Youssef and Krishnamurthy *Journal of Cloud Computing: Advances, Systems and Applications* (2017) 6:17
DOI 10.1186/s13677-017-0087-y

Journal of Cloud Computing: Advances, Systems and Applications

RESEARCH Open Access

Burstiness-aware service level planning for enterprise application clouds

Anas A. Youssef[1*] and Diwakar Krishnamurthy[1,2]

Abstract

Enterprise applications are being increasingly deployed on cloud infrastructures. Often, a cloud service provider (SP) enters into a Service Level Agreement (SLA) with a cloud subscriber, which specifies performance requirements for the subscriber's applications. An SP needs systematic Service Level Planning (SLP) tools that can help estimate the resources needed and hence the cost incurred to satisfy their customers' SLAs. Enterprise applications typically experience bursty workloads and the impact of such bursts needs to be considered during SLP exercises. Unfortunately, most existing approaches do not consider workload burstiness. We propose a Resource Allocation Planning (RAP) technique, which allows an SP to identify a time varying allocation plan of resources to applications that satisfies bursts. Extensive simulation results show that the proposed RAP variants can identify resource allocation plans that satisfy SLAs without exhaustively generating all possible plans. Furthermore, the results show that RAP can permit SPs to more accurately determine the capacity required for meeting specified SLAs compared to other competing techniques especially for bursty workloads.

Keywords: Cloud management, Service level planning, Workload burstiness

Introduction

Enterprises are beginning to increasingly rely on cloud computing systems for hosting their applications. Users of many enterprise applications, e.g., Web servers, require fast responses to their requests. Unfortunately, existing cloud infrastructures e.g., Amazon Web Services Elastic Compute Cloud (EC2), typically do not provide such performance guarantees. Typically, the enterprise subscriber of the cloud service is responsible for provisioning resources such that their applications' performance objectives are met.

In this paper, we consider a performance-aware cloud system that is capable of providing performance guarantees to its subscribers. In such an environment, a cloud Service Provider (SP) enters into a Service Level Agreement (SLA) with a subscriber *prior* to deploying the subscriber's applications. Service Level Objectives (SLOs) are specified for a subscriber's applications as part of their SLA with the SP. In contrast to a traditional cloud system, the SP is responsible for provisioning resources such that the specified subscriber SLAs are met.

In a performance-aware cloud system, the SP needs systematic Service Level Planning (SLP) tools to guide the process of formulating performance SLAs with subscribers and translating them to application resource allocations. For example, consider a subscriber that stipulates a certain mean response time threshold as an SLO for their application's transactions. An SP should consider the application's workload as well as the workload of other applications on the cloud to determine whether there is adequate capacity to satisfy this performance requirement. If this requirement cannot be satisfied, the SP may suggest a less stringent requirement. Alternatively, the SP may determine the additional capacity needed to satisfy the request and use that information to present a revised cost estimate to the subscriber for satisfying the desired performance requirement. Our work focuses on SLP tools that can help SPs undertake such exercises.

SLP tools designed for enterprise application clouds need to address the phenomenon of workload burstiness. Past studies have shown that enterprise workloads suffer from burstiness [1–8]. *Burstiness* refers to serial correlations in workload patterns such as correlation

*Correspondence: anas.youssef@ci.menofia.edu.eg
[1]Computer Science Department, Faculty of Computers and Information, Menoufia University, Menoufia, Egypt
Full list of author information is available at the end of the article

between successive arrivals of transactions and correlations in the resource consumption patterns at various system resources. Typically, applications encounter periods of sustained workload peaks followed by periods of sustained workload troughs. For an application with a bursty workload, resource allocation based on the average behaviour, e.g., average rate of request arrivals, may lead to insufficient resources to handle the peaks thereby causing SLO violations. In contrast, allocation based on the peak may lead to over provisioning and under utilization of resources leading to increased costs, which can adversely impact the competitiveness of the SP. To deliver SLOs in a cost-effective manner, an SP therefore needs to deduce a time varying allocation plan of resources to any given application that matches the burstiness pattern of that application. Deducing such burstiness-aware resource allocation plans while considering the workloads and SLOs of all applications hosted on the cloud is a challenge.

Existing approaches have not explicitly addressed the challenges of SLP in the presence of burstiness. This paper proposes a novel Resource Allocation Planning (RAP) method to identify a time varying allocation plan of resources to applications that meets SLO objectives while satisfying an SP's resource constraints. Specifically, the technique takes as input traces that capture the burstiness of a set of applications over a time period of interest. Given an application's trace and a candidate time varying resource allocation plan, a trace-driven performance modeling technique [9] is used to predict the impact of the application's burstiness on the SLO. RAP searches for a resource allocation plan that globally minimizes SLO violations over the input set of applications while satisfying resource constraints imposed by the SP.

Since the SLP problem outlined above cannot be solved by traditional optimization solvers, we describe and evaluate three different RAP variants to search for optimal or near-optimal solutions. In particular, we describe an intelligent search technique that identifies the optimal solution without resorting to an exhaustive search of all possible resource allocation plans. Furthermore, we propose two variants that are computationally more efficient than this intelligent technique while providing near-optimal solutions. Our simulation results show that all three RAP variants can permit SPs to more accurately determine the capacity required for delivering specified SLOs compared to burstiness agnostic techniques. Specifically, the RAP variants discover plans where resource allocations fluctuate over time in keeping with the observed burstiness. Such plans can in turn lower the costs required by the SP to deliver a given SLO. Our results also establish the superiority of RAP to the prevalent practice of considering SLO targets based on resource utilization thresholds.

This paper represents a significant extension of the work we describe in our previous conference papers [10, 11]. Specifically, this paper proposes two new variants of the RAP technique, which improve the optimality of resource allocation solutions. It presents new experiments that evaluate in detail the optimality and computational complexity of the different RAP variants. Finally, this paper also includes a more exhaustive discussion of related work.

The remainder of the paper is organized as follows. "Resource allocation planning (RAP)" section describes the RAP variants. "Evaluation setup" section outlines the simulation setup we use to evaluate RAP. Results that compare RAP's performance with other baseline approaches and that characterize the behaviour of the various RAP variants are presented in "Results" section. A discussion of related work is presented in "Related work" section and conclusions are presented in "Conclusions and future work" section.

Resource allocation planning (RAP)

We provide a high-level overview of RAP in "Overview" section. "Formulation" section formulates resource allocation as an optimization problem. "RAP variants" section describe three RAP variants to solve this problem.

Overview

Figure 1 shows an overview of the RAP approach. Studies have shown that enterprise workloads have predictable hour-of-the-day, day-of-the-week, and day-of-the-month patterns [12]. RAP exploits this phenomenon to discover burstiness aware resource allocation plans. As shown in Fig. 1, for each application under study, a cloud SP uses a collection of traces that embody the workload characteristics of that application. We refer to these traces as a *workload scenario*. In particular, a *session trace* captures information about user sessions related to the application over a period of time. A *session* is defined as a group of inter-related requests that fulfill a certain task, e.g., a web transaction to purchase a product. The session trace includes information such as arrival instants of sessions, number and type of requests issued within a session, and the user *think* time, i.e., idle time, between successive requests in a session. Furthermore, *resource traces* capture the application's use, i.e., utilization, of low-level resources such as processor cores over the same period.

Application traces can be obtained in several ways. For existing applications, a cloud SP can use historical traces obtained from the application's deployment in the cloud. Traces for new applications can be obtained in several ways. For example, cloud subscribers can provide traces from environments external to the cloud where the application was previously deployed. Alternatively, traces can

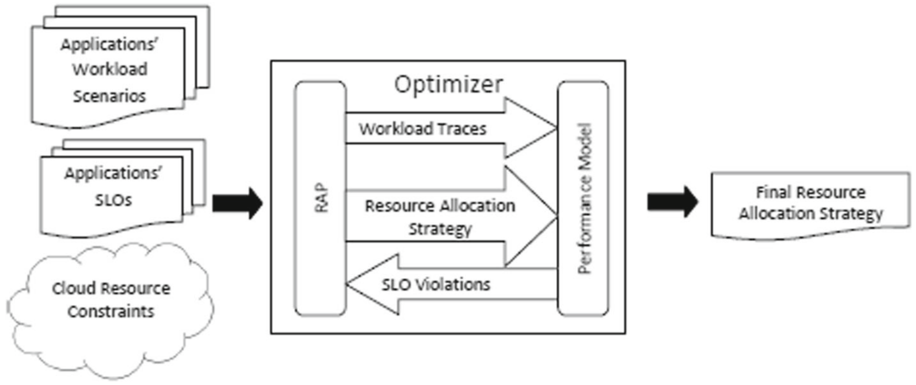
Applications' Workload Scenarios
Applications' SLOs
Cloud Resource Constraints
Optimizer
RAP
Workload Traces
Resource Allocation Strategy
SLO Violations
Performance Model
Final Resource Allocation Strategy

Fig. 1 RAP overview

be synthetically generated based on subscriber estimates of mean or peak workload behaviour.

From Fig. 1, RAP also takes as input information on the numbers and types of resources, e.g., web servers and database servers, available for allocation. This captures any resource constraints an SP might face and provides a mechanism for limiting resource costs incurred by the SP. Finally, RAP takes as input SLOs for the applications. While many types of SLOs can be considered by RAP, we use an SLO based on a target mean application request response time in this paper. We define the *SLO violation percentage* for an application as the percentage of deviation of the mean request response time of the application for its workload scenario from the target mean request response time that defines the application's SLO. The *global SLO violation metric* is defined as the mean SLO violation percentage for the input set of applications. We note that RAP can be easily adapted to accommodate other global SLO violation metrics.

Given the aforementioned inputs, RAP searches for a time varying resource allocation plan that minimizes the global SLO violation metric for the input set of applications. We define a *resource allocation plan* as the number and type of resources allocated to a set of applications over a given number of resource allocation intervals within a defined planning horizon. The *planning horizon* represents the time period over which an SP is obliged to satisfy an application's SLO. RAP allows an SP to emulate elastic or time-varying resource allocation at fine-grained time intervals within the planning horizon. For a given planning horizon, resources can be allocated for applications at the start of these fine-grained intervals and relinquished at the end of these intervals. Each of these intervals is referred to as the *resource allocation interval*. For example, cloud SPs can study the effect of dynamically changing resource allocations to applications at resource allocation intervals of 30 min within a planning horizon of 24 h.

The main power of RAP comes from exploring a resource allocation interval smaller than the planning horizon and exploiting the burstiness across these intervals. This is a key difference from other SLP techniques proposed in literature. Exploring finer time scales allows RAP to discover allocation plans that use the least amount of resources to minimize SLO violations when application workloads are bursty. Figure 2 provides an example that shows the relationship between the planning horizon and the resource allocation intervals over time for two bursty applications with non overlapping peaks, i.e., applications have their peak request arrival rates in two different resource allocation intervals. If the two applications compete for the same resource to satisfy their peaks, they can both be allocated this resource over the planning horizon but over different resource allocation intervals. This is not possible if the planning horizon is of the same granularity as the resource allocation interval. Figure 2 shows how the SP's resource allocation can be modulated according to the burstiness patterns of the applications. RAP uses application workload scenarios to automatically discover such opportunities.

From Fig. 1, similar to other SLP studies RAP relies on a performance model to search for an optimum resource allocation plan. Given a candidate resource allocation plan and the workload scenario for an application, the performance model predicts the application's mean request response time and hence the SLO violation percentage. We use the Weighted Average Method (WAM) [9] performance modeling technique, which has been shown to be more accurate than other traditional performance modeling techniques in predicting SLO violations under burstiness. RAP iteratively generates various resource allocation plans and uses WAM to evaluate the global SLO violation metric of these plans. It finally selects the plan that results in the least violation.

