Dynamic SLA Based Elastic Cloud Service Management: A SaaS Perspective

Bipin B. Nandi, Ansuman Banerjee, Sasthi C. Ghosh
Indian Statistical Institute
Kolkata, India
{mtc1112,ansuman,sasthi}@isical.ac.in

Nilanjan Banerjee
IBM Research - India
New Delhi, India
nilanjba@in.ibm.com

***Abstract*—*Elasticity* is one of the most attractive features of cloud computing. While elastic resource requirements of a cloud service user often exhibit periodic patterns, the absence of a mechanism to specify precisely the elastic resource requirements forces the cloud service provider to either provision unbounded resources or offer services with rigid and static resources, none of which is an efficient or economic proposition to the tenant. We propose, in this paper, a novel model of *dynamic service level agreement (SLA)* for the tenants to specify their elastic resource requirements so that the cloud service provider can take an informed decision for resource management, thereby optimizing on resource utilization, user expenses and generated revenue. We study the problem from a Software-as-a-Service (SaaS) perspective. We propose an intuitive model of dynamic SLA that can capture the anticipated license requirement variation of an SaaS user. We also propose ILP and greedy-heuristic based optimization approaches for the service provider to solve the tenant onboarding problem with dynamic SLA constraints. Experimental results show that our solution produces optimal or near-optimal (for heuristic based approach) performance in terms of overall resource utilization and the economy for both the service provider and the tenants.**

I. Introduction

In recent times, cloud computing has emerged as a popular paradigm for large scale computing needs and a key factor behind its emergence is elastic management of resource requirement driven by advancement in virtualization and networking technologies. In cloud computing, a service user (often called a tenant) accesses a resource or a service over the network instead of hosting the service on his own premise, with the implicit assumption that the provisioning of adequate resources is the responsibility of the service provider (often called a vendor) alone. Depending on the types of resources accessed, this is called Infrastructure as a Service (IaaS) or Platform as a Service (PaaS) or Software as a Service (SaaS). Generally, a more subsuming term, XaaS (Everything as a Service), is used in cloud computing parlance to indicate that any resource could be offered to a customer by a cloud service provider, who on the other hand, leverages the economy of scale by sharing the resources in the cloud to as many customers as possible.

Typically, in a services ecosystem, when users wish to use a certain service with some performance guarantees, they are required to negotiate a Service Level Agreement (SLA) with a service provider specifying specific conditions of the service. In the user-driven SLA negotiation paradigm, the end users usually tend to negotiate the SLA with the service provider in terms of service level metrics representing the quality of their experience or quality of service (QoS), for example the latency or time for completion of a job submitted to a SaaS provider. To strictly honor such SLAs in the face of dynamic and unpredictable workload, a service provider needs to ensure elastic provisioning of resources to adequately mitigate the anticipated surge in resource requirements. Such unbounded resource provisioning is obviously inefficient for a service provider. Hence, cloud service providers typically tend to negotiate SLAs with the users in terms of absolute maximum resource requirements. This is a service provider driven SLA negotiation paradigm. For example, an IaaS user could lease a virtual machine with large, medium or small compute capacities or a SaaS provider could lease a software with a fixed number of licenses. Users are not allowed to change the SLAs dynamically and there is no guarantee that their variable demands will be handled elastically beyond the resources committed during the initial SLA negotiations. In case of the demand in excess of the committed resources, the users either have to lease additional resources and over-provision or fail to satisfy the desired QoS.

Static SLA negotiations often turn out to be inefficient in terms of actual resource usage for both the user and the service provider, especially in the case of anticipated variable workload. Thus, there is clearly a need of addressing this gap by capturing the variability in user resource requirement and enabling the service provider to leverage that for provisioning adequate resources accordingly. It is not that the user workload is completely unpredictable - usually the users do have an idea of the variability of the load [12] and the time-varying need of resources for a particular job or the *bounded elasticity* support required from the service provider. We bridge the gap by proposing a novel model for user driven, dynamic SLA specification that could be leveraged by the service provider to take an informed decision in provisioning the necessary resources more efficiently and economically, while honoring the specific user SLA requirements.

While our model and the solution are applicable to any cloud based service (XaaS), we present the rest of the paper from a SaaS perspective for the sake of illustration only.

The SaaS Perspective

In Software-as-a-Service (SaaS), a software is deployed as a hosted service and accessed over the Internet. The move from on-premise license-based software usage, the traditional way software has been delivered to customers for decades, to the centrally hosted, subscription-based SaaS model is a huge change for software vendors and customers offering unique business benefits. The ownership of the software now truly shifts to the provider, as does the responsibility for providing technology infrastructure and management. The customer organization is spared the burden of purchasing and maintaining server hardware for the applications. From the vendor perspective, the additional costs of hardware and management are more than offset by leveraging the economies of scale that arise from being able to serve a large number of customers.

The SaaS delivery model is focused on exploiting economies of scale by offering the same instance of an application (or parts thereof) to as many customers as possible. SaaS vendors support different customer needs from the same common base (application, infrastructure) through appropriate sharing and customizations (termed multi-tenancy in SaaS parlance). Multi-tenant aware applications thus allow providers to provision the hardware and software stack for the application once and run multiple customers (or tenants) on the same infrastructure. Salesforce, a popular SaaS provider, for example, sells various services like CRM, analysis and forecasting, to individuals and enterprises for a licensing fee charged per user that varies from some \$65 to \$250 per month, depending on service capacities. This is the service provider driven SLA negotiation paradigm, where the user has no option of defining the SLA dynamically for temporary surge in the requirement of software licenses.

