrates, the virtual infrastructure is reconstructed accordingly. The number of micro instances is increased from 0 to 27, while the number of small instances is decreased from 31 to zero. Since the micro instance type is offered at a very low price (see Table I), the system now can improve the investment proportion for other instance types with larger computing capacity: the number of large instances is increased from 0 to 1, and the number xlarge instances is increased from 1 to 4. As a result, the $TIC$ is increased by 22% to 95.2 with no need to increase the budget.

Figure 3 shows how the performance of the infrastructure changes in the first 800 seconds. In this figure, there are two obvious inflection points (encircled in the figure) which indicates the significant growth of capacity for the infrastructure with micro instances. The first inflection point is after 90 seconds before only one VM is running in the infrastructure of micro instance type; afterwards, the cloud broker completes migration processes for 11 VMs and restarts them. The second inflection point is after around 280 seconds, when 20 more VMs are rebooted after migrations. After 610 seconds, the performance of the infrastructure with micro instance surpasses the one without micro instances, and the difference expands increasingly as time elapses as illustrated in Figure 4. In this case, we can conclude that, it is worthy to perform migration if the infrastructure is to run for more than 10 minutes. This evaluation demonstrates that our cloud...
Fig. 3. Performance improvement with and without the micro instance type.

Fig. 4. Performance improvement with and without the micro instance type.

Brokering mechanisms can handle the scenarios with new instance types. Interestingly, the proposed mechanisms can accurately determine the break-off point when the improved performance resulting from migration outweighs the migration overhead.

C. Scenario II: Prices change

In this second experiment, we first simulate an imaginary scenario where cloud providers offer a price discount of 20% during the night time due to less energy consumption. To study the effect of this, we increase the budget from $5 per hour to $60 per hour in 55 steps. We then calculate the $TIC$ values under three different scenarios: static placement with old prices, static placement with new prices ignoring migration overhead, and dynamic placement with new prices and consideration of migration overhead.

We observe in Figure 5 that, for lower budgets, the performance improvement due to price discounts is more significant. The performance difference between two price offers (i.e., original prices and prices after discount) is notable, and despite the consideration of migration overhead, the new optimal $TIC$s are very closed to the values in the static scenario, especially when the budget is lower than $20 per hour. However, when the budget is higher than $48 per hour, there is no difference among the three scenarios. This is because the budget is excessive compared to the VMs to be deployed and the price offers, and hence, the broker does not migrate any VM even if the prices are lowered. To use all the budget, the broker may suggest the service owner to deploy more VMs (as discussed in the next scenario), so that the performance can be improved further.

Fig. 5. VM placement with and without price discount.

We also explore the behaviour of our model under the condition that only one of the cloud provider (i.e., GOGRID) offers price discounts. We set $Budget = $5 per hour. Due to the load balancing constraint (4), each cloud hosts at least 30% of the VMs (notably, $32 \times 30\% \approx 10$) and thus at most 12 VMs. Since GOGRID is the most expensive cloud provider, to fulfil the minimum requirement for loading balancing, the cloud broker assigns only 10 VMs (of small instance types) to it, and obtains a $TIC = 99$ (see Figure 6).

As illustrated in Figure 6, the cloud broker manages to obtain higher TICs as the discount offered by GOGRID

Fig. 6. VM placement with varying prices discount by GOGRID.
increases. The number of VMs hosted in GOGRID is increased from 10 to 12. The cloud broker first tries to increase the number of VMs of larger instance types, e.g., when the price discount is 30%, the number of small instances increases from 0 to 1, while the total number of VMs located in GOGRID does not change. When the discount is larger (i.e., ≥ 60%), the number of VMs of small instance types is scaled up to 5 and the total number of VMs located in GOGRID increases to 11.

Resources allocation for instance type medium, large and xlarge in GOGRID cloud is comparatively time-consuming (see Table II), and therefore the cloud broker does not assign any medium, large or xlarge instance in GOGRID even when the prices discount increases to 60%. However, 7 xlarge instances and 1 medium instance are employed when the discount comes to 80% which means that the cost for hosting more VMs or upgrading VMs with more computing power in GOGRID is inexpensive enough and the benefit from it suppresses the overhead arises from VM migrations.

In these experiments, we do not consider the overhead of re-migrating the infrastructure when the day time returns at the end of the discount period. One way of incorporating this could be to simply multiply MC by 2 (migration to and from new infrastructure), but this is a simplification as the previous infrastructure needs not be optimal, unless we know that we after the discount period will re-migrate the infrastructure to the original layout.

To summarize, this evaluation demonstrates that our cloud mechanism can cope with scenarios with changes in price. Performance change, as well as transformation of VM distribution across cloud providers evolved with prices change can be precisely calculated through the proposed approach.

D. Scenario III: Service performance elasticity

In this scenario, the service owner needs to increase the infrastructure capacity due to business growth. Before the expansion, $5 is invested per hour, and the service owner obtains TICs of 115, 108, 102 and 99 per hour under load balancing (LB) constraints 0%, 10%, 20%, and 30% respectively. To fulfil the new business demands, the service owner needs to increase the budget so as to obtain a new TIC of 230 per hour. This goal can be done either through adding certain amount of new VMs without migrating any running VMs, or by migrating some running VMs and meanwhile adding some new VMs.