A naive way to arrive at an optimal resource allocation plan is to enumerate all possible resource allocation plans

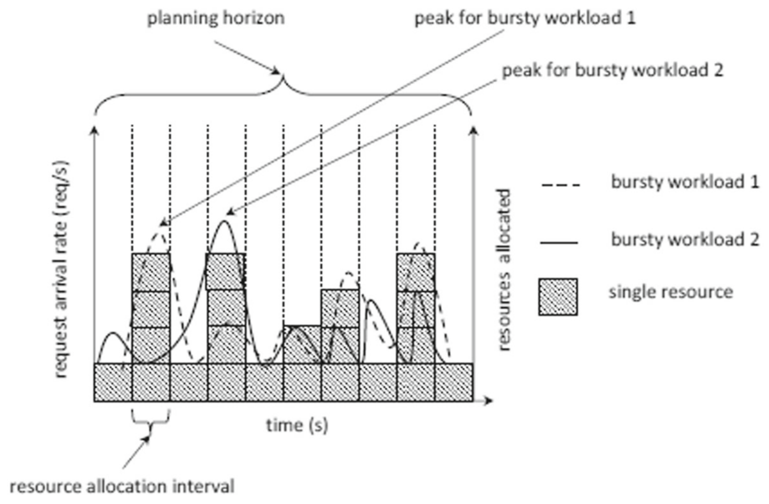

Fig. 2 SLP under burstiness

for a given set of applications under given cloud resource constraints. However, an exhaustive enumeration might not be feasible if the number of applications, the number of resource allocation intervals, and the number of tiers within each application are large. This necessitates the need for the non-exhaustive search techniques described in "RAP variants" section.

Formulation

This section describes in detail the optimization problem underlying RAP. Please refer to Table 1 for a summary of all symbols used in the paper. The resource allocation problem has two cost factors: cost of resources allocated and penalties due to applications' SLO violations. It should be noted that both costs can be equally weighted by the SP or one of them can be assigned a higher priority. This paper focuses on minimizing the SLO violations objective.

We consider a given set of A applications. Typical of enterprise applications, each application a employs a multi-tier architecture which is composed of N_a tiers. For ease of explanation, we assume that all applications considered for resource allocation have the same number of application tiers. However, this assumption can be relaxed by invoking the appropriate performance model for each application depending on the number of its tiers.

For ease of discussion, each application tier is also assumed to be associated with a specific type or *flavour* of resource instances. For example, the web tier of an application might use a large number of resource instances each containing small number of cores and memory while the database tier might use a single instance with larger number of cores and memory. This assumption can also be relaxed by supporting multiple flavours of resource instances at each application tier and storing a performance profile for each resource flavour. This performance profile can be used to select between different flavours when allocating resources to an application tier.

As mentioned in "Overview" section, each application a is characterized by a workload scenario, denoted by W_a. We assume a planning horizon divided into T equal sized resource allocation intervals. Resource constraints place an upper limit on the number of resources available to each tier. The maximum number of resources available to all applications at tier n in a resource allocation interval t is denoted by $C_{n,t}^{max}$. The total number of resource instances allocated to tier n of application a in resource allocation interval t is denoted by $C_{a,n,t}$.

As described previously in "Overview" section, RAP relies on a WAM performance model to predict applications' SLOs given applications' workloads scenarios and resources allocated to the tiers. The SLO violation percentage V_a for an application a is defined as:

$$V_a = \begin{cases} \frac{R_a - RT_a}{RT_a} * 100\% & \text{if } R_a > RT_a \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where R_a is the mean response time of application a over the planning horizon as predicted by the performance model and the SLO of application a, denoted by SLO_a, is specified as a target mean response time RT_a over the planning horizon.

The global SLO and resource allocation optimization problem can be defined by the following set of equations.

$$min \sum_{a=1}^{A} Pen_a \tag{2}$$

$$Pen_a = f(V_a) \tag{3}$$

Table 1 Summary of symbols

Symbol	Definition
A	Number of applications
AM_a	Arrival process model for application a
a_{max}	Application with the top most SLO violation percentage across all A applications
$C_{a,n,t}$	Number of resource instances allocated to tier n of application a at interval t
$C_{n,t}^{max}$	Maximum number of resources available for allocation to all applications at tier n in resource allocation interval t
$D_{a,n}$	Mean service demand of application a at tier n
F_a	Distribution of number of requests per session for application a
I_a	Index of dispersion of session inter-arrival time for application a
N_a	Number of application tiers for application a
$n_{bottleneck}$	Bottleneck tier of an application
Pen_a	SLO violation penalty cost of application a
R_a	Mean request response time of application a predicted by the performance model over the planning horizon
$R_{a,t}$	Mean request response time of application a predicted by the performance model over interval t
RT_a	Target mean request response time of application a over the planning horizon
S_a	Number of user sessions in the workload scenario of application a
SCV_a	SCV of session inter-arrival time for application a
SLO_a	SLO of application a
SM_a	Service process model of application a
T	Number of resource allocation intervals of the planning horizon
t_{max}	Resource allocation interval with the maximum mean request response time over the planning horizon
V_a	SLO violation percentage of application a
W	Set of workload scenarios of all applications
W_a	Workload scenario of application a
Z_a	Distribution of think time between session requests in application a
λ_a	Mean session arrival rate for application a

$$s.t.\ C_{a,n,t} \geq 1\ \forall a \in \{1, \ldots A\}, n \in \{1, \ldots N_a\}, t \in \{1, \ldots T\} \tag{4}$$

$$s.t.\ \sum_{a=1}^{A} C_{a,n,t} \leq C_{n,t}^{max}\ \forall n \in \{1, \ldots N_a\}, t \in \{1, \ldots T\} \tag{5}$$

$$R_a = f(W_a,\ C_{a,n,t}\ \forall n \in \{1, \ldots N_a\}, t \in \{1, \ldots T\}) \tag{6}$$

Equation (2) specifies the objective to minimize SLO violation penalties over all applications. Equation (3) defines an SLO violation penalty cost Pen_a for a given application a as function of the application violation percentage V_a defined in Eq. (1). The penalty cost due to an SLO violation of a given application may increase with higher violation percentage. For example, an application with a 50% violation percentage over the planning horizon may incur higher violation penalty than an application that has a violation percentage of 10%. Equations (4) and (5) place upper and lower constraints on the size of the SP's resource pool, respectively. Equation (4) shows that each application, a, should be allocated at least one resource of an appropriate flavour in each tier, n, in every interval, t. Equation (5) places a maximum limit on the number of available resource instances for all A applications at each tier n in each resource allocation interval t. Equation (6) relates application mean response time R_a as a function of its workload scenario W_a and the resources allocated to all tiers $n \in \{1,\ldots N_a\}$ of application a for all resource allocation intervals $t \in \{1,\ldots T\}$.

The formulation of the optimization problem provides flexibility with respect to the overall objective of the SLP process. From Eqs. (2) and (3) the Sum of Violation percentages (SV) over all applications can be minimized by simply setting Pen_a to V_a for each application a. Therefore the SLO objective can be modified to:

$$min\ SV = \sum_{a=1}^{A} V_a \tag{7}$$

If the cloud SP is interested in minimizing the number of applications violating their SLOs, we can assume that the violation penalties for all applications are equal. Specifically, each Pen_a value in Eq. (2) can be defined as a binary variable which takes 0 for no violation and 1 for any positive violation percentage. In this manner the objective function defined in Eq. (2) leads to minimizing the total number of applications with positive Pen_a values. Minimizing the SV objective defined in Eq. (7) is the focus of this work.

As discussed in "Overview" section, the output of the optimization is a resource allocation plan represented by the number of resources allocated to each tier n per application a, in each resource allocation interval t. If any SLO violations occur, a list of these violations for all A applications should be returned by the optimization solver.

The objective function represented by Eq. (2) relies on the mean response time estimates provided by Eq. (6). In general, the function shown in Eq. (6) is non-linear. However, it does not have a closed-form representation. Performance predictions are obtained using WAM, which uses an iterative algorithm to arrive at the mean response time estimate R_a. Consequently, we cannot use traditional non-linear optimization techniques. Instead, we propose three heuristic algorithms in the next section to solve this optimization problem.

RAP variants

As mentioned previously, an exhaustive search over all possible resource allocation plans may not be feasible. We propose three variants of the RAP approach that avoid exhaustive searching. The three variants differ with respect to the trade off between computational complexity and optimality.

In general, these approaches start with the basic resource allocation plan as captured previously by Eq. (4) and then traverse a sequence of decision stages. Each *decision stage* generates an optimal or near optimal resource allocation plan given that one additional resource is available for allocation relative to the previous decision stage. This involves using WAM to carry out the following tasks:

1. Identify a candidate application for allocating the one additional resource.
2. For the identified application, identify the interval over which this resource has to be allocated.
3. For the identified interval, select the tier to which the resource has to be allocated.

These tasks are carried out such that the lowest SV value as shown in Eq. (7) is achieved. These tasks are repeated over several stages till resources are exhausted or all applications meet their SLOs.

We now describe in more detail the different approaches to obtain an optimal or near-optimal resource allocation plan at a given decision stage. The naive way is to evaluate all possible resource allocation plans at each decision stage. As shown in Table 2, this involves evaluating all tiers in all resource allocation intervals for all applications which results in invoking the performance model $A^*T^*N_a$ times. To reduce the computation time at each decision stage, we introduce three variants of RAP namely: *RAP-Intelligent-Exhaustive (RAP-IE)*, *RAP-AllApps*, and *RAP-OneApp*. Table 2 summarizes the operation and computational complexity of the different RAP variants relative to exhaustively enumerating all possible resource allocation plans at each decision stage.

Table 2 illustrates the difference between RAP-IE and the naive enumeration. RAP-IE considers only the *bottleneck* tier of each application in all T resource allocation intervals rather than considering all tiers as per the naive approach. Among all application tiers, the bottleneck tier has the most effect on the mean response time of an application and consequently the application's V_a. Specifically, at each decision stage for each application in each interval, one resource of the appropriate flavour is allocated to the bottleneck tier. We apply the utilization law [13] to identify the bottleneck tier. WAM is then invoked to compute Eq. (6) and hence Eq. (7). This process is then repeated for all applications. The resource allocation plan which gives the least SV value is selected and used to obtain the resource allocation plan in the next stage. As shown in Table 2, the number of performance model invocations required by RAP-IE is reduced to A^*T.

RAP-IE is essentially an intelligent exhaustive search technique and as with any exhaustive search technique it is guaranteed to produce an optimal solution. The intelligence stems from not considering non-bottleneck resources while addressing the problem of where to allocate an additional resource. We only explore resource allocations at the bottlenecks since such a strategy is proven to minimize response time and hence service level violation [14, 15]. Essentially, resources are incrementally allocated in stages, each time allocating a resource that mitigates the most serious bottleneck from among the set of all possible bottlenecks. Though better than naive enumeration, RAP-IE is still computationally expensive since the performance model has to be invoked for all applications in all intervals. To alleviate this problem, we propose two more computationally efficient variants namely: RAP-AllApps and RAP-OneApp. These two variants trade-off optimality for computation time.