Typically, in an enterprise, not every user needs all the licenses of a given software at all times. Usually there is a time of the day when the demand of a particular software peaks, e.g., the demand of analysis and forecasting software could peak in the last working hour of a day when the daily reports are being generated. In the current practice, due to the lack of a SLA specification capturing the variable workload, an enterprise is compelled to buy the licenses corresponding to the peak demand, which is obviously an expensive proposition to the enterprise. A better proposition is to have the enterprise study its dynamic software requirements over a day and negotiate with Salesforce based on a dynamic SLA specification that captures the daily variation in workload in the enterprise. To this end, we have proposed our definition of dynamic SLA, which could be used by the enterprise to communicate its dynamic requirement - a critical need of a specific number of licenses in the peak hour and a throughput oriented requirement for the rest of the day - to Salesforce, who could then provision and charge resources accordingly towards mutual benefits of economy and efficiency of resource utilization.

The main contributions of this paper are as follows:

- A formal definition of dynamic SLA specification that a user can use to specify their variable service requirements over a period in time.
- A formal model for resource management by the cloud service provider according to the dynamic SLA specification.
- Optimal and Heuristic based solutions for efficient and economic resource management by the cloud service provider.

The rest of the paper is organized as follows. Section II presents the formal model for dynamic SLA specification, while Section III deals with the resource management problem formulation for a SaaS provider from a tenant selection perspective. Section IV discusses the various optimization approaches proposed for the tenant onboarding problem. Section V presents experimental results of the proposed solution approaches for the tenant onboarding problem. Section VI discusses related work and Section VII concludes the paper with a note on our future work.

II. The Formal Model

In this section, we present a formal model of a SaaS system for the tenants to specify their dynamic SLA requirements. We assume a static set of $\mathcal{P}$ tenants $\mathbb{C} = \{\mathbb{C}_1, \mathbb{C}_2, \mathbb{C}_3, \ldots, \mathbb{C}_{\mathcal{P}}\}$ and a SaaS vendor $\mathbb{S}$ in our SaaS system. Each tenant has several distinct *modes* of operation, each having different values for the tenant work parameters. We explain the formalisms in the following discussion. The SaaS vendor has a planning span of τ slots, each of same duration. For example, the SaaS vendor may plan the tenant selection (the set of tenants to be chosen for serving / onboarding) strategy on a daily basis (τ = 24 in this case). In this work, we consider one such window of τ slots and analyze the onboarding problem within it. The SaaS vendor has a total number of $\mathbb{L}$ licenses on sale for its service.

Formally, each tenant requirement or the dynamic SLA for the tenant is represented as the following tuple $\mathbb{C}_i = <\mathbb{M}_i, \mathbb{D}_i, \mathbb{Q}_i, \mathbb{F}_i, \mathbb{G}_i, \mathbb{T}_i>$ where:

- $\mathbb{M}_i = \{m_{i1}, m_{i2}, \ldots m_{i|\mathbb{M}_i|}\}$ is the set of modes the tenant operates in. $|\mathbb{M}_i|$ denotes the total number of modes in $\mathbb{M}_i$.
- $\mathbb{D}_i$ is the deadline (desired completion slot), where 1 $\leq \mathbb{D}_i \leq \tau$. We take this as a hard deadline.
- $\mathbb{Q}_i$: $\mathbb{M}_i \rightarrow \mathbb{N}$ is the license requirement function. This expresses the license needs of the tenant when operating in a particular mode at any slot.
- $\mathbb{F}_i$: $\mathbb{M}_i \rightarrow \mathbb{R}$ is the percentage execution function. This expresses the percentage of job that the tenant can execute in one slot when operating in a particular mode.
- $\mathbb{G}_i$: $\mathbb{M}_i \rightarrow \mathbb{R}$ is the price function. Intuitively, this expresses the price the tenant is ready to pay when allotted one slot in a particular mode.
- $\mathbb{T}_i$ is the *criticality constraint*. This is expressed as a set of pairs of the form (d, m) where $d \in \{1, 2, \ldots \mathbb{D}_i\}$ and $m \in \mathbb{M}_i$. Intuitively, this expresses the slot allocation constraints that the tenant might have, in addition to the guarantee of completion within the specified deadline.

In the above, $\mathbb{N}$ and $\mathbb{R}$ denote the set of natural and real numbers respectively. The model of the SLA described above is inspired by a similar model of power mode specification used in [11]. In this work, for the sake of simplicity, we take τ as the maximum deadline value across all tenants, i.e. τ = $max_i\{\mathbb{D}_i : 1 \leq i \leq \mathcal{P}\}$ or a fixed constant (e.g. 24). Also, it may be noted that even within its deadline, it is not imperative to choose each tenant at every slot (i.e. in some slot, the tenant may be discarded and not assigned any operating mode at all). The following example illustrates the concept in greater detail.