Figures 7, 8, 9 and 10 illustrate how the minimum budget and infrastructure reconfiguration overhead (IRO) evolve with the number of new VMs added for these two options. In this experiment, we define the IRO the sum of resource release time for VMs shut down weighted with VM size and resource allocation time for VM booted weighted with VM size, and it is given by:

\[ IRO = \sum_{V_i \text{ is shut down}} (RT_i \times ComputingCapacity_{V_i}) + \sum_{V_i \text{ is booted}} (AT_i \times ComputingCapacity_{V_i}) \]

where \( RT \) denotes recourse release time of shutting down a VM, \( AT \) denotes resource allocation time of booting a VM, and the computing capacity of VM depends on its instance type, i.e., \( ComputingCapacity_{V_i} = C_j \) if \( VM_i \) is placed with instance type \( j \). IRO indicates the capacity loss when re-constructing an existing infrastructure. Notably, IRO is a dynamic form of \( MC \) mentioned in Section III-B, and it can also be expressed through assigning \( \beta_i = 1 \) and \( M_{i,j' \cdot j,k'} = k' \) as follows:

\[ M_{i,j' \cdot j,k'} = RT_{j'k'} \times C_{j'} + AT_{jk} \times C_j \]  \hspace{1cm} (11)

where values for \( RT \) and \( AT \) can be found in Table II, and \( RT_{j'k'} = 0 \) if a VM is newly added.

![Fig. 7. Illustration of performance scale-up (LB constraint: 0%).](image)

![Fig. 8. Illustration of performance scale-up (LB constraint: 10%).](image)

Figure 7 illustrates that, without load balancing constraints, the performance can be doubled to 230 per hour by replicating the number of VMs (i.e., adding 32 VMs) without any VM migration using twice the budget ($10 per hour).

In cases where no migration is performed, it is not possible to achieve a solution until 8 (or 9, if LB constraint is 30%)
new VMs are added. Another interesting finding is that, in some cases, IROs with migration are higher than IROs without migration, whereas the opposite is true in other cases. The rationale behind this is the fact that, according to the statistics in Table II, it is possible that in some cases, the time for shutting down a VM and booting a new one is shorter than the time for only booting a new VM of some other type. For example, increasing the TIC (to be higher than 7) of an infrastructure with 1 VM of small instance type in EC2-US can be implemented by shutting down the small instance and booting an xlarge instance, which takes 85 seconds (21 seconds for shut-down, and 64 seconds for booting), or only starting a large instance using 90 seconds.

We can also observe from Figure 9 and Figure 10 that load balancing (LB) constraints impose a significant impact on infrastructure cost and IRO when migration is prohibited and few VMs (less then 11) are allowed to assign. Compared with Figure 7 and Figure 8, when the LB constraint is as 20%, to fulfill the minimum performance requirement, and meanwhile comply with the LB constraint, the broker has to place some VMs with large size in the least cost-efficient provider (i.e., GOGRID), which is harmful for the infrastructure cost and IRO. However, as the number of VMs that are added increases, the distances between solutions with migration and solutions without migration are narrowed down again, since the broker is able to place VMs of small size (instead of larger size) in GOGRID in order to comply with the LB constraint and performance constraint.

This experiment demonstrates the ability of the cloud brokering mechanism to handle the tradeoff between vertical (resizing VMs) and horizontal elasticity (adding VMs), as well as to improve decision making in complex scale-up scenarios with multiple options for service reconfiguration, e.g., to decide how many new VMs to deploy, and how many and which VMs to migrate.

Through the evaluations above, it is demonstrated that our model can support a wide range of dynamic scenarios, and by proper parametrizations, many interesting behaviours can be achieved. Finally, we point out that values in matrix $M$ in real world applications are normally much higher than they are in Section IV-A. This is because VM migration across cloud providers located in different regions is a tedious task due to the fact that establishing a high-speed network tunnels to transfer VM images (that usually consist of Gigabytes of data) is time-consuming and costly.

V. CONCLUSIONS AND FUTURE WORK

With the emergence of cloud computing as a paradigm, users can buy computing capacity from public cloud providers to minimize investment cost rather than purchasing physical servers. However, users are faced with the complexity of integrating various cloud services as the cloud computing market grows and the number of cloud providers increases. Despite the existence of a large number of efforts targeting cloud brokerage mechanisms, dynamic cloud scheduling issue remains largely unexplored. We present a linear integer programming model for dynamic cloud scheduling via migration of VMs across multiple clouds, which offers the flexibility of expressing different levels of migration overhead when restructuring an existing infrastructure. By proper parametrization, this model can be applied to handle changes both in infrastructure (new providers, prices, etc.) and services (elasticity in terms of sizes and/or number of VMs in a service). The proposed model is evaluated against commercial clouds offering settings, and it is demonstrated that our model is applicable in dynamic cloud scheduling cases aiming at cost-efficiency and performance-efficiency solutions.

Future directions for our work include investigation of mechanisms for model parametrization for dynamic cloud scheduling use cases, i.e., finding suitable values for parameters in the proposed model for different scenarios. Additionally, SLA violation compensation for users has not been taken into account in our model. Another interesting topic would be to apply our model to real world services.
ACKNOWLEDGEMENT

We thank Ruben Montero, Rafael Moreno-Vozmediano, and Ignacio Llorente for their valuable contributions to this work, as well as for providing the foundation [13] of this paper. We are also grateful to the anonymous reviewers for their constructive comments. Financial support has in part been provided by the European Community’s Seventh Framework Programme ([FP7/2010-2013]) under grant agreement no. 257115 (OPTIMIS [26]) and the Swedish Government’s strategic effort eSSENCE.

REFERENCES