Unlike RAP-IE which invokes the performance model for all applications in all intervals at each stage, RAP-AllApps invokes the performance model for all applications only in the intervals with the *highest mean response time*. This interval represents the most bursty and congested interval for a given application and so the allocation of one more resource instance to the bottleneck tier of such an interval is likely to result in the most reduction in the application's V_a when compared to other intervals. As shown in Table 2, at each stage RAP-AllApps invokes the performance model A times. In contrast to RAP-IE, the

Table 2 Computational complexity of RAP variants

At each decision stage	Exhaustive enumeration	RAP-IE	RAP-AllApps	RAP-OneApp
Applications evaluated	All	All	All	Application with the highest V_a
Resource allocation intervals evaluated	All	All	Interval with the highest mean response time	Interval with the highest mean response time
Tiers evaluated	All	bottleneck tier	Bottleneck tier	Bottleneck tier
Number of performance model invocations	$A^*T^*N_a$	A^*T	A	1

algorithm has no dependence on the number of resource allocation intervals, T.

RAP-AllApps will deviate from optimality when the bottleneck tier is the not the same in all intervals for a given application. To obtain an optimal solution in this case, one has to evaluate all intervals since different resource flavours can have different impact on the value of V_a. We will show later in Figs. 6 and 7 that solutions diverge from optimal when systems tiers have very different configurations and very different service demands.

RAP-OneApp makes the simplifying assumption that targeting the application with the highest V_a likely yields the least SV value. Accordingly, at each stage it ranks all applications in a descending order in terms of their V_a and selects the top most application. The allocation of one additional resource is explored only for the bottleneck tier of this application rather than for all applications as in RAP-IE and RAP-AllApps. Thus, RAP-OneApp invokes the performance model only once at each decision stage as shown in Table 2. RAP-OneApp is not guaranteed to produce an optimal solution since targeting the application with the highest V_a might not yield the highest reduction in SV. We experimentally evaluate the optimality and computational complexity of the three RAP variants in “Results” section.

We now describe RAP-OneApp in greater detail. In the remaining sections RAP-OneApp and RAP are used interchangeably. If the other two variants of RAP, i.e., RAP-IE and RAP-AllApps, are involved then all variants are explicitly referenced by their names. Figure 3 shows the pseudo-code of the RAP-OneApp algorithm. As described previously in “Overview” section, the inputs to the algorithm are the set W of workload scenarios with one workload scenario per application a, the SLO_a for each application a over the planning horizon and $C^{max}_{n,t}$ resources available for allocation to all applications at each tier n in each resource allocation interval t.

From Fig. 3, initially each application a is allocated one resource instance of the appropriate flavour at each tier n and for each resource allocation interval t. The WAM performance model is then invoked by RAP through the function *WAM-QNM()* as shown in Fig. 3. This function takes as inputs the workload scenario W_a and the number of resource instances assigned to application a at each tier n per resource allocation interval t. The outputs returned by this function are the mean response time R_a of application a over the planning horizon and the mean response time $R_{a,t}$ of application a over each resource allocation interval t. After this step, V_a is calculated for each application a.

The RAP algorithm then enters a loop. In each iteration of this loop, the application a_{max} with the top most SLO violation percentage, i.e., maximum V_a, is selected first. For the selected application a_{max} the resource allocation interval t_{max} with the maximum $R_{a,t}$ over all T resource allocation intervals is selected. This represents the interval where the application is most heavily loaded. It should be noted that t_{max} can only be an interval for which the system has free resources remaining to be allocated. The bottleneck application tier $n_{bottleneck}$ is then determined for a=a_{max} and t=t_{max}. Finally one additional resource instance of the appropriate flavour is allocated to application a_{max} in tier $n_{bottleneck}$ at resource allocation interval t_{max}. Resource availabilities are updated to reflect this resource allocation. The algorithm terminates when either all the available resources are allocated in all resource allocation intervals or the maximum V_a is 0 which means that all applications have achieved their SLOs.

Evaluation setup

We first provide a description of the parameters that we use to generate the synthetic workload scenarios for this study in “Workload scenario parameters” section. “WAM performance model” section describes the WAM performance model we use in our study. “Experiment factors and levels” section summarizes the SLP and workload factors in the study. A description of the baseline techniques that we use to illustrate the gains from RAP are presented in “Baseline methods” section.

Workload scenario parameters

As mentioned previously, RAP requires for each application a a workload scenario W_a. W_a is associated with an arrival process model AM_a, which governs how sessions arrive at the application. In this study, we inject controlled levels of burstiness in session arrivals. To achieve this, we characterize AM_a by the set of parameters S_a, F_a, Z_a, λ_a, SCV_a, and I_a.

The parameter S_a is the number of user sessions in the workload scenario. F_a and Z_a specify the distributions of the number of requests per session and the think time between session requests, respectively. λ_a denotes the mean session arrival rate.

The variability of the session inter-arrival time, i.e., time between successive session arrivals, distribution is captured by SCV_a. The *Squared Coefficient of Variation* (SCV) of the session inter-arrival times is defined as:

$$SCV_a = \left(\frac{\sigma_a}{\mu_a}\right)^2 \quad (8)$$

where σ_a and μ_a are the standard deviation and mean of session inter-arrival times, respectively.

Finally, the degree of burstiness in the arrival of sessions is summarized by the parameter I_a, which is the index of dispersion of session inter-arrival times. I_a considers the autocorrelation between session inter-arrival times. *Autocorrelation* is defined as the degree of similarity between a given time series and a lagged version of that time series

```
Input: $W$, $SLO_a$ $\forall a \in \{1,2,\ldots,A\}$, $C^{max}_{n,t}$ $\forall$ $n \in \{1,2,\ldots,N_a\}$, $t \in \{1,2,\ldots,T\}$
Output: $C_{a,n,t}$ $\forall a \in \{1,2,\ldots,A\}$, $n \in \{1,2,\ldots,N_a\}$, $t \in \{1,2,\ldots,T\}$, $V_a$ $\forall a \in \{1,2,\ldots,A\}$
For $a$ = 1 to $A$
    # for each application $a$ allocate one resource instance to each tier $n$ in each interval $t$
    $C_{a,n,t} = 1$ $\forall$ $a \in \{1,2,\ldots,A\}$, $n \in \{1,2,\ldots,N_a\}$, $t \in \{1,2,\ldots,T\}$
    # update resource availability based on this information
    $C^{avail}_{n,t} = C^{max}_{n,t}$ - $C_{a,n,t}$ $\forall$ $n \in \{1,2,\ldots,N_a\}$, $t \in \{1,2,\ldots,T\}$
    # invoke the WAM performance model for each application workload scenario $W_a \in W$, $\forall a \in \{1,2,\ldots,A\}$
    $[R_{a,t}, R_a]$ = WAM-QNM ($W_a$, $C_{a,n,t}$ $\forall n \in \{1,2,\ldots,N_a\}$ $t \in \{1,2,\ldots,T\}$)
    $V_a$ = CalculateViolationPercentage($R_a$, $SLO_a$)

While (ResourcesAvailable($C^{avail}_{n,t}$ $\forall$ $n \in \{1,2,\ldots,N_a\}$, $t \in \{1,2,\ldots,T\}$)
    [$a_{max}$, MaxViolationPercentage] = GetAppwithMaxViolationPerentage()
    # if all applications satisfy their violation percentages
        If MaxViolationPercentage == 0
            Return $C_{a,n,t}$ $\forall a \in \{1,2,\ldots,A\}$, $n \in \{1,2,\ldots,N_a\}$, $t \in \{1,2,\ldots,T\}$,
                $V_a = 0$ $\forall a \in \{1,2,\ldots,A\}$
        End If
    $t_{max}$= GetFreeIntervalwithMaxRmean($a_{max}$, ($R_{amax,t}$ $\forall t \in \{1,2,\ldots,T\}$),
                                         ($C^{avail}_{n,t}$ $\forall$ $n \in \{1,2,\ldots,N_a\}$, $t \in \{1,2,\ldots,T\}$ )
                                   )
    $n_{bottleneck}$=GetBottleneckTier($a_{max}$, $t_{max}$)
    # allocate one additional resource instance to tier $n_{bottleneck}$ of application $a_{max}$ in interval $t_{max}$
    $C_{amax,nbottleneck,tmax} = C_{amax,nbottleneck,tmax} + 1$
    $C^{avail}_{nbottleneck,tmax} = C^{avail}_{nbottleneck,tmax}$ - 1
    # re-invoke The WAM performance model for the application $a_{max}$ only
    $[R_{amax,tmax}, R_{amax}]$ = WAM-QNM ($W_{amax}$, $C_{amax,n,t}$ $\forall n \in \{1,2,\ldots,N_{amax}\}$, $t \in \{1,2,\ldots,T\}$)
    #update the violation percentage for this application
    $V_{amax}$ = CalculateViolationPercentage($R_{amax}$, $SLO_{amax}$)
End While
Return $C_{a,n,t}$ $\forall a \in \{1,2,\ldots,A\}$, $\forall n \in \{1,2,\ldots,N_a\}$, $t \in \{1,2,\ldots,T\}$, $V_a$ $\forall a \in \{1,2,\ldots,A\}$
```

Fig. 3 RAP Algorithm

over successive time intervals. The *lag-k autocorrelation*, denoted by ρ_k, is the correlation between a given time series and itself shifted by k time steps. Given SCV_a and a set of ρ_k where $k \geq 1$, I_a is defined as:

$$I_a = SCV_a \left(1 + 2\sum_{k=1}^{\infty} \rho_k\right) \tag{9}$$

As shown in Eq. (9), I_a depends on SCV_a and the degree of autocorrelation between session inter-arrival times given by the infinite summation of ρ_k values. Session inter-arrival times that are exponentially distributed, i.e., have a Poisson arrival process, have an I_a value of 1 [5, 16]. This implies that the I_a value represents the deviation of any observed set of session inter-arrival times from the burstiness free Poisson process. A high value of SCV_a characterizes high variability in session inter-arrival times but not necessarily a bursty pattern of session inter-arrival times. The degree of autocorrelation between session inter-arrival times determines the degree of burstiness encountered in the workload. Workloads with bursty session arrivals will have very large values for I_a in the order of hundreds or thousands to indicate a clear deviation from the Poisson process. Caniff et al. presented evidence of burstiness in real-life workloads [17]. In particular, the authors showed that the classic FIFA World Cup trace of web request arrivals has an index of dispersion value of 8400.

Measuring the value of I_a based on Eq. (9) is not practically feasible due to the infinite summation of the ρ_k values. Consequently, we use the approximate method proposed by Casale et al. [5] to calculate I_a.

Finally, W_a is associated with a service process model SM_a. SM_a captures the service demands placed by a request on application system resources, e.g., CPUs and disks. As mentioned previously, each application a has a number of application tiers N_a. The application's service process model SM_a is characterized by the set $D_{a,n}$ representing the mean service demands at various tiers, i.e., $n \in \{1, 2, \ldots N_a\}$.