Example 1: A tenant $\mathbb{C}_1$ can operate in three possible modes, namely, m_{11}, m_{12} and m_{13}. When allotted a slot in mode m_{11}, $\mathbb{C}_1$ needs 5 licenses, pays 100 dollars and progresses toward its completion by 20%, whereas those values become 10, 180 dollars and 30% respectively when in mode m_{12} for a slot. When working in a slot in mode m_{13}, it needs 2 licenses, pays 75 dollars and makes 10% progress. In addition, $\mathbb{C}_1$ has a hard deadline of 10 slots for its completion. In slot 3, $\mathbb{C}_1$ needs to work in mode m_{12}, and in slot 7, working in mode m_{13} is required, failing which the terms are not acceptable to him. This requirement is specified as the criticality constraint $\mathbb{T}_i = \{(3, m_{12}), (7, m_{13})\}$. □

III. Resource Management: The Tenant Onboarding Problem

Given a collection of tenants, each expressing its requirement as above, it is the task of the SaaS vendor to analyze whether it is possible to onboard all tenants, while honoring their deadlines along with other criticality constraints (if any), with $\mathbb{L}$ licenses. If the answer is in the affirmative, we have a solution strategy to onboard all tenants, while honoring their SLAs. The strategy essentially specifies a mode allocation scheme to the tenants for each of the slots within their respective deadlines. Evidently, there may be multiple onboarding strategies (mode allocation schemes) and the vendor may select one of them, either arbitrarily, or driven by specific objectives (maximum revenue etc.) corresponding to different business situations.

It may as well be possible that for a given set of tenants, it is infeasible to onboard all of them, given the restriction on the total number of licenses. This may happen quite often, and in several situations. A simple scenario where this may arise is when multiple tenants have criticality restrictions of being scheduled in specific modes at the same slot, and the sum of their license needs exceeds $\mathbb{L}$. In such cases, the vendor has to resort to a *choice* of a subset of tenants, either arbitrarily or with a specific optimization objective (may be driven by economy or an effort to maximize its customer base).

A. Constraint Formulation

We analyze here multiple facets of the onboarding problem from a constraint satisfaction perspective. Let us first define the following binary variables for $1 \le i \le \mathbb{P}$, $1 \le$ j $\le |\mathbb{M}_i|$, $1 \le k \le \mathbb{D}_i$.

$$x_{ijk} = \begin{cases} 1 & \text{if tenant } \mathbb{C}_i \text{ is operating in mode } \mathbb{M}_{ij} \text{ at} \\ & \text{the } k^{th} \text{ slot} \\ 0 & \text{otherwise.} \end{cases}$$

$$y_i = \begin{cases} 1 & \text{if tenant } \mathbb{C}_i \text{ is served} \\ 0 & \text{otherwise.} \end{cases}$$

Mode exclusivity constraint

Each tenant can be in at most one mode at a given slot, if the tenant is served. To ensure this, the following constraint must hold for each tenant $\mathbb{C}_i$ and each slot.

$$\sum_{j=1}^{|\mathbb{M}_i|} x_{ijk} \le y_i, \ \forall \ 1 \le i \le \mathbb{P}, 1 \le k \le \mathbb{D}_i \qquad (1)$$

Completion constraint

In order to serve a tenant, the tenant must reach its 100% completion of the whole job within its deadline. To ensure this, the following constraint must hold for all the tenants which are served.

$$\sum_{k=1}^{\mathbb{D}_i} \sum_{j=1}^{|\mathbb{M}_i|} \mathbb{F}_i(j) x_{ijk} \ge 100 y_i, \ \forall \ 1 \le i \le \mathbb{P} \qquad (2)$$

Maximum license constraint

The vendor has a constraint on the maximum number of licenses $\mathbb{L}$ that can be used simultaneously at any slot. In order to satisfy this maximum license constraint, the following must hold for each slot.

$$\sum_{i=1}^{\mathbb{P}} \sum_{j=1}^{|\mathbb{M}_i|} \mathbb{Q}_i(j) x_{ijk} \le \mathbb{L}, \ \forall \ 1 \le k \le \tau \qquad (3)$$

Criticality constraints

In order to satisfy the criticality requirements specified by the tenants, the following must hold for each tenant served.

$$x_{ijk} = y_i \ \forall (k, j) \in \mathbb{T}_i, \ 1 \le i \le \mathbb{P}, \qquad (4)$$

The above constraint ensures that the selected tenant is guaranted to operate in mode $\mathbb{M}_{ij}$ at the k^{th} slot.

The constraints (1) through (4) collectively define the restrictions on the the deadline requirements, peak licenses, and the criticality constraints for the tenants to be served. If the above set of constraints are satisfiable, the SaaS vendor is guaranteed to be able to serve all of them, while honoring their deadlines and criticality requirements. However, this may not always be the scenario depending on the tenant requirements and the license availability. In the following discussion, we elaborate on these two situations separately.

Case I: All tenants can be onboarded

With $y_i = 1; \quad \forall \ 1 \le i \le \mathbb{P}$, if the family of constraints are collectively satisfiable for a given value of $\mathbb{L}$, we have a strategy for onboarding all the tenants while meeting their deadlines. An assignment to the family of Boolean variables $\{x_{ijk} : 1 \le i \le \mathbb{P}, 1 \le j \le |\mathbb{M}_i|, 1 \le k \le \mathbb{D}_i\}$ that satisfies the conjunction of the Boolean constraints (1) through (4) represents a possible mode allocation scheme. There may be more than one satisfying assignments and thus more than one scheme. We can determine an optimum schedule that maximizes total revenue or minimizes the number of licenses or a combination of the two by framing optimization problems with (1) through (4). This may have ramifications in different stages of the business development of the SaaS vendor. In the early stages of the business evolution, the vendor might be interested to know the minimum number of licenses needed to on-board all the tenants. In the later stages, when the business matures, the vendor might be interested more in the economic perspective or a combination of the two. We capture these different outlooks in the following.