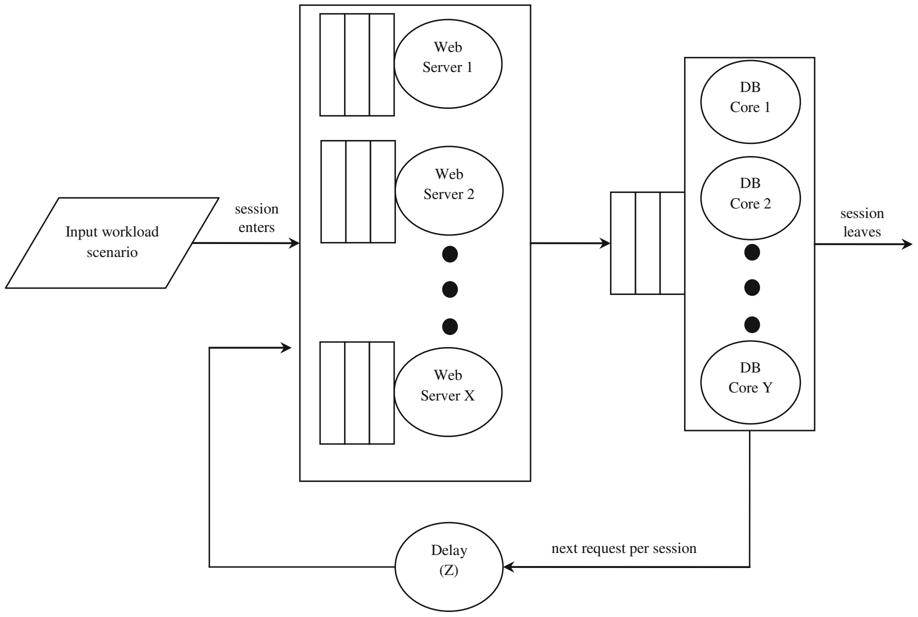

Fig. 4 WAM model of a two tier application

WAM performance model

Figure 4 shows the WAM performance model used to represent the performance behaviour of an application in our study. From the figure, WAM takes as input a workload trace of sessions conforming to the arrival process model AM_a in W_a. It also takes as input the Queueing Network Model (QNM) shown in Fig. 4. The service demands of the resources in the QNM are defined by the service process model SM_a. WAM peruses the session trace to estimate a distribution of the number of concurrent sessions at the application. This is defined as the *population distribution*. For each population, i.e., the number of concurrent sessions, in this distribution, WAM solves the QNM analytically to obtain a mean response time estimate for the population. Finally, the per-population mean response times are weighted by their corresponding probabilities and session throughputs to obtain a mean response time estimate for the application when it is subjected to the session trace. By incorporating the population distribution instead of merely using the mean population to solve the QNM, WAM is able to better capture the impact of burstiness in the session trace [9].

We now describe the QNM used by WAM in more detail. As shown in Fig. 4, the QNM represents a two tier application consisting of a web tier and a database tier. Similar to best practices used in enterprise applications, the web and database tiers employ different flavors of resource instances. The web tier employs horizontal scalability. X resource instances each containing one processing core can be allocated at the web tier. The database tier exploits vertical scalability. One resource instance with Y cores can be allocated to the database tier. The values of X and Y are varied by RAP during the SLP process to achieve application SLO objectives.

Experiment factors and levels

The experiments use a variety of synthetic workload scenarios with characteristics as shown in Table 3 to assess the behaviour of RAP under varying levels of session arrival variability and burstiness. Unless otherwise stated, the service demands at the web and database tiers for all experiments are fixed as per Table 3. The session length and think time distributions of AM_a are chosen to match empirical distributions observed at a real web-based application system [18].

Table 3 Experiment factors and levels

Factor	Levels
Planning horizon	$\sim$ 4 and 8 h
Resource allocation interval	1 h
S_a	{75,000, 85,000, 100,000}
F_a	Empirical distribution with mean 9.4 request/session [18]
Z_a	Empirical distribution with mean 40 s [18]
λ_a	3.33, 2.86, 2.5 session/s
SCV_a	1, 3, 4, 5
I_a	{1, 100, 500, 1000, 10,000}
$D_{a,1}$	Exponential distribution with mean 20 ms
$D_{a,2}$	Exponential distribution with mean 10 ms

Experiments presented in the paper employ a planning horizon of 4 and 8 h and a resource allocation granularity of 1 h. All applications in the experiments have an application SLO defined as a target mean response time that is twice the service demand at the bottleneck resource. Practitioners often use such an SLO as a measure to indicate an upper bound for the queuing that can be tolerated at the bottleneck resource.

Baseline methods

To better illustrate the characteristics of RAP, we consider several baseline SLP methods. We first consider two techniques that do not consider burstiness. The first approach is referred to as the *whole* approach. Unlike RAP, the allocation of resource instances in this approach is carried out for the whole planning horizon at each decision stage without considering finer-grained resource allocation intervals. Specifically, this approach ranks applications in terms of their SLO violations. Starting from the application with the highest V_a value it allocates an additional resource instance to an application tier over the entire planning horizon until the application's SLO is satisfied or until all resources have been allocated.

The second approach is referred to as the *basic interval* approach. Similar to RAP, this approach attempts resource allocation at a finer time scale than the planning horizon starting with the application having the highest V_a value. However, it differs from RAP in the way it selects the candidate resource allocation interval for the additional resource instance allocation at each decision stage. In RAP the candidate resource allocation interval is the resource allocation interval with the highest mean response time to account for burstiness. On the contrary, the basic interval approach applies a simple technique to select resource allocation intervals chronologically. Specifically, after selecting the application with the highest V_a value, the additional resource instance is allocated to the resource allocation interval which comes in sequence starting from the first resource allocation interval in the planning horizon. If no resource instances are available for allocation in this interval, then the next interval chosen. After each decision stage the applications are re-ranked in terms of their V_a values and the process is repeated until all application SLOs are satisfied or all resources in the cloud are exhausted for all resource allocation intervals.

Apart from these burstiness agnostic approaches, we also consider an approach used by practitioners that is based on setting a resource utilization threshold, e.g., a CPU utilization target, instead of SLOs based on response times. An advantage of this technique is its simplicity. The technique does not require a performance model to estimate application response times. An example of a utilization-based technique is the *Quartermaster* capacity manager service proposed by Rolia et al. [19]. Quartermaster allows resource utilization thresholds to be set as an indirect mechanism for achieving desired application response times.

We use a modified version of RAP that allows resource utilization targets to be specified for applications in lieu of response time based SLO targets. Since resource utilization in any interval can be computed with just the knowledge of the request arrival rate and service demand over that interval, there is no need for employing the WAM model.

Results

This section is organized as follows. "Comparison of RAP variants" section compares the three RAP variants. "Importance of burstiness aware SLP" section compares RAP with the burstiness agnostic approaches. A comparison with the approach that uses resource utilization thresholds is presented in "Comparison with utilization based SLP" section.

Comparison of RAP variants

We first perform two controlled experiments to show the degree of optimality achieved by the three RAP variants. In the first experiment four different applications are subjected to a non-bursty workload, i.e., exponential session arrivals, over a planning horizon of 4 h with a resource allocation interval of 1 h. In each resource allocation interval, the number of database instance cores is kept constant at 1 for each application. Initially, all applications are allocated one web server instance in each resource allocation interval. The maximum number of available web server instances per resource allocation interval is set to 7. These settings allow the exhaustive enumeration of all possible resource allocation plans for these applications. The service demands are set as per Table 3.

The second experiment explores more bursty workloads. The four applications are characterized by more session arrival variability, i.e., $SCV_a = 3$, and progressively higher degrees of session arrival burstiness, i.e., $I_a = 1, 100, 1000$ and 10,000. The maximum number of available web server instances per resource allocation interval is limited to six in order to enumerate all possible resource allocation plans. All other settings are the same as in the previous experiment.

Table 4 shows some statistics about the two experiments described in the above paragraphs. As shown in the table, exhaustive enumeration requires the generation of approximately 1.5 million and 50,000 solutions for the two experiments, respectively. Clearly, exhaustive enumeration is prohibitive even for small scale SLP exercises.

Figure 5 shows the optimality of the solutions obtained by the three RAP variants. It compares the mean SLO

Table 4 Statistics of the results obtained in Fig. 5

	Exhaustive enumeration	RAP-IE	RAP-AllApps	RAP-OneApp
Performance model invocations per decision stage	N/A	16	4	1
Number of solutions explored in Fig. 5a	1.5 million	192	48	12
Number of solutions explored in Fig. 5b	50,000	128	32	8

violation percentages of the solutions obtained by each of the three RAP variants against the mean SLO violation percentages of the solutions obtained by exhaustive enumeration. Specifically, the figure organizes the exhaustive enumeration of all possible solutions obtained based on the decision stages explored by the three RAP variants. The x-axis represents the decision stages through which the RAP variants proceed. At decision stage 0, an initial resource allocation plan is generated by allocating one web resource instance to each application over each resource allocation interval. For each subsequent decision stage *i* the black dots represent the mean SLO violation percentages obtained for the various possibilities of allocating *i* additional web server instances to the initial resource allocation plan generated at decision stage 0.

Figure 5a shows the results obtained for the exponential arrivals experiment while Fig. 5b shows results for the bursty arrivals experiment. It can be observed in

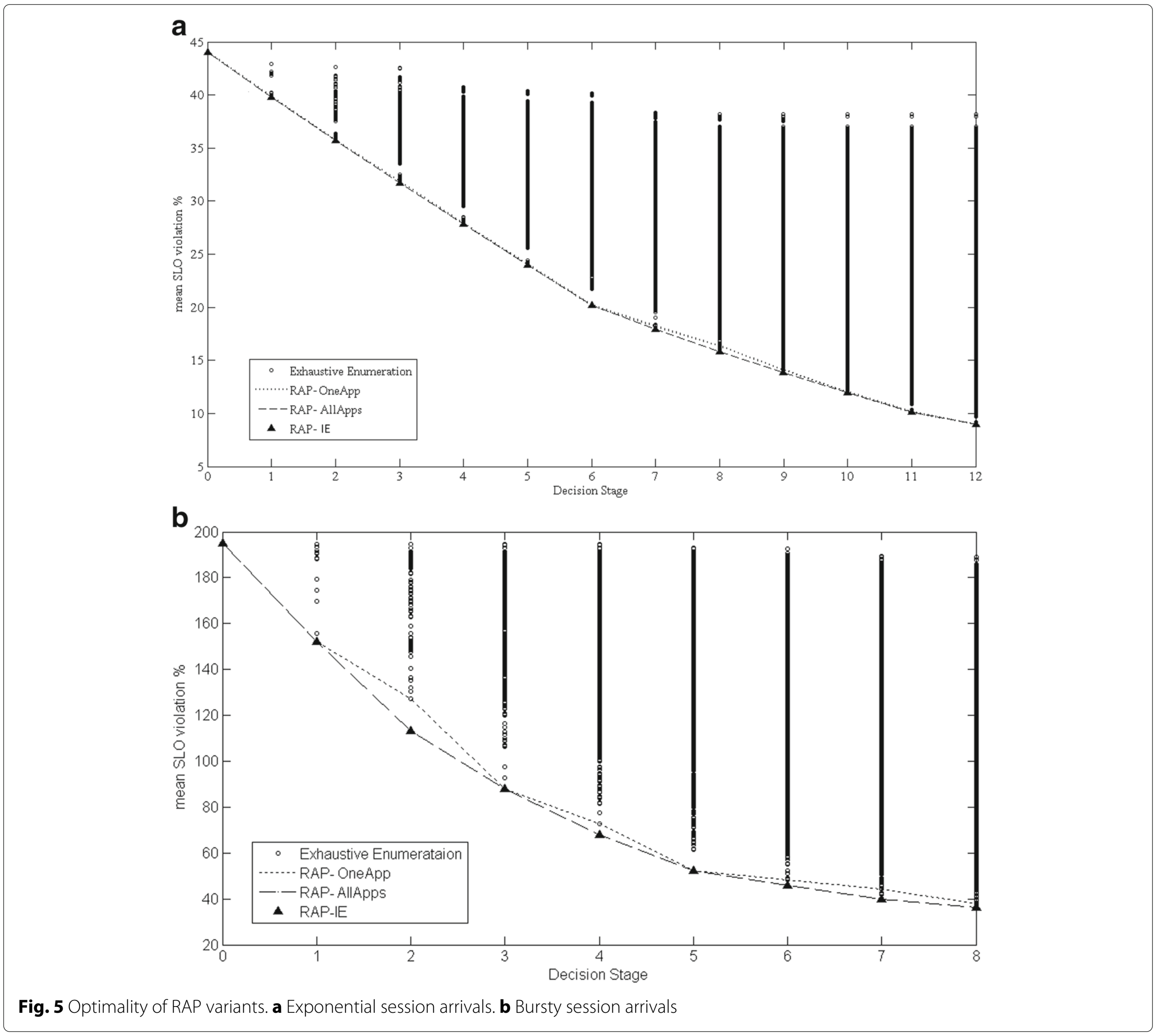

Fig. 5 Optimality of RAP variants. **a** Exponential session arrivals. **b** Bursty session arrivals

Fig. 5a that both RAP-IE and RAP-AllApps are able to generate the optimal plans at each decision stage while RAP-OneApp can generate the optimal plan in all stages except decision stages 7, 8, and 9. In most decision stages the application which causes the most reduction in the overall SV value has also the highest V_a value. The difference in behaviour between RAP-IE and RAP-AllApps on one side and RAP-OneApp on the other side can be observed more clearly in Fig. 5b with bursty workloads. However, RAP-OneApp's solution is still very close to the solutions obtained by the other two variants.

From Fig. 5, there is no difference between the solutions obtained by RAP-IE and RAP-AllApps in all decision stages. This is because the workloads considered in the two experiments have the same bottleneck tier, i.e., web tier, in all resource allocation intervals at all decisions stages. To illustrate the difference between the solutions obtained by RAP-IE and RAP-AllApps, another experiment is conducted with the same settings used in the experiment shown in Fig. 5a. However, the maximum number of web server instances and database instance cores per resource allocation interval are set to much higher values than the values used in the first experiment. This is done to allocate enough resources to all applications so that they can satisfy their SLO requirements. To force the workloads to change their bottleneck tiers in some resource allocation intervals at some decision stages, the mean database service demand is increased from 10 to 18 ms to be close to the mean web service demand of 20 ms.

Figure 6 shows slight differences in the solutions obtained by RAP-AllApps from the optimal solutions obtained by RAP-IE. These differences occur at decision stages 7 to 13 and 18 to 25. This is because in each of these decision stages the bottleneck tier is not the same in all resource allocation intervals for some of the workloads considered in the experiment. This affects the optimality of the solutions obtained by RAP-AllApps, which evaluates at each decision stage the performance of all applications in only the resource allocation intervals with the highest mean response time. Figure 6 shows that the three RAP variants are eventually able to converge to the optimal solution which results in zero mean SLO violation percentage. This happens because, as described previously, the resource limits per each resource allocation interval are set to very high values which are sufficient for each RAP variant to eventually obtain the optimal resource allocation plan.

Finally, we explore the sensitivity of the RAP variants to the degree of homogeneity in resource scaling among application tiers. In the experiments described so far the web tier is modeled by a multi-server resource while the database tier is modeled by a multi-core resource. Therefore, the effect of adding one more resource instance to the web tier on the overall application response time is not the same as adding one more resource instance to the database tier. This is referred to as *heterogeneous resource scaling* among application tiers. In the experiments we present now, the service demands of the web and database tier are 20 and 18 ms, respectively. However, in contrast to the previous experiments, the database tier is modeled by a multi-server service center instead of a multi-core service center. In this way, the web and database tier are allocated same flavours of resource instances which yields *homogeneous* scaling of resources in both tiers.

Figure 7 compares homogeneous and heterogeneous scaling for the scenario where the 4 applications are subjected to bursty workloads, respectively. From the

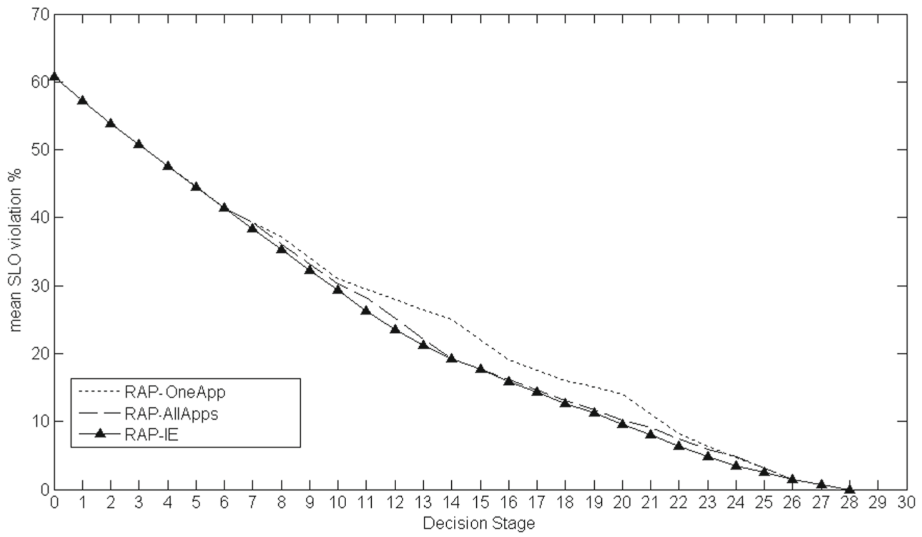

Fig. 6 Effect of similarity in service demands

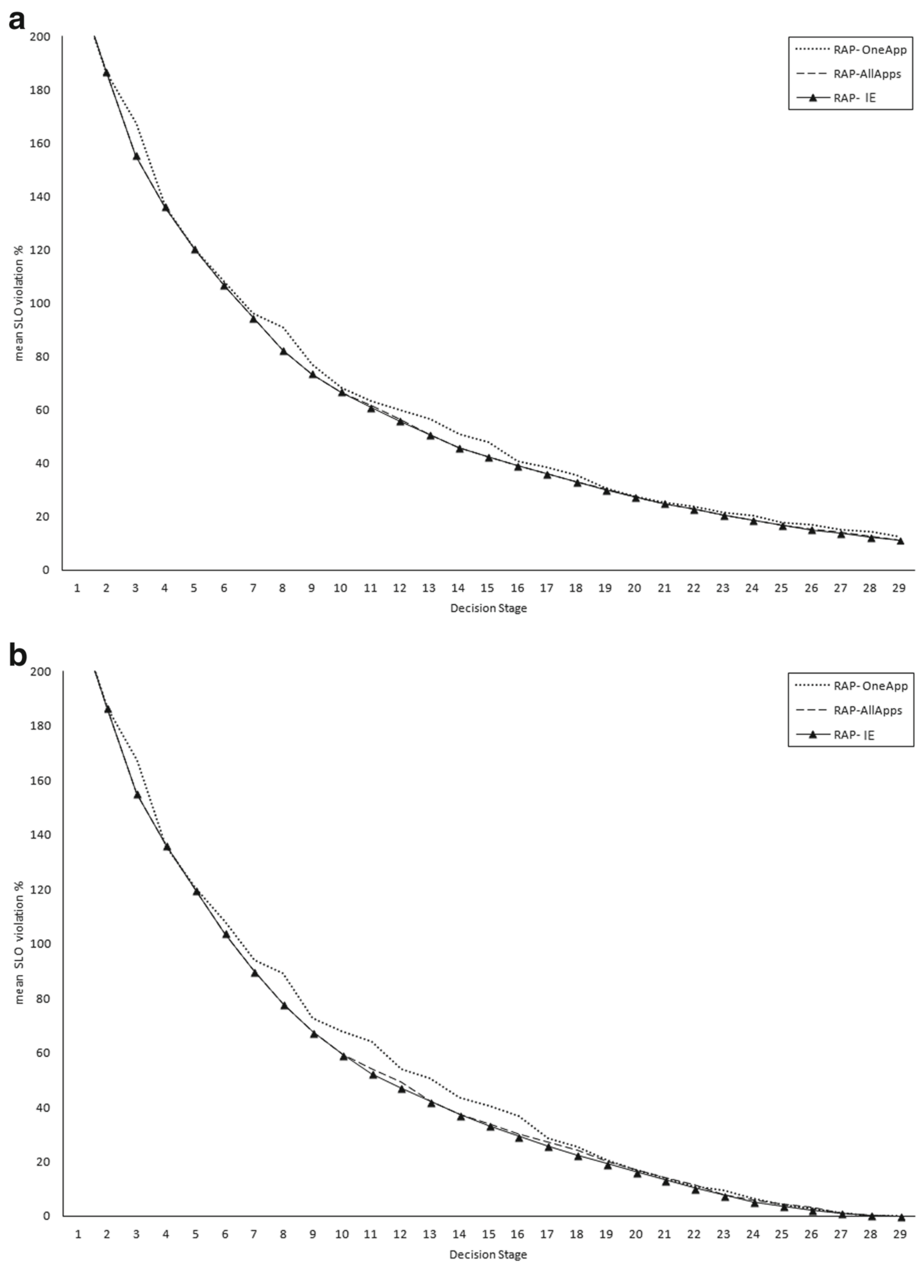

Fig. 7 Effect of Homogeneous and Heterogeneous scaling. **a** Bursty Arrivals - Homogeneous Scaling. **b** Bursty Arrivals - Heterogeneous Scaling

figures, the solutions of the RAP variants diverge from one another slightly more when there is heterogeneous scaling. This is because heterogeneous scaling triggers more changes in the bottleneck tier over the planning horizon than homogeneous scaling. Consequently, RAP-AllApps and RAP-OneApp are likely to report only near optimal solutions. We however note that even in this scenario RAP-AllApps and RAP-OneApp solutions are quite close to the optimal solution.

In summary, the more the degree of similarity in resource service demands between application tiers and the more the degree of heterogeneity in resource scaling among application tiers, the more the deviation in the solutions obtained by RAP-AllApps and RAP-OneApp from those obtained by RAP-IE. However, both heuristic RAP variants, i.e., RAP-AllApps and RAP-OneApp, are able to obtain close to optimal solutions in most of the experiments conducted for both exponential and bursty workloads. Furthermore, these approaches are significantly less computationally expensive as shown in Table 4. From Table 4, for the experiment shown in Fig. 5a, RAP-OneApp reduces the number of solutions explored per decision stage by 94 and 75% relative to RAP-IE and RAP-AllApps, respectively. Consequently, RAP-OneApp is better suited for analyses involving a large number of applications.

We characterize the computation times of the different RAP variants on a desktop machine with a 2.4 GHz Intel Xeon processor and 4 GB RAM. Each WAM model invocation takes about 20 s. For the experiment shown in Table 4, RAP-IE, RAP-AllApps, and RAP-OneApp invoke WAM 16 times, 4 times, and once per decision stage, respectively. Therefore, the computation times per decision stage for RAP-IE, RAP-AllApps, and RAP-OneApp are 5.3 min, 1.3 min, and 20 s, respectively. We note that the WAM invocations in each decision stage are completely independent of one another. Consequently, computation times of RAP-IE and RAP-AllApps can be reduced by exploiting hardware parallelism. While we have implemented multi-threaded versions of these RAP variants, we omit providing detailed computation time statistics for these versions for the sake of brevity. These can be found in the doctoral thesis of Youssef [20].

Importance of burstiness aware SLP

The experiments in this section illustrate the advantages of exploiting burstiness in SLP exercises. To achieve this, we compare RAP to the burstiness agnostic whole and basic interval approaches described previously in "Baseline methods" section. The RAP version referred to in the remaining sections of the paper is RAP-OneApp. As mentioned previously, the whole approach considers a resource allocation interval that is the same as the planning horizon. In contrast, the basic interval approach considers a resource allocation interval with finer time scale granularity than the planning horizon. However, in contrast to RAP, it does not consider the relative burstiness of these intervals.

Figure 8 shows the number of web server instances and database instance cores allocated to a combination consisting of five identical application workload scenarios, i.e., in terms of variability and burstiness, over an 8 h planning horizon with 1-h resource allocation intervals. Specifically, the five workload scenarios are characterized by medium variability and extremely bursty session arrivals, i.e., $SCV_a = 3$ and $I_a = 10,000$. All other settings are as per Table 3. RAP is setup to allocate as many resources needed to satisfy the SLOs of all the five applications.