Maximizing revenue

If the vendor wants to maximize the revenue gained, then the objective function of the optimization would be

$$\max \quad \sum_{i=1}^{\mathbb{P}} \sum_{j=1}^{|\mathbb{M}_i|} \sum_{k=1}^{\mathbb{D}_i} \mathbb{G}_i(j) x_{ijk} \tag{5}$$

Minimizing the number of licenses

In the above discussion, we have assumed a fixed value of $\mathbb{L}$. However, it would help the vendor to know what is the minimum value of $\mathbb{L}$ that can satisfy all tenant needs. This would enable him reduce license wastage and allocate only the required number of licenses to this tenant set, while keeping the remaining as available to attract more customers. The objective function for this can be expressed as follows:

$$\min \quad \mathbb{L} \tag{6}$$

A combined objective

If the primary objective of the vendor is to minimize the total number of licenses and the secondary objective is to maximize revenue, the combined objective function can be stated below:

$$\min \quad \infty(\mathbb{L}) - \sum_{i=1}^{\mathbb{P}} \sum_{j=1}^{|\mathbb{M}_i|} \sum_{k=1}^{\mathbb{D}_i} \mathbb{G}_i(j) x_{ijk} \tag{7}$$

where ∞ represents a big number. The above style of constraint formulation ensures that on one hand, maximizing revenue contributes to the overall minimization objective, while on the other hand, reducing $\mathbb{L}$ also has the same effect. Reducing $\mathbb{L}$ even by 1 can reduce ∞ in the objective, which implies maximizing revenue is clearly the secondary choice. This formulation essentially helps the constraint solver in navigating the search space. One possible value for ∞ is $\sum_{i=1}^{\mathbb{P}} \mathbb{D}_i \times \sum_{j=1}^{|\mathbb{M}_i|} \mathbb{G}_i(j) + 1$, where this quantity essentially represents one plus the maximum possible revenue that can be achieved by onboarding all tenants.

TABLE I: Client Specification

$ClientNo.$	$\mathbb{D}_i$	$\mathbb{M}_i(j)$	$\mathbb{Q}_i(j)$	$\mathbb{F}_i(j)\%$	$\mathbb{G}_i(j)$
$\mathbb{C}_1$	2	m_{11}	3	20	40
		m_{12}	5	50	90
$\mathbb{C}_2$	3	m_{21}	4	30	35
		m_{22}	8	55	60
$\mathbb{C}_3$	4	m_{31}	4	20	30
		m_{32}	8	30	40
		m_{33}	10	40	70
$\mathbb{C}_4$	2	m_{41}	4	25	30
		m_{42}	7	50	50

Example 2: Consider the scenario with 4 tenants $\mathbb{C}_1$, $\mathbb{C}_2$, $\mathbb{C}_3$ and $\mathbb{C}_4$. Table I gives the requirement for each tenant. The tenant $\mathbb{C}_1$ imposes a criticality constraint that it needs to operate in mode m_{12} in the first slot. Consider a SaaS vendor with 24 licenses planning for a window of 4 slots. Fortunately, the vendor can on-board all tenants with a maximum revenue of 660. It can be shown that with 20 licenses only, the vendor can in fact schedule all the tenants and generate a revenue of 610. □

Case II: All tenants cannot be on-boarded

If the family of constraints (1) through (4) are unsatisfiable for a given value of $\mathbb{L}$, it means $\exists\ i$ such that $y_i = 0$ for some i, $1 \leq i \leq \mathbb{P}$. Essentially, all tenants cannot be on-boarded in this case and the vendor has to resort to a tenant selection activity with different objectives as discussed below. We consider three objectives here, namely, (a) maximum tenant on-boarding (b) revenue maximization and (c) a combination of revenue and tenant onboarding. However, our model is generic to handle others as well.

Maximizing the number of tenants on-boarded

If the vendor wants to serve as many tenants as possible, then the objective function of the optimization is expressed as:

$$\max \quad \sum_{i=1}^{\mathbb{P}} y_i \tag{8}$$

Maximizing revenue

If the vendor wants to maximize the revenue gained, then we have the following objective:

$$\max \quad \sum_{i=1}^{\mathbb{P}} \sum_{j=1}^{|\mathbb{M}_i|} \sum_{k=1}^{\mathbb{D}_i} \mathbb{G}_i(j) x_{ijk} \tag{9}$$

Combined objective

If the primary objective of the vendor is to serve the maximum number of tenants and the secondary objective is to chose the maximum set of tenants such that the revenue is maximized, the combined objective function can be stated below:

$$\max \quad \infty(\sum_{i=1}^{\mathbb{P}} y_i) + \sum_{i=1}^{\mathbb{P}} \sum_{j=1}^{|\mathbb{M}_i|} \sum_{k=1}^{\mathbb{D}_i} \mathbb{G}_i(j) x_{ijk} \tag{10}$$

where ∞ represents a big number. One possible value for ∞ is $\sum_{i=1}^{\mathbb{P}} \mathbb{D}_i \times \sum_{j=1}^{|\mathbb{M}_i|} \mathbb{G}_i(j) + 1$. The significance of the above objective can be easily explained in the light of the earlier discussion.