Figure 8 illustrates the operation of each allocation approach. In the whole approach all intervals are allocated the same number of web server instances and database instance cores. In the basic interval approach the interval resource allocations follow almost a chronological order. Specifically, for intervals 1 to 6, allocations in any given interval either remain the same or drop when compared to the allocations in the previous interval. Interval 7 violates this observation since the bottleneck tier switches from the web tier to the database tier in one of the workload scenarios. RAP selects the interval for resource allocation to be the one with the highest mean response time. Consequently, its resource allocations show a pattern that reflects the combined burstiness characteristics of the application workload scenarios. From the figure, RAP estimates a requirement of at most 14 web server instances and 9 database instance cores in some intervals to satisfy application SLOs. The other approaches estimate higher numbers since they do not exploit burstiness.

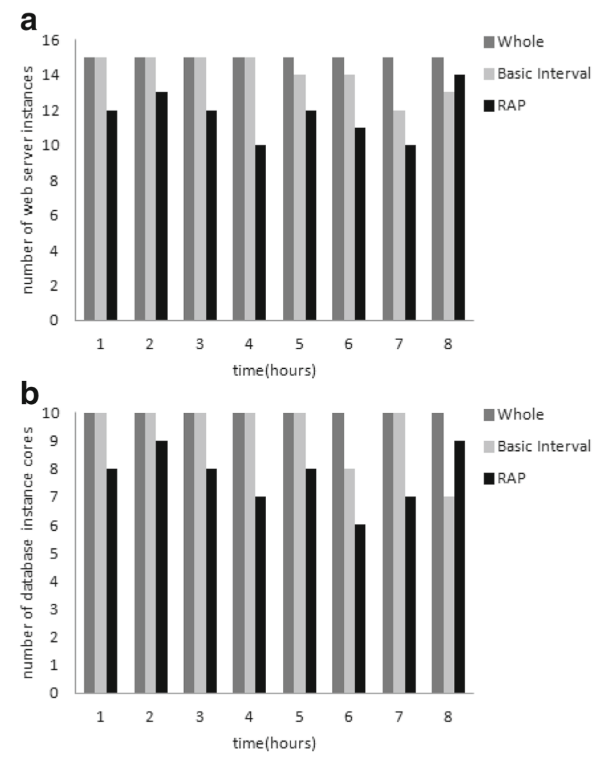

Fig. 8 RAP vs. Burstiness agnostic approaches. **a** Web server instances. **b** Database instance cores

The sensitivity of the three policies to burstiness is now explored. Figure 9 shows the total number of web server instances and database instance cores allocated over eight hours using the three resource allocation approaches mentioned above to satisfy SLOs of four different combinations of workload scenarios. The four combinations are characterized by progressively higher degrees of session arrival burstiness. Each combination consists of five workload scenarios which have statistically identical session arrival process characteristics. Table 5 lists the values of the parameters which characterize the session arrival process of each combination of workload scenarios shown in Fig. 9. Figure 9 confirms that RAP estimates the least number of resources in all cases and the gains

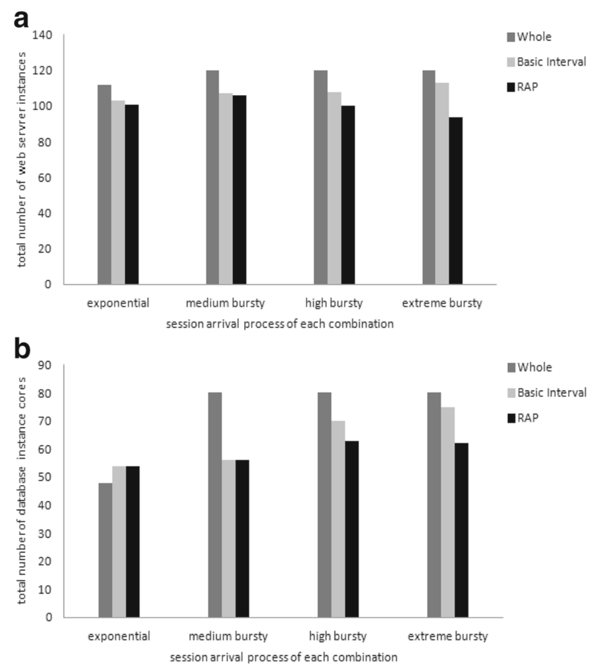

Fig. 9 Impact of degree of burstiness. **a** Web server instances. **b** Database instance cores

are significant especially when there is a high degree of burstiness.

We now explore a resource constrained scenario to show the effect of each of the three approaches on the mean violation percentage. For this experiment, we consider the four combinations of workload scenarios shown in Table 5. Each combination consists of ten applications subjected to the statistically similar workloads. The number of web server instances and the number of database instance cores available per interval is limited to 16 each so that application SLOs are violated. Similar to the previous experiment, we consider a planning horizon of 8 h and a resource allocation interval of 1 h. All other settings are as per Table 3. Figure 10 shows the SLO violation percentages that result from the three resource allocation approaches.

As shown in Fig. 10a, all three resource allocation approaches estimate the same mean SLO violation percentage for the non-bursty exponential combination of workload scenarios. Thus it does not matter which resource allocation approach is used for exponential workload scenarios. However, the whole and basic interval approaches present resource allocation plans that cause very large SLO violations for the highly bursty combinations of workload scenarios. In particular, the plans estimated by both of these approaches cause mean SLO violation percentages of 155 and 226%, respectively, for the extremely bursty combination of workload scenarios while RAP's plan causes only a 71% mean violation. The mean SLO violation percentage of RAP is still high because a very tight constraint is enforced on the number of web server instances and database instance cores available per interval. A lower violation percentage can be achieved by relaxing the resource constraints.

Figure 10b provides a more detailed view of the extremely bursty combination of workload scenarios evaluated in Fig. 10a. This detailed view shows the individual violation percentages of the ten applications constituting that combination. From the figure, the whole approach achieves very low violation percentages for some workload scenarios while very high violation percentages for other workload scenarios. This behaviour stems from the way the whole approach works. As described in "Baseline methods" section, the whole approach allocates resources to an application over the whole planning horizon without taking into consideration burstiness in specific resource allocation intervals. Given the resource-constrained scenario shown in Fig. 10b, the whole approach exhausts the available resources quickly by giving a bulk of additional resources over the whole planning horizon to one application with the highest violation percentage at each decision stage. Therefore, the applications which are allocated resources in the early decision stages have a higher chance to obtain much lower violation percentages than the applications that are allocated resources in the decision stages that come later since most of the resources will be exhausted at these late decisions stages. This explains why some applications have very low violation percentages while others have very high violation percentages using the whole approach.

This is not the case with both basic interval and RAP approaches. Both basic interval and RAP-OneApp allocate resources on a resource allocation interval basis, which does not allow resources to be exhausted as quickly as is the case with the whole approach. Therefore, these two approaches achieve an SLO violation balance across the ten applications. However RAP has the added advantage of obtaining lower per-application violation percentages.

In summary, RAP can take advantage of burstiness characteristics of applications hosted on a cloud to suggest cost-effective resource allocations. In resource constrained scenarios, it can suggest strategies that lead to lower overall SLO violations. Furthermore, plans

Table 5 Arrival processes of workload scenario combinations

Session arrival process of each combination	S_a	I_a
Exponential	1	1
Medium bursty	3	100
High bursty	3	1000
Extreme bursty	3	10,000

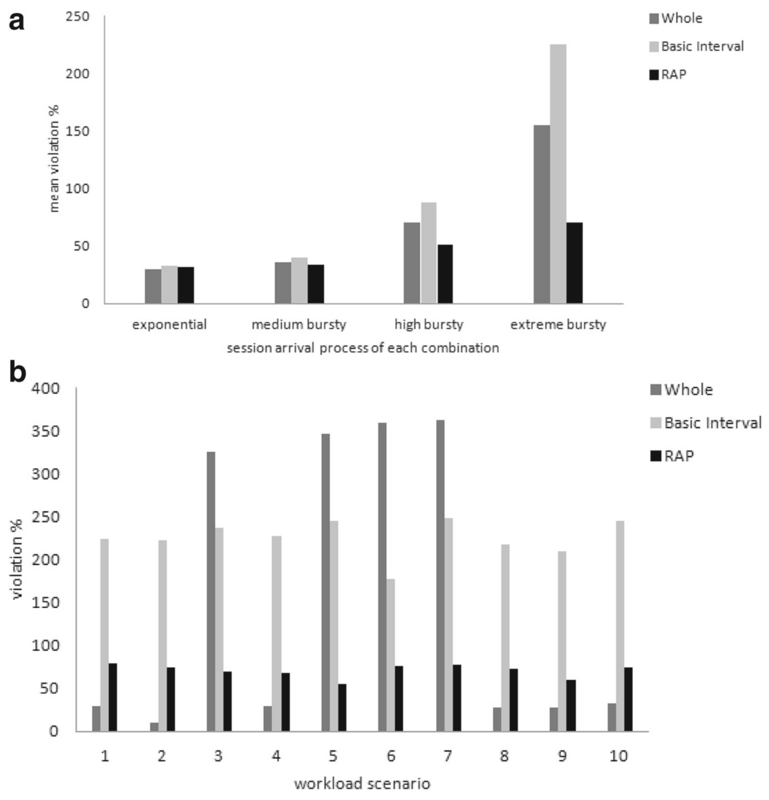

Fig. 10 SLO violation percentages. **a** Mean violation percentage. **b** Violation percentages of the extreme bursty combination

generated by RAP also balance SLO violations across applications. As a result, they can minimize the risk of SPs incurring penalties due to extreme SLO violations.

Comparison with utilization based SLP

We now investigate a technique that only considers resource utilization thresholds, e.g., a CPU utilization target, instead of SLOs based on response times. We use RAP as the allocation approach in all cases. However, we explore two different variants. We denote the approach taken in the previous sections where SLOs are specified based on mean response times as *RAP-Resp*. The other variant accepts an SLO threshold based on a mean resource utilization target of the bottleneck resource over the planning horizon. We refer to this as *RAP-Ux%* where x refers to the value of the threshold. Specifically, we experiment with x values of 30, 50 and 60%. We explore four different combinations of workload scenarios where each combination has five applications. Similar to the previous section, the four combinations have progressively higher degrees of burstiness. All other settings are as per Table 3. Figure 11 compares the total number of web server instances and database instance cores allocated using *RAP-Ux%* with different x values and *RAP-Resp*.

Figure 11 illustrates several potential problems with a utilization based approach for bursty workloads. From the figure, as expected, the lower the utilization target the higher the number of resources estimated by RAP. However, the number of resources estimated for a given utilization threshold remains the same regardless of the degree of burstiness in session arrivals. In contrast, RAP-Resp adapts the number of resources provisioned depending on the level of burstiness. This indicates that merely relying on utilization-based targets may not be an appropriate method for guaranteeing SLOs of applications with high degree of burstiness.

Figure 12 also shows the effect of using utilization-based targets on application response time violations. For each of the RAP-Ux% estimated allocation plans, we calculate mean response time SLO violations as defined in "Resource allocation planning (RAP)" section. RAP-Resp is not shown in the figure since it identifies a plan that eliminates SLO violations. From Fig. 12, response time violations increase with burstiness for a given utilization target. Furthermore, the degree of violation is larger for higher thresholds. From the figure, the likelihood of minimizing response time violations for bursty workloads increases if one were to choose very low utilization targets, i.e., less than 30%. However, such low utilization

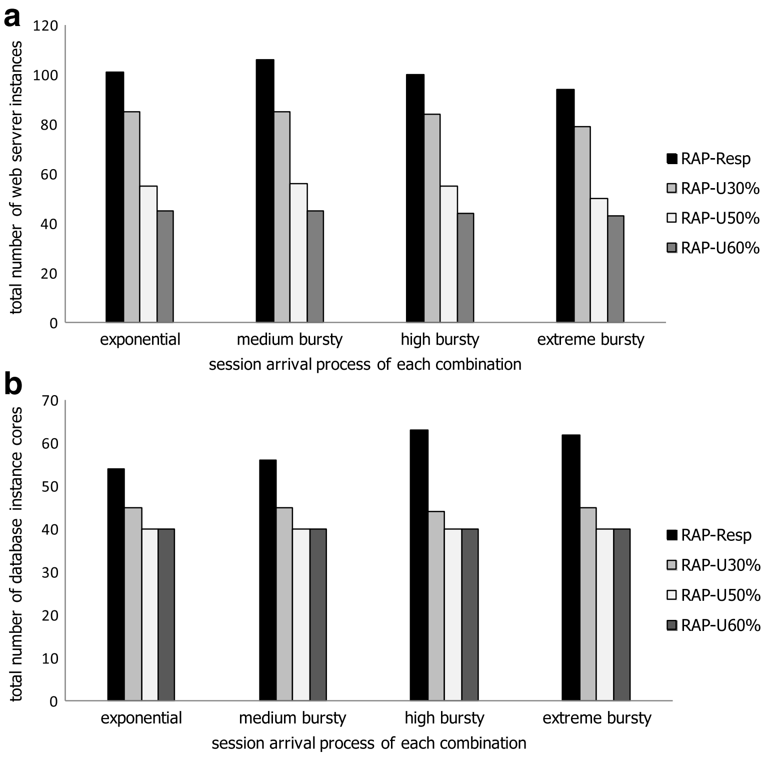

Fig. 11 Response time vs. Utilization based SLP. **a** Web server instances. **b** Database instance cores

targets may allocate excessive resources over the planning horizon thereby increasing costs.

In summary, these results show the challenges in meeting response time targets for applications with bursty workloads by controlling their resource utilizations. We argue that SLP frameworks must incorporate models that can help SPs directly specify response time based SLOs.

Related work

"Burstiness in enterprise workloads" section briefly discusses the prevalence of workload burstiness in enterprise applications. A description of related work in the areas of runtime resource management and offline capacity planning are provided in "Runtime resource management" and "Offline capacity planning" sections, respectively. Finally, a brief overview of current commercial capacity planning tools is provided in "Commercial capacity planning tools" section.

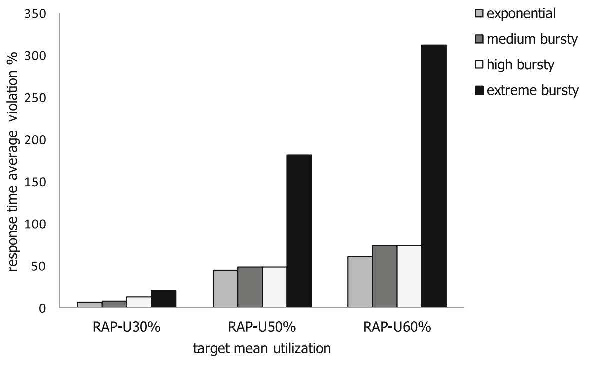

Fig. 12 Violations with utilization based SLP

Burstiness in enterprise workloads

Several studies have characterized the behaviour of enterprise application workloads [1, 2]. Many of these studies have indicated the presence of workload burstiness in real workloads [3–8]. Menascé et al. [2] characterized two weeks of activity at two real e-business sites. The authors reported very strong burstiness in the arrival of requests to these sites. Vallamsetty et al. [1] confirmed the above findings by characterizing traffic arriving at two other real e-commerce sites. This suggests that workload burstiness should be taken into consideration and exploited during resource planning and provisioning exercises [17].

Runtime resource management

Significant research efforts have focused on techniques that can dynamically reconfigure an application such that its performance objectives are satisfied even in the presence of workload fluctuations. These studies focus on computing resource allocations at runtime. This is in contrast to our work that is intended to be used by an SP to proactively determine resource allocations that exploit predictable patterns in application workload characteristics. Broadly, work in this area can be categorized into approaches that rely on queueing performance models and those that rely on control theory [21].

Queueing-model based techniques

Queueing-model based techniques employ a QNM to predict user-level SLO metrics such as throughput and response time of the application being managed. This performance model is usually embedded within a runtime controller that controls resource allocations to the managed application so as to satisfy its SLO targets. These techniques are based on the assumption that the application system being controlled is in a steady state and as a result they have been shown to be effective for decisions spanning medium term time horizons, e.g., 30 min [21].

Urgaonkar et al. proposed a performance-model based approach to dynamically reconfigure an application based on observed and predicted changes to its workload behaviour [22]. In contrast to our approach, this work is concerned with "local" optimization of a single application. Moreover, it does not take into account resource availability constraints and workload burstiness.

Li et al. [23–25] combined an extended QNM called a Layered Queueing Model (LQM) with bin-packing and a linear programming approach based on Network Flow Models (NFMs) for fast, optimal deployment of a given set of applications in a resource pool. Similar to our proposed work, the authors address computational complexity by formulating the NFM in a manner that permits a large number of applications to be handled. Unlike our work, their work does not consider the impact of bursty workloads.

Ardagna et al. proposed a runtime resource allocation scheduler that assigns multi-tier applications to physical resources at fine timescales to satisfy cost and performance objectives [26, 27]. The approach uses a multi-class QNM and a heuristic search algorithm that solves a mixed non-linear programming problem. The authors demonstrate that their solution techniques execute faster and provide better resource allocation solutions than traditional optimization techniques. Similar to the approach of Li et al., this work does not consider workload burstiness.

Control-theoretic techniques

Control-theoretic techniques employ an online feedback controller to adjust resource allocations in response to short time scale workload fluctuations. These techniques can accurately model system transients and adjust the system resource configuration within a very short time frame, e.g., a few minutes.

Kostentinos et al. derive runtime models that deduce the relationship between resource utilizations, data center power usage, and application performance [28]. The models are used to dynamically adapt CPU frequency, CPU usage, and memory usage of applications to meet power and performance requirements. In contrast to our work, the study did not focus on how burstiness impacts data center resource allocations.

Gmach et al. proposed an approach to integrate two types of controllers: a long-term workload placement controller and a short-term workload migration controller [29, 30]. Another integrated workload management architecture composed of multiple resource controllers was proposed by Zhu et al. to consolidate different application workloads having SLOs on a large data center [31]. In contrast to our work, both these approaches determined workload placements based on CPU utilization and did not consider the effect of the placements on response time. Lu et al. develop controllers that dynamically manipulate resource control settings offered by Virtual Machine Monitors (VMMs) in response to workload fluctuations [32]. While this work considers response time based SLOs, it does not focus on optimizing resource allocation by leveraging knowledge about application burstiness.

Malkowski et al. proposed a multi-model controller for dynamically provisioning Virtual Machines (VMs) to multi-tier applications in clouds [33]. A knowledge base is used to store all VM configurations encountered during previous deployments of the applications and the measured performance, e.g., request throughput, of such configurations. This knowledge base is used to drive future provisioning decisions. The authors point out that a large collection of performance data spanning multiple different configurations, workload conditions, and SLA specifications are needed for the knowledge base to be effective.

Caniff et al. presented an online resource provisioning algorithm, *Fastrack*, which exploits workload burstiness to guide dynamic resource allocation in multi-tier systems [17]. Fastrack detects the bursty periods in the application's workload and accordingly allocates more resources to these periods. This algorithm is quite similar to our work in trying to exploit workload burstiness to save the number of resources allocated. However, unlike our work which considers all applications in a cloud simultaneously while making resource allocation decisions, this work only focuses on a single application.

Offline capacity planning

While runtime resource provisioning techniques are important, they need to be complemented by offline techniques that provide pre-deployment insights to SPs [34]. A number of such offline approaches have been introduced in the literature [12, 19, 35–39].

Rolia et al. [19] proposed a capacity manager service, Quartermaster, for enterprise applications sharing a pool of resources. Quartermaster relies on historical traces of observed demands, e.g., CPU, disk, and network demands, of applications. Similar to our work, this technique exploits predictable patterns in historical traces of applications, e.g., time-of-the-day, day-of-the-week, and month-of-the-year patterns, to produce cost-effective resource estimates. Quartermaster allows resource utilization thresholds to be set as an indirect mechanism for achieving adequate application response times. Our work allows both utilization and response time thresholds to be specified directly as part of an application's SLO.

Jung et al. proposed an approach which combined a LQM with a heuristic optimization algorithm to generate optimal server configurations [36]. These configurations were then encoded in the form of rules and policies to be used while the system is running. In contrast to our work, the configuration generation exercise did not consider fine timescale resource allocation strategies that exploit workload burstiness. Furthermore, LQMs are not capable of reflecting the impact of burstiness unless they are integrated with a trace-based technique such as the WAM technique that we use in this paper.

Mylavarapu et al. proposed a Monte Carlo technique in conjunction with a genetic algorithm to obtain an optimized VM assignment scheme that ensures peak application demands are satisfied while avoiding over-provisioning of resources [37]. In contrast to our study, this work only considers SLOs based on CPU utilization targets.

Meng et al. proposed a trace-based approach for capacity planning in compute clouds through VM multiplexing [39]. Instead of assigning VMs on a one-by-one basis to each application workload, a joint-VM is allocated to accommodate a group of application workloads at a time. The applications to group within a VM are selected such that their workload peaks do not overlap with one another. This approach is similar to our work in the sense that workload properties of applications are exploited to drive the SLP process. However, this work only considered VM CPU service demands as the workload metric of interest while our work considers several service and arrival process parameters that are of importance to SLP.

Finally, as the problem of resource scheduling in cloud environments is seen as NP-hard, approaches such as genetic algorithms, particle swarm optimization, and ant colony optimization have been found to be effective [40]. These algorithms are similar to RAP in the sense that they can obtain near optimal solutions without exhaustively enumerating all possible solutions. However, to the best of our knowledge, these approaches have not explicitly considered the impact of burstiness.

Commercial capacity planning tools

Various commercial offline capacity planning tools have been offered by different vendors. Examples of such tools include *VMware Capacity Planner* [41] , *NetIQ PlateSpin Recon* [42], and *Lanamark Suite* [43]. To the best of our knowledge based on their data sheets, these tools cannot support SLP based on response time targets. Moreover, they do not explicitly consider SLP for complex workloads characterized by burstiness.

Conclusions and future work

This paper presents RAP, a technique that can allow cloud SPs to assign resources to applications to meet SLOs. RAP exploits the predictable, bursty workload patterns experienced by enterprise applications. Given traces that capture the workload behavior of a set of applications, RAP automatically discovers time varying resource allocation plans that globally minimize SLO violations across the applications. In particular, RAP exploits burstiness to suggest plans that use lesser resources to satisfy SLOs than burstiness agnostic techniques. In resource constrained scenarios, RAP balances out SLO violations uniformly across applications thereby reducing the risk of SPs incurring large penalties due to extreme violations. We also show that the approach outperforms the prevalent practice of enforcing resource utilization thresholds.

We develop several variants of RAP that provide mechanisms to trade off optimality for computational efficiency. The RAP-IE variant can identify the optimal resource allocation plan. Our experiments indicate that RAP-AllApps can identify the optimal plan when there is no bottleneck switch across the resource allocation intervals. For other scenarios, it is able to identify close to optimal plans. The RAP-OneApp variant is more computationally efficient than the other variants and is hence better suited for analyses involving a large number of applications. For all the scenarios we study, RAP-OneApp's solutions are very close to those identified by the other two variants.