TABLE II: Client Specification

$ClientNo$	$\mathbb{D}_i$	$\mathbb{M}_i(j)$	$\mathbb{Q}_i(j)$	$\mathbb{F}_i(j)\%$	$\mathbb{G}_i(j)$
$\mathbb{C}_1$	2	m_{11}	2	25	40
		m_{12}	5	45	100
$\mathbb{C}_2$	3	m_{21}	3	25	50
		m_{22}	7	50	100
$\mathbb{C}_3$	2	m_{31}	3	20	30
		m_{32}	5	50	60
		m_{33}	7	60	90
$\mathbb{C}_4$	3	m_{41}	3	25	60
		m_{42}	7	50	120
$\mathbb{C}_5$	4	m_{51}	4	25	40
		m_{52}	10	50	60

Example 3: Consider a scenario with 5 tenants $\mathbb{C}_1$, $\mathbb{C}_2$, $\mathbb{C}_3$, $\mathbb{C}_4$ and $\mathbb{C}_5$. Table II gives the complete specification values for the tenant set. $\mathbb{C}_1$, $\mathbb{C}_2$ and $\mathbb{C}_4$ have also some criticality constraints. In the first slot, $\mathbb{C}_1$ wants to operate in mode m_{12} whereas $\mathbb{C}_2$ and $\mathbb{C}_4$ want to operate in mode m_{21} and m_{41}, respectively. Consider a SaaS vendor with 15 licenses and planning for a window of 4 slots. Unfortunately, in this case, it is not possible to on-board all the tenants. The maximum number of tenants that can be on-boarded with 15 licenses is 4 out of the 5 tenants. With 15 licenses, the vendor can achieve a maximum revenue of 800. It can be noted that if vendor provides licenses to tenants $\mathbb{C}_1$, $\mathbb{C}_2$ and $\mathbb{C}_4$, then it will get this maximum revenue. It shows the fact that maximizing revenue is not necessarily equivalent to maximizing the number of on-boarded tenants. Suppose the vendor's primary objective is to maximize the number of on-boarded tenants and then look for revenue. For the tenants ($\mathbb{C}_1$, $\mathbb{C}_3$, $\mathbb{C}_4$ and $\mathbb{C}_5$) it can maximize its revenue to 770. $\square$

IV. Tenant Onboarding Problem: Optimization approaches

The optimization problems framed in the previous section can be solved using standard ILP solvers (e.g. CPLEX [3]). However, given the inherent complexity of ILP and known capacity issues, we propose a couple of simple-minded heuristics that achieve close to optimal solutions for different objectives.

A. License Minimization: A Greedy Approach

The objective of the approach proposed here is to serve all tenants with minimum number of licenses possible. Let L_k be the total licenses required for slot k. Initially $L_k = 0$ for all k. Once some tenant $\mathbb{C}_i$ is allocated at slot k in mode m_{ij}, L_k will be updated as $L_k = L_k + \mathbb{Q}_i(j)$. Therefore after considering all the tenants, final value of L_k will give the total licenses required for that slot. Finally the maximum of L_k $\forall k$ will give the minimum license requirement of the vendor. Let $R^{rem}_{i,k}$ be the average percentage of completion required for client $\mathbb{C}_i$ after executing its job in previous k slots. Initially $R^{rem}_{i,0} = \frac{100}{\mathbb{D}_i}$ for all i. Algorithm 1 presents the greedy heuristic.

Algorithm 1 Heuristic for minimum number of licenses

```
Procedure GreedyHeuristicMinLicense($C$)
 1: for $i = 1$ to $\mathbb{P}$ do
 2:     Sort the modes ($m_{ij} \in \mathbb{M}_i$) of $\mathbb{C}_i$ according to $\mathbb{Q}_i(j)$
 3:     in ascending order and let $\mathbb{M}^*_i$ be this sorted order.
 4:     for all critical slots $k$ such that $(k, j) \in \mathbb{T}_i$ do
 5:         Allocate mode $j$ at slot $k$ and update $x_{ijk}$, $L_k$
 6:         and $R^{rem}_{i,k}$ accordingly.
 7:     end for
 8:     Set $k = \mathbb{D}_i$.
 9:     while $k \geq 0$ do
10:         if $k$ is not a critical slot then
11:             Pick the first mode $j$ from $\mathbb{M}^*_i$ for which
12:             $\mathbb{F}_i(j) \geq R^{rem}_{i,\mathbb{D}_i-k}$ and update $x_{ijk}$, $L_k$ and
13:             $R^{rem}_{i,k}$ accordingly.
14:             if job completed till slot $k \geq 100$ then
15:                 Client is successfully on-boarded within
16:                 the deadline and update $y_i = 1$.
17:             else
18:                 $k = k - 1$.
19:             end if
20:         end if
21:     end while
22: end for
23: return $\max_k L_k$
EndProcedure
```

Example 4: We now explain the working of our algorithm on the scenario used in example 2. Note that this time, vendor wants to find the minimum number of licenses using which it can on-board all the tenants. The tenants are considered in the order $\mathbb{C}_1$, $\mathbb{C}_2$, $\mathbb{C}_3$ and $\mathbb{C}_4$. Consider the tenant $\mathbb{C}_1$ and note that its deadline is 2 so $R^{rem}_{1,0}$ becomes $\frac{100}{2} = 50$. Vendor will assign modes for slot 2 first then slot 1. It will assign mode m_{12} in slot 2 because it is the only mode which has greater percentage completion value than $R^{rem}_{1,0}$. After that $R^{rem}_{1,1}$ becomes $\frac{100-50}{1} = 50$, so vendor will assign the same mode m_{12} for slot 1 too. Thus tenant $\mathbb{C}_1$ is served and the licenses required for both the slots 1 and 2 become 5. Now, the vendor will consider tenant $\mathbb{C}_2$. Since it has a deadline of 3, $R^{rem}_{2,0}$ is $\frac{100}{3} = 33.33$. Proceeding similarly, the vendor will allocate mode m_{22} for slot 3. After that $R^{rem}_{2,1}$ becomes $\frac{100-55}{2} = 22.5$. For the slot 2, both the modes have greater percentage completion value than $R^{rem}_{2,1}$ but the vendor will assign m_{21}, because it takes the least number of licenses. Similarly, the vendor will allocate mode m_{21} at slot 1 too. Thus tenant $\mathbb{C}_2$ is served. Proceeding similarly, vendor will allocate mode m_{32} for both the slots 4 and 3, mode m_{31} for both the slots 2 and 1 for tenant $\mathbb{C}_3$. After serving 3 tenants, the license required values for slot 1, 2, 3 and 4 become 13, 13, 16 and 8 respectively. Finally, for tenant $\mathbb{C}_4$, the vendor will allocate mode m_{42} for both the slots 2 and 1. After serving all the tenants, the license requirement values become 20, 20, 16 and 8 respectively for slots 1, 2, 3 and 4. So the maximum value 20 among them gives the minimum licenses required. $\square$

B. Maximum tenant selection: A Greedy Approach

The objective of the approach proposed here is to serve as many tenants as possible with a given number of licenses. For a given slot and for a given tenant, we need to choose an appropriate mode so that all constraints are honored. To achieve this, we schedule the tenants according to the ascending order of their deadlines (Earliest Deadline First), with an expectation that more tenant deadlines will be met (honoring criticality constraints, if any), thereby, fulfilling our overall objective.

Let $L^{rem}_{k,i-1}$ be the remaining licenses available at the k^{th} slot after considering all the tenants up to the $(i-1)^{th}$ tenant. Intuitively, $L^{rem}_{k,i-1}$ is basically the quantity $\mathbb{L} - \sum_{i'=1}^{i-1} \mathbb{Q}_{i'}(j) x_{i'jk}$. Let $F^{rem}_{i,k-1}$ be the remaining percentage of completion for tenant $\mathbb{C}_i$'s job after the $(k-1)^{th}$ slot. Similarly, $F^{rem}_{i,k-1}$ is basically the quantity $100 - \sum_{k'=1}^{k-1} \mathbb{F}_i(j) x_{ijk'}$. Initially for all i, $F^{rem}_{i,0} = 100$ and for all k, $L^{rem}_{k,0} = \mathbb{L}$.

We present a greedy algorithm based on the percentage completion value in each mode and deadline value of the tenant. The procedure $GreedyHeuristic$ takes $\mathbb{C}$ and $\mathbb{L}$ as input and returns the number of tenants on-boarded. The algorithm is described in Algorithm 2.

Example 5: We now explain the working of our algorithm with an example. Consider again the scenario of the SaaS

Algorithm 2 Heuristic for maximim tenant onboarding

```
Procedure GreedyHeuristic($\mathbb{C}$, $\mathbb{L}$)
 1: Sort the tenants according to the deadline $\mathbb{D}_i$ in ascending order (resolving ties with index values). Let $\mathbb{C}_1^*, \mathbb{C}_2^*, \ldots, \mathbb{C}_{\mathbb{P}}^*$ be this sorted order.
 2: for $i = 1$ to $\mathbb{P}$ do
 3:     Sort the modes ($m_{ij} \in \mathbb{M}_i$) of $\mathbb{C}_i^*$ according to $\mathbb{F}_i(j)$
 4:     in descending order and let $\mathbb{M}_i^*$ be this sorted order.
 5:     for all critical slots $k$ such that $(k, j) \in \mathbb{T}_i$ do
 6:         if $\mathbb{Q}_i(j) \leq L_{k,i-1}^{rem}$ then
 7:             Allocate mode $j$ at slot $k$ and update $x_{ijk}$
 8:             and $F_{i,k}^{rem}$ accordingly.
 9:         else
10:             Discard that tenant and goto Line 32.
11:         end if
12:     end for
13:     Set $k$ = 1.
14:     while $k \leq \mathbb{D}_i$ do
15:         if $k$ is not a critical slot then
16:             If $k < \mathbb{D}_i$, pick the first mode $j$ from $\mathbb{M}_i^*$
17:             for which $\mathbb{F}_i(j) \leq F_{i,k-1}^{rem}$ and $\mathbb{Q}_i(j) \leq L_{k,i-1}^{rem}$.
18:             If such $j$ does not exist or $k = \mathbb{D}_i$, pick mode
19:             $j$ from $\mathbb{M}_i^*$ for which $\mathbb{F}_i(j)$ is minimum satis-
20:             fying $\mathbb{Q}_i(j) \leq L_{k,i-1}^{rem}$ and $\mathbb{F}_i(j) > F_{i,k-1}^{rem}$.
21:             Update $x_{ijk}$ and $F_{i,k}^{rem}$ accordingly.
22:             if $F_{i,k}^{rem} \leq 0$ then
23:                 Tenant is successfully on-boarded within
24:                 the deadline $\mathbb{D}_i$ and goto Line 34.
25:             else
26:                 $k = k + 1$.
27:             end if
28:         end if
29:     end while
30:     if $F_{i,\mathbb{D}_i}^{rem} > 0$ then
31:         Discard the tenant.
32:         Update $y_i = 0$ and $x_{ijk} = 0$ for all $j$ and $k$.
33:     else
34:         Update $y_i = 1$ and $L_{k,i}^{rem}$ $\forall k$.
35:     end if
36: end for
37: return $\sum_{i=1}^{n} y_i$
EndProcedure
```