RAP requires a performance model that can quantify the impact of scaling strategies. As an example, we have used a model that can look at either horizontal scaling or vertical scaling. If a model can capture the impact of hybrid scaling, i.e., simultaneously scaling vertically as well as horizontally, then RAP algorithms can leverage such a model for resource allocation planning. We also note that scaling strategies can incur overheads, e.g., adding cores to a database server might likely require restarting the database service thereby negatively affecting the application's performance. RAP can take into account such scenarios by using a performance model that captures the impact of such overheads.

An SP can deploy applications on the cloud based on the estimates provided by RAP and monitor SLO violations over time. If any changes occur in application workloads after deployment leading to SLO violations, revised traces in addition to any changes in resource instance limits can be input to RAP again to consider changes in resource allocations. This can also be used to negotiate new SLAs with subscribers to reflect workload changes. We also note that SLO violations predicted by RAP need to be compared continually with those observed under deployment. If discrepancies are related to the performance model, then the model needs to be calibrated to offer better SLO predictions.

Future work will focus on refining RAP in a number of ways. Currently, RAP avoids over provisioning resources by allocating them in increments of one. Future work will address more explicit ways to specify resource constraints, e.g., enforcing a budget based on the costs for different types of resource flavours. We will also incorporate constraints that minimize the number of resource instance migrations across applications. In this paper, we do not address the objective of balancing the number of resources among the applications for the sake of fairness [44, 45]. Objective functions based on fairness measures will be investigated in future work. We will also modify RAP to accommodate multiple resource flavours for a given tier.Finally, future work will explore experimental validation using benchmark multi-tier applications.

Authors' contributions
All authors read and approved the final manuscript.

Competing interests
The authors declare that they have no competing interests.

Publisher's Note
Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Author details
[1]Computer Science Department, Faculty of Computers and Information, Menoufia University, Menoufia, Egypt. [2]Department of Electrical and Computer Engineering, University of Calgary, Calgary, Canada.

Received: 6 October 2016 Accepted: 24 July 2017
Published online: 15 August 2017

References

1. Vallamsetty U, Kant K, Mohapatra P (2002) Characterization of E-commerce Traffic. In: Proceedings of IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems, (WECWIS). IEEE, Newport Beach, pp 137–144
2. Menascé D, Almeida V, Riedi R, Ribeiro F, Fonseca R, Meira Jr W (2000) In search of invariants for e-business workloads. In: Proceedings of the ACM Conference on Electronic Commerce, (EC). ACM, Minneapolis, pp 56–65
3. Casale G, Mi N, Smirni E (2010) Model-driven system capacity planning under workload burstiness. IEEE Trans Comput 59(1):66–80
4. Mi N, Zhang Q, Riska A, Smirni E, Riedel E (2007) Performance impacts of autocorrelated flows in multi-tiered systems. Perform Eval 64(9-12): 1082–1101
5. Casale G, Mi N, Cherkasova L, Smirni E (2012) Dealing with burstiness in multi-tier applications: models and their parameterization. IEEE Trans Softw Eng 38(5):1040–1053
6. Casale G, Mi N, Smirni E (2008) Bound analysis of closed queueing networks with workload burstiness. In: Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems. ACM, Annapolis, pp 13–24
7. Xue J, Yan F, Birke R, Chen LY, Scherer T, Smirni E (2015) Practise: Robust prediction of data center time series. In: Proceedings of the International Conference on Network and Service Management (CNSM). IEEE, Barcelona, pp 126–134
8. Birke R, Chen LY, Smirni E (2015) Usage patterns in multi-tenant data centers: a large-case field study. In: Handbook on Data Centers. Springer, New York, pp 1257–1266
9. Krishnamurthy D,Rolia J,Xu M (2011) WAM -the weighted average method for predicting the performance of systems with bursts of customer sessions. IEEE Trans Soft Eng 37(5):718–735
10. Youssef A, Krishnamurthy D (2013) Cloud service level planning under burstiness. In: Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS). IEEE, Toronto, pp 107–114
11. Youssef A, Krishnamurthy D (2011) A trace-based service level planning framework for enterprise application clouds. In: Proceedings of the International Conference on Network and Service Management (CNSM). International Federation for Information Processing, Paris, pp 422–426
12. Gmach D, Rolia J, Cherkasova L, Kemper A (2007) Capacity management and demand prediction for next generation data centers. In: Proceedings of IEEE International Conference on Web Services (ICWS). IEEE, Salt Lake City, pp 43–50
13. Menascé DA, Almeida VA, Dowdy LW, Dowdy L (2004) Performance by design: computer capacity planning by example. Prentice Hall Professional, Upper Saddle River
14. Lazowska ED, Zahorjan J, Graham GS, Sevcik KC (1984) Quantitative System Performance: Computer System Analysis Using Queueing Network Models. Prentice-Hall, Inc., Upper Saddle River
15. Denning PJ, Buzen JP (1978) The operational analysis of queueing network models. ACM Comput Surv 10(3):225–261
16. Gusella R (1991) Characterizing the variability of arrival processes with indexes of dispersion. IEEE J Sel Areas Commun 9(2):203–211
17. Caniff A, Lu L, Mi N, Cherkasova L, Smirni E (2010) Fastrack for taming burstiness and saving power in multi-tiered systems. In: International Teletraffic Congress (ITC). IEEE, Amsterdam, pp 1–8
18. Ariltt M, Krishnamurthy D, Rolia J (2001) Characterizing the scalability of a large web-based shopping system. ACM Trans Internet Technol 1(1):44–69
19. Rolia J, Cherkasova L, Arlitt M, Andrzejak A (2005) A capacity management service for resource pools. In: Proceedings of the International Workshop on Software and Performance (WOSP). ACM, Palma, pp 229–237
20. Youssef A (2014) Burstiness and uncertainty aware service level planning for enterprise clouds. PhD thesis, University of Calgary
21. Tanelli M, Ardagna D, Lovera M, Zhang L (2008) Model identification for energy-aware management of web service systems. In: Proceedings of the International Conference on Service-Oriented Computing (ICSOC), pp 599–606
22. Urgaonkar B, Shenoy P, Chandra A, Goyal P (2005) Dynamic provisioning of multi-tier internet applications. In: International Conference on Autonomic Computing (ICAC). IEEE, Seattle, pp 217–228
23. Li J, Chinneck J, Woodside M, Litoiu M (2009) Fast scalable optimization to configure service systems having cost and quality of service constraints. In: Proceedings of the International Conference on Autonomic Computing (ICAC). ACM, Barcelona, pp 159–168
24. Li J, Chinneck J, Woodside M, Litoiu M, Iszlai G (2009) Performance model driven qos guarantees and optimization in clouds. In: Proceedings of ICSE Workshop on Software Engineering Challenges of Cloud Computing (CLOUD). IEEE Computer Society, Vancouver, pp 15–22
25. Li J, Woodside M, Chinneck J, Litoiu M (2011) CloudOpt: multi-goal optimization of application deployments across a cloud. In: International Conference on Network and Service Management (CNSM). International Federation for Information Processing, Paris, pp 1–9
26. Ardagna D, Trubian M, Zhang L (2007) SLA based Resource Allocation Policies in Autonomic Environments. J Parallel Distrib Comput 67(3):259–270
27. Ardagna D, Panicucci B, Trubian M, Zhang L (2012) Energy-Aware Autonomic Resource Allocation in Multitier Virtualized Environments. IEEE Trans Serv Comput 5(1):2–19
28. Kostentinos Tesfatsion S, Wadbro E, Tordsson J (2016) Autonomic resource management for optimized power and performance in multi-tenant

clouds. In: Proceedings of the IEEE International Conference on Autonomic Computing (ICAC 2016). IEEE, Wurzburg
29. Gmach D, Rolia J, Cherkasova L, Belrose G, Turicchi T, Kemper A (2008) An integrated approach to resource pool management: policies, efficiency and quality metrics. In: IEEE International Conference on Dependable Systems and Networks (DSN). IEEE, Anchorage, pp 326–335
30. Gmach D, Rolia J, Cherkasova L (2009) Satisfying service level objectices in a self-managing resource pool. In: IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO). IEEE, San Francisco, pp 243–253
31. Zhu X, Young D, Watson B, Wang Z, Rolia J, Singhal S, McKee B, Hyser C, Gmach D, Gardner R, Christian T, Cherkasova L (2008) 1000 islands: integrated capacity and workload management for the next generation data center. In: Proceedings of the International Conference on Autonomic Computing (ICAC). IEEE, Chicago, pp 172–181
32. Lu L, Zhu X, Griffith R, Padala P, Parikh A, Shah P, Smirni E (2014) Application-driven dynamic vertical scaling of virtual machines in resource pools. In: 2014 IEEE Network Operations and Management Symposium, NOMS 2014, Krakow, Poland, May 5-9, 2014. IEEE, Krakow, pp 1–9
33. Malkowski S, Hedwig M, Li J, Pu C, Neumann D (2011) Automated control for elastic n-tier workloads based on empirical modeling. In: Proceedings of the International Conference on Autonomic Computing (ICAC). ACM, Karlsruhe, pp 131–140
34. Hyser C, McKee B, Gardner R, Watson B (2007) Autonomic virtual machine placement in the data center. Technical Report HPL-2007-189, HP Labs
35. Anselmi J, Amaldi E, Cremonesi P (2008) Service consolidation with end-to-end response time constraints. In: Euromicro Conference on Software Engineering and Advanced Applications (SEAA). IEEE, Parma, pp 345–352
36. Jung G, Joshi K, Hiltunen M, Schlichting R, Pu C (2008) Generating adaptation policies for multi-tier applications in consolidated server environments. In: International Conference on Autonomic Computing (ICAC). IEEE, Chicago, pp 23–32
37. Mylavarapu S, Sukthankar V, Banerjee P (2010) An optimized capacity planning approach for virtual infrastructure exhibiting stochastic workload. In: Proceedings of the ACM Symposium on Applied Computing (SAC). ACM, Sierre, pp 386–390
38. Gmach D, Rolia J, Bash C, Chen Y, Christian T, Shah A, Sharma R, Wang Z (2010) Capacity planning and power management to exploit sustainable energy. In: International Conference on Network and Service Management (CNSM). IEEE, Niagara Falls, pp 96–103
39. Meng X, Isci C, Kephart J, Zhang L, Bouillet E, Pendarakis D (2010) Efficient resource provisioning in compute clouds via vm multiplexing. In: Proceedings of International Conference on Autonomic Computing (ICAC). ACM, Washington, pp 11–20
40. Zhan ZH, Liu XF, Gong YJ, Zhang J, Chung HS-H, Li Y (2015) Cloud computing resource scheduling and a survey of its evolutionary approaches. ACM Comput Surv 47(4):63–16333
41. VMware Inc., VMware Capacity Planner. http://www.vmware.com/products/capacity-planner.html. Accessed 17 Sept 2016
42. NetIQ PlateSpin Recon. https://www.netiq.com/products/recon/. Accessed 17 Sept 2016
43. Lanamark Suite. http://www.lanamark.com/product/overview. Accessed 17 Sept 2016
44. Ghodsi A, Zaharia M, Hindman B, Konwinski A, Shenker S, Stoica I (2011) Dominant resource fairness: Fair allocation of multiple resource types. In: NSDI, Vol. 11. USENIX Association, Boston, pp 24–24
45. Wang H, Varman PJ (2014) Balancing fairness and efficiency in tiered storage systems with bottleneck-aware allocation. In: FAST. USENIX Association, Santa Clara, pp 229–242