vendor with $\mathbb{C}_1$, $\mathbb{C}_2$, $\mathbb{C}_3$ and $\mathbb{C}_4$ as used in example 2. We consider that the SaaS vendor has 20 licenses. The tenants in ascending order of their deadlines are $\mathbb{C}_1, \mathbb{C}_4, \mathbb{C}_2, \mathbb{C}_3$. Consider tenant $\mathbb{C}_1$ and note that $\mathbb{C}_1$ can operate in two modes m_{11} and m_{12} in which the completions are 20% and 50% respectively. Therefore $\mathbb{M}_1^*$ becomes $\{m_{12}, m_{11}\}$. As per the criticality constraint of $\mathbb{C}_1$, the vendor will assign him in mode m_{12} in slot 1. The execution percentage in mode m_{12} is 50%, hence the remaining completion becomes 50% too. For slot 2, it will also pick mode m_{12} as its completion is equal to 50% (same as remaining value) and also it takes 5 licenses which is less than 20, the number of available licenses. Thus, the tenant $\mathbb{C}_1$ is served and generates revenue 180 and the license remaining values for both slot 1 and 2 become 15. Proceeding similarly, the vendor will consider tenant $\mathbb{C}_4$ and allocate mode m_{42} for both the slots 1 and 2. Therefore, $\mathbb{C}_4$ will be served and will give revenue of 100. After serving tenants $\mathbb{C}_1$ and $\mathbb{C}_4$, the remaining licenses for both slots 1 and 2 become 8. Now the vendor will consider $\mathbb{C}_2$ and allocate modes m_{22} in slot 1 and m_{21} in slot 2 and remaining completion will become 15%. Since completion of both the modes m_{21} (30%) and m_{22} (55%) are greater than 15%, vendor will choose mode m_{21} in slot 3. Thus, the tenant $\mathbb{C}_2$ is served and generates revenue of 130. After serving $\mathbb{C}_2$, remaining licenses in slots 1, 2 and 3 are 0, 4, 16 respectively. Finally, tenant $\mathbb{C}_3$ will be considered, and vendor will allocate mode m_{31} in slot 2, mode m_{33} in slot 3 and 4 respectively. The revenue generated by onboarding $\mathbb{C}_3$ is 170. So all tenants are eventually served with 20 licenses and a total revenue of 580 is obtained. □

V. Results

In this section, we present simulation based experimental results to evaluate our proposed strategies. We consider a vendor with 30 tenants where each tenant has different operating modes ranging from 2 to 10. The percentage completion values for each mode varies from 20% to 100%. Without loss of generality, we assume the license requirement of a mode varies from 10 to 50 and the corresponding revenue values vary between 20 and 500. For experimentation, we have generated test cases with uniform random values for each of the parameters described above. We have used CPLEX [3] as the ILP solver for our experimentation. The greedy heuristic is implemented as an in-house tool in C++.

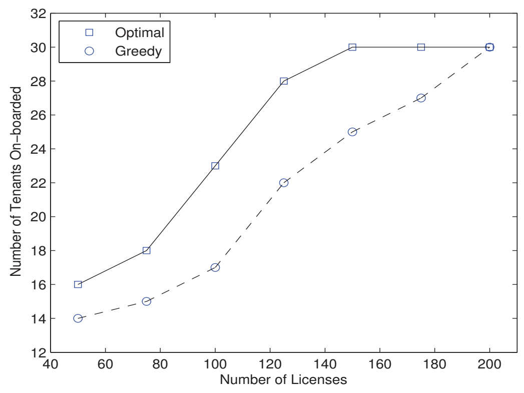

Fig. 1: Greedy vs. ILP : $\mathbb{L}$ vs tenant on-boarded

Figure 1 shows the number of tenants on-boarded corresponding to a given $\mathbb{L}$ value as obtained by our greedy heuristic and ILP. Expectedly, it can be seen from the figure that more tenants can be on-boarded with higher number number of licenses. However, the number of on-boarded tenants saturates when maximum possible tenants are already on-boarded. For a given number of licenses $\mathbb{L}$, the number of tenants on-boarded by the greedy heuristic is comparable to the value obtained by the ILP.

Figure 2 shows the total revenue obtained corresponding to a given value of $\mathbb{L}$, as obtained by our greedy heuristic and ILP. As expected, more revenue can be obtained with more number of licenses. However when maximum possible tenants are already on-boarded the revenue saturates. For a given number of licenses $\mathbb{L}$, the revenue obtained by the greedy heuristic is comparable to the value obtained by the ILP.

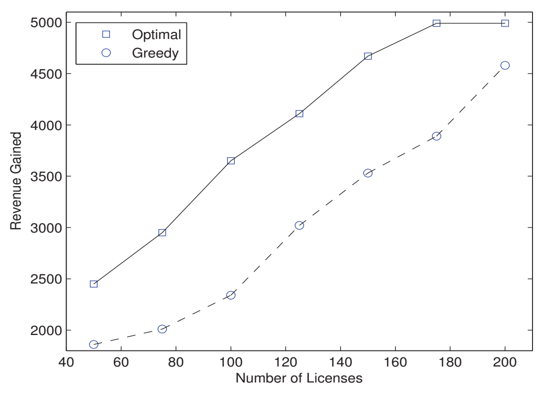

Fig. 2: Greedy vs. ILP: $\mathbb{L}$ vs revenue gained

Figure 3 shows the minimum licenses required to on-board a given number of tenants by our greedy heuristic and ILP. It also shows a worst case value of the minimum number of licenses as obtained assuming that all tenants are scheduled at their maximum license consumption mode at each slot, which would ensure that each of them complete their job earliest. In other words, in this case, $\mathbb{L} = \sum_{i=1}^{\mathcal{P}} \max_{1 \le j \le |\mathbb{M}_i|} \mathbb{Q}_i(j)$. Naturally, to serve more number of tenants, more licenses are required. For a given number of tenants, the minimum number of licenses required to serve all by the greedy heuristic is close to the value obtained by the ILP. The worst case value (without using ILP or the greedy approach) however, gives a much higher estimate, which justifies our greedy solution.

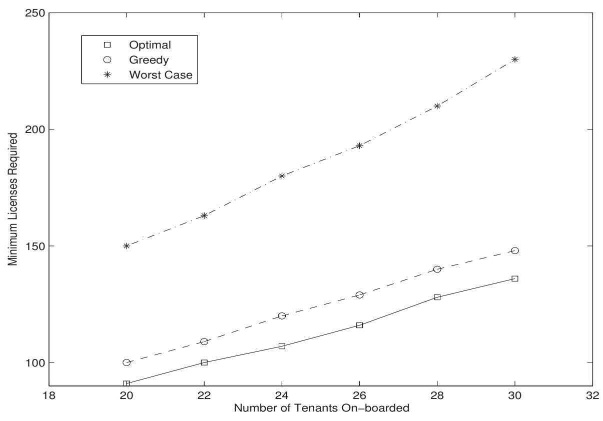

Fig. 3: Tenant onboarded vs. minimum license

VI. Related Work

As discussed earlier, SLA specification usually happens at two levels for cloud based services. Either the user specifies the service level requirements [14] oblivious to its implication to resource provisioning by the service provider or the service provider provides the users with some rigid and static set of service guarantees on the services offered and asks the users to choose from that set [1], [2], [4]. As argued before, in the former case the service provider needs to ensure *infinite* unbounded elasticity in resources provisioned, often by outsourcing jobs to third parties [15], to handle variable workload while maintaining the SLAs negotiated. In the latter case, the users often need to over provision to handle variable workloads. Thus both the approaches result in inefficient resource utilization for the users and the service providers. There has been some related effort in modelling the workload using queuing theory [5], [9], [8] and other statistical models [7]. Stochastic models have been used for multiplexing variable user workload [10], [13], but none of them addresses the issue of user specified variable SLAs that enables the service provider to provision the resources in a more planned and efficient manner. In [6], the authors have looked at tenant onboarding from an optimization perspective by reducing different problems arising in the SaaS context to Knapsack and its variants, but did not consider variable workload.

VII. Conclusion

In this paper we have proposed a model for specifying dynamic SLA to capture elastic resource requirements of tenants in a cloud computing system. We adopt a simple and intuitive model for expressing variations in tenant's resource requirement over time and propose optimization based formulations for solving the tenant onboarding problem for the service provider from a SaaS system perspective. The foundation of the models proposed in this paper is quite generic and can be applied for defining and analyzing other types of cloud computing systems beyond SaaS. We are currently working on heuristic solutions to handle multiple objectives in the generic framework.

References

[1] Amazon simple storage service. http://aws.amazon.com/s3/.

[2] Google app engine. http://code.google.com/intl/fr/appengine/.

[3] Ibm ilog cplex optimizer. http://www-01.ibm.com/software/in/integration/optimization/cplex/.

[4] Microsoft bpos. http://www.microsoft.com/online/business-productivity.mspx.

[5] A. Ali-Eldin, J. Tordsson, and E. Elmroth. An adaptive hybrid elasticity controller for cloud infrastructures. In *IEEE Network Operations and Management Symposium (NOMS), 2012*, pages 204 – 212.

[6] A. Banerjee, S. C. Ghosh, and N. Banerjee. Pack your sack for the cloud. In *ISEC*, pages 157–16, 2012.

[7] N. Bonvin, T. G. Papaioannou, and K. Aberer. Autonomic sla-driven provisioning for cloud applications. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGRID '11, pages 434–443, Washington, DC, USA, 2011. IEEE Computer Society.

[8] V. Cardellini, E. Casalicchio, F. L. Presti, and L. Silvestri. Sla-aware resource management for application service providers in the cloud. In *First International Symposium on Network Cloud Computing and Applications 2011*, pages 20 – 27.

[9] H. Li, G. Casale, and T. Ellahi. Sla-driven planning and optimization of enterprise applications. In *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*, WOSP/SIPEW '10, pages 117–128, New York, NY, USA, 2010. ACM.

[10] B. B. Nandi, A. Banerjee, S. C. Ghosh, and N. Banerjee. Stochastic vm multiplexing for datacenter consolidation. In *IEEE International Conference on Service Computing, 2012*.

[11] S. Ray, P. Dasgupta, and P. P. Chakrabarti. A new pseudo-boolean satisfiability based approach to power mode schedulability analysis. In *VLSI Design*, pages 95–102, 2007.

[12] N. Roy, A. Dubey, and A. Gokhale. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *IEEE International Conference on Cloud Computing (CLOUD), 2011*, pages 500 – 507.

[13] B. Speitkamp and M. Bichler. A mathematical programming approach for server consolidation problems in virtualized data centers. *Services Computing, IEEE Transactions on*, 3(4):266 –278, oct.-dec. 2010.

[14] L. Wu, S. K. Garg, and R. Buyya. Sla-based resource allocation for software as a service provider (saas) in cloud computing environments. *Cluster Computing and the Grid, IEEE International Symposium on*, 0:195–204, 2011.

[15] J. O. F. and In igo Goiri and J. Guitart. Sla-driven elastic cloud hosting provider. In *18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2010*, pages 111 – 118